

PMD: A Parallel Fortran 90 Module to Solve Elliptic Linear Second Order Equations

Jalel Chergui* - Rémi Jacques[†] - Patrick Lequéré[‡] - Olivier Daube[§]

CNRS/IDRIS

E-mail: Jalel.Chergui@idris.fr

Phone : 01.69.35.85.53

Fax : 01.69.85.37.75

19 mai 2003

RÉSUMÉ. *PMD* (Parallel Multi-Domain decomposition) est un ensemble de modules Fortran 90 dans lequel est implémentée une méthode parallèle de décomposition de domaine sans recouvrement basée sur le complément de Schur dual. La matrice du complément de Schur est construite à l'aide d'une technique de matrice d'influence. Cet article décrira brièvement l'implémentation. Il présentera ensuite un cas d'étude ainsi que les performances obtenues sur trois machines parallèles : IBM SP2, SGI/CRAY T3E-600 et Fujitsu VPP300. Enfin, la localisation de *PMD* sur un serveur public sera communiquée ainsi que son mode d'installation.

ABSTRACT. *PMD* (Parallel Multi-Domain decomposition) is a set of Fortran 90 modules which implements a parallel none-overlapping domain decomposition method based on the so called dual-Schur complement. The Schur matrix is built using an influence matrix technique. In this paper, we will briefly describe the implementation. A case study will then be presented and performance results will be shown on three parallel machines: IBM SP2, SGI/CRAY T3E-600 and Fujitsu VPP300. At the end, the public location of *PMD* will be given together with the installation procedure.

MOTS-CLÉS : performance, parallèle, elliptique, linéaire, second ordre, décomposition de domaine, complément de Schur, matrice d'influence, *PMD*, SP2, T3E, VPP300, LAPACK, ScaLAPACK, MPI

KEYWORDS: performance, parallel, elliptic, linear, second order, domain decomposition, Schur complement, influence matrix, *PMD*, SP2, T3E, VPP300, LAPACK, ScaLAPACK, MPI

* Institut du Développement et des Ressources en Informatique Scientifique. CNRS/IDRIS, Bât. 506, BP167, F-91406 Orsay cedex, France. <<http://www.idris.fr>>

[†] UMR6600, Université de Technologie de Compiègne. BP 20529, 60200 Compiègne, France

[‡] Laboratoire d'Informatique et de Mécanique pour les Sciences de l'Ingénieur. CNRS/LIMSI, UPR 3251, BP133, F-91403 Orsay cedex, France. <<http://www.limsi.fr>>

[§] Laboratoire d'Informatique et de Mécanique pour les Sciences de l'Ingénieur. CNRS/LIMSI, UPR 3251, BP133, F-91403 Orsay cedex, France

1. Introduction

Elliptic linear second order systems are encountered in many engineering applications where physical phenomena are modeled for further mathematical or numerical resolution. These equations can cover large application fields as fluid mechanics, electromagnetic wave propagation and solid structure deformation. Each time one has to solve the following system:

$$\begin{cases} Lu = s & \text{inside the domain} \\ + \text{ Initial condition} & \text{onside the domain} \\ + \text{ Boundary condition} & \text{at physical boundaries} \end{cases}$$

where s is the source term, u is the required solution and L can be any positive definite elliptic linear second order operator. A very famous one is the Laplace operator commonly denoted by Δ .

In order to solve in parallel such kind of equations, PMD implements a Parallel non-overlapping Domain Decomposition ([QUA 90], [ABD 98], [MAM 97]) inside a set of Fortran 90 modules, where generic subroutines are defined. To this end, the computational domain is split into subdomains leading to a classical interface problem. Such a problem is overcome by the dual-Schur complement method ([FAR 93], [TAL 91], [GUY 98]) together with an influence matrix technique [DAU 91] to build the Schur matrix and to solve the interface problem [JAC 97]. Nevertheless, the user has to provide two routines to the PMD solver. One to solve the local problem and the other to compute the interface normal derivatives.

A high level overview of PMD will be given in Section 2. Section 3 describes the use of some PMD routines with a classic example as solving the LAPLACE problem. Section 4 gives performance measurements obtained on IBM SP2, SGI/CRAY T3E [CHER 97] and Fujitsu VPP300 parallel machines. Section 5 presents the advantages and restrictions of PMD. Section 6 is an installation and testing guide of PMD. Finally, future trends of PMD will be discussed.

2. Overview

PMD's current version defines a set of routines to solve 1D and 2D boundary condition problems while the global domain is cut along one axis into non-overlapping subdomains without interface cross-points (Figure 1). In this paper, such a domain decomposition will be denoted by 1DD.

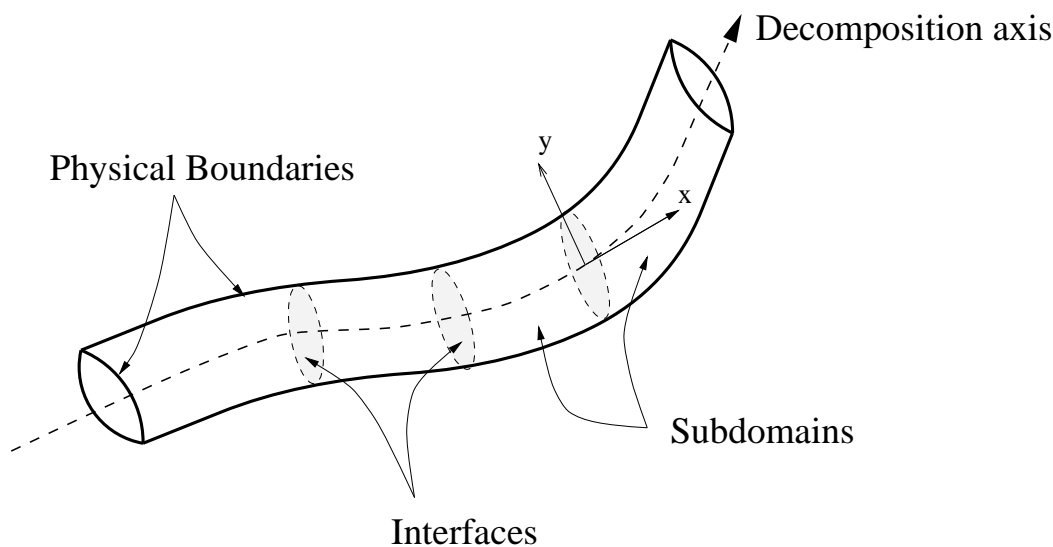


FIG. 1 – 1DD type Domain Decomposition.

For portability and performance issues, PMD routines are built on top of MPI [SNI 96], LAPACK [AND 95] and ScaLAPACK [BLA 97] libraries (Figure 2).

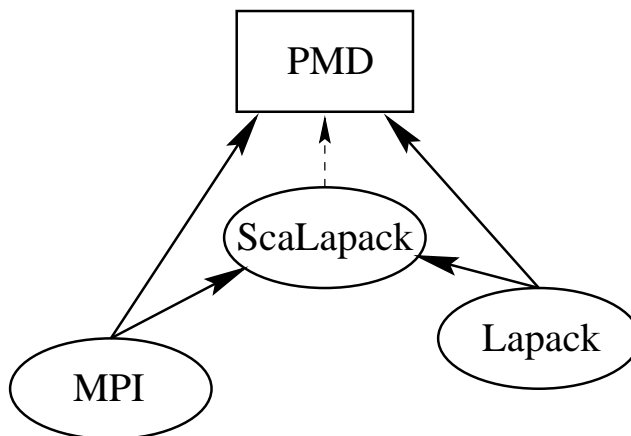


FIG. 2 – *PMD software layers.*

Up to now, PMD includes four Fortran 90 modules:

1. **PMD_Types**: Defines constants, base data type and derived data type objects.
2. **PMD_Tools**: Defines some routines to handel warning and error messages.
3. **PMD_Norms**: Defines routines to compute some classical norms.
4. **PMD_Solver_1DD**: Defines the solver core of PMD.

The user has access to all functionality of PMD by including a `USE PMD` statement into each affected scoping unit. To this end, the PMD module file includes the previous four modules as it is shown in Figure 3.

```

MODULE PMD
  USE PMD_Types
  USE PMD_Tools
  USE PMD_Norms
  USE PMD_Solver_1DD
END MODULE PMD
    
```

FIG. 3 – *PMD module file.*

The decomposition of the physical computational domain is such that each process, of the user MPI process group, deals with one and only one subdomain.

At the user application level, one routine (`PMD_Init_1DD`) must be called prior to any other PMD routine in order to initialize the PMD environment. Conversely, to free all previous PMD settings, a call to the `PMD_End_1DD` routine has to be performed to leave the PMD environment. Between the previous two environment routine calls, the user has to set the local operator matrix and the physical boundary conditions before calling three main PMD subprograms (Figure 4) to solve in parallel the global problem. This means that the third subroutine returns the final solution corresponding to the subdomain to which is attached the calling process. This step represents the core of the PMD solver which main goal is to solve the interface problems.

Today it defines:

1. A parallel influence matrix technique to build the Schur matrix

2. Two parallel algorithms to solve distributed linear systems:
- A direct method for general operators which implements a LU-factorization¹ algorithm
 - An iterative method for symmetric operators which implements a Preconditioned Conjugate Gradient algorithm

```

PROGRAM Laplace
  USE PMD
  !... Declare User and PMD type objects
  ...

  CALL MPLINIT(...)
  !... Begin PMD
  CALL PMD_Init_1DD(...)

  !... Set Local Operator Matrix and Boundary Conditions
  ...

  !... Build the Schur matrix
  CALL PMD_Schur_1DD(...)
  !... Factor the Schur Matrix
  CALL PMD_Schur_Factor_1DD(...)
  !... Solve the interface problem
  CALL PMD_Solve_1DD(...)

  ...

  !... End PMD
  CALL PMD_End_1DD(...)
  CALL MPLFINALIZE(...)
END PROGRAM Laplace
  
```

FIG. 4 – *PMD subroutine calling sequences.*

Starting from a short example, we will see in the next section how to declare the **PMD** subroutine parameters and how to set these objects supposing M , the user local matrix associated to the operator L , u , the required local solution and s , the local source term. **PMD** needs to know about all these variables to build the Schur matrix and to further solve the interface problem.

3. Application examples

Let us consider the following classical 1D Laplace problem formulated in a cartesian coordinate system:

$$\begin{cases} -\frac{\partial^2 u}{\partial x^2} = s(x) = 4.\pi^2 \sin(\pi x) \cos(\pi x) & ; \quad x \in [0,1] \\ u(0) = u(1) = 0. \end{cases}$$

The previous system admits the analytic solution:

1. Based on **PxGETRF** and **PxGETRS** subprograms of the **ScaLAPACK** library.

$$u_a(x) = \sin(\pi x) \cos(\pi x) \quad ; \quad x \in [0,1]$$

Let us denote by N_{sd} the number of subdomains². Each subdomain is divided into N_x regular meshes (Figure 5) so that the spatial discretization step is defined by:

$$h = \frac{1}{N_{sd}(N_x - 1)}$$

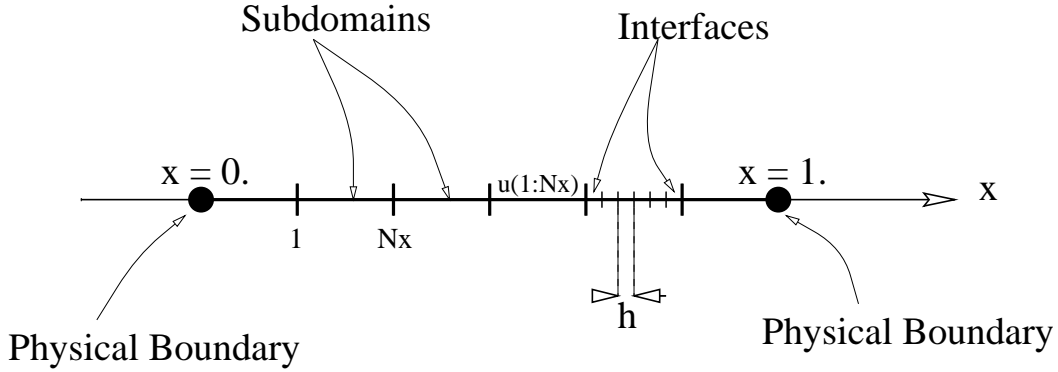


FIG. 5 – Splitting and Discretizing the 1D domain.

In each subdomain $k = 0, \dots, N_{sd} - 1$, the required numerical solution u_i is defined at nodes:

$$x_i = (i + (N_x - 1)k - 1)h \quad ; \quad i = 1, \dots, N_x$$

Standard second order centered differences have been used for the second order operator $\frac{\partial^2 u}{\partial x^2}$ which now reads:

$$\left[\frac{\partial^2 u}{\partial x^2} \right]_i \simeq \frac{1}{h^2} (-u_{i-1} + 2u_i - u_{i+1}) = s_i$$

$$\text{for } \begin{cases} i = 2, \dots, N_x & \text{if } k = 0 \\ i = 1, \dots, N_x & \text{if } k = 1, \dots, N_{sd} - 2 \\ i = 1, \dots, N_x - 1 & \text{if } k = N_{sd} - 1 \end{cases} \quad [1]$$

Assuming given physical and interface conditions³, from a user point of view, each process has to solve a local tridiagonal linear system which reads:

$$\frac{1}{h^2} \begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & 0 & -1 & 2 & -1 \\ 0 & \dots & \dots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_2 \\ u_3 \\ \vdots \\ \vdots \\ u_{N_x-2} \\ u_{N_x-1} \end{pmatrix} = \begin{pmatrix} \frac{u_1}{h^2} + s_2 \\ s_3 \\ \vdots \\ \vdots \\ s_{N_x-2} \\ \frac{u_{N_x}}{h^2} + s_{N_x-1} \end{pmatrix} \quad [2]$$

2. N_{sd} also represents the size of the MPI communicator.

3. The interface conditions are transparent to the user since they are computed inside PMD.

Only the three diagonals L , D and U of the local operator matrix should be stored. Figures 6 and 7 show the Laplace main program containing all the subroutines calling sequences.

The `SetLocalOperator` subroutine sets the local operator matrix as it is defined in the left hand side of Equation [2]. The `SetLocalSourceTerm` subroutine sets the previous source term s and the analytic solution u_a . `UserLocalSolver` routine solves the previous local linear system and returns the local solution u assuming a given operator matrix, source term s and physical boundary conditions. For each calling process $k = 0, \dots, N_{sd} - 1$, `UserFirstDerivative` subroutine returns the normal derivatives $\left(\frac{\partial u}{\partial x}\right)_{i=1}$ and $\left(\frac{\partial u}{\partial x}\right)_{i=N_x}$ at the interface nodes $i = 1$ and $i = N_x$. In fact, these quantities are computed from the Equation [1] which can be rewritten as:

$$\frac{1}{h^2} ([u_i - u_{i-1}] - [u_{i+1} - u_i]) = s_i \quad [3]$$

Taking into account the contribution of the subdomain neighbours in order to keep second order accuracy on the computed first derivative, Equation [3], expressed at the interface nodes $i = 1$ and $i = N_x$, leads to:

$$\begin{aligned} \left(\frac{\partial u}{\partial x}\right)_{i=1} &= \frac{1}{h^2} (u_{i=2} - u_{i=1}) + \frac{s_{i=1}}{2} \\ \left(\frac{\partial u}{\partial x}\right)_{i=N_x} &= \frac{1}{h^2} (u_{i=N_x} - u_{i=N_x-1}) - \frac{s_{i=N_x}}{2} \end{aligned}$$

```

PROGRAM Laplace
USE PMD
IMPLICIT NONE
!... Local constants and variables
INCLUDE 'mpif.h'
TYPE(PMD_Comm_1D)           :: Comm
TYPE(PMD_R8_GELU)          :: Factor
TYPE(PMD_R8_Schur)         :: Schur
TYPE(PMD_R8_Tr_Operator)   :: Operator
INTEGER, PARAMETER         :: Nx=1001
REAL(KIND=R8),DIMENSION(Nx) :: L, D, Up, s, residu
REAL(KIND=R8),DIMENSION(0:Nx+1) :: u, ua
REAL(KIND=R8)              :: Norm
INTEGER                    :: Rank, Nsd, Info
!... External routines
EXTERNAL MPLINIT,MPLFINALIZE,UserLocalSolver,UserFirstDerivative
    
```

FIG. 6 –. A Fortran program using PMD to solve the 1D Laplace problem (to be continued...).

As one can notice from Figures 6 and 7, some parameters of the PMD generic subroutines are declared as derived data types. These data types are mainly important in the way that they direct the generic subroutine calls to the appropriate PMD internal routines according to the floating point data precision (R4 or R8), to the domain dimension (PMD_Comm_1D or PMD_Comm_2D), to the domain decomposition technique (PMD_R[4 or 8]_Schur), to the matrix factorization algorithm (PMD_R[4 or 8]_GELU) if a direct method has to be applied or to the preconditioner type (PMD_R[4 or 8]_PCG_Jacobi) if an iterative method (here a PCG solver) has to be performed.

`UserLocalSolver` and `UserFirstDerivative` are user defined subroutines. Nevertheless, they must respect the rules of the Fortran 90 interfaces shown in Figure 8.

```

!... Initiate MPI
CALL MPLINIT(Info)
!... Initiate PMD
CALL PMD_Init_1DD(MPLCOMM_WORLD, Comm)
CALL PMD_Comm_Info_1DD(Comm, Rank, Nsd)
!... Set local operator matrix (here Tridiagonal)
CALL SetLocalOperator(Comm, L, D, Up)
!... Set Operator object
CALL PMD_Operator_1DD(Comm, L, D, Up, Operator)
!... Build Dual-Schur complement
CALL PMD_Schur_1DD(Comm, UserLocalSolver, UserFirstDerivative, Operator, Schur)
!... Parallel LU-Factor the Schur matrix
CALL PMD_Schur_Factor_1DD(Comm, Schur, Factor)
!... Free the Schur Matrix memory allocation
CALL PMD_Schur_Free_1DD(Comm, Schur)
!... Set source term and analytic solution
CALL SetLocalSourceTerm(Comm, s, ua)
!... Physical boundary conditions
u(1) = ua(1)      ! WEST side (x=0)
u(Nx) = ua(Nx)   ! EAST side (x=1)
!... Parallel solve using a General Direct Solver
CALL PMD_Solve_1DD(Comm, UserLocalSolver, UserFirstDerivative, Operator, Factor, s, u)
!... Compute the norm of the residu
residu(:) = ABS(u(1:Nx) - ua(1:Nx))
Norm = .Norm.residu / real(Nx * Nsd)
!... Check results
IF( Rank == 0 ) &
    PRINT *, "Gap between analytic & numerical solutions: ", Norm
!... Leave PMD
CALL PMD_End_1DD(Comm)
!... End MPI
CALL MPLFINALIZE(Info)
END PROGRAM Laplace

```

FIG. 7 –. A Fortran program using PMD to solve the 1D Laplace problem.

```

INTERFACE
SUBROUTINE UserLocalSolver(Comm, M[or L,D,U], s, u)
  TYPE(PMD_Comm_1[or 2]D), INTENT(IN)      :: Comm
  REAL(KIND=R4[or 8]),DIMENSION(:,:),INTENT(IN)  :: M[or L,D,U]
  REAL(KIND=R4[or 8]), DIMENSION(:,:), INTENT(IN)  :: s
  REAL(KIND=R4[or 8]), DIMENSION(:,:), INTENT(OUT) :: u
END SUBROUTINE UserLocalSolver

SUBROUTINE UserFirstDerivative(Comm, u, s, Du_West, Du_East)
  TYPE(PMD_Comm_1[or 2]D), INTENT(IN)      :: Comm
  REAL(KIND=R4[or 8]), DIMENSION(:,:), INTENT(IN)  :: s, u
  REAL(KIND=R4[or 8]),DIMENSION(:), INTENT(OUT)  :: Du_West
  REAL(KIND=R4[or 8]),DIMENSION(:), INTENT(OUT)  :: Du_East
END SUBROUTINE UserFirstDerivative
END INTERFACE

```

FIG. 8 – *UserLocalSolver* and *UserFirstDerivative* subroutine interfaces.

2D Laplace problem can be solved in parallel following the same general steps. In order to extend the source program of Figures 6 and 7 to a 2D case, the user has to introduce a few changes as declaring *Comm* as an object of type *PMD_Comm_2D*, *M* (or *L*, *D*, and *U*), *s* and *u* as 2D arrays and *Du_West* and *Du_East* as 1D arrays. Thanks to the generic Fortran 90 interface of the PMD solver which will internally orient the calls to the appropriate subroutines. The global 2D domain is cut into equal sized slices (Figure 9).

Physical boundary

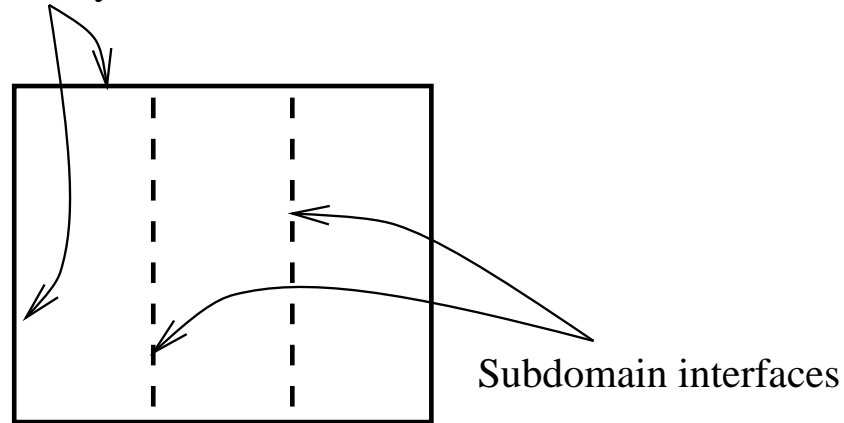


FIG. 9 – *PMD Splitting of a 2D domain.*

The procedure then consists in seeking a local solution *u* in each subdomain which satisfy the following global problem:

$$\begin{cases} -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = s(x,y) = 2\pi^2 \sin(\pi y) \cos(\pi x) & ; (x,y) \in [0,1] \times [0,1] \\ u(0,y) = \sin(\pi y) \\ u(1,y) = -\sin(\pi y) \\ u(x,0) = u(x,1) = 0 \end{cases}$$

In the next section, we will present some performance measurements relative to the below defined 2D Laplace operator case study.

4. Performances

Timing measurements will concern the following three main parts of the PMD solver:

1. Building the Schur matrix with `PMD_Schur_1DD`.
2. LU-Factor the Schur matrix with `PMD_Schur_Factor_1DD`.
3. Solve the final problem with `PMD_Solve_1DD`.

Global performance rates of the application will also be presented in terms of sustained 64 bit floating point operations per second (Mflops/sec) on three parallel machines whose characteristics and compiler options are summarized in Table 1.

Machines	IBM SP2	SGI/CRAY T3E	Fujitsu VPP300
#Processors	127	256	8
Processor	IBM P2SC thin	Dec Alpha EV5	Fujitsu Vector
Processor memory	256 MB	128 MB	2048 MB
Memory data cache	1 level cache 128 KB	2 level caches 8 KB + 96 KB	-
Processor peak performance	500 MFlops/sec	600 MFlops/sec	2200 MFlops/sec
Interconnexion network	Omega Multilevel Switch 90 MB/sec	Bidirectional 3D Torus 600 MB/sec	Bidirectional Crossbar 615 MB/sec
Operating system	AIX 4.2	Unicos/mk 2.0	UXP/V V10L20
Compiler	xl (version 4.1)	f90 (Version 3.0)	frt (version L97121)
Optimization options	-O3	-O3,unroll2, pipeline3	-Of -Wv,-Of

TAB. 1 –. *Hardware and software characteristics.*

PMD modules and Laplace source codes have been compiled and run as is on each machine without any particular hardware dependent tuning. Table 2 shows how the elapsed time is spread among the three main subroutines on each parallel machine. As we can see, the Schur matrix building step is largely time consuming with respect to the parallel factorization and resolution steps on the three machines. This suggests that PMD can be particularly efficient in solving time-dependent problems since the Schur matrix and its factored form can be built before entering the time-loop. Especially noteworthy is the low communication costs on the T3E and on the VPP300.

This case study performs 20 times faster on the VPP300 with respect to the T3E-600 and 39 times faster with respect to the SP2. This suggest that PMD is rather a vector oriented application.

However, profiler analysis tools on the T3E and on the VPP300 show that almost 70% of the elapsed time is actually spent in the user local solver defined in `UserLocalSolver` subroutine which is passed to `PMD_Schur_1DD` and `PMD_Solve_1DD` routines. Consequently, the user has to concentrate his optimization effort (i.e. reducing memory access conflicts, inlining, using high performance scientific libraries, etc.) on that routine only.

	IBM SP2	SGI/CRAY T3E	Fujitsu VPP300
Build Schur matrix (sec.)	2979.6	1340.	63.
Factor Schur matrix (sec.)	74.7	183.7	10.
Solve (sec.)	3.6	1.8	0.1
Total elapsed time (sec.)	3125.	1525.6	74.
Communication time (sec.)	15.5	0.6	0.2
Total MFlops/sec./processor	22.	42.	859.

TAB. 2 – *Elapsed execution time and performance rate to solve the 2D Laplace problem using PMD. Processor (or subdomain) number: $N_{sd} = 4$. Mesh size in each subdomain: $N_x = 801$, $N_y = 601$.*

The scalability of the method can be evaluated assuming a fixed number ($N = N_{sd} \times N_x \times N_y = \text{constant}$) of global mesh points in the domain and measuring the restitution time for different values of the processor number N_{sd} . Consequently, the computational work on each processor reduces as N_{sd} increases (Figure 10) since the Schur matrix is distributed among a greater number of processors. Globally, the application seems to scale better on the SP2 than on the T3E.

Nevertheless, the application runs faster on the T3E than on the SP2 for $N_{sd} < 36$, while beyond 36 processors, it performs a few better on the SP2 (Figure 10).

This could be explained by the fact that, as N_{sd} increases, the computations is done on a smaller set of data in the processor memory. One could suggest that beyond 36 processors, local data in the cache memory are better reused on the POWER2-SC processor of the SP2.

If we compare the execution time obtained when using four vector processors of the VPP300 to the one obtained when using RISC processors of the SP2 or of the T3E (Figure 10), we observe that we need approximately 64 RISC processors to achieve almost an equal execution time with 4 vector processors to perform the same global problem size.

The Schur matrix factorization step is based on the ScaLAPACK subroutine PxGETRF (x=S or x=D). However, the distribution of the matrix obtained at the end of the building step does not fit the ScaLAPACK bloc-distribution need. Therefore, a redistribution of the Schur matrix has to be performed in a 2D BLACS virtual process grid before using PxGETRF. The redistribution and the factorization are applied in the PMD_Schur_Factor_1DD routines. As we observe in (Figure 11), this routine scales globally better on T3E, although it runs quite faster on SP2 for small number of processors ($N_{sd} \leq 16$). The curve shows a strange behaviour for $N_{sd} = 25$ on SP2. This situation is reproducible and occurs at the communication layer. It may be related to the BLACS topology and to the process placements on the SP2 processors.

Figure 11 shows also that 9 RISC processors performs this step almost as fast as 4 vector processors.

Figure 12 shows how the restitution time evolves for growing number of processors to solve the final problem with the PMD_Solve_1DD routine. In this routine, the final interface problem is solved using the ScaLAPACK subroutine PxGETRS. It has to be noted that PMD_Solve_1DD routine is the less consuming time with respect to the Schur matrix building and factorization steps. It scales quite well on the SP2, although it runs faster till 36 processors on the T3E. This routine is fit to give optimal performance to solve time-dependent problems where this routine has to be called probably thousands of time in a timestep loop.

Also, as we can see in Figure 12, we need more than 64 RISC processors to perform this step as fast as 4 vector processors of the VPP300.

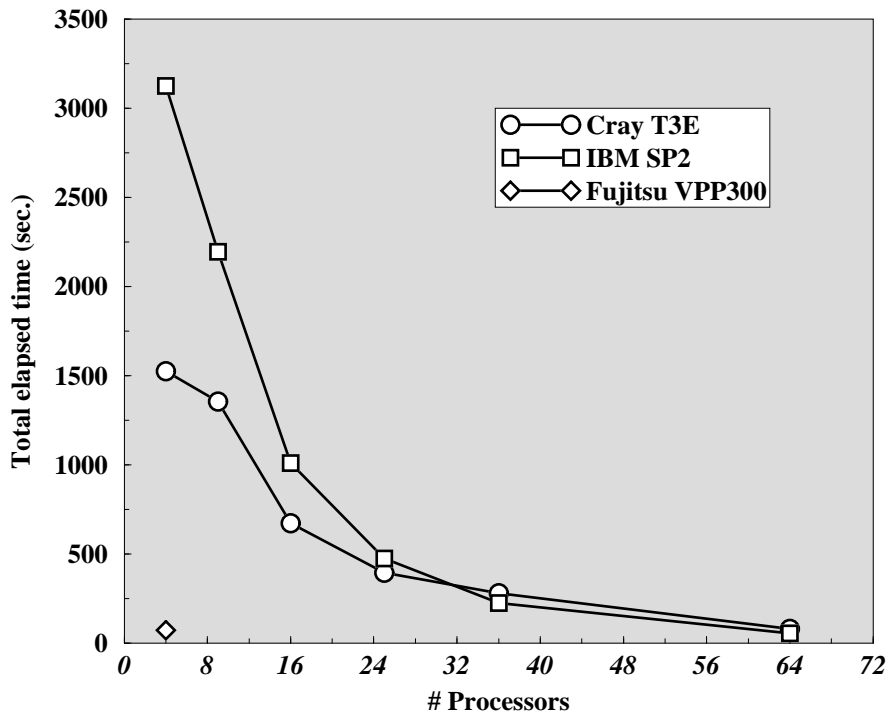


FIG. 10 – Scalability of PMD. Elapsed execution time versus the processor number N_{sd} at a fixed global mesh size ($N \simeq 1,923,801$) of the domain. $N_x = 801$; $N_y = 601$; 267; 153; 99; 69; 37.

At fixed local mesh size $N_x \times N_y$ in each subdomain, the restitution time evolves as a linear function of the number N_{sd} of processors (Figures 13 and 14). It has to be noted however that as the processor number N_{sd} increases, the global mesh size grows as $N = N_{sd} \times N_x \times N_y$. In such situation, as we know, mono-domain solvers usually provide timings which evolve as $N \times f(N)$ to be compared to timings which evolve here as $\alpha \times N$, where the slope α remains constant whatever N .

However, the slopes of the T3E and of the SP2 curves are clearly different. Actually, this mainly reflects not only the processor hardware and software characteristics of the machine, but also the dependency with the local mesh size $N_x \times N_y$. In fact, Figure 13 shows that the restitution time is better on SP2 than on T3E whatever the number of processors. On the contrary Figure 14 shows that it performs better on T3E for a greater local mesh size. This has to be related to a better compiler optimisation on the T3E processeur that leads to a better data cache reuse when local data are large enough.

5. Advantages and restrictions

The aim of PMD is to hide for the user the complexity of developing a domain decomposition method on a parallel distributed memory machine. Its a high level approach which provides a set of high performance Fortran 90 generic routines to solve elliptic linear systems. PMD offers to the user mainly two levels of freedom. The user chooses its discretization method and applies its own local solver. Nevertheless, because of some numerical and software

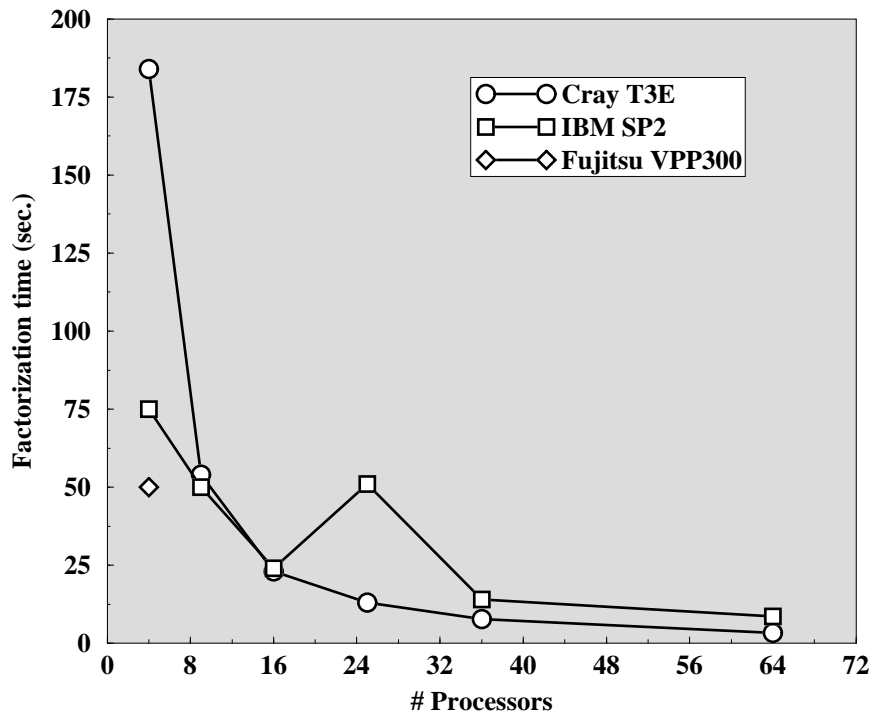


FIG. 11 – Elapsed time versus the processor number to factor the Schur matrix. The global mesh size is fixed to $N \simeq 1,923,801$. $N_x = 801$ in the domain. $N_y = 601; 267; 153; 99; 69; 37$.

restrictions, some classes of problems are solved when using the current PMD version.

1. The computational domain must be simply connected.
2. Along the domain decomposition axis, physical boundary conditions are assumed to be non-periodic.
3. The current version of PMD stands for 1D Domain decomposition only.
4. The interface normal vectors must be parallel to the domain decomposition axis.
5. It is up to the user to compute the interface first normal derivatives.
6. When using direct parallel solvers, which here is based on the ScaLAPACK library, the BLACS process grid must be square. This means that if N_p denotes the process grid dimension, then N_{sd} is such that $N_{sd} = N_p^2$. However this restriction doesn't stand when using the parallel preconditioned conjugate gradient solver.
7. The parallel preconditioned conjugate gradient method, which can be used to solve the final problem, has an inconvenient property which is the number of iterations to convergence increases as the number of subdomains grows. However, it remains better adapted to solve steady state problems since building the Jacobi preconditionner is much faster than LU-factoring the Schur matrix. Actually, the Jacobi preconditioning step performs almost 40 to 150 times faster than the LU-factorization one.

6. Installation and testing guide

PMD assumes MPI, LAPACK, and ScaLAPACK libraries already installed.

The PMD Fortran 90 modules, the example source files and the documentation can be obtained from the following web link:

<http://www.idris.fr/data/publications/PMD/PMD.htm>

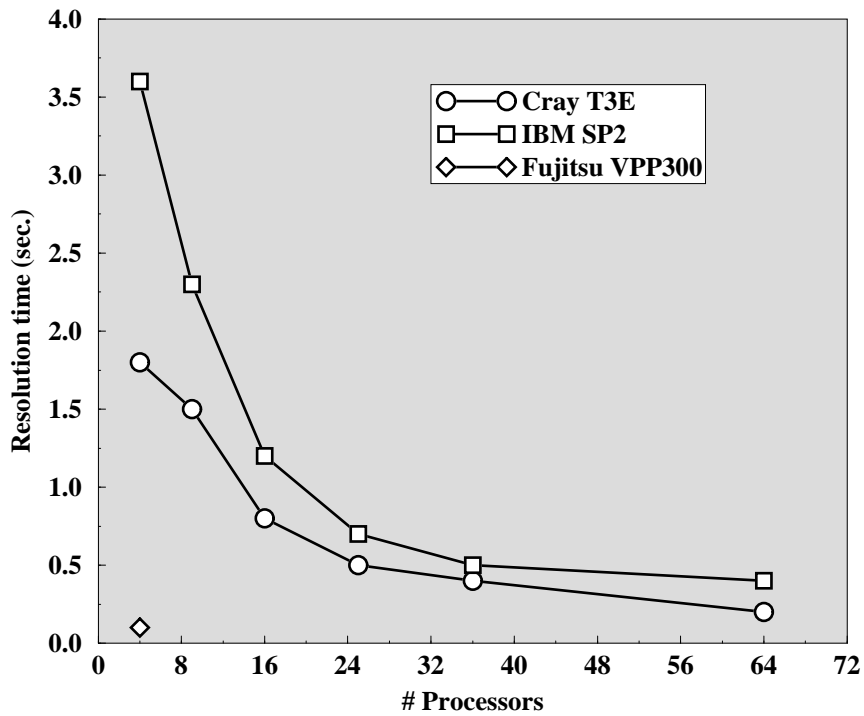


FIG. 12 – Elapsed time versus the processor number to solve the final problem. The global mesh size is fixed to $N \simeq 1,923,801$ in the domain. $N_x = 801$; $N_y = 601; 267; 153; 99; 69; 37$.

- i) Transfert the file `pmd.tar.gz`.
- ii) Decompress and detar the file using the following commands:

```
zcat pmd.tar.gz | tar -xvf -
```
- iii) The PMD package contains the following files and directories:

```

PMD/
  README
  CHANGES
  Makefile
  Manuals/
  Modules/
    Makefile
    PMD_Types.f
    PMD_Tools.f
    PMD_Norm.f
    PMD_Solver_1DD.F
    PMD.f
  MakeDep/
    Make.inc
    Make.SP2.inc
    Make.T3E.inc
    Make.VPP300.inc
    Make.SX4.inc

```

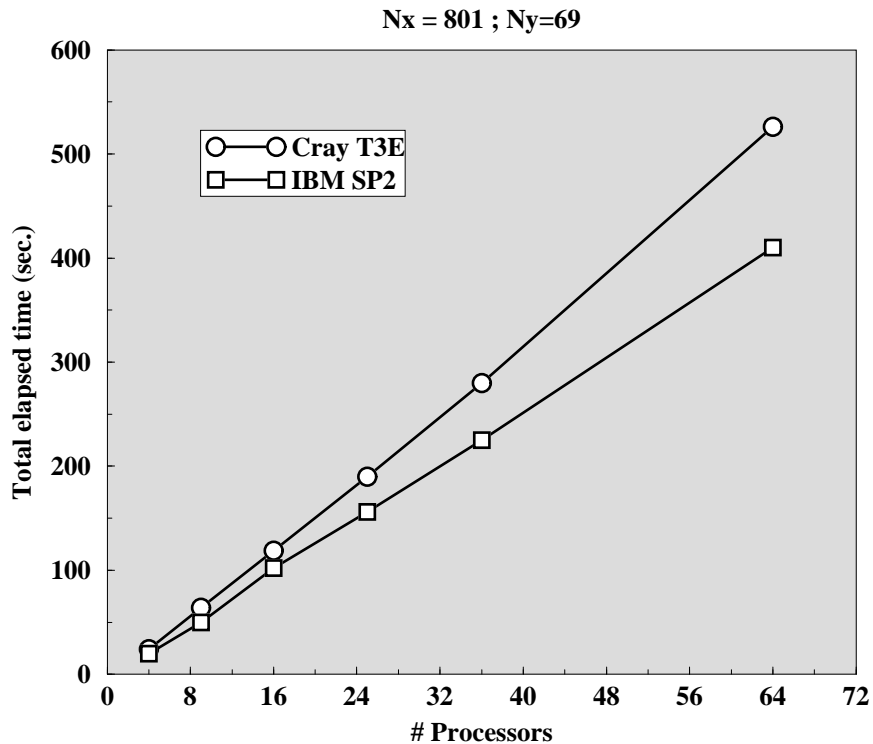


FIG. 13 – Elapsed execution time to solve the 2D Laplace problem using PMD versus the processor number. The local mesh size is fixed to ($N_x = 801; N_y = 69$) in each subdomain.

Examples/

**Helmholtz_1D_1DD/
Helmholtz_2D_1DD/
Laplace_1D_1DD/
Laplace_2D_1DD/
PseudoLaplace_1D_1DD/
PseudoLaplace_2D_1DD/
NS_2D_1DD**

Please note that all Fortran files are in free format. Therefore, options like `-qfree=f90` on the IBM SP2 or `-ffree` on the Cray T3E or `-Free -Am -X9` on the Fujitsu VPP300, may be needed. On some systems, however, the MPI header file `mpif.h` might be in fixed format. Note also that on CRAY machines, double precision references does not exist. Consequently, the `-F -DCRAY` directives have to be used at a preprocessing stage to select the 64 bit CRAY single precision references only.

- iv) Change to the `MakeDep` directory and configure the `Make.inc` file to point to the right compiler and libraries on your machine. Note that on some systems, the compiler has to know about the location of the Fortran 90 compiled module files to compile the sources and to generate the executable file. To this end, the compiler option can be `-I[modulepath]` on the IBM SP2 and the Fujitsu VPP300 or `-p[modulepath]` on the Cray T3E.
- v) Change back to the `PMD` directory and execute the `make` command to compile all the source files of the `Modules` subdirectory.

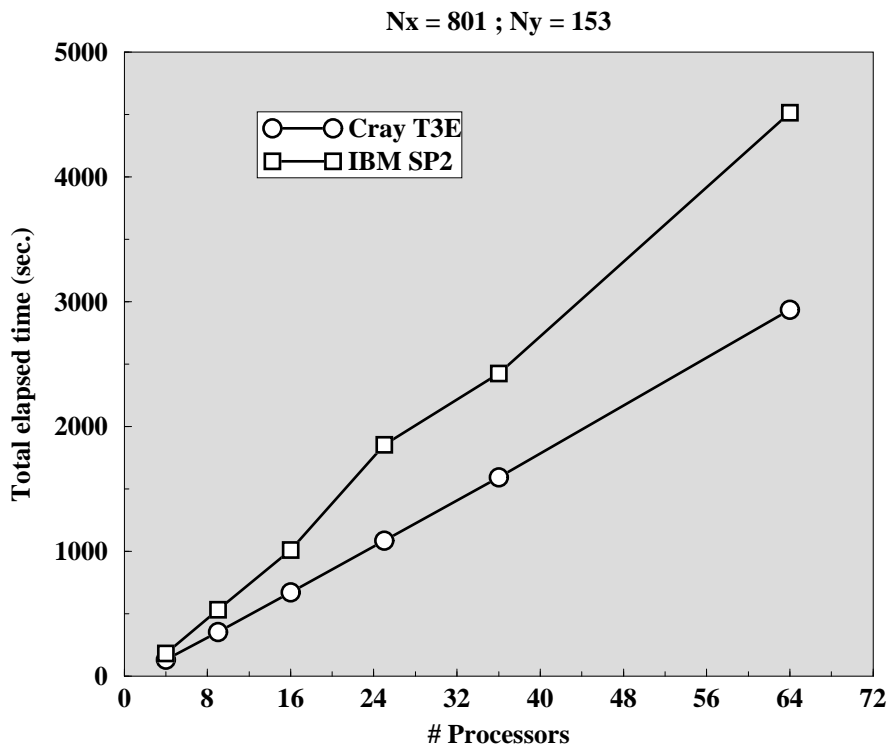


FIG. 14 –. Elapsed execution time to solve the 2D Laplace problem using PMD versus the processor number. The local mesh size is fixed to $(N_x = 801; N_y = 153)$ in each subdomain.

- vi) Run the “make examples” command to compile and to load all the examples of the Examples subdirectory.

7. Conclusion

In this study, we presented an object oriented Fortran 90 module called PMD which defines a set of high performance generic routines built on top of MPI, LAPACK and ScaLAPACK libraries to solve in parallel elliptic linear second order definite positive systems. These routines implement the dual-Schur complement method together with an influence matrix technique to overcome the subdomain interface problem. PMD can easily be used as a blackbox to solve such systems. Also it can be extended straightforwardly to use higher degree domain decomposition, others iterative and direct parallel solvers and to take into account other local operator matrix shape. Timing measurements on parallel distributed memory machines like the IBM SP2, the CRAY T3E-600 and the Fujitsu VPP300 have shown good performance rate on parallel vector processors and good scalability on parallel RISC processors. Further work may implement 2D and 3D Domain decomposition applied on 2D and 3D domains with possibly including others parallel solvers.

Acknowledgments

This project was jointly supported by IDRIS and LIMSI. The authors would like to thank

IDRIS where a large part of this work has been performed and CNUSC⁶ for kindly helping us to use their IBM SP2 machine.

8. Références

- [AND 95] E. ANDERSON, Z. BAI, C. BISHOF, J. DEMMEL, J. DONGARRA, LAPACK users' Guide, Second Edition. SIAM, Philadelphia, PA, 1995.
- [BAR 97] R. BARRETT, M. BERRY, T.F. CHAN, J. DEMMEL, J. DONATO, J. DONGARRA, V. EIJKHOUT, R. POZO, C. ROMINE, H. VAN DER VORST, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, PA, 1994.
- [ABD 98] A. BEN ABDALLAH, Méthode de projection pour la simulation des grandes structures turbulentes sur calculateurs parallèles. Thèse de Doctorat de mécanique. Université Paris VI. Octobre 1998.
- [MAM 97] A. BEN MAMOUN, Une méthode de projection parallèle pour la résolution des équations de Navier-Stokes - Applications à l'approximation spectrale des écoulements inter-disques. Thèse de Doctorat de Mathématiques Appliquées. Université Paris VI. Décembre 1997.
- [BLA 97] L. S. BLACKFORD, J. CHOI, A. CLEARY, E. D'AZEVEDO, J. DEMMEL, I. DHILLON, J. DONGARRA, S. HAMMARLING, G. HENRY, A. PETITET, K. STANLEY, D. WALKER, R. C. WHALEY, ScaLAPACK users' Guide. SIAM, 3600 University City Science Center, Philadelphia, PA 19104-2688. May 1997.
- [CHER 97] J. CHERGUI, Performance ratio analysis between CRAY T3E and T3D machines (in French). *Calculateurs Parallèles*, Hermes publisher, 75004 Paris, France. Vol. 9. No 4. December 1997.
- [DAU 91] O. DAUBE, An influence matrix technique for the resolution of the 2D Navier-Stokes equations in velocity-vorticity form. Notes et documents LIMSI: 91-11. CNRS/LIMSI, B.P. 133 - 91403 Orsay cedex, France. Octobre 1991.
- [FAR 93] C. FARHAT, F. X. ROUX, Implicit Parallel Processing in Structural Mechanics. College of Engineering University of Colorado Campus Box 429. Boulder, Colorado 80309. November 1993.
- [GUY 98] M. GUYON, G. MADEC, F.-X. ROUX, CH. HERBAUT, M. IMBARD, P. FRAUNIE, Domain Decomposition Method as a Nutshell for Massively Parallel Ocean Modeling with the OPA Model. Research Report. IPSL, Université Pierre et Marie Curie, B 102 - T15-E5, 4 place Jussieu, 75252 Paris cedex 05, France. December 1998.
- [JAC 97] R. JACQUES, P. LEQUÉRE, O. DAUBE, Parallel Numerical Simulation of Turbulent Flows in Closed Rotor-Stator Cavities. The paper will be published in the notes on Numerical Fluid Mechanics, Vieweg Publisher.
- [QUA 90] A. QUARTERONI, Domain Decomposition Method for the Numerical Solution of Partial Differential Equations. UMSI 90/246. University of Minnesota Supercomputer Institute. Research report. December 1990.
- [SNI 96] M. SNIR, S. OTTO, S. HUSS-LEDERMAN, D. WALKER, J. DONGARRA, *MPI: The Complete Reference*. The MIT Press Cambridge, Massachusetts London, England. 1996.
- [TAL 91] P. LE TALLEC, Y. H. DE ROECK, M. VIDRASCU, Domain decomposition methods for large linearly elliptic three-dimensional problems. *Journal of Computational and Applied Mathematics* 34 (1991) 93-117. Elsevier Science Publishers B.V. (North-Holland).

⁶. Centre National Universitaire Sud de Calcul, BP7229, 34184 Montpellier, cedex 4, France. <<http://www.cnusc.fr/>>.