PVP Optimization for a Finite Element Tidal Model

S. Chumbe, M. González, M. Espino, M. García, F. Hermosilla and *A. S-Arcilla*, Laboratori d'Enginyeria Marítima, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain

ABSTRACT: *This paper reports a vector/parallel implementation of a numerical algorithm to simulate the propagation of tides using a spectral model. The Finite Element Method (FEM) is employed in the discretization of the differential equations. The preconditioned Conjugate Gradient Squared (CGS) method is applied to the solution of the discretized equations. Since the efficiency of the CGS iterative method is determined primarily by the performance of the matrix-vector product, the preconditioner solver and the storage scheme, a Compressed Row Storage (CRS) format is used to store the sparse matrices. The storage scheme, the preconditioner and the CGS method are implemented with an emphasis on their potential for parallelism. The code is parallelized with the help of Fortran Compiler Directives. A Cray YMP/232 has been used, but the implementation is suitable for all MIMD shared-memory systems.*

1. Introduction

The *Laboratori d'Enginyeria Marítima* of the Catalonia University of Technology (LIM/UPC) deals with hydrodynamic research in the sea. This research is carried out by :

a. fields campaign,
b. physical modelling,
c. numerical modelling.

Over the last ten years the LIM/UPC's research team has been collecting information about ocean dynamics. These data fields are very important to know the hydrodynamic behaviour within this area. They are also useful for comparing the output of physical and numerical models with real data.

The CIEM Flume is one of the most important physical models at the LIM/UPC. In this canal (100 m. long, 3 m. wide and 5 m. deep) it is possible to simulate different kinds of wave propagation. It is also useful to study the breaker zone at a reduced scale.

However, not all the marine phenomena can be studied by experimental campaign at the sea or by models at a reduced scale in the laboratory. Because of that, numerical models have been developed to simulate complex marine phenomena. They have become powerful tools thanks to recent mathematical and computational advances. These models are available to simulate the real hydrodynamic by solving partial differential equations (PDE) which include the variables and physical parameters of the marine system.

When considering numerical modelling of marine hydrodynamics, one must consider large data volumes and millions and millions of intensive calculations. All of this leads us to the need for improved numerical efficiency of our models by using optimization techniques and high-performance computers.

In this work, the optimization process carried out to port the sequential code of the numerical MAREAS model from a scalar machine to the Cray YMP/232 vectorcomputer is presented. The model MAREAS has been developed to simulate the tide induced current in marine domains where the shallow water hypothesis can be applied.

2. Outline of the numerical model

The model solves the transient shallow water equations using a harmonic decomposition technique in time. This technique leads to one non-linear steady problem per each frequency, which is linearized with a standard Picard method and is solved by a preconditioned iterative solver.

In this report, the physical hypotheses and the mathematical formulation of the model are assumed as a starting-point. More information can be found in reference [7]. However, it is important to mention that the model uses the Finite Element Method (FEM) to discretize the differencial equations in order to obtain the correspondent nonlinear system equations. For details on the discretization we will refer to [7].

In very abstract terms, the matrix form of the non-linear system equation can be represented by:

$$\sum_{i=1}^{I} \left[\mathbf{A}^{(i)}_{\alpha\beta} \cdot \mathbf{x}^{(i)}_{\beta} \right] = \sum_{i=1}^{I} \mathbf{z}^{(i)}_{\alpha} \qquad (2.1)$$

The superindex (i) represents the values of each element and Σ is the assembler operator. **A**, **x** and **z** are the coefficient matrix, the vector of unknowns and the vector of independent terms respectively (for each element). α and β are the indexes of the elemental matrix.

The model solves a matrix equation similar to (2.1) for each tide frequency ω_k

3. Algorithm of the MAREAS Model

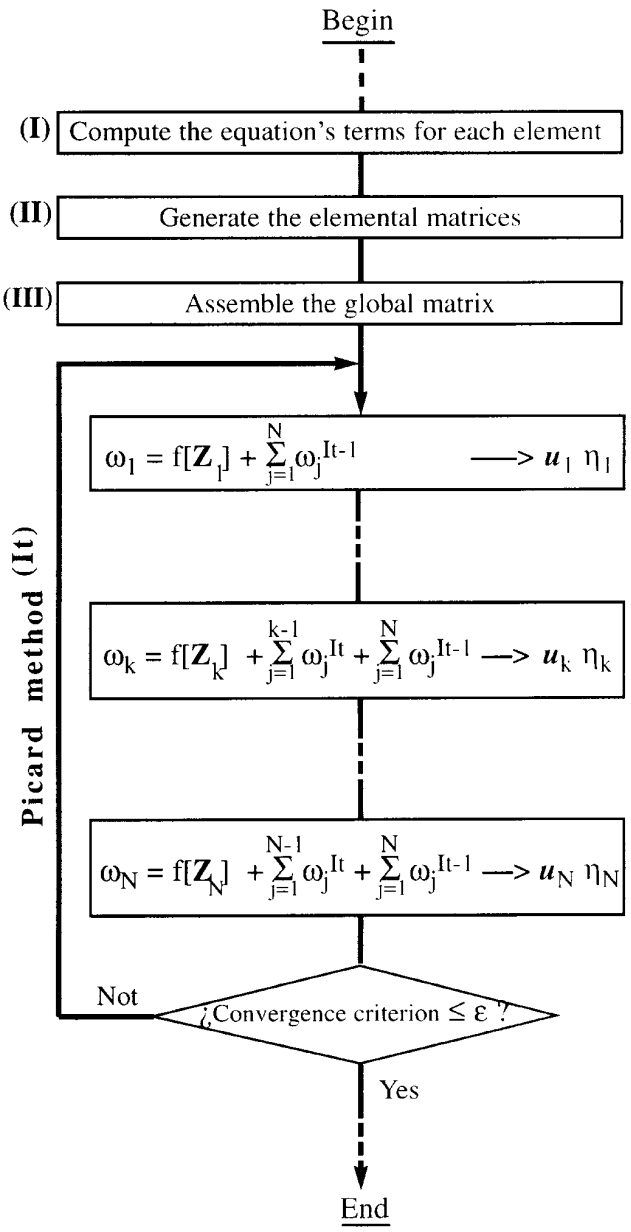

Fig. 3.1 Flowchart of the MAREAS model

For each frequency ω_k, (k = 1,2, ... ,N) the equation (2.1) is solved by a preconditioned CGS iterative method. This process consumes around 40 % of the total CPU time and executes the largest number of floating point operations. Therefore, it has a considerable influence on the model performance. Also, the proces-

ses (I), (II) and (III) (see fig. 3.1) associated with the generation of the elemental matrices and the assembly of the global matrix have an important effect on performance. The optimization strategy has been applied to these three processes and the solver.

4. Computer architecture

The computer architecture available to the LIM/UPC is characterized as follows:

Cray Y-MP/232
2 vector-processors
256 Mb of RAM
333 MFLOPS by processor
6 ns. by clock cycle
MIMD shared-memory
64 bits of word length

A powerful advantage of the Cray YMP/232 is its vectorial architecture. The compiler of this computer is available to produce a high vectorial ratio with a little help. The programmer only needs to avoid the data dependences and some classical statements which inhibit the vectorization. It must also be remembered that the compiler only vectorizes the most internal nested loop [4]. Moreover, there are tools such as *Flowtrace*, *hpm* and *atexpert*, which make the optimization work more easily [10].

5. Optimization strategy

For a FEM hydrodynamic simulation, we have observed that an effective implementation of a vector/parallel algorithm for numerical models requires:

- the independent generation of elemental matrices,
- an optimum storage of sparse matrices,
- iterative solvers for the linear systems,
- algorithms focused to parallelism.

Before attempting vector/parallel optimization our first step was to implement the most appropiate mathematical algorithm for solving the MAREAS model on the Cray platform while keeping its portability on scalar/sequential machines. This means, for example, that we do not use domain decomposition and do not develop special routines to perform the load-work balancing between processors. Therefore, our first issue was to develop an advantageous scheme for storing sparse matrices used in solving the preconditioned CGS solver on parallel platforms.

The parallelization strategy reported in this work has two baselines:

5.1.- To exploit a favorouble characteristic of the FEM (each element contains its own information without any data dependence between elements), which allows us to operate on the element contributions to the global matrix independently and simultaneously (as a pseudo coarse granularity parallelization).

5.2- To make use of sparse matrix storage schemes adapted to the vectorial/parallel algorithms which involve iterative solvers.

Finally, it is important to remark that using vector-processors in conjunction with an algorithm specially for such computers, computation speed increases up to a factor of 10 have been reported [5]. As a consequence of that, theoretically, a speed-up of 20 could be reached on the Cray YMP/232 vector-computer.

6. Vector/parallel implementation

During all of this work we keep in mind the following idea: It is important to consider the vectorial and parallel characteristics of the code at the same time in order to consider both the good and bad effects of each optimization on one another. Also, the whole program has to be considered as a set of coupled parts because modifying one aspect of the program may lead to significant changes in the whole code.

According to the optimization strategy established in (5), the routines which perform the generation of the elemental matrices and the assembly of the global matrix are first analysed to find the inherently scalar and sequential parts. Next, we determine suitable vector/parallel algorithms for these parts and, finally, the most appropriate alternative is implemented. Therefore, the vectorization consisted (for each one of these routines) in putting the largest loop as the innermost loop and avoiding the data dependences and statements which inhibit vectorization of this loop. These routines are then parallelized by dividing the computation of the whole mesh among the two available processors equally (see Fig. 6.1). FORTRAN compiler directives are used to force the parallelization where it was necessary.

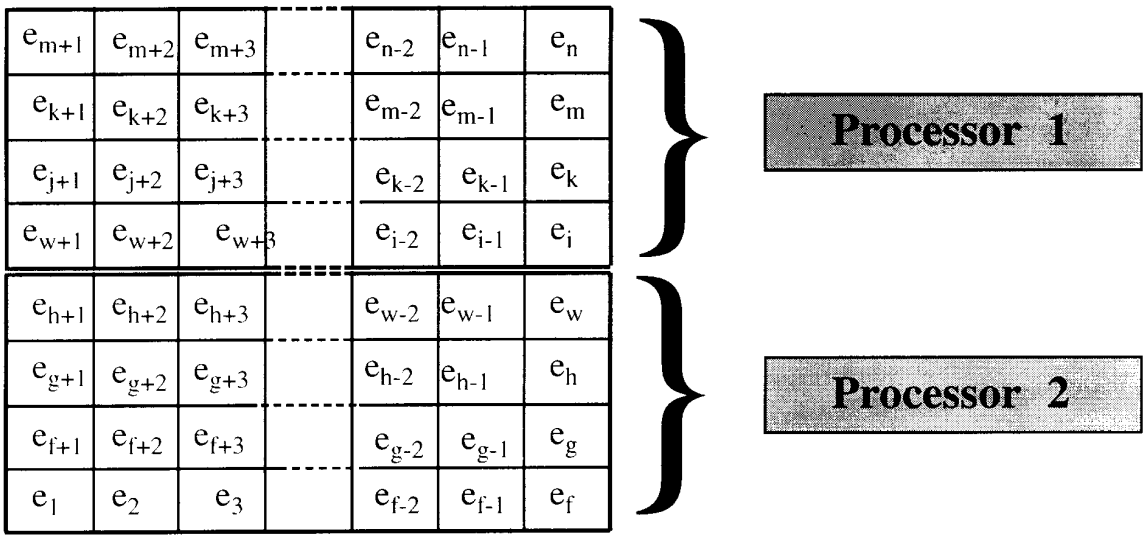

Fig. 6.1. The generation of the elemental and global matrices has been parallelized by an even distribution of the computation of the mesh elements among the two processors.

In order to implement the second part of the optimization strategy (5.2) we carried out with restructuring of the data-storage, the development of a vectorial preconditioned CGS solver and parallel programming (by using *microtasking* and *autotasking* features).

6.1. Restructuring the data-storage

As is well-known , the efficiency of any iterative solver and its vector/parallel implementation is determined primarily by the performance of the preconditioner solver and the matrix-vector products. Both of them are highly dependent on the storage scheme used for the global matrix. Therefore, we have implemented a storage scheme more appropiate for our specific application.

On the other hand, large-scale linear systems of the form **Ax = b** can be most efficiently solved if the zero elements of **A** are not stored. For instance, the global matrix of MAREAS model has less than 95 nonzero elements by row (in practical problems the matrix order can be more than 5000).

There are many schemes for storing sparse matrices (see for instance Saad [11] and Eijkhout [6]). We have implemented the Compressed Row Storage (CRS) scheme because our original matrix is not banded (actually its bandwidth varies strongly if it is not renumbered previously) and this scheme does not store any unnecessary elements.

The CRS format puts the subsequent nonzeros of the matrix row in contiguous locations. We have created 3 vectors: one for floating-point numbers (*val*), and the other two for integers (*col_ind*, *row_ind*). The *val* vector stores the values of the nonzero elements of **A**, as they are traversed in a row-wise fashion. The *col_ind* vector stores the column indexes of the elements in the *val* vector (that is, if $val(k)=a_{i,j}$ then $col\_ind(k)=j$). The *row_ind* vector stores the locations in the *val* vector that start a row (that is, if $val(k)=a_{i,j}$ then $row\_ind(i) \le k < row\_ind(i+1)$). The storage savings for this approach is significant. Instead of storing n^2 elements, we need only $2nz + n + 1$ storage locations, where nz is the number of nonzeros in **A** and n is the order of **A**.

The matrix-vector product $y = \mathbf{A}x$ using a CRS scheme can be expresed in the usual way:

$$y_i = \sum_{j = row\_ind(i)}^{row\_ind(i+1)-1} val(j) * x(col\_ind(j)) \qquad i = 1,\ldots,n$$

Since this scheme only multiplies nonzero matrix

entries, the operation count is 2 times the number of nonzero elements in **A**, which is a significant savings over the dense operation requirement of $2n^2$. Moreover, it avoids any kind of data dependences in the nested loop.

Next, we implemented a storage scheme for the preconditioner. There are many sophisticated preconditioners for iterative methods (see, for example [2,3,13]). In this work, we have chosen an incomplete factorization of the global matrix **A** stored in the CRS format as a first approach. A variant of the CRS format focused to the parallelization has been used to store and to perform the incomplete factorization. It is recalled Modified Row Storage (MRS) format.

For the factorization we need to store the preconditioner matrix, **M**, the diagonal elements and their respective column and row pointers. To do that we require 3 vectors.

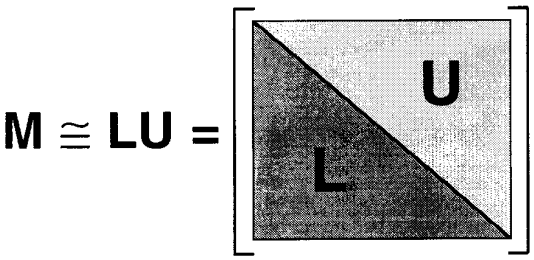

Fig. 6.2. **M** is the preconditioner matrix for CGS solver.

The *val_lu* vector has (nz + n + 1) elements, the *col_lu* vector also has (nz + n + 1) elements, and the *row_u* vector stores nz elements. The *val_lu* vector stores the inverses of the diagonal elements and the values of the nonzero elements of the matrix **M**, row by row (see Fig. 6.3). The *col_lu* vector stores the locations in the *val_lu* vector that start a row, that is, its n+1 first elements stores the column where the **L** part of **M** starts. Its following elements contain the column indexes of the elements in the *val_lu* vector. The *row_u* vector stores the locations in the *val_lu* vector where the **U** part of **M** starts, row by row.

6.2. Vectorial preconditioned CGS solver

The coefficent matrix (**A**) of the MAREAS model is a nonsymmetric definite nonpositive and large sparse matrix. Because of this we have chosen the Conjugate Gradient Squared (CGS) iterative solver which does not involve computations with $\mathbf{A}^T$ and reaches successful convergence with nonsymmetric matrices if the initial guess is not close to the solution [2]. The pseudocode for the preconditioned CGS method is given in the next figure.

Compute $r^{(0)} = b - \mathbf{A}x^{(0)}$ for some initial guess $x^{(0)}$
Choose $\omega = r^{(0)}$
for i = 1,2, ...
  $\rho^{i-1} = \omega^T \mathbf{r}^{(i-1)}$
  if $\rho^{i-1} = 0$ then method fails
  if i = 1
    $u^{(i)} = r^{(0)}$
    $p^{(1)} = u^{(1)}$
  else
    $\beta_{i-1} = \rho^{i-1} / \rho^{i-2}$
    $u^{(i)} = r^{(i-1)} + \beta_{i-1} q^{(i-1)}$
    $p^{(i)} = u^{(i)} + \beta_{i-1}(q^{(i-1)} + \beta_{i-1} p^{(i-1)})$
  end if
  solve My = p$^{(i)}$
  v = **A***y*
  $\alpha_i = \rho^{i-1} / \omega^T \mathbf{v}$
  $q^{(i)} = u^{(i)} - \alpha_i v$
  solve My = *u*$^{(i)}$ + q$^{(i)}$
  $x^{(i)} = x^{(i-1)} + \alpha_i y$
  v = **A***y*
  $r^{(i)} = r^{(i-1)} - \alpha_i v$
  check convergence; continue if necessary

Fig. 6.4. Pseudo-code of the preconditioned CGS solver.

The basic time-consuming kernels of CGS solver are: inner products, matrix-vector products and primarily preconditioner solvers. These critical parts have been emphasised by bold characters in the pseudo-code (Fig. 6.4).

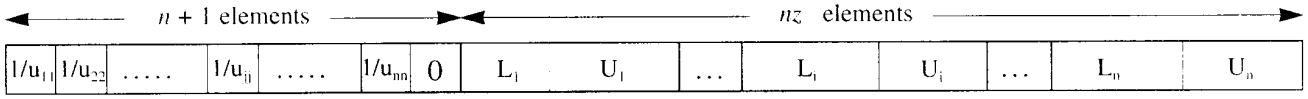

Fig. 6.3 The MRS format for the matrix **M** ≅ **LU** needs three vectors. Here it is a scheme of the *val_lu* vector which uses (n+1+nz) elements.

The computation of an inner product of two vectors can be easily vectorized and parallelized. In general, for shared-memory machines, the accumulation of local inner products can be implemented as a critical section where all the processors add their local result in turn to the global result. However, because of the Cray YMP/232 architecture[1], it was more efficient to vectorize the inner product routine rather than parallelize.

In order to reach a high ratio of vectorization, for the matrix-vector product, we have implemented a variant of its usual way. The pseudo-code to perform the product $\mathbf{A}x$, using the CRS storage scheme, is:

```
Choose i = 0 and iv = col_ind(1)
For j = 1,...., nz
    if col_ind(j) ≤ iv then i = i + 1
    y(i) = y(i) + val(j) . x(col_ind(j))
    iv = col_ind(j)
end
```

Here, the loop has a large number (nz) of iterations and it has not any data dependence.

Although preconditioning is often the most problematic part of optimization of a CGS solver, we have improved the performance of the solver by vectorization of the incomplete factorization preconditioner using the MRS storage scheme. By taking in advantage the Cray YMP/232 vectorial architecture, we have implemented a pseudo code to perform the forward/backward solver for $\mathbf{M}x \cong \mathbf{LU}x = b$. The forward part is represented as :

```
Choose ω(1) = b(1) , i = 1 and iv = k = pivot = 0
for j = n + 1, .... , nz + n
    if col_lu(i) + k < row_u(i) then
        pivot = pivot + val_lu(j)*ω(col_lu(j))
    end if
    if col_lu(j+1) ≤ iv then
        i = i + 1
        set k = pivot = 0
    end if
    ω(i) = b(i) - pivot
    iv = col_lu(j+1)
end
```

As is apparent, only one large loop is necessary to perform the forward process. The backward part may be similarly performed by changing the direction of the recursive process.

6.3. Multitasking

For the parallel implementation, we have focused on *Multitasking*, which allows parallel execution on multiple CPUs of the Cray systems [4]. Specifically, the *microtasking* and the *autotasking* features have been combined in this work to accomplish *multitasking* in FORTRAN language. Thus, parallelism is exploited at the loop or block level in the MAREAS code.

We insert *directives* in the code to indicate sections that can be executed on the two CPUs simultaneously when the compiler is unable to detect parallelism or to solve inhibition problems automatically. The idea is that if it is possible, innermost loops are vectorized and outermost loops are multitasked. Therefore, we reprogrammed the sections of code which are well-suited for parallel processing by using *inlining* or *unrolling* techniques [4]. In order to do that, we have used *directives* but any kind of *switches*.

As we have related previously in (3), there are two principal tasks to be done for FEM hydrodynamic modelling : (a) the generation of the elemental matrices and the assembly of the global matrix (process II and III), and (b) the linearization process which involves the solution of the $\mathbf{A}x=b$ system.

For the first task, we developed a small and simple algorithm to carry out the even distribution of the work load among processors. The delay-time is minimized when we assign a similar number of mesh elements to each processor (see Fig. 6.1). Because of the absence of data dependence between elements, the exchange of information between CPUs is lower.

The routines involved in the calculation of the terms associated with each elemental matrix have been parallelized by forcing the simultaneous processing of the loops, which perform the numerical integration. Because we use four Gauss-points for the integration, the work load is balanced.

The linearization process and the preconditioner solver are typical sequential processes and we have seen that the vectorization has a good influence over the

[1] An important advantage of a Cray vectorprocessor is that it does not use *cache* memory, which is good for the *locability* and the synchronization between CPUs and RAM memory.

model performance in these cases.

7. Test cases

In order to evaluate the performance and the scalability of the optimized MAREAS model we have defined five test cases on a rectangular domain of 1353 km. by 600 km. Two of the four boundaries are assumed to be closed, while at the third and fourth boundary a periodic tidal movement is imposed. The characteristics of the FEM meshes are shown in the Table 1. The FEM mesh has 512 elements (NELEM) and 561 nodes (NNODE). NEQU is the number of equations (order of the global matrix).

Case	NVDF	SP	NEQU
1	1	1.6 %	1634
2	2	1.1 %	2756
3	3	0.7 %	3878
4	4	0.5 %	5000
5	5	0.3 %	6122

Table 1. Test cases. Here NEQU = 2*NNODE*NVDF + NELEM, NVDF is the number of vertical degrees of freedom and SP measures the percentage of nonzeros of the sparse matrix.

For these tests, the analytical problem of the double Kelvin wave has been simulated . An angular velocity (ω) equal 1.45441E-4 rad/s., a frequency of 10E-4 1/s. and a depth of 100 m. have been imposed as physical conditions. The 2DH problem is simulated for one vertical degree of freedom . For NVDF > 1 the MAREAS model is able to simulate quasi-3D problems. Five vertical degrees of freedom is enough for hydrodynamic real cases [7].

8. Benchmark

8.1. Scalar/sequential performance

We did not test the scalar/sequential performance of the MAREAS code on the Cray because it is well-known that if the *pipelining* and the *chaining* features are not used on a vectorprocessor the processing time will be exaggerated [5,9]. In any case, for reference, the CPU times spent for the test cases (running sequentially) on a Digital Alpha 7000/630 scalarprocessor is shown in Table 2. The Alpha computer has a theoretical peak performance of 300 MFLOPS per processor and RISC architecture.

CPU time and Speedup obtained by applying a Sparse-matrix technique for inner products, matrix-vector products and the ILU(0) preconditioner

Case	Original	Optimized	Speedup
1	95 s.	28 s.	3.39
2	384 s.	65 s.	5.92
3	933 s.	123 s.	7.57
4	1858 s.	217 s.	8.56
5	3422 s.	366 s.	9.35

Table 2. Performance profile of the original and the optimized codes on the Alpha computer

It is very important to note the improvement of the MAREAS model performance produced by using only a sparse matrix technique. We can compare (see Table 2) how the *speedup* increases as well as the SP decreases (see Table 1) proportionality.

8.2. Parallel/Vector Performance

We are interested in benchmarking the performance produced for the optimization strategy applied on the code of MAREAS model. Therefore, this performance profiling involves three optimization phases:

- **Original** means performance produced by the code before applying any sparse matrix storage scheme. For this, we have used the *-Zv* and *-Wf"-o aggress"* compiler options in order to allow full vectorial optimization (automatically)[4]. The CGS solver is used.

- **ManVector** means vectorization obtained by avoiding conditions that inhibit vectorization manually. The principal techniques applied are: *inlining*, *unrolling* and *splitting* innermost loops, elimination of any reference to external code that cannot be vectorized, removal of obsolete and extra conditional statements, prevention of memory conflicts and avoiding data dependencies. We do not use vectorial directives. In this case, the CRS/MRS storage scheme and the CGS solver are used

- **ParaVector** means optimization of the MAREAS model by parallelizing its ManVector version using *multitasking* techniques. To do this, we have applied

parallel directives beginning with CMIC$, CDIR$, CMIC@ and CDIR@. The code was compiled by using the *-Zp* option.

Tables 3 and 4 show the performance profile of the PVP optimization. Table 3 shows the reduction of *Real* and *CPU times* profiled by each test case. Table 4 shows the increase in MFLOPS in the vectorial PVP optimization phase. Finally, the comparative study of the total PVP optimization *speedup* versus the problem

Case	Original	ManVector	Para/Vector
1	73 / 61	15 / 13	11 / 17
2	241 / 219	36 / 34	26 / 45
3	592 / 524	65 / 61	51 / 73
4	1130/1020	110/103	94 / 124
5	1897/1760	165/155	147 / 186

Table 3. Performance profile. Real time/ CPU time. (in seconds)

Case	Original MFLOPS	ManVector MFLOPS	Vectorial Speedup
1	4.1	48	4.7
2	3.4	43	6.4
3	3.1	39	8.6
4	2.6	35	9.9
5	2.4	34	11.3

Table 4. Efficiency and vectorial speedup produced by CRS/MRS scheme.

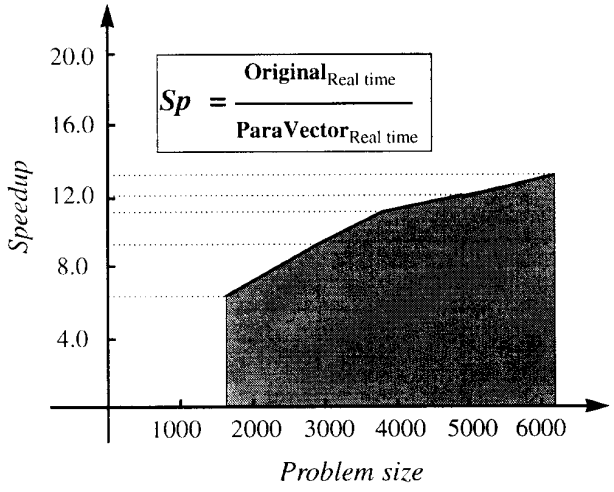

Fig. 8.1. PVP Speedup (*Sp*) vs. Problem size

9. Applications

Two real applications taken from the sea-shore of the Basque Country in the north of Spain have been chosen to ilustrate the capability of the PVP optimized MAREAS model.

a. Zumaia outlet

FEM mesh : 480 elements
542 nodes
Problem size : 19264 equations
Tide frequencies : 4
Vertical degrees of freedom : 4

b. Orio river mouth

FEM mesh : 608 elements
693 nodes
Problem size : 36912 equations
Tide frequencies : 6
Vertical degrees of freedom : 4

The following Table shows the *Real* and *CPU times* (in minutes), the *Total Speedup (CRS/MRS - CGS+P/V)* and the *Parallel Speedup* reached for these two real applications on the Cray YMP/232.

Application	Real/CPU time	Total PVP Speedup	Parallel Speedup
Zumaia	15 / 11	36	1.18
Orio	22 / 19	54	1.15

Table 5. Applications benchmarking.

As is proved, it now takes only minutes to simulate real cases, such as the Zumaia outlet and the Orio River mouth, whereas these simulations previously took hours to complete. The increased capability of the vectorized/parallelized version of the MAREAS code makes it possible to simulate large and complex domains, such as the Bizcaya Gulf (located in Northern Spain), which would not have been possible with the original code. On the other hand, because only two processors were available and we did not use a complete mesh decomposition, the parallel speedup is not significant.

size (global matrix order) is summarized in Figure 8.1.

10. Concluding remarks

- This work proves that although experience has revealed that the use of vector/parallel techniques may reduce the computation time considerably, many successes may be obtained after developing new algorithms that are particularly designed and tuned for Parallel Vector Processing (PVP). In our case such an algorithm consists of the CRS/MRS Sparse Matrix Scheme for improving the efficiency of the preconditioned Krylov methods such as the CGS method applied in this work.

- For the routines which perform the generation of the elemental matrices and the assembling of the global matrix, a size mesh multiple of two is desirable to reach more efficiency in the load-balancing when is targeted a Cray YMP/232. However, for large meshes in real applications, this aspect is less important and the delay time or the time overhead is not as significant.

- The MAREAS Model PVP code is highly portable because it does not use any external references (libraries or commercial software), and only uses shared-memory compatible compiler directives for parallelization.

- Scalar and vector optimization are very useful on the Cray-YMP/232. Also, these changes are not complicated for marine researchers and engineers to perform. We have proved that the reformulation of the matrix assembly process, the use of Sparse Matrix Storage techniques and an efficient solver are more important than the informatic aspect in PVP optimization.

- The CRS/MRS storage scheme applied to a FEM model using a CGS solver will reduce Real/CPU times even on scalar/sequential machines. This is an additional advantage for users who work with PC's or workstations. However the advantages are not as spectacular as on a vectorial machine.

- Because the original algorithm of the MAREAS model was not suitable for parallelization, the automatic parallelization by the compiler was not useful on its own, but it was useful to analyze the parallelism and for detecting the bottle-necks in the sequential program.

- Vector-processors with a few very powerful processors have their limitations in further increasing the computing power for more complex applications. Moreover, they are expensive because they rely on the use of expensive advanced technology. Therefore, to further increase the speed of computation, massive parallel processing has to be applied. A computer such as the Cray T3D would be useful for present hydrodynamic modelling using HPCN technology.

Acknowledgement

We are most grateful to Enric Torres of Cray Research Inc. (Spain) and Oleg Mercader of CESCA for their technical support and useful discussions. The suggestions and advice of José María Cela of CEPBA in numerical aspects is very much appreciated. In particular, we would like to thank the AZTI Company for providing us with its field data to perform the real applications of the MAREAS model. This research is part of the EC *TRIMODENA/PACOS* Project, carried out by the LIM-AZTI agreement and is partially supported by the *ESPRIT* European Programme. The parallel/vectorial optimization was done on the Cray YMP/232 of CESCA in Barcelona, Spain.

References

[1] Axelsson O. and V. Eijkhout, "*Vectorizable preconditioners for elliptic difference equations in three space dimensions*", J. Comput. Appl. Math., 27, 1989, pp. 299-321.

[2] Barrett R., M. Berry et. al., "*Templates for the solution of linear systems: Building blocks for iterative methods*", Tech Report of the Office of Energy Research, U.S. (SIAM Publications), 1993.

[3] Dongarra J., I Duff, D Sorensen and H. Van der Vorst, "*Solving Linear Systems on vector and Shared Memory Computers*", SIAM, Philadelphia, PA, 1991.

[4] "*CF77 Optimization Guide*", SG-3773 6.0. Cray Software Documentation. Cray Research Inc. MN 55120 USA. 1994.

[5] Dongarra J., "*Performance of various computers using standard sparse linear equations solving techniques*", J. Dongarra and W. Gentzsch, eds., Elsevier Science Publishers B.V., New York, 1995.

[6] Eijkhout V., "*LAPACK working note 50: Distributed sparse data structures for linear algebra operations*", Tech. Report CS 92-169, Computer Science Department, University of Tennessee, Knoxville, TN, 1992.

[7] González M. "*Un modelo numérico en elementos finitos para la corriente inducida por la marea. Aplicaciones al estrecho de Gibraltar*". Degree Thesis. ETSECCPB, UPC. Barcelona. 1994.

[8] Lin X., H. Ten cade et. al., "*Parallel simulation of 3-D flow and transport models within the NOWESP project*", Int. Journal of the Fed. of European Simulation Societies, Elsevier eds., vol. 3, Nros. 4-5, 1995, pp. 257-272.

[9] Ortega J., "*Introduction to Parallel and Vector Solution of Linear Systems*", Plenum Press, New York and London, 1988.

[10] "*Performance Utilities Reference*", SR-2040. Cray Software Documentation. Cray Research Inc. MN 55120 USA. 1994.

[11] Saad Y., "*SPARSKIT: A basic tool kit for sparse matrix computation*", Tech. Report CRSD TR 1029, CSRD, University of Illinois, Urbana, IL, 1990.

[12] Sonneveld P., "*CGS, a fast Lanczos-type solver for nonsymmetric linear systems*", SIAM J. Sci. Statist. Comp., 10, 1989, pp. 36-52.

[13] Tong C., "*A comparative study of preconditioned Lanczos methods for nonsymmetric linear systems*", Tech. Report SAND91-8240, Sandia Nat. lab., Livermore, CA, 1992.

[14] Van der Vorst H., "*The convergence behavior of preconditioned CG and CGS in the presence of rounding errors*", O. Axelsson and L. Kolotilina eds., vol. 1457 of Lecture Notes in mathematics, Berlin, New York, 1990.