Proceedings of the 8th WSEAS International Conference on APPLIED MATHEMATICS, Tenerife, Spain, December 16-18, 2005 (pp43-48)

Parallelization of a Method for the Solution of the Inverse Additive Singular Value Problem

GEORGINA FLORES-BECERRA[†‡], VICTOR M. GARCIA[†] and ANTONIO M. VIDAL[†]
†Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia
SPAIN
‡Departamento de Sistemas y Computación
Instituto Tecnológico de Puebla.
Av. Tecnológico 420, Colonia Maravillas, C.P. 72220, Puebla
MEXICO
gflores@dsic.upv.es, vmgarcia@dsic.upv.es, avidal@dsic.upv.es

Abstract: - This paper describes the parallelization of a method (proposed by Chu in [7]) to solve the Inverse Additive Singular Value Problem (IASVP). The IASVP is a problem whose solution requires a high computational cost, both in time and in memory. For example, the complexity of Chu's method is $O(n^4)$ in time and $O(n^3)$ in memory. Using parallel computing, the time needed to solve the problem has been substantially reduced. The parallel algorithm developed has shown good experimental performance, confirming the theoretical performance predicted and showing an acceptable scalability.

Key-Words: - Inverse Singular Value Problems, Parallel Algorithms, Newton-type methods

1 Introduction

The Inverse Eigenvalue Problem (IEP) and the Inverse Singular Value Problem (ISVP) appear in many science and engineering problems, such as medical tomography, image processing, circuit design or curve fit [6,11,12]. Both problems have as goal the reconstruction of a matrix with given structure and with pre-established eigenvalues or singular values. A particular case of the ISVP is the Inverse Additive Singular Value Problem, defined by Chu as [7]:

Given n+1 real $m \times n$ matrices $A_0, A_1, ..., A_n$ ($m \geq n$) and a set of real numbers $\sigma^* = \{S^*_1, S^*_2, ..., S^*_n\}$, where $S^*_1 \geq S^*_2 \geq ... \geq S^*_n$, find a real vector $c = [c_1, c_2, ..., c_n]^t$, such that σ^* are the singular values of

$$A(c) = A_0 + c_1 A_1 + ... + c_n A_n \qquad (1)$$

The IASVP can be stated as a system of nonlinear equations which can be solved with iterative Newton-like methods. This was done in [10], adapting the so-called Method I proposed by Friedland et al., to solve the IEP [11]. The IASVP can be formulated as well as a minimum square problems; a technique solution can be found in [10]. This solution is based on the Lift&Project method, proposed by Chen et al. in [4] to solve the IEP.

A different type of method for the solution of the IASVP was developed by Chu in [7]; we denote this method as MIII in this paper. MIII generalizes an iterative process described first by Friedland et al., in [11] to solve the ISVP. This is an iterative Newton-like method, of fast convergence [2] and high accuracy. However, as with most Newton-like algorithms, its convergence relies heavily on the quality of the initial approximation.

MIII has been experimentally tested in [2,7] for problems of size m=5, n=4 and in [9] for problems of sizes $5 \leq m = n \leq 50$, using sequential algorithms. Experimental tests with larger sizes (for example $O(10^2, 10^3)$) would have very long execution times, since MIII has high complexity in time ($O(n^4)$) and in space ($O(n^3)$).

The goal of this paper is the design of a parallel version of the MIII method, so that larger problems can be solved. This parallel algorithm complete a set of parallel algorithms based on Newton-type methods to solve the IASVP [9,10]. The main idea is to incorporate the parallel MIII algorithm to a specialized library for the resolution of the IEP and the ISVP. This library is currently in the design and development stages.

The MIII method shall be briefly described in Section 2. In Section 3 we will describe the parallel SPMD algorithm; it has been implemented on a distributed memory architecture. In Section 4 the theoretical performance of the algorithm is discussed, and some numerical results are presented. Finally, in Section 5 we give our conclusions.

2 Method MIII

The basic operation of the MIII method [7] consists of finding the intersection between the set $G(\sigma^*)$ of the matrices whose singular values are the set σ^*, and the set $L(c)$ of the matrices that can be written as in (1). $G(\sigma^*)$ and $L(c)$ can be expressed as:

$G(\sigma^*) = \{US^*V^t \mid U \in \Re^{mxm}, V \in \Re^{nxn}, \text{orthogonal}\}$

$L(c) = \{A(c) \mid c \in \Re^n\}$,

where $S^* = \text{diag}(S^*_1, S^*_2, ..., S^*_n)$.

MIII finds an approximation to the intersection of $G(\sigma^*)$ and $L(c)$ through an iterative method with two distinct stages in each iteration.

Thus, in the iteration k, given the matrix $X^{(k)} \in G(\sigma^*)$, the first stage is to find a line that is tangent to the manifold $G(\sigma^*)$ at $X^{(k)}$ and which intersects with $L(c)$ at $A(c^{(k+1)})$.

As $X^{(k)} \in G(\sigma^*)$, there exist orthogonal matrices $U^{(k)}$ and $V^{(k)}$ such that

$$X^{(k)} = U^{(k)} S^* V^{(k)t}. \quad (2)$$

Furthermore, as proved in [7], the vector tangent to $G(\sigma^*)$ which starts in the point $X^{(k)}$ and reaches the set $L(c)$ at the point $A(c^{(k+1)})$ can be expressed as:

$$X^{(k)} + X^{(k)} \Lambda^{(k)} - \Theta^{(k)} X^{(k)} = A(c^{(k+1)}), \quad (3)$$

where $\Lambda^{(k)}$ and $\Theta^{(k)}$ are skewsymmetric matrices which, along with $c^{(k+1)}$, are the unknowns of equation (3). Using (2), (3) can be written as:

$$S^* + S^* L^{(k)} - H^{(k)} S^* = W^{(k)}, \quad (4)$$

where:

$$L^{(k)} = V^{(k)t} \Lambda^{(k)} V^{(k)}, \; H^{(k)} = U^{(k)t} \Theta^{(k)} U^{(k)} \text{ and}$$

$$W^{(k)} = U^{(k)t} A(c^{(k+1)}) V^{(k)}. \quad (5)$$

By equating the diagonal elements in (4) the following linear system is obtained [7]:

$$J^{(k)} c^{(k+1)} = b^{(k)}$$

where:

$$J^{(k)} = [u_i^{(k)t} A_j v_i^{(k)}]_{i,j=1,n} \quad (6)$$

$$b^{(k)} = S^* - [u_i^{(k)t} A_0 v_i^{(k)}]_{i=1,n} \quad (7)$$

Once this is solved, $c^{(k+1)}$, $A(c^{(k+1)})$ and $W^{(k)}$ are obtained. So, one of the unknowns in (4) is computed, ending the first stage.

If the off-diagonal elements in (4) are equated, the unknowns $H^{(k)}$ and $L^{(k)}$ can be computed as follows (see [7] for the details):

$$H_{ij}^{(k)} = -H_{ji}^{(k)} = -\frac{W_{ij}^{(k)}}{S_j^*}; \qquad i=n+1,m; \; j=1,n \quad (8)$$

$$H_{ij}^{(k)} = -H_{ji}^{(k)} = \frac{S_i^* W_{ji}^{(k)} + S_j^* W_{ij}^{(k)}}{(S_i^*)^2 - (S_j^*)^2}; \; 1 \le i < j \le n \quad (9)$$

$$L_{ij}^{(k)} = -L_{ji}^{(k)} = \frac{S_i^* W_{ij}^{(k)} + S_j^* W_{ji}^{(k)}}{(S_i^*)^2 - (S_j^*)^2}; \; 1 \le i < j \le n \quad (10)$$

The goal of the second stage in the iteration k is to find a matrix $X^{(k+1)}$ in $G(\sigma^*)$ which approximates $A(c^{(k+1)})$:

$$X^{(k+1)} \approx A(c^{(k+1)}).$$

Matrix $X^{(k+1)}$ could be obtained from matrix $X^{(k)}$ by expressing $U^{(k+1)}$ and $V^{(k+1)}$ in the way

$$U^{(k+1)} = e^{H(k)} U^{(k)}$$

and

$$V^{(k+1)} = e^{L(k)} V^{(k)},$$

(see [7] for details).

Since $e^{H(k)}$ and $e^{L(k)}$ can be approximated as

$$e^{H(k)} \approx I + H^{(k)} + \frac{1}{2} H^{(k)2} \approx$$

$$\approx \left(I + \frac{1}{2} H^{(k)} \right) \left(I - \frac{1}{2} H^{(k)} \right)^{-1}$$

and

$$e^{L(k)} \approx I + L^{(k)} + \frac{1}{2} L^{(k)2} \approx$$

$$\approx \left(I + \frac{1}{2} L^{(k)} \right) \left(I - \frac{1}{2} L^{(k)} \right)^{-1},$$

if we define the orthogonal matrices

$$R^{(k)} = \left(I + \frac{1}{2} H^{(k)} \right) \left(I - \frac{1}{2} H^{(k)} \right)^{-1}$$

and

$$T^{(k)} = \left(I + \frac{1}{2} L^{(k)} \right) \left(I - \frac{1}{2} L^{(k)} \right)^{-1},$$

then an estimation of matrix $X^{(k+1)} = U^{(k+1)} S^* V^{(k+1)t}$ can be obtained from the matrices

$$U^{(k+1)} = R^{(k)T} U^{(k)t}$$

and

$$V^{(k+1)} = T^{(k)T} V^{(k)}.$$

Expression (3) guarantees that:

$$X^{(k+1)} \approx R^{(k)t} (e^{H(k)} A^{(k+1)} e^{-L(k)}) T^{(k)},$$

(see [7] for details).

Therefore, to compute $X^{(k+1)}$ the following systems of equations (with multiple right hand sides):

$$\left(I + \frac{H^{(k)}}{2} \right) U^{(k+1)} = \left(I - \frac{H^{(k)}}{2} \right) U^{(k)} \quad (11)$$

$$\left(I + \frac{L^{(k)}}{2} \right) V^{(k+1)} = \left(I - \frac{L^{(k)}}{2} \right) V^{(k)} \quad (12)$$

must be solved to obtain $U^{(k+1)}$ and $V^{(k+1)}$. This completes a single iteration of the method MIII.

In [2] it was shown that MIII converges quadratically to the solution of the IASVP, denoted as c^*. The full MIII algorithm is:

Sequential MIII Algorithm (SMIII)

1. Compute $A(c^{(0)})$ in accordance with (1)
2. Compute $\text{svd}(A(c^{(0)})) = U^{(0)}S^{(0)}V^{(0)t}$
3. For $k = 0,1,\ldots$, while $\|U^{(k)t}A^{(k)}V^{(k)} - S^*\|_F > tol$
4. Compute $J^{(k)}$ in accordance with (6)
5. Compute $b^{(k)}$ in accordance with (7)
6. Solve the system $J^{(k)}c^{(k+1)} = b^{(k)}$ for $c^{(k+1)}$
7. Compute $A(c^{(k+1)})$ in accordance with (1)
8. Compute $W^{(k+1)}$ in accordance with (5)
9. Compute $H^{(k+1)}$, $L^{(k+1)}$ in accordance with (8-10)
10. Compute $U^{(k+1)}$, $V^{(k+1)}$ in accordance with (11-12)
11. End For

3 Parallel MIII Method

The parallelization of the MIII has been carried out using the SPMD paradigm on a message passing environment [15]. If P processors are available, each one will perform the same set of instructions. Matrices and vectors are distributed among the processors using the standard ScaLAPACK distribution [3], that is, matrices and vectors are block-partitioned, and these blocks are distributed cyclically in a two-dimensional mesh of the $P=P_r \times P_c$ processors (P_r rows and P_c columns). For example, for the case P_r=2, P_c=2, a vector b (of size 8×1) and a matrix A (of size 8×8), would be distributed as in Figure 1.

Figure 1. Block cyclic distribution of a vector and a matrix in a mesh of processors

b		A							
0	b_1	0	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}
	b_2		a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	a_{26}	a_{27}	a_{28}
1	b_3	1	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	a_{36}	a_{37}	a_{38}
	b_4		a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	a_{46}	a_{47}	a_{48}
0	b_5	0	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}	a_{56}	a_{57}	a_{58}
	b_6		a_{61}	a_{62}	a_{63}	a_{64}	a_{65}	a_{66}	a_{67}	a_{68}
1	b_7	1	a_{71}	a_{72}	a_{73}	a_{74}	a_{75}	a_{76}	a_{77}	a_{78}
	b_8		a_{81}	a_{82}	a_{83}	a_{84}	a_{85}	a_{86}	a_{87}	a_{88}

Some operations in MIII can be parallelized directly using distributed linear algebra routines of libraries such as ScaLAPACK and PBLAS [5], others can be parallelized designing specific routines based on calls to ScaLAPACK/PBLAS routines and on calls to other non distributed libraries, as LAPACK/BLAS [1,14]. The message passing is made with communication routines of BLACS [8] and MPI [13].

To minimize the communications between processors, vectors S^* and c are replicated in all the processors.

Steps 1 and 7 of the algorithm SMIII can be perfectly parallelized, since c is replicated in all the processors, and each processor computes (1) with the blocks of the matrices A_i (i=0:n) which has locally stored. Each processor performs this step locally with the BLAS routine *daxpy*. The step 2 is parallelized through the routine *pdgesvd* of ScaLAPACK.

The matrix J can be computed in parallel (see (6)), with a single matrix-matrix product A_jV with the routine *pdgemm* of PBLAS. The obtained matrix is multiplied column by column with U, using the distributed dot product (*pddot* of PBLAS) and the result is stored in the appropriate component of J (with the *pdelset* routine of ScaLAPACK). With this procedure the step 4 is parallelized. The computation of the vector b (step 5 of SMIII) is analogous to the computation of the matrix J because it consists of the same kind of operations (see (7)).

The step 6 of SMIII is parallelized through a call to the ScaLAPACK routine *pdgesv*, which solves a linear system of equations. This last routine leaves the c vector distributed among the processors. The algorithm has been designed assuming that the vector c is replicated in all the processors, therefore, this vector must be broadcasted to all the processors.

Since W and W^t are needed to compute H and L (see (8-10)), step 8 in SMIII is parallelized with two distributed matrix-matrix products *pdgemm* (to obtain W) and a redistribution of W (to obtain W^t). This redistribution is quite costly in terms of communications, but it is essential to compute H and L.

The step 9 of SMIII can be parallelized without communications, since the computations to be performed are made component by component, and all processors have all the data needed (S^*) in its local memory.

Finally, the parallelization of the step 10 is made through the PBLAS routine to multiply distributed matrices (*pdgemm*) and through the routine to scale vectors (*dscal* of BLAS). The ScaLAPACK routine *pdgesv* is needed as well to solve the multiple right hand sides linear systems (11,12). All together, the parallel algorithm for method MIII is as follows (*pdlange* is a ScaLAPACK routine which computes the Frobenius norm of a distributed matrix):

Parallel MIII Algorithm (PMIII)

In Parallel For $Proc$ = 0,1,…,P-1

1. Compute $A(c^{(0)})$ in parallel
   using *daxpy* of BLAS
2. Compute $\text{svd}(A(c^{(0)})) = U^{(0)}S^{(0)}V^{(0)t}$ in parallel
   using *pdgesvd* of ScaLAPACK
3. For k = 0,1,…, while $\text{pdlange}(U^{(k)}A^{(k)}V^{(k)} - S^*) > tol$
4. Compute $J^{(k)}$ in parallel

using *pdgemm*, *pddot* of PBLAS and *pdelset* of ScaLAPACK

5. Compute $b^{(k)}$ in parallel
   using *pdgemm*, *pddot* of PBLAS and *pdelset* of ScaLAPACK
6. Solve $J^{(k)}c^{(k+1)} = b^{(k)}$ for $c^{(k+1)}$ in parallel
   using *pdgesv* of ScaLAPACK
6. Reduction of $c^{(k+1)}$ to construct it in processor 0
   using *pdgemr2d* of BLACS
7. Broadcast of $c^{(k+1)}$ to all processors
   using *dgebs2d* and *dgebr2d* of BLACS
8. Compute $A(c^{(k+1)})$ in parallel
   using *daxpy* of BLAS
9. Compute $W^{(k+1)}$ and $W^{(k+1)t}$ in parallel
   using *pdgemm* and *pdcopy* of PBLAS
10. Compute $H^{(k+1)}$ and $L^{(k+1)}$ in parallel
    using *daxpy* of BLAS
11. Compute of $U^{(k+1)}$, $V^{(k+1)}$ in parallel
    using *pdgemm* of PBLAS, *dscal* of BLAS and *pdgesv* of ScaLAPACK

End Parallel For

4 Performance of Parallel MIII

Some numerical experiences have been carried out to analyse the performance of SMIII and PMIII algorithms. We present results about execution time, speedup, efficiency and scalability of PMIII by comparing it with its sequential version SMIII. As target matrices we have chosen random matrices of sizes m=n={1000,2000,3000}. The singular values to be assigned have been chosen randomly too.

SMIII and PMIII have been tested in a cluster of 2GHz biprocessor Intel Xeon, composed of 20 nodes, each one with 1 Gbyte of RAM, disposed in a 4x5 mesh with 2D torus topology and interconnected through a SCI network.

Tests with a specific version of MPI for this platform (Scali MPI) showed a latency of 5 μs and a bandwidth of 166 Mbytes/s. All the algorithms were implemented in Fortran 90. Several mathematical libraries were used. First, ScaLAPACK and BLACS parallel libraries were used to distribute the data and to carry out some operations such matrix-matrix products, solving linear systems of equations or computing svd in parallel. Optimized versions of the sequential BLAS and LAPACK libraries were used to perform basic local operations on each processor.

4.1 Execution times

The theoretical execution times of SMIII and PMIII, when m=n, can be approximated, respectively, by the expressions:

$$T(n,k) = \left\{ \frac{53n^3}{3} + 2kn^4 + O(kn^3) \right\} t_f$$

and

$$T(n,k,P) = \left\{ \frac{53n^3}{3P} + \frac{2kn^4}{P} + O\left(\frac{kn^3}{\sqrt{P}} \right) \right\} t_f +$$

$$+ \left\{ \frac{14\log(P)n^2}{\sqrt{P}} + \frac{\log(P)kn^3}{\sqrt{P}} + O\left(\sqrt{P}kn^2\right) \right\} t_v +$$

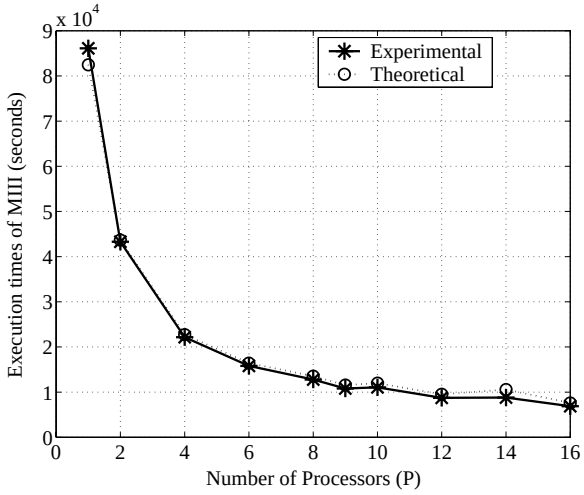
$$+ \left\{ 2\log\left(\sqrt{P}n\right) + \sqrt{P}kn^2 + O\left(\sqrt{P}n\right) \right\} t_m .$$

Here, k is the number of iterations, t_f is the execution time for a single floating point operation, t_m is the network latency and t_v is the inverse of the bandwidth.

These expressions show the high computational cost of the algorithms and the good benefit obtained with the parallelization.

We have estimated the parameters t_f, t_m and t_v, corresponding to our target cluster and we have compared the experimental results with those predicted by the theoretical model for the case m=n=3000. As it is shown in Figure 2, theoretical results are a good approximation for the experimental ones. Thus, we have a good tool to analyse the behaviour of the algorithms in hypothetical situations.

Figure 2. Experimental v.s. Theoretical Runtime (seconds) of MIII for m=n=3000



As the parallelization of MIII algorithm has been carried out at iteration level, we show in the sequel experimental results for one iteration only. In Table 1 we present the execution time of one iteration of the algorithm for several sizes and different number of processors. It can be seen the large complexity of the problem (O(n^4) per iteration) and how the runtime decreases as the number of processors increases.

Table 1. Runtime (seconds) of MIII

P	**1000**	**2000**	**3000**
1	1181	17124	86131
2	692	9407	43306
4	402	4973	22160
6	351	3586	15813
8	265	2909	12774
9	237	2493	10792
10	265	2747	11059
12	200	2111	8745
14	233	2108	8790
16	169	1686	6882

4.2 Speedup and Efficiency

From the theoretical point of view PMIII is an asymptotically optimum parallel algorithm:

$$\lim_{m\to\infty} S(m,P) = \lim_{m\to\infty} \frac{T(m,k)}{T(m,P,k)} = P\,.$$

Figure 3. Experimental Speedup of MIII

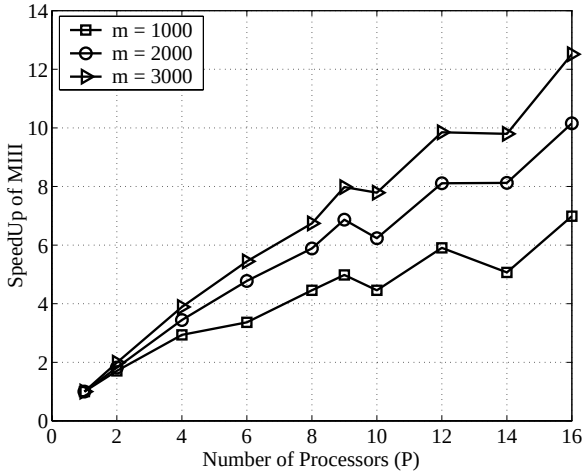


Table 2. Experimental Efficiency of MIII

P	**1000**	**2000**	**3000**
1	100 %	100 %	100 %
2	85 %	91 %	99 %
4	73 %	86 %	97 %
6	56 %	79 %	90 %
8	55 %	73 %	84 %
9	55 %	76 %	88 %
10	44 %	62 %	77 %
12	49 %	67 %	82 %
14	36 %	58 %	69 %
16	43 %	63 %	78 %

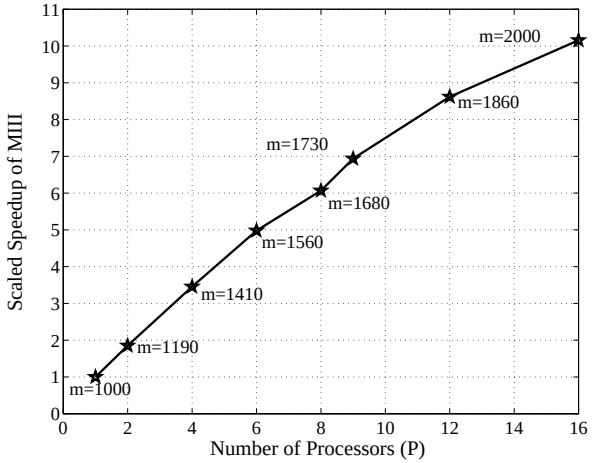
In Figure 3 and Table 2 the experimental speedup and efficiency are shown for different sizes and number of processors. It can be verified the good behaviour of the algorithm specially for large size problems. Thus, for m=n=3000 efficiency is always above 69%.

4.3 Scalability

We use the scalability approach provided by [15]. The scalability of a parallel system is a measure of its capacity to increase speedup in proportion to the number of processing elements. It reflects a parallel systems ability to utilize increasing processing resources effectively [15]. Different metrics for evaluating the scalability can be found in the specialized literature; however it is convenient to have in mind the characteristics of the problem in order to chose the adequate metric of scalability [16]. In our case we use the scaled speedup as defined in [15] due to its ease of use when experimental data are available. The scaled speedup is defined as the speedup obtained when the problem size is increased linearly with the number of processing elements. A system is considered scalable if the scaled-speedup is close to linear with respect to the number of processing elements.

To analyze the scalability the problem size must be scaled with the number of processors. We define the problem size as the number of computations carried out in the best sequential algorithm. In this case we consider that the theoretical computational cost of the sequential algorithm is W=O(n^4). Thus, scaling the number of processors P and the problem size in the same ratio implies that if we scale P as P=kP_0 then W must be scaled as kO(n^4) and n must be scaled as $k^{1/4}n_0$. In the practical experiments we have taken the Speedup with m=n={1000, 1190, 1410, 1560, 1680, 1732, 1861, 2000} in correspondence with P={1, 2, 4, 6, 8, 9, 12, 16}. This is plotted in Figure 4 and it shows a quite good behaviour of the scalability of the parallel algorithm.

Figure 4. Experimental Scaled Speedup of MIII



5 Conclusions

We have designed a parallel algorithm (PMIII) that solves the IASVP efficiently. Some operations of SMIII algorithm have been parallelized directly using portable and efficient distributed linear algebra routines of parallel libraries (PBLAS, ScaLAPACK), supported by linear algebra routines of sequential libraries (BLAS, LAPACK). Others SMIII operations have been parallelized designing specific routines based on calls to ScaLAPACK, PBLAS, LAPACK, BLAS, BLACS and MPI routines. The communications among processors have been carefully minimized.

We have estimated the theoretical execution times of SMIII and PMIII. They indicate that the execution time of PMIII is smaller than that corresponding to SMIII and it has good performance asymptotically. We have verified these facts experimentally.

We have experimented with different sizes of problems ($1000 \leq m,n \leq 3000$), greater than those reported on the previous literature. The experiments have been executed in a cluster and we have obtained execution times that improve the sequential execution times substantially. Speedups and efficiency are quite good for small number of processors (2 and 4), and remain acceptable when the number of processors grows as long as the problem size is large enough. Accordingly, the scalability observed is fairly good.

Acknowledgement

This work has been supported by Spanish MCYT and FEDER under Grant TIC2003-08238-C02-02 and SEIT, SUPERA-ANUIES (México).

References:

[1] Anderson, E., Bai, Z., Bishof, C., Demmel, J. and Dongarra, J., *LAPACK User Guide*, SIAM, 1995.

[2] Bai, Z., Morini, B. and Xu, S., On the local convergence of an Iterative Approach for Inverse Singular Value Problem, *Journal of Computational and Applied Mathematics*, 2005.

[3] Blackford, L., Choi, J. and Clearly, A., *ScaLAPACK User's Guide*, SIAM, 1997.

[4] Chen, X. and Chu, T., On the Least Squares Solution of Inverse Eigenvalue Problems, *SIAM, Journal on Numerical Analysis*, Vol. 33, No. 6, 1996, pp 2417-2430.

[5] Choi, J., Dongarra, J., Ostrouchov, S., Pettitet, A. and Walker, D., *A Proposal for a Set of Parallel Basic Linear Algebra Subprograms*, Technical report ut-cs-95-292, Department of Computer Science, University of Tenessee, 1995.

[6] Chu, M., Inverse Eigenvalue Problems, *SIAM, Review*, Vol. 40, 1998.

[7] Chu M., Numerical Methods for Inverse Singular Value Problems. *SIAM, Journal Numerical Analysis*, Vol. 29, 1992, pp 885-903.

[8] Dongarra, J. and Van de Geijn, A., *Two Dimensional Basic Linear Algebra Comunications Subprograms*, Technical report st-cs-91-138, Department of Computer Science, University of Tenessee, 1991.

[9] Flores-Becerra, G., García, V.M. and Vidal, A.M., Numerical Experiments on the Solution of the Inverse Additive Singular Value Problem. *Lecture Notes in Computer Science*, Vol. 3514, 2005, pp 17-24.

[10] Flores, G. and Vidal, A.M., Parallel Global and Local Convergent Algorithms for Solving the Inverse Additive Singular Value Problem. *Wseas, Transaction on Circuits and Systems*, Vol. 3, 2004, pp 2241-2246.

[11] Friedland, S., Nocedal, J. and Overton, M. L., The Formulation and Analysis of Numerical Methods for Inverse Eigenvalue Problems, *SIAM, Journal on Numerical Analysis*, Vol. 24, No. 3, 1987, pp 634-667.

[12] Groetsch, C., *Inverse Problems. Activities for Undergraduates*, The mathematical association of America, 1999.

[13] Groupp, W., Lusk, E. and Skjellum, A., *Using MPI: Portable Parallel Programming with Message Passing Interface*. MIT Press, 1994.

[14] Hammarling, S., Dongarra, J., Du Croz, J,. and Hanson, R., *An Extended Set of Fortran Basic Linear Algebra Subroutines*, ACM Trans. Methemathical Software, 1988.

[15] Grama, A., Gupta, A., Karypis, G. and Kumar, V., *Introduction to Parallel Computing*, 2nd edition. Pearson Education Limited, 2003.

[16] Martin, I. and Tirado, F., Relationships between efficiency and execution time of full multigrid methods on parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 6, June 1997.