数理解析研究所講究録
941 巻 1996 年 113-122

14.

Massively Parallel Computer Algebra for Control Engineering by Using SIMD parallel computer

Osami SAITO(豊橋技科大)
Atsushi Watanabe(〃)
Taiichi YUASA(〃)

14.1 Introduction

The various general computer algebra systems such as REDUCE, Mathematica and Maple have been released. At present they are applied to many fields in science and engineering. When they are applied to the complex problems, the ability of conventional computer algebra system, specially computing speed is considered to be not enough. One idea for improving the speed, is to parallelize the computer algebra functions.

In the last few years, there exist few literatures[1] which studied the parallel computer algebra from view point of parallel algorithm and parallel processing. However almost studies realized the distributed system by connecting many conventional computer algebra systems with the network.

This paper deals with the massively parallel computer algebra system by using SIMD parallel computer. In this study, the parallel computer algebra system is considered consistently through the parallel computer architecture, the parallel symbolic language and the parallel algorithms. And the original massively parallel computer algebra system is realized based on SIMD parallel computer and parallel LISP which are implemented by our group.

The conventional general systems cover many functions such as integration, differentiation of functions, matrix operation and so on. In this study, at first the massively parallel functions are developed for determinant of matrix of which element consists of multi-variable polynomial. And it is extended to the functions of inverse matrix. The realized functions are evaluated in computing speed, comparing with sequential calculation in each problem.

14.2 SIMD parallel computer and parallel LISP

The parallel computer adopted here, is SM-1 which is designed and fabricated by one of outhers, YUASA. And the parallel LISP is TUPLE (Toyohashi University Parallel Lisp Enviroment). This chapter explains the outline of SM-1 and TUPLE.

SM-1 is designed based on SIMD architecture. It has the front end processor (FE) and 1024 number of the processing element (PE). The single instructions are broadcasted from FE to PE's and PE's execute them parallelly with different data.

The SPARCStaion is used as the FE. 1024 number of PE are allocated as (32*32) 2-dimensional mesh array. FE and PE array are connected by the coprocessor interface of SPARCStation.

Each PE has an 8-bit ALU, 20 bytes of resister file , a 64 bit shift register, 160 bytes of RAM and 1 Mbyte of off-chip DRAM. As SM-1 has 1 Mbyte huge local memories, it is enabled to develop here the massively parallel computer algebra. Each PE has the number (processor number) from 0 to 1023 in register pn for identifing PE.

Our system is written by TUPLE, which is the extended parallel Common LISP for the SIMD computer. The features of TUPLE are executing functions parallelly and programming as same as conventional Common Lisp. TUPLE consists of FE functions and PE functions. The FE functions have a full-set Common Lisp and run on FE. The PE functions, running on each PE, have a subset of Common Lisp and are similar to FE ones.

TUPLE has also local communication function, connecting each PE to its four nearest neighbors and global one , communicating with assigned PE's. The local communication can save much time comparing with the global one. In chapter 5, the communication time will be big problems. The readers who is interesting in SM-1 architecture and /or TUPLE, can refer the book[2].

14.3 Massively Parallel Algorithms

In this chapter, the massively parallel algorithms for SIMD parallel computer SM-1 are ex-

plained. They are designed fundamentally based on the concept models of SIMD parallel computer -PRAM- (Parallel Random Access Machine).

14.3.1 Determinant Algorithms

Here two parallel algorithms for calculating the determinant of matrix of which elements consiste of mutivariable polynomials, are explained. First, Laplace method is adopted for the massively parallel algorithm. Hereafter it is called as DL algorithm. It is wellknown that the determinant is expanded as

$$|A_n| = \sum_{l=1}^{n} (-1)^{k+l} a_{kl} |A_{n-1}^{kl}| \tag{1}$$

where n is matrix size,a_{kl} is (k,l) element,A_{n-1}^{kl} means (n-1) dimensional square matrix which is obtained by removing k-th column and l-th row. By using this method, the calculation of n-d(imensional) matrix determinant is divided into n times calculations of (n-1)-d matrix determinant. This expansion as (1) is named as level 1 division. As SM-1 has many PE's, the division are continued. At level 2, Laplace method is applied again to (n-1)-d matrix determinant. Consequently n-d matrix determinant is divided into n*(n-1) number of (n-2)-d matrix determinants. The process is continued and finally it goes to n! number of scaler matrix ones at level (n-1). This is called as the topdown process. After the topdown process, the unification (bottomup) process is executed, following up inversely the topdown process. That is , (n-1)! number of 2-d matrix determinants are calculated by using the level n-1 results. And successively this process are continued and finally $|A_n|$ are obtained.

This algorithm consists of two processes, division of matrix (topdown) and unification of results (bottomup). The above processes are performed parallelly with PE's as much as possible in order to derive the ability of SM-1. As the SIMD parallel computer SM-1 has the limitation such that each PE executes the same process (Single Instruction) with different data (Multi Data), elaborate processes are necessary to perform the above algorithms on SM-1. It is carefully designed how to make the different matrices in the right hand of (1) and unify the results by using same instruction with only key of processor number pn on each PE. First the topdown process is explained as follows. The n! number of PE are prepared because the parallel computation of n-d matrix determinant is divided into n! parallel computations at final level (n-1). At level 0, all data of are broadcasted from FE to n! PE's with the network between FE and PE's. At level 1,$|A_n|$ is expanded with respect to 1 column with (1). At this level, (n-1)! number of PE, which have pn number from 1 to (n-1)!, obtain the same (n-1)-d matrix A_{n-1}^{11}. And (n-1)! number of PE

from (n-1)!*j+1 to (n-1)!*(j+1) do similarly A^{1l}_{n-1}. PE's which make same matrix, are considered to belong to the same group. Then n! PE's are divided into n groups. The new number from 1 to (n-1)!, called as GroupNum, is assigned to each PE, belonging to same group. The PE, which has GroupNum 1, is called as a parent PE and the pn number of a parent PE is described as LPN in the following. The parent PE will become the root of calculation of $|A^{1l}_{n-1}|$. At level i, n! PE's are divided into n!/(n-i)! groups and GroupNum of each group become 1 though (n-i)!. Also the PE of GroupNum 1 is a parent PE and the root of $|A_{n-1}|$. This process is continued recusively and at level n-1, n!/2 groups are obtained. In each group, 2-d matrix is divided into scalar matrices The topdown process is finished and the bottomup process is began as follows. At level n-1, each PE has the scaler matrix. As the determinant of scaler one is equal to the element, each PE calculates $(-1)^{1+l}a_{11}a_{22}$ or $(-1)^{1+2}a_{12}a_{12}$ and sends it to a parent PE of the group to which the PE belongs at level n-2 with the global communication. The communicated PE is determined with data of GroupNum and LPN. The parent PE performs the addition of sent results and obtains $|A_2|$. Continuing these processes, it reaches the level 1. The pn number 1 becomes the parent PE at level 1 and $|A_n|$ is obtained.

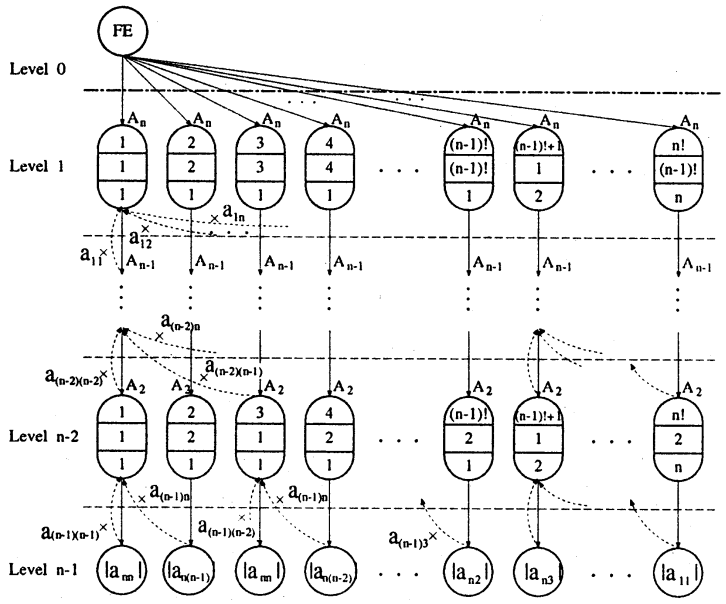

図 1 schematic diagram of massively parallel determinant computation with DL algorithm

Fig. 1 shows schematically the above topdown and bottomup processes. Solid and dashed arrows mean topdown and bottomup processes respectively. The circulus are PE's and their upper , middle and lower numbers are pn numbers, GroupNums and the removed columns numbers respectively. As the bottomup process is going up, it is necessary to add more polynomials. The

addition is not executed sequentially with one PE. it is done by using many PE's. Each PE adds two polynomials and the results are unified successively with addition of two polynomials as binary tree. That is because the addition of polynomials consists of processing complex lists and takes much time. If the processing is done parallelly, much time can be saved. Next the another parallel algorithm for determinant is considered in the follows. The definition of determinant is given as

$$|A_n| = \sum_{e(j_1,j_2,\cdots,j_n)} e(j_1,j_2,\cdots,j_n)a_{aj_1}a_{2j_2}\cdots a_{nj_n} \tag{2}$$

The parallel algorithm is designed based on the above definition and called as DD algorithm. It also consists of the topdown and the bottomup processes. Because the right hand of (4) has n! terms, n! PE's are prepared. First, all data of are broadcasted from FE to n! PE's. The topdown process begins. In each PE, the permutation $(j_1, j_2, \cdots, j_n)$ is generated with the PE number pn as a key and pick up the elements as $a_{1j_1}, a_{2j_2}, \cdots, 'a_{nj_n}$ from A_n. Subsequently multiplication of n terms,$e(j_1, j_2, \cdots, j_n)a_{1j_1}a_{2j_2}\cdots a_{nj_n}$ is performed parallelly and n! terms are obtained simultaneously. The bottomup processes are adding results which all PE's have. Because this bottomup process consists of only addition, the shufulexchange connection is applicable in this algorithm. It is implemented based on so-called reduction function in SIMD parallel computer which can use shuffle-exchange network and consequently save the PE's communication time.

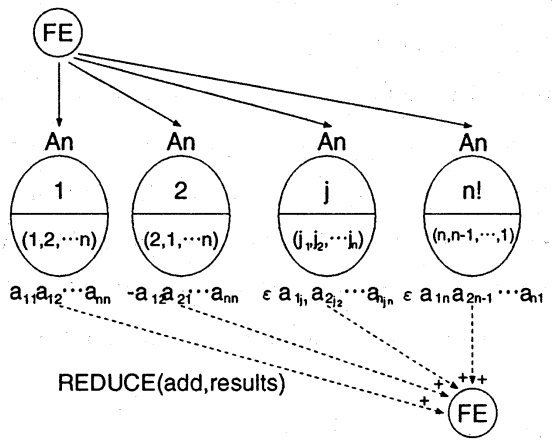

図 2 Massively parallel determinant computation with DD algorithm

Fig.2 shows schematically the DD algorithm.

14.3.2 Inverse algorithms

The calculation of inverse matrix of multivariable polynomial is mentioned. Usually, Gaussian method is applied to inverse matrix in numerical computation. Considering here the parallelizing, inverse matrix with the adjoint matrix is adopted, given as

$$A_n^{-1} = \frac{adj\ A_n}{det\ A_n} \tag{3}$$

(3) needs determinant calculations. Then they are calculated by applying the above parallel calculations based on DL or DD algorithms. In (3) the numerator and denominator polynomials have common factors occasionally. So the greatest common divisor(GCD) of both polynomials must be calculated with Euclidean method. It consists of polynomial division. In our system pseudo-division of polynomials is applied. Then number of PE calculate parallelly the GCD of each element of inverse matrix.

14.4 Implementation

The parallel algorithms mentioned in chapter 3 are implemented on SM-1 as follows. The computer algebra system is generally implemented as list processing. The canonical expression of multivariabel polynomial in our system is represent by the list structure, given as

(*poly*VAR(E1,C1)(E2,C2)....(En,Cn)),

where poly is a tag for recognizing a polynomial, VAR means the maine variable of polynomial, Ei (E1¿E2¿....¿En) is the exponent of power product and Ci is coefficient of power. The variable has the lexicographical ordering as A¿B¿....¿Z. Our parallel system is implemented by using parallel lisp -TUPLE-, based on the above list data structure. The various problems must be considered in programming the above parallel algorithms with TUPLE. Here two problems i.e. the parallel programming technique and the limitation of the number of PE are explained.

14.4.1 Technique of parallel programming

The parallel lisp programming for SIMD parallel computer must be carefully implemented because the active PE's and inactive PE's exist mixeturely in parallel programming. It is seemed that all PE's work simultaneously in SIMD computer. But in practice, the situation that some

PE's are active and other PE's are inactive, occurs depending on the programming. Then the parallel programming must be written for avoiding the situation as much as possible.

14.4.2 Limitation of PE number

The parallel algorithm in chapter 3 are designed without considering the limitation of PE number. However SM-1 has 1024 PE's actually. Then it can be applied directly to the small size problem,but not to huge size one. If the necessary PE number is over 1024 at level i, the division process is stopped at level i-1. That is , if the inequality

$$\frac{n!}{i!} < 1024 < \frac{n!}{(i-1)!} \tag{4}$$

is satisfied at level i, each PE calculates sequentially the determinant of (n-i)-d matrix. And the bottomup process is started. In the case of n¡7, the division process can be performed to level (n-1) and the scaler matrix can be obtained. But in our system, the process is stopped at level (n-2) and the bottomup process is started after calculating determinant of 2-d matrix for saving the communication time.

In applying the DD algorithm, the another approach must be considered. The program must be improved so that some PE's play no less than one PE, if the necessary PE number is over 1024. At present our system has been implemented with this approach. However, the programming load increases. Now it is trying to be revised by using the virtual processing element. The idea of virtual PE is kept for the next issue.

14.5 Results and Evaluation

This chapter evaluates how much the computation speed can be improved by using massively parallel computer. At first, the case of determinant with the DL algorithm is investigated.

Fig.3 depicts the ratio of the sequential computation time to the parallel one. The vertical and horizontal axis are the ratio and matrix dimension respectively. The value of each point is the average of 5 examples. The computation time is measured without the garbage collection time, because it deeply depend on the current computer situation and isnt constant. The sequential computation is executed with one PE.

A solid line is the result in which all elements of matrix consist of the numeric. In the case, the availability of parallel computation is increasing as the matrix dimension becomes lager and

the about 500 time ratio is realized in 9-d matrix. A dashed line is the case in which elements are linear polynomials of one variable. A chain line is the quadratic polynomial of one variable. And also a pointed chain line is the quadratic polynomial of two variables. The under numbers of horizontal axis mean the used PE numbers.

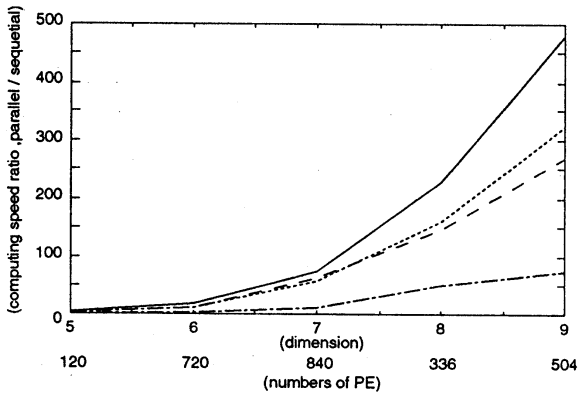

図 3 Speed ratio of parallel to sequential in determinant computation

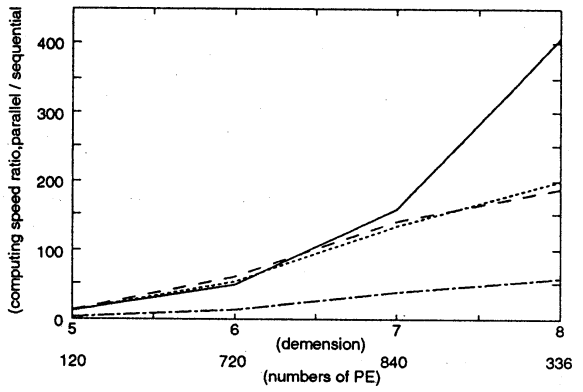

図 4 Speed ratio of parallel to sequential in inverse matrix computation

Fig.4 depicts the results of inverse matrix with the DL algorithm. The shaPE's of graph are similar to Fig.3, because the algorithm is same as the determinant except for GCD calculation. The determinant calculation of lager dimensional matrix than 9-d can not be executed because of lack of local memory capacity. Then Fig.3 and Fig.4 show the results through 9-d or 8-d matrices.

The above results clarify that the parallelizing of computer algebra can save the computing time. For example, the ratio of 9-d matrix determinant in the case of quadratic polynomials becomes 80 times.

Lets investigate the results more precisely. The parallel computation can realized about only

one hundred time speed ratio although 1024 PE's are applied. The results are not so far as the our expected value. The reason of this problem is considered to be the overhead communication time in parallel computation.

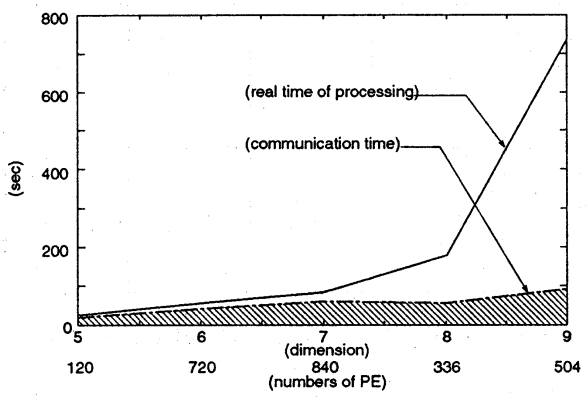

図 5 Communication time to total processing time

Fig.5 shows real data how much time the communication occupies to the total processing time, in the case of matrix determinant of quadratic polynomials of two variables. The vertical axis is time(sec) and the horizontal axis is matrix dimension. In the figure, a solid and a dashed lines mean the total processing time and the communication time respectively. The reason why the communication spends so much time, is that the bottomup process in LD has to use the global communication between PE's. It needs more time than the local communication or the shuffle exchange communication.

So the DD algorithm which can use the shufull exchange communication, is investigated. The function based on the DD algorithm is applied to the some examples and the computation times are measured.

Table 1 lists the results of comparing DL and DD functions. The numeric of each column means the computing speed ratio of DD function to the DL function in calculating the matrix determinants of numeric and quadratic polynomials of two variables. The results show that the DD function can be executed two times as faster as the DL one because the shuffle exchange can save the communication time in the bottomup process of DD algorithm. It is concluded that massively parallel computer algorithm must be designed in considering the PE's communication time.

Table 1 lists only through 4 to 6 dimensional cases because 7-d case needs 5040 PE's and consequently DD algorithm can not be applied directly to more than 7-d matrices. From only view point of PE's communication, DD is more suitable to massively parallel computation than

DL. However DL can deal more easily with the limitation of PE's number than DD. Then they have both merits and demerits.

element \ degree	4	5	6
numerical ratio(DL/DD)	1.807	2.113	3.186
two variable polynomial ratio(DL/DD)	2.151	2.668	2.542

表 1 Computing speed ratio of DD to DL

Though the detail of parallel programming is here omitted, the programming with care of parallel programming technique can execute in 20

14.6 Conclusion

This paper deals with the development of massively parallel computer algebra system. The effectiveness of realized parallel functions is evaluated, comparing with the sequential computation. And it can be concluded that the parallel computation is available for computer algebra. At present our system has only few functions. In near future it will be extended to essential functions of computer algebra.

References

[1]Siegl,K.:Parallelizing Algorithms for Symbolic Computation using MAPLE,In 4th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming,pp 179-186 (1993)

[2]T.YUASA and et.al.:SM-1 and Its Language System, Parallel Languages and Compiler Research in Japan, Ed.by Nicolau and Sato, M.Kluwer Academic Press 1995