Numerical Platon: a unified linear equation solver interface for industrial softwares

Bernard Sécher, Michel Belliard, Christophe Calvin

HAL Id: hal-00274480
https://hal.archives-ouvertes.fr/hal-00274480

Submitted on 18 Apr 2008

Numerical Platon: a unified linear equation solver interface for industrial softwares

Bernard Sécher [b,*], Michel Belliard, [c] Christophe Calvin [d]

French Atomic Energy Commission (CEA), Nuclear Energy Division (DEN), France

[b]*CEA Saclay DM2S/SFME/LGLS, Bât. 454, F-91191 Gif-sur-Yvette Cedex*

[c]*CEA Cadarache DER/SSTH/LMDL, Bât. 238, F-13108 Saint-Paul-lez-Durance Cedex*

[d]*CEA Saclay DM2S/SERMA/LLPR, Bât. 470, F-91191 Gif-sur-Yvette Cedex*

Abstract

This paper describes a tool called Numerical Platon developed by the French Atomic Energy Commission (CEA). It provides an interface to a set of parallel linear equation solvers for high-performance computers that may be used in industrial software written in various programming languages. This tool was developed as part of considerable efforts by the CEA Nuclear Energy Division in the past years to promote massively parallel software and on-shelf parallel tools to help develop new generation simulation codes. After the presentation of the package architecture and the available algorithms, we show examples of how Numerical Platon is used in CEA codes.

Key words: Parallel linear solvers, Unified interface, PETSc, HyPre

* Corresponding author. Tel.: +33.(0)1 69 08 73 78, Fax : +33.(0)1 69 08 10 87
  Email address: bsecher@cea.fr (Bernard Sécher).

1 Introduction

For years, the scientists working for the Nuclear Energy Division of the French Atomic Energy Commission (CEA/DEN) have developed numerical simulation codes in various programming languages. These developments aim at making significant steps towards building a numerical nuclear reactor. This is particularly the case of the European Platform for NUclear REactor SIMulations (NURESIM). Most of these codes need to solve linear systems. However, the resolution of these linear systems uses a large share of program's CPU load, e.g. up to 90% of the total CPU time. Moreover, these resolution methods are shared between codes. From a strategic viewpoint, it was important for CEA to mutualise and to optimize this part of the simulation process as efficiently as possible. This optimization step requires using parallelism both in hardware architecture and in software. Nuclear simulation has come to heavily rely on massively parallel machines, and developed codes must use these machines in the best way. This is why it is very important to have high-performance parallel linear solvers.

At the beginning of this project, numerous libraries for the resolution of linear systems were available in open-source projects. Rather than concentrate on developing a new library, the CEA preferred to benefit from what already existed. Furthermore, to avoid depending on any given library, in particular for obvious reasons of continuation, the CEA decided to develop a tool that uses several of these available libraries, while offering the possibility to add in-house algorithms according to the user's needs. The choice thus went towards the development of a standardized interface to several existing libraries. This tool was to be available in various programming languages like C, C++, FORTRAN, Ocaml or python so it could be used by the many simulation codes developed by the CEA/DEN. The interest in having

several libraries accessible from a unique tool should make it possible for each user to reach the best existing algorithm to solve the specific problem.

To promote code reusability, flexibility, portability and upgradeability, the CEA/DEN decided to develop Numerical Platon (NP). It is a standardized interface which allows users to reach multiple available libraries of linear equation solvers from a single interface without any change in the code, while being able to manage distributed and shared parallelism in a transparent way. It supports data and processing parallelism, but is optimal on scalar machines. It offers primitives of read/write on file while hiding the problems of parallel accesses. Numerical Platon is ported and can be installed on many different architectures like PC/linux, IBM/aix, alpha/osf, SUN/solaris, SGI/irix, HP/ux, ...

The package can be accessed from an application through a straightforward interface defined in the form of procedure calls. NP includes many mechanisms needed within numerical works, such as vector and matrix structure operators. The library is organized hierarchically, enabling users to employ the most appropriate level of abstraction for a particular problem. For instance, users can directly use a NP conjugate gradient routine or can write their own conjugate gradient through a combination of vector and matrix routines. The designers of NP sought to make use of the most generic interfaces of a number of existing sequential or parallel numerical libraries, rather than selecting one of them and adopting it as the standard for the CEA/DEN. The current NP V3.0 version is based on PETSc 2.3.3 (Balay et al., 2001) and HyPre 2.0.0 (Falgout and Yang, 2002) for the distributed memory machines and CEA OpenMP routines (Dagum and Menon, 1998) for the shared memory machines. Thus, the CEA/DEN has developed several numerical solvers and precondionners in OpenMP (cf. Table 1). NP is maintained by the CEA and is licensed under the terms of the GNU Lesser General Public license.

3

At the time the project NP was launched, no equivalent tool existed though various equivalent projects have since appeared. The Trilinos project, for example, (Heroux et al., 2005) developed by the Sandia laboratory and built around the Aztec library of iterative solvers (Tuminaro et al., 1999), offers interfaces towards tools which provide either direct solvers or linear algebra. Moreover, the PETSc library notably enriched its interfaces towards external packages.

This paper will first discuss software for linear algebra that is freely-available on the web. We have separated these tools into three groups: the linear algebra elementary libraries, the integrated linear solvers, and the interfaces towards external linear solver libraries. The Numerical Platon library is then examined, with a description of its architecture based on three levels and the different available algorithms. The next section presents two examples of how the Numerical Platon library is used in CEA codes. In the first example, Numerical Platon makes it possible to parallelize the linear solvers of a typical industrial sequential code. In the second example, Numerical Platon is used in an already parallelized code. Finally, we conclude and discuss possible extensions of the library.

2 A brief survey of software for linear algebra freely-available on the web

This section discusses software for linear algebra that is freely-available on the web, including support for questions and bug reports. This survey is not exhaustive and interested readers can consult the Dongarra web site for more details (Dongarra, 2006). We have classified this software into three types as follows (from simple to complex): tools for linear elementary operations (as dot product, . . .), tools for integrated linear solvers (as conjugated gradient, . . .) and toolkits integrating the two above-mentioned types of tools.

2.1 Linear algebra elementary operations

The most popular tools for sequential linear algebra elementary operations are the BLAS (Blackford et al., 2002), FLAME (Gunnels et al., 2001) and ATLAS (Whaley et al., 2001) libraries for dense, triangular, banded or tridiagonal matrices. ATLAS provides C and FORTRAN 77 interfaces to an efficient BLAS implementation, as well as a few routines from LAPACK. SparsKit, developed by Y. Saad (Saad, 1990) supplies support routines for sparse matrices (to add two sparse matrices, to reorder a sparse matrix, etc. . . .). One of the goals of the SparsKit package is to provide basic tools to facilitate exchange of software and data (for example the Harwell/Boeing collection of matrices) between researchers in sparse matrix computations.

There are some libraries for parallel linear algebra operations in distributed memory, such as the libraries provided by the Trilinos package (Epetra and Teuchos) from the Sandia National Laboratory (SNL) (Heroux et al., 2005).

2.2 Integrated linear solvers

Other very popular libraries providing direct solvers for dense matrices are the sequential LAPACK (Anderson et al., 1999) and FLAME libraries. ScaLAPACK (Blackford et al., 1997) is a distributed memory parallel version of LAPACK, based on MPI or PVM. The MUMPS (Amestoy et al., 2001) (Esprit IV European project PARASOL), the SuperLU (Demmel et al., 1999; Li and Demmel, 2003) and Trilinos/Amesos packages (Sala et al., 2006) are a few good examples of distributed memory general sparse matrix direct solvers.

The two most broadly used libraries are the PETSc and the HyPre libraries for

iterative solvers in a parallel distributed memory context. They have several specificities. PETSc from the Argonne National Laboratory (ANL) addresses dense and sparse matrices, as well as parallel direct solvers. HyPre from the Lawrence Livermore National Laboratory also works with shared memory matrices and is known to have powerful algebraic multigrid solvers such as BoomerAMG, and conceptual interfaces for structured or block structured and finite element space discretization. It is worth pointing out that PETSc includes interfaces to functions of other packages such as HyPre. Among these interfaces we can name the SPAI library for sparse approximated inverse matrix computation, the Blocksolve95, or the SuperLU (sequential, parallel shared or distributed memory) for incomplete LU preconditioner. It is also worth mentioning the pARMS library from Y. Saad (Saad and Sosonkina, 2004) based on a preconditioned Krylov subspace approach and domain decomposition methods, and the Aztec library (Tuminaro et al., 1999) from SNL concerning iterative solvers in distributed memory.

2.3   Toolkits

Among initiatives similar to NP, once again the Trilinos package developed at SNL is worth citing. The Trilinos Project intends to develop and implement robust parallel algorithms using modern object-oriented software design, while still leveraging the value of established numerical libraries. It is built around the Aztec library which provides iterative solvers. This tool gives a collection of compatible software packages that support direct solvers, parallel linear algebra computations, solution and optimization of linear, non-linear, transient and eigen systems of equations and related capabilities.

In the last few years, the PETSc library from ANL has developed many interfaces

toward other packages, and in this sense became a tool equivalent to NP.

However, this does not minimise the interest that the CEA has for Numerical Platon. It simply means that the CEA does not have to depend on only one external tool and will be able to better meet user needs by adding new available libraries and CEA/DEN numerical methods. Moreover, NP does not have its own format of distributed data, but directly takes the internal format of the library used like PETSc or HyPre. Using PETSc or HyPre via NP does not result in any more copies of objects than directly using one of these libraries.

3 Numerical Platon architecture

3.1 The NP infrastructure

Figure 1 shows a diagram of the relationships between the different levels of the Numerical Platon infrastructure. It is built on three levels. The lower level represents the data structures of the libraries used such as PETSc and HyPre, or the OpenMP CEA/DEN solvers. In sequential runs or shared memory (OpenMP), the user data pointers are transmitted to the NP data structures without any memory duplication. In distributed memory (MPI), the initially sequential or distributed user data structures are spread over the processor memories. The middle level represents the NP API available in different programming languages. Lastly, the upper level represents the numerical methods used in CEA/DEN codes. It would be interesting in the future to include a high level numerical method in the NP API from a CEA/DEN code if this method can interest other users.
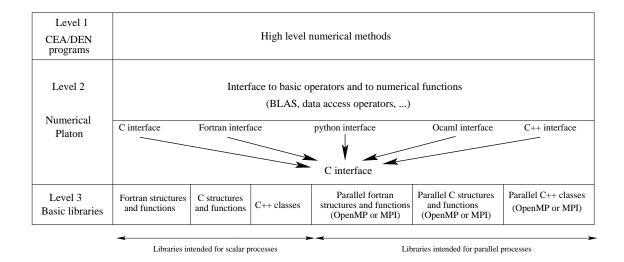
Level 1 CEA/DEN programs	High level numerical methods				
Level 2 Numerical Platon	Interface to basic operators and to numerical functions (BLAS, data access operators, ...)				
	C interface	Fortran interface	python interface	Ocaml interface	C++ interface
			C interface		
Level 3 Basic libraries	Fortran structures and functions	C structures and functions	C++ classes	Parallel fortran structures and functions (OpenMP or MPI)	Parallel C structures and functions (OpenMP or MPI)	Parallel C++ classes (OpenMP or MPI)

Libraries intended for scalar processes   Libraries intended for parallel processes

Fig. 1. The three layers of the Numerical Platon architecture.

3.2 NP components

Numerical Platon consists of a variety of components (Colombet and Sécher, 2004). In most cases, each component manipulates two abstract types named NP_vector and NP_matrix using particular encapsulating data structures. Access to the internal representation of vectors and matrices is not necessary and is discouraged. This allows algorithm improvement or data structure modifications without adjustment of application programs using the package. In other words, vectors and matrices are defined as new data types manipulated by the corresponding supporting routines. The basic operations are implemented in order to allow the programming of linear algebra algorithms in a natural way. Numerical Platon version V3.0 provides five modules dealing with:

- Environment
- Vectors
- Matrices (sparse and dense)
- IO
- Solvers and preconditioners

Each of these components consists of generic interface routines to specific mathematical library routines - PETSc, HyPre or CEA OpenMP routines for instance - in order to promote code reusability, flexibility and portability. Moreover, this approach separates the issues of parallelism from the choice of algorithms.

3.2.1 Environment

This section contains the functions designed to manage parallelism, time measures, error checking and trace for debugging.

The NP programs begin with a phase of initialization which automatically initializes MPI if this has not done before and sets the number of threads given by option on command line in OpenMP environment. If this option is not present, the maximum number of available processors will be used, e.g. two on bi-processors and four on quadri-processors. All NP programs should finalize the NP environment as their nearly final statement. If needed, the user can obtain the total number of processes or threads, or obtain the number of the current process or thread.

The Numerical Platon user has the possibility of imposing the parallel environment of the code inside the sources, in distributed memory or in shared memory. The user can also decide to choose the parallel environment dynamically, depending of the value of an environment variable: NP_MEMORY. If this variable is set to DIST, the code will work in distributed memory, but if the variable is set to SHARE, the code will work in shared memory. So the same executable code can run either with message passing or with threads, without any change in it.

Each Numerical Platon function of C API returns an error status. If this status is null, no error has occurred, otherwise an error has occurred. Users can do whatever

9

they want with errors. Fortran API also returns error status. C++, Ocaml and python API return exceptions on error.

Numerical Platon offers a trace mode. When users take this option, each NP function sends information on standard output. This information is generally the values of function input and output data. This possibility is offered to help the users to debug their codes. This possibility must be used for this purpose only and not in production codes for obvious reasons of performance.

3.2.2 *NP vectors and matrices*

Numerical Platon does not define a single internal format for vectors and matrices, but directly uses the internal format of available libraries used. If the user runs PETSc through NP, the format will be PETSc for vectors and matrices, if the user runs HyPre through NP, the format will be HyPre for vectors and matrices, and if the user runs OpenMP, the format will be the specific Numerical Platon format.

Vectors and matrices are created in two steps. The first step allocates a new NP object structure with only a string for object name. The NP function returns the object pointer. The second step allows the user to specify the characteristics of the object: sequential or parallel, the data type: integer, float or double, the object size, the kind of distribution in case of parallelism, the size of the local part of the object in case of distributed parallelism (NP gives a default distribution if this option is not used). Numerical Platon allows the user to create six formats of matrices: dense format: all the components of the matrices are stocked, symmetric dense format: the upper triangular part of the matrices is stocked, CSR for compressed sparse row: all the non zero components of the matrices are stocked row by row, symmetric CSR: the non zero components of the upper triangular part of the matrices are stocked row
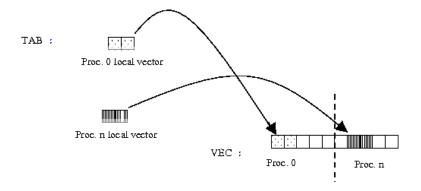
10

Fig. 2. NP vector: data set only in local part of the object.

by row, CSC for compressed sparse column: all the non zero components of the matrices are stocked column by column, symmetric CSC: the non zero components of the upper triangular part of the matrices are stocked column by column. All these formats are available in OpenMP, but in the case of distributed memory, only internal formats of available libraries are accessible (CSR and CSR_SYM for PETSc and CSR for HyPre).

NP has functions to set values in objects. There are two kinds of setting functions in case of distributed parallelism. The first setting is interesting when the NP objects are assembled in parallel and allows the user to set values in the local part of the object only, see Fig. 2. The second setting is interesting when the linear solver only is parallelized and the NP objects are assembled in sequential by only one processor, which allows the user to set values from one processor to the others, see Fig. 3. In the case of distributed parallelism, the user has no access to information on the internal format used, so there is a copy of the values from the input array to the NP object. However, in the case of shared parallelism, NP can directly handle the user arrays to the internal NP object format, which means there is no data recopy.

In a distributed memory run, the NP assembly procedure distributes, if necessary, the object via message-passing communications after having set the object values and, only for a sparse matrix, computes an optimal compression of the matrix. In
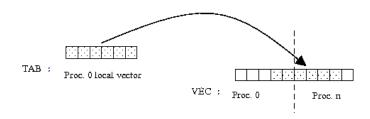
11

Fig. 3. NP vector: data set from one processor to the others.

the particular case where only one process contributes to build the entire matrix, it may be interesting to perform several flushings before building the entire object to avoid having all this object in the local memory before assembling. In a shared memory run, the NP object assembling is only useful on NP sparse matrices. This assembling computes an optimal compaction of the sparse matrix.

Some NP functions allow the user to:get values from NP objects, copy the values of an object from one to another, multiply the values of an object by a single value, compute the dot product of vectors or the following linear combinations of vectors: $x = ax + by$, $w = ax + by$ and $y = ax + y$, compute the matrix vector product, send the object's values to the standard output, and then to destroy NP object in memory.

3.2.3   NP IO

The user can save NP objects in files. These files have a binary format built with XDR standard. In the case of distributed memory parallelism, all processes send data to process zero. Only process zero writes data in files, one NP object by file. The user can read NP objects from files. In the case of distributed memory parallelism, all processes read their local data in the same file.

12

3.2.4 NP solvers and preconditioners

In the current Numerical Platon version which uses PETSc, SuperLU and HyPre packages, the different solvers available are (cf. Table 1): Cholesky and LU factorisations associated with different renumberings like natural, reverse Cuthill-McKee, nested dissection, one way dissection, row length, minimum degree or quotient minimum degree. The different iteratives solvers available are: conjugate gradient, GMRES, biconjugate gradient stabilized, conjugate gradient squared, transpose free quasi-minimal residual, Richardson and multigrid methods.

Some preconditioners are available in association with these solvers: symmetric successive over relaxation (SSOR), incomplete Cholesky factorisation, incomplete LU factorisation (ILU(k)), dual threshold incomplete LU factorisation (PILUT), diagonal, sparse approximate inverse (SPAI), additive Schwarz and polynomial preconditioners

**Here: Table 1 **

4   Numerical Platon in industrial codes

As little knowledge is required on parallelisation techniques (MPI or OpenMP),the effort to incorporate NP routines in user codes is rather insignificant. Sequential (Grandotto et al., 1989; Grandotto and Obry, 1996) or parallel (Calvin et al., 2002) CEA industrial software have included NP solvers and have increased their numerical method capacities, both in sequential runs on desktop computers and in parallel ones on massively parallel computers. This is particularly crucial for CFD software with projection schemes for which the CPU time spent in the iterative pressure solver can be as high as 90%.

Moreover, Numerical Platon provides users with the possibility of designing the same code for a large range of computers, from one-processor PCs to massively parallel mainframes. An even more interesting aspect is the possibility of using a broad range of sequential or parallel linear solvers in different existing numerical libraries like PETSc or HyPre without any change in the user interface. With a given specific library implementation, the change in solver library often involves re-writing an important part of the user code with an extra change in the data structures. This flexibility largely balances the slight overhead due to the NP interface between the user code and specialized libraries. This makes it easier to use the best appropriate solver for the user problem without change or extra development. It can be understood in terms of numerical method choice (direct methods, iterative methods for symmetric or non-symmetric matrix, Schwarz, algebraic multigrid, ...). Also it can be understood in terms of versatility considering the target computer system (distributed or shared memory, mono-processor or multi-processors, ...). In particular, the choice of the computer memory type is done via an environment variable, by specifying the type or letting NP decide by itself.

At the CEA, we have so far been able to prove that there is no overhead involved in using NP rather than directly using a library like Petsc or HyPre. Performances in CPU time or in memory are the same, and effort to incorporate NP into a given code is similar to directly incorporating one of the above-mentioned solvers like Petsc or HyPre (cf. Table 2). CPU times on one processor are explained by the fact that sequential jobs use dual core, but parallel jobs use single core.

Today, many CEA codes have introduced NP APIs that have also been evaluated at EDF, the main French electric power supplier. This is done to be able to have linear system parallel resolutions in the context of sequential data or to increase the choice of parallel solvers in the context of distributed data. The following section reviews a particular example of each type of target code architecture.

**Here: Table 2 **

4.1 NP and sequential data codes

In this category we consider *classical* industrial codes initially written as sequential codes for which the data are located on a unique node (shared memory with one or many processors). Even if the parallel build of the matrix is not allowed by the code design, NP provides a way to solve the linear systems in parallel. With this goal, NP was successfully introduced into the following CEA codes: Genepi (Grandotto et al., 1989; Grandotto and Obry, 1996; Belliard, 2001), Flica IV (Toumi et al., 2000), Apollo II (Sanchez et al., 1988), Ovap (Kumbaro and Seignole, 2001; Kumbaro et al., 2002), Alliances project (Montarnal et al., 2006). Below is a review of the Genepi code.

4.1.1 The Genepi code case

The CEA Genepi code is devoted to simulating the 3D steady two-phase flow in steam generators (SG) of French nuclear power plants. The French SG manufacturer considers this code as a benchmark in the industrial computation chain for SG studies. Consequently, it is not possible to deeply modify its structure to transform it into a distributed memory parallel code.

15

The re-engineering of the code needed to introduce the use of the NP library (FOR-TRAN API) was easy due to the modularity and the orthogonality of the data and procedures in Genepi. The matrix building is always done in sequential. In the context of shared memory parallelism, CSR data and vector data pointers are put in the `NP_vector` and `NP_matrix` structures, without extra memory cost, and in-house openMP solvers are run. In the context of distributed memory parallelism, we set up parallel computation servers in a master/slave manner. Only one task (master, processor rank 0) manages the computation algorithm and the data in a sequential way. The other tasks (slave, processor rank $\neq 0$) are specialized as servers of parallel services for distributed parallelism, see Figure 4. All the tasks contribute to the data distribution and the parallel resolution of linear systems. In this case, CSR and vector data are copied in the `NP_vector` and `NP_matrix` structures, with an extra memory cost.

Moreover, the original Genepi solver toolkit is largely improved by the NP library without code change, even for sequential runs. Once the software is modified so it can interface with a high-level NP solver, no more work is to be done to use any other NP solver. More specifically, the independent choice of the preconditioner and the solver (generally not the case in an in-house code) leads to very versatile algorithms adapted to the target matrix. Even if the high-level solvers provided by NP are very efficient, in-house Genepi solvers were also parallelized with the NP parallel elementary operations listed in Section 3.2.2.

4.1.1.1 An example: the pressure computation by a projection scheme. In computational fluid dynamics, projection schemes (Gresho and Chan, 1990; Ferziger and Peric, 1996) are widely used to compute the velocity - pressure couple for the Navier - Stokes equation. After a prediction step in which a predicted velocity is
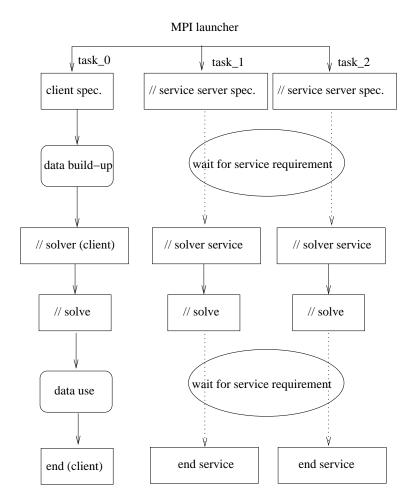
16

Fig. 4. Master/slave mode for the parallel resolution in the Genepi code

obtained from a given pressure, a projection step allows the computation of a pressure increment and a corrected velocity. This last step needs solve an elliptic PDE. Initially in the Genepi code, the linear system for this PDE was solved by the LU method in sequential. It is very important to precisely solve the linear system since the mass conservation depends on it.

For example, a test case involving 150,000 cells needs 1.9 GB memory, which is nearly the maximum memory available with a 32-bit computer architecture. Well-known memory limitations result from the filling of the matrix during the factorization phase, and iterative methods are needed to increase the number of cells with the same memory amount. We recall that the matrix is assembled on a unique node before the sequential or parallel solve phase. By using a matrix compact storage

and the diagonal preconditioned conjugate gradient (PDCG) provided by PETSc through the NP library, a 500,000 cell test case can be run using only 1.7 GB. Using a stop criterion of 10^{-9} for the relative residual diminution, the pressure increment resolution is very CPU-time-consuming for a large number of cells. For 500,000 cells in sequential, the CPU time for one pressure computation is about 90% of one time step (CPU time) and about 10,000 time steps must be run to reach the stationary regime. The DIGITAL EV68 1.25 Ghz processor needs about half an hour, see Table 3. Hence, parallel computations are an absolute necessity. Table 3 presents parallel computation results in a distributed memory context. The speed-up values are very good and we believe that full transient computations will be possible in the future.

**Here: Table 3 **

4.2   NP and distributed data codes

In this category, we consider *advanced* industrial codes initially written as parallel codes for which the data are distributed on nodes (distributed memory with one or many processors by node): Trio-U (Calvin, 2003a; Calvin et al., 2002; Calvin, 2003b), Alliances (Montarnal et al., 2006) or Ovap (Kumbaro and Seignole, 2001; Kumbaro et al., 2002). NP provides a way to increase the choice of available parallel solvers. Below is a review of the Trio-U code.

4.2.1   Trio-U code case

Trio-U was designed to solve large 3D structured or unstructured CFD problems. The code is intrinsically parallel, and an object-oriented design, UML, is used. The

implementation language chosen is C++. All the parallelism management and the communication routines have been encapsulated. Parallel I/O and communication classes over standard I/O streams of C++ have been defined, which allows the developer to easily use the different modules of the application without dealing with basic parallel process management and communications. Moreover, the encapsulation of the communication routines guarantees the portability of the application, thus providing the efficient tuning of basic communication methods in order to achieve the best performance level for the target architecture. This new generation code was developed from the beginning in distributed memory context. Hence, the code was already a parallel one, but NP is used to reach new parallel solvers from PETSc or HyPre libraries without large new developments. The matrix assembly and the resolution are done in parallel (SPMD).

4.2.1.1 Trio-U and NP interface The Trio-U code focuses on the resolution of the linear system in pressure resulting from the resolution of the Navier-Stokes equations. In the incompressible case, the resolution of the Navier-Stokes equations with an Euler description implies that the pressure matrix is constant and depends on the discretization only. The resolution algorithms are dependent neither on the discretization, nor on the solved equations. In the Trio-U design, the Navier-Stokes equation carries the linear system to be solved. An assembler object, which specializes according to the discretizations, has the main function of assembling the matrix. Another object, the `SolveurSys` has a method to resolve the linear system.

In the Trio-U code, the pressure matrices are symmetric, sparse and unstructured. They are stored in the symmetrical Compressed Sparse Row format. However, other matrix formats are also used (dense, diagonal or by blocks) for other systems or when certain algorithms are required. The matrices per blocks are used in particular

19

for the parallel cases.

The main algorithm of resolution used in Trio-U is the conjugate gradient algorithm. In order to improve convergence of this algorithm, it is necessary to use preconditioning. Currently the SSOR algorithm is generally used, although it is also possible to use an incomplete factorization LU.

Trio-U is a parallel code, thus it is important to have effective parallel algorithms for the resolution of linear systems. The method of parallelization in Trio-U is based on a domain decomposition into as many domains as there are processors used for the calculation. Domain overlapping is used in order to ensure the coherence of the results calculated in parallel. The vectors are thus represented by distributed vectors which have a real part representing the local data with the domain considered, and a virtual part corresponding to overlapping with the neighbouring domains.

The use of NP for Trio-U consists in the possibility of proposing parallel linear solvers that are more efficient than those currently available. The main interfacing difficulty lies in the fact that the data - matrices and vectors - are already distributed in the code. It was thus necessary to have the initial distribution of the data coincide with the distribution suggested by Numerical Platon.

The principle of the data distribution in NP is simple and complies with the majority of the parallel scientific libraries. The matrix is distributed per consecutive blocks of lines on the various processors. The vectors concerned follow the same distribution. This imposes two conditions: a processor must have complete lines of the matrix and, considering one processor, the lines of the matrix must be contiguous. In Trio-U, the matrices are mainly sparse and thus the first condition is met. With regard to the second condition, it is necessary to renumber the elements correctly to ensure single global numbering so that all the domains satisfy the second condition.
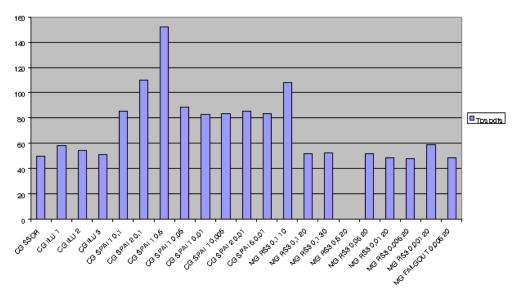
Fig. 5. Performance of the different algorithms in the sequential case in the Trio_U code (average time of one time step in seconds)

Various tests were performed while varying the different algorithms of resolution and machine architectures available. The benchmark was a calculation of a T-piece of mixture in a Finite Element Volume discretization of 170,000 tetrahedrons with a Large Eddy Simulation model. Simulation was performed on ten time steps. We were interested in measuring the performance at the average time of one time step.

Tests were run on a massively parallel mainframe and on a department PC cluster.

4.2.1.2   Results on HP cluster   The HP/COMPAQ SC280 machine was made up of 70 nodes ES40 (quadri processors EV68 to 833 MHz with 1 Gb of shared memory) and inter-connected via a Quadrics network. Figure 5 gives the results of the tests performed in a sequential case with the different algorithms: in-house CG SSOR in comparison with some NP solvers.

It can be noticed that, except for ILU preconditioning, the parameters of the different algorithms are numerous and their variation involves considerable differences
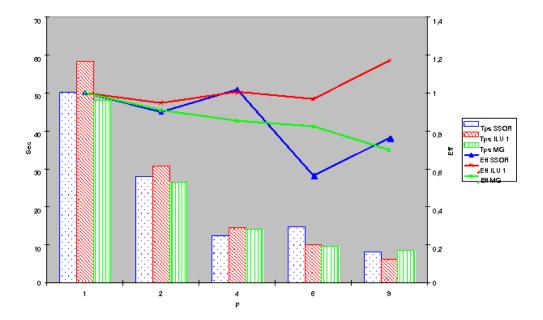
21

Fig. 6. Performance of the different algorithms in the parallel case in the Trio_U code (average time of one time step in seconds and parallel efficiency)

in performance. If the average time of a time step obtained by using the standard algorithm of Trio-U is used as a reference base, i.e. the conjugate gradient combined with a SSOR preconditioning, the best performance levels are obtained with an ILU pre-conditioning and a multigrid algorithm. However the profit does not exceed 3 to 4%.

After having tested the sequential performances of the various algorithms, we evaluated these algorithms in parallel, by taking the optimal parameters of the sequential one. The results obtained are summarised in Figure 6. We observed that the parallel efficiency is good regardless of the algorithm used.

4.2.1.3 Results on PC cluster This cluster was made up of 56 given Pentium IV processors from 1.7 to 2.4 GHz, having 1 Gb of RAM and inter-connected via a fast SCI network. The main performance measurements were given on an algorithm of conjugate gradient with preconditioning. We compared the standard
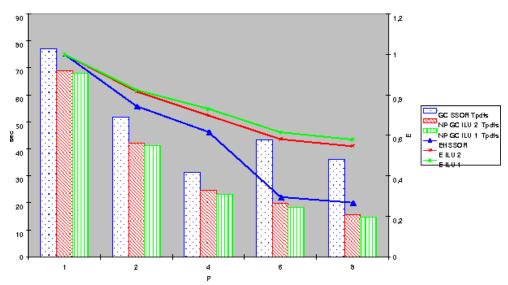
22

Fig. 7. Parallel performance on PC cluster in the Trio_U code (average time of one time step in seconds and parallel efficiency)

preconditioning (in-house Trio-U SSOR) with ILU preconditioner provided by NP. The other algorithms were tested, but revealed a bug implementation of MPI on the fast network of the PC cluster. Figure 7 shows the results of performance on the cluster (average time of one time step and parallel efficiency defined by the sequential/parallel time ratio).

It was remarked that the gain in term of performance levels was notable. This was true for the sequential case (about 13%) but especially for the parallel case where we obtained an efficiency close to 0.6, whereas it was only 0.25 with a standard algorithm (60% gain in term of CPU time).

5 Conclusion

This paper has presented the NP project and its final purpose, which is to provide a unified interface towards several freely available software (mainly PETSc and

23

HyPre) for parallel linear algebra operations ranging from vector and matrix operations to integrated linear solvers. Special attention was given to minimising the data copy between structures as well as possible in order to preserve the performance levels.

The main features of the APIs, the NP routines and the abstract types of the NP structures have been presented. Interfaces with various programming languages (C, C++, FORTRAN, Ocaml and python) allow easy implementation for the user. Moreover, coming with a LGPL license and interfaces to freely available libraries, the sources of the codes are available. This is of great importance for industrial software stability in time. In order to be compatible with the external libraries, the necessary periodic efforts are only done once by the NP's development team and not many times by each client software using PETSc or HyPre separately. This is an important aspect.

Among all the CEA codes having incorporated the NP library (Alliances, Apollo II, Flica IV, Genepi, Ovap, Trio-U), we have shown how the linear algebra solvers of a typical industrial sequential code (Genepi) can be parallelized without considerable extra development cost, which opens new perspectives for meshes with a large number of cells. Furthermore, we have presented an example of a CEA new generation parallel code (Trio-U) using the NP interface to get flexibility in the choice of linear algebra solvers. This gave us the opportunity to test new methods and adapt the solver choice to a particular problem to solve.

Concerning future prospects for Numerical Platon, we are expecting the integration of the CEA SLOOP library (Meurant, 2001; Colombet et al., 2004) Parallel Object Oriented Linear Solvers, in French) including, among other methods, efficient algebraic multigrid algorithms written in C++.

References

Amestoy, P., Duff, I., L'Excellent, J.-Y., Koster, J., 2001. A fully asynchronous multifrontal solver using distributed dynamic scheduling. SIAM Journal on Matrix Analysis and Applications 23 (1), 15–41.

URL http://mumps.enseeiht.fr

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D., 1999. LAPACK Users' Guide, 3rd Edition. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

URL http://www.netlib.org/lapack/index.html.

Balay, S., Buschelman, K., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., Zhang, H., 2001. PETSc Web page.

URL http://www.mcs.anl.gov/petsc

Belliard, M., 2001. Introduction de la librairie Numerical Platon dans le logiciel Génépi. Technical report DEN/DTN/STH/LTA/2001-27, CEA, Cadarache, (in French).

Blackford, L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R. C., 1997. ScaLAPACK Users' Guide. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

URL http://www.netlib.org/scalapack

Blackford, L. S., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., Heroux, M., Kaufman, L., Lumsdaine, A., Petitet, A., Pozo, R., Remington, K., Whaley, R. C., 2002. An updated set of basic linear algebra subprograms (blas). ACM Trans. Math. Soft. 28 (2), 135–151.

URL http://www.netlib.org/blas/index.html

25

Calvin, C., 2003a. A development framework for parallel CFD applications: Trio-U project. In: SuperComputing in Nuclear Applications (SNA-2003). Paris, France.

Calvin, C., 2003b. L'interfacage de Numerical Platon avec le code Trio-U. Technical report DEN/DTP/SMTH/LDTA/2003-078, CEA, Grenoble, (in French).

Calvin, C., Cueto, O., Emonot, P., 2002. An object-oriented approach to the design of fluid mechanics software. M2AN 36 (5), 907–921.

Colombet, L., Meurant, G., al., 2004. Manuel utilisateur de la bibliotheque (SLOOP) 3.2 SLOOP 3.2 users manual. Technical report, CEA/DIF/DSSI/SNEC.

Colombet, L., Sécher, B., 2004. NUMERICAL PLATON User Guide and Reference Manual. Technical report DM2S/SFME/LGLS/RT/01-001, CEA, Saclay.

Dagum, L., Menon, R., 1998. OpenMP: An Industry-Standard API for Shared-Memory Programming. IEEE Computational Science and Engineering 5 (1), 46–55.
URL http://www.openmp.org

Demmel, J. W., Gilbert, J. R., Li, X. S., 1999. An asynchronous parallel supernodal algorithm for sparse gaussian elimination. SIAM J. Matrix Analysis and Applications 20 (4), 915–952.
URL http://crd.lbl.gov/ xiaoye/SuperLU

Dongarra, J., 2006. Freely available software for linear algebra on the web.
URL http://www.netlib.org/utk/people/JackDongarra/la-sw.html

Falgout, R., Yang, U., 2002. Hypre: a Library of High Performance Preconditioners., c.j.k. tan. j.j. dongarra, and a.g. hoekstra Edition. Vol. 2331 of Lecture Notes in Computer Science. Springer-Verlag, pp. 632–641.
URL http://www.llnl.gov/CASC/linear_solvers/

Ferziger, J. H., Peric, M., 1996. Computational Methods for Fluid Dynamics.

Springer, New York, USA.

Grandotto, M., Bernard, M., Gaillard, J., Cheissoux, J., De Langre, E., November 1989. A 3D finite element analysis for solving two-phase flow problems in PWR steam generators. In: 7th International Conference on Finite Element Methods in Flow Problems. Huntsville, Alabama, USA.

Grandotto, M., Obry, P., 1996. Calculs des écoulements diphasiques dans les échangeurs par une méthode aux éléments finis. Revue Européenne des Eléments Finis 5 (1), 53–74, (in French).

Gresho, P., Chan, S., 1990. On the theory of semi implicit projection methods for viscous incompressible flow and its implementation via finite element method that also introduces a nearly consistent matrix. i, theory. International Journal for Numerical Methods in Fluids 11 (5), 587–620.

Gunnels, J. A., Gustavson, F. G., Henry, G. M., van de Geijn, R. A., Dec. 2001. FLAME: Formal Linear Algebra Methods Environment. ACM Transactions on Mathematical Software 27 (4), 422–455.
URL `http://www.cs.utexas.edu/users/flame`

Heroux, M. A., Bartlett, R. A., Howle, V. E., Hoekstra, R. J., Hu, J. J., Kolda, T. G., Lehoucq, R. B., Long, K. R., Pawlowski, R. P., Phipps, E. T., Salinger, A. G., Thornquist, H. K., Tuminaro, R. S., Willenbring, J. M., Williams, A., Stanley, K. S., 2005. An overview of the trilinos project. ACM Transactions on Mathematical Software 31 (3), 397–423.
URL `http://trilinos.sandia.gov`

Kumbaro, A., Seignole, V., 2001. Two-Phase Flow Computing with OVAP Code. In: Trends in Numerical and Physical Modeling of Industrial Two-phase Flow. Cargèse, Corsica, France.

Kumbaro, A., Toumi, I., Seignole, V., April 14-18 2002. Numerical Modeling of Two-Phase Flows Using Advanced Two Fluid Systems. In: Tenth International

Conference on Nuclear Engineering (ICONE-10). Arlington, Virginia, USA.

Li, X. S., Demmel, J. W., June 2003. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. ACM Trans. Mathematical Software 29 (2), 110–140.

URL http://crd.lbl.gov/ xiaoye/SuperLU

Meurant, G., 2001. Numerical experiments with algebraic multilevel preconditioners. Electronic Transactions on Numerical Analysis 12, 1–65.

Montarnal, P., Bengaouer, A., Chavant, C., Loth, L., 2006. ALLIANCES: Simulation platform for nuclear waste disposal. In: Proc. of CMWR06. Copenhagen, Denmark.

Saad, Y., 1990. Sparskit : A basic tool kit for sparse matrix computations. Technical report RIACS-90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA, USA.

URL http://www-users.cs.umn.edu/ saad/software

Saad, Y., Sosonkina, M., 2004. pARMS : A package for the parallel iterative solution of general large sparse linear systems user's guide. Technical report UMSI2004-8, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, USA.

URL http://www-users.cs.umn.edu/ saad/software

Sala, M., Stanley, K., Heroux, M., 2006. Amesos: A set of general interfaces to sparse direct solver libraries. In: Proceedings of PARA'06 Conference, Umea, Sweden.

URL http://trilinos.sandia.gov/packages/amesos

Sanchez, R., Mondot, J., Stankovski, Z., Cossic, A., Zmijarevic, I., 1988. APOLLO II: A user-oriented, portable, modular code for multigroup transport assembly calculations. Nucl. Sci. Eng. 100, 352.

Toumi, I., Bergeron, A., Gallo, D., Royer, E., Caruge, D., 2000. FLICA-4 code:

a three-dimensional two-phase flow computer code with advanced numerical methods for nuclear applications. Nuclear Engineering and Design 200 (1-2), 139–155.

Tuminaro, R. S., Heroux, M., Hutchinson, S. A., Shadid, J. N., 1999. Official Aztec User's Guide: Version 2.1.

URL http://www.cs.sandia.gov/CRF/aztec1.html

Whaley, R. C., Petitet, A., Dongarra, J., 2001. Automated empirical optimization of software and the atlas project. Parallel Computing 27 (1-2), 3–35.

URL http://math-atlas.sourceforge.net

List of Figures

List of Tables

Methods		PETSc+SuperLU	HyPre	multi-threaded	
				SuperLU	CEA OpenMP
Solvers	Conjugate gradient	parallel	parallel	-	parallel
	GMRES	parallel	parallel	-	parallel
	BICGSTAB	parallel	parallel	-	parallel
	CGS	parallel	-	-	parallel
	TFQMR	parallel	-	-	-
	Richardson	parallel	-	-	-
	multigrid	-	parallel	-	-
	LU factorisation	parallel	-	parallel	-
	Cholesky factorisation	parallel	-	parallel	-
Preconditioners	SSOR	sequential	-	-	sequential but can be used with parallel solver
	ILU(k)	parallel for k=0 sequential for k>0	parallel	-	sequential but can be used with parallel solver
	PILUT	-	parallel	-	-
	diagonal	parallel	parallel	-	parallel
	SPAI	parallel	parallel	-	-
	polynomial	parallel only for CG	-	-	parallel only for CG
	additive schwarz	parallel	-	-	-

Table 1

Available solvers and preconditioners in Numerical Platon

	1 proc	4 proc	9 proc	16 proc	25 proc	64 proc
HYPRE	657.2	307.5	181.0	95.4	67.1	18.3
NP/HYPRE	641.1	308.9	181.2	93.6	67.3	18.6

Table 2

NP+HyPre/HyPre comparison: CPU time (in seconds) to solve linear system on a cluster

with conjugate gradient without pre-conditioning

Proc. Number	CPU time (s)	Speed-up
1	1531	-
4	415	3.7
12	120	12.7
16	95	16.1

Table 3

Distributed memory parallel computations of the projection step in the Genepi code (CEA

HP ES45 cluster: 4 proc. DIGITAL EV68 1.25 GHz / 4 GB. 1 task by node)