# Developing the TELEMAC system for HECToR (phase 2b & beyond)

Zhi Shang

HECToR Distributed CSE Support Technical Meeting, Manchester 4th October 2011

## Outline of the Talk

Introduction to the TELEMAC System and to TELEMAC-2D

Code Developments

Data Reordering Strategy

Results

Conclusions

Future Work

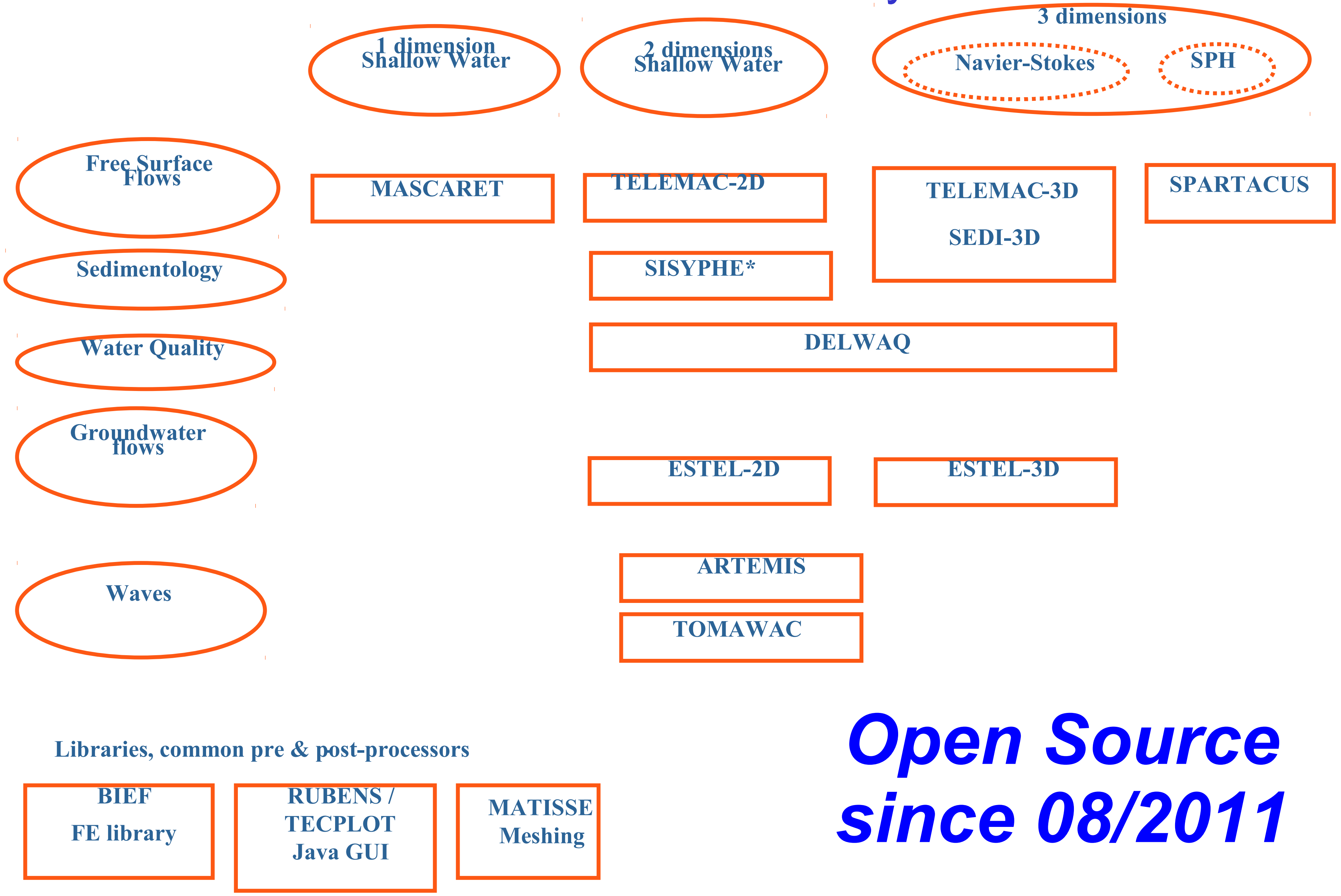
## Introduction to the TELEMAC System

The TELEMAC system uses the Finite Element Method to simulate free-surface flows, mainly environmental, dealing with current distribution, sediment transport, wave propagation, flooding and/or water quality treatment.

The system is written in Fortran 90 and contains a total of 250,000 lines of code, split into several libraries (160,000 lines for TELEMAC-2D to run).

The TELEMAC system is open source (GPL/LGPL licensing).

# Introduction to TELEMAC System



***Open Source since 08/2011***

## Introduction to TELEMAC-2D

The Shallow Water equations (derived from depth-integrating the Navier-Stokes equations) are implemented in TELEMAC-2D to simulate free-surface flows. Those equations apply when the horizontal length scale is much larger than the vertical length scale. Conservation of mass implies that the vertical velocity of the fluid is small.

$$\frac{\partial h}{\partial t} + u \cdot \nabla(h) + h \nabla \cdot (u) = S_h$$

$$\frac{\partial u}{\partial t} + u \cdot \nabla(u) = -g \frac{\partial z}{\partial x} + S_x + \frac{1}{h} \nabla \cdot \left( h v_t \nabla(u) \right)$$

$$\frac{\partial v}{\partial t} + u \nabla(v) = -g \frac{\partial z}{\partial y} + S_y + \frac{1}{h} \nabla \cdot \left( h v_t \nabla(v) \right)$$

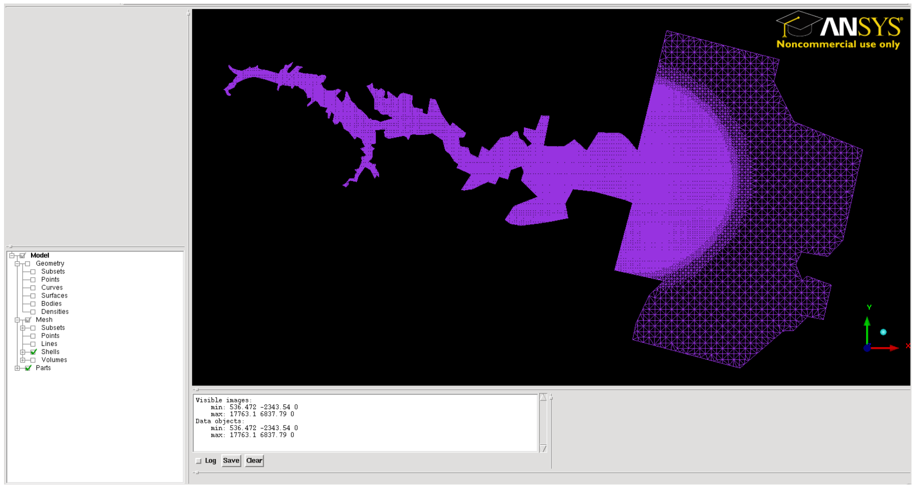
# Introduction to TELEMAC-2D

## How to run TELEMAC-2D?

Generation of a grid of triangular elements with a mesh generator, and interpolation of the bathymetry at each node of the grid.

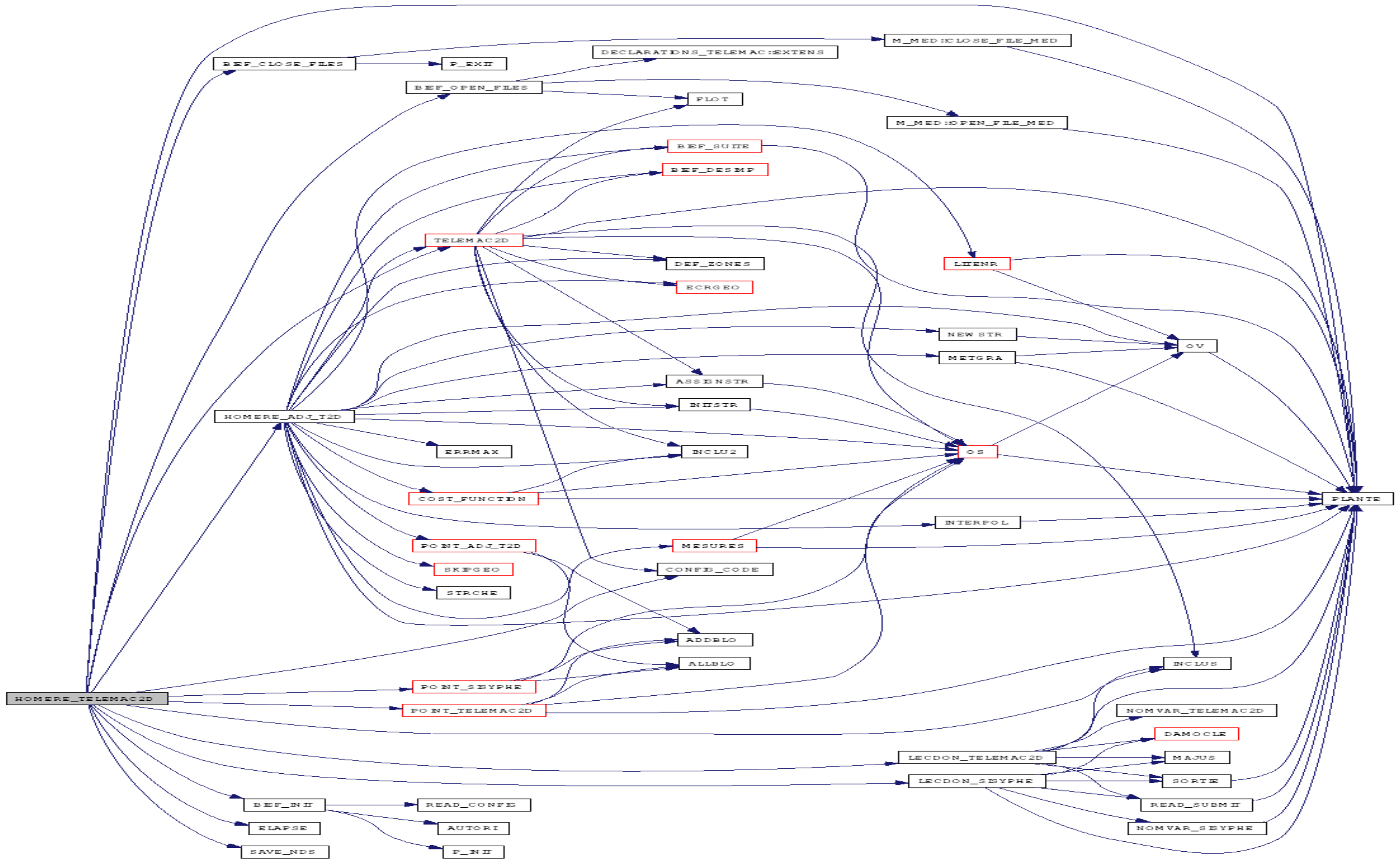
If run parallel, a pre-processing stage is required to distribute information over to each MPI task.

TELEMAC-2D computes the Shallow Water equations (ex: Malpasset Dam).



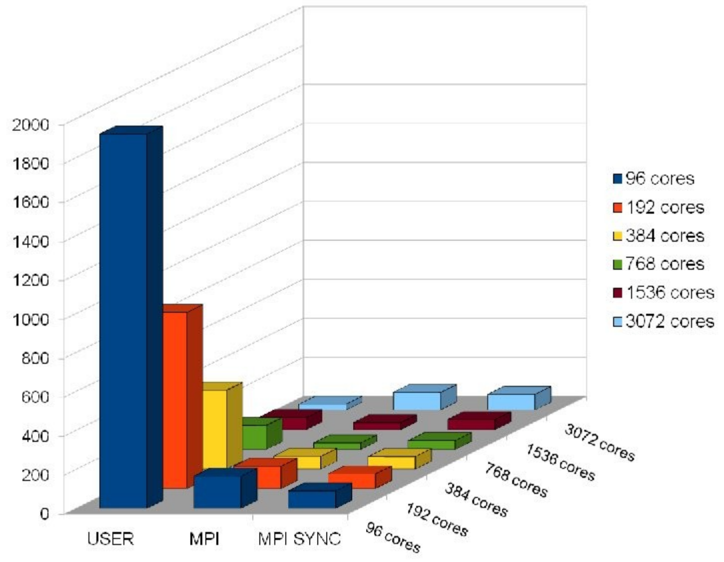
```
Visible images:
    min: 536.472 -2343.54 0
    max: 17763.1 6837.79 0
Data objects:
    min: 536.472 -2343.54 0
    max: 17763.1 6837.79 0
```

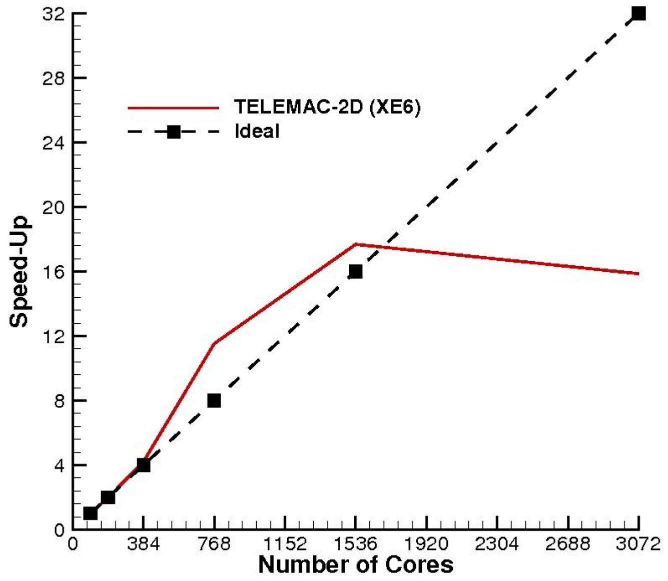
# Introduction to TELEMAC-2D



# Introduction to TELEMAC-2D

MPI is already implemented in the TELEMAC system, to account for large spatial or temporal environmental situations.





Example of a 12M-element grid using TELEMAC-2D (Pure MPI)

## Introduction to TELEMAC-2D

Hybrid programming approach using both MPI and openMP is proposed to improve the scalability of TELEMAC-2D on massively parallel architectures.

Getting good performance out of openMP implementation in the TELEMAC system is not an easy task, because indirect addressing is used in a lot of loops of the various codes, due to the unstructured nature of the Finite Element Method.

## Code Developments

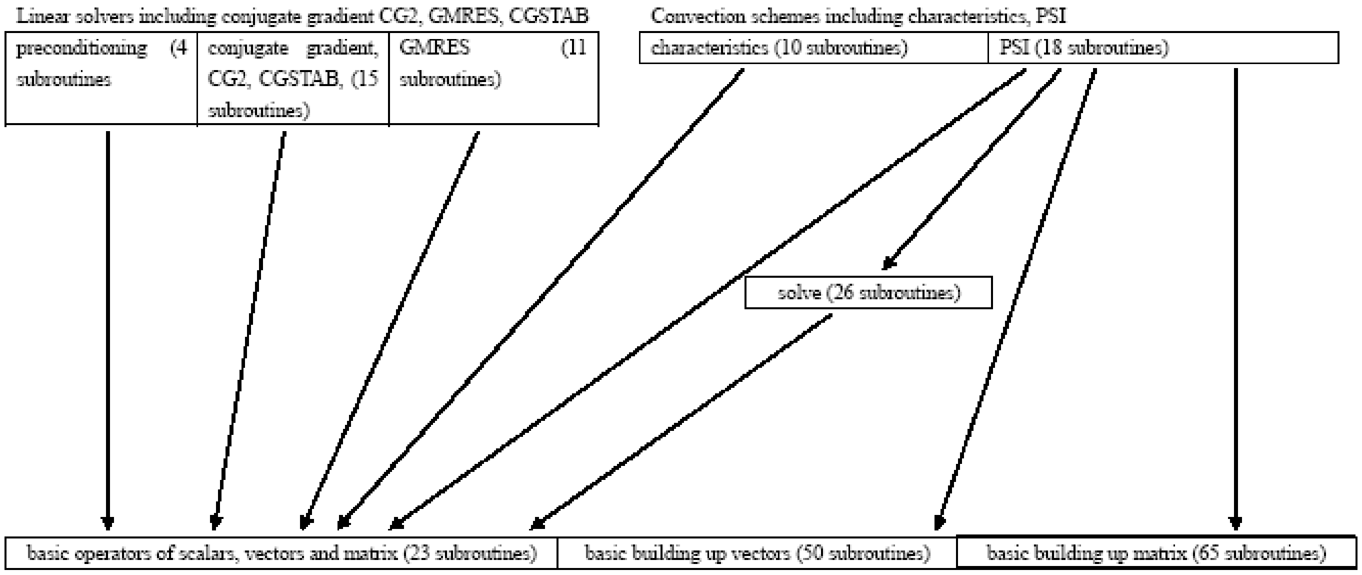
An analysis of TELEMAC-2D performance shows that, depending on the time-step used, on the configuration (different for dam-break and for flooding studies, for instance), most of the time is spent either by the linear solver to solve the wave propagation equation (giving the solution of the Shallow Water equations), or by the advection scheme.

The lower level of the system (BIEF Finite Element library) contains subroutines performing elementary operations on scalars, vectors and matrices.

This work focuses on the implementation of openMP in the Conjugate Gradient (CG) solver and in the Positive Streamwise Invariant (PSI) scheme.

## Code Developments

A total of about 200 subroutines have been updated
by openMP directives, but not all the combinations have been tested.



Linear solvers including conjugate gradient CG2, GMRES, CGSTAB
preconditioning (4 subroutines
conjugate gradient, CG2, CGSTAB, (15 subroutines)
GMRES (11 subroutines)
Convection schemes including characteristics, PSI
characteristics (10 subroutines)
PSI (18 subroutines)
solve (26 subroutines)
basic operators of scalars, vectors and matrix (23 subroutines)
basic building up vectors (50 subroutines)
basic building up matrix (65 subroutines)

## Code Developments

openMP programming seems easier than MPI programming, but still requires a lot of care when indirect addressing occurs, for instance.

The directives identifying the openMP parallelization are used to identify the parallel region.

Within the openMP parallel region, directives for loops and synchronizations are used.

However if there is a data dependent argument inside a loop, data dependence needs to be broken, when possible.

An example of such a situation is presented in the following.

## Code Developments

The original code segment including the data dependence from one of the subroutines used for communication (BIEF library) reads:

```
K = 1
IF (IAN.EQ.3) THEN
   DO J=1,NPLAN
         DO I=1,IKA
              II=NH_COM(I,IL)
              IF (ABS(BUF_RECV(K    ,IL)).GT.ABS(V1(II,J)))     V1(II,J)=BUF_RECV(K    ,IL)
              IF (ABS(BUF_RECV(K+1,IL)).GT.ABS(V2(II,J)))     V2(II,J)=BUF_RECV(K+1,IL)
              IF (ABS(BUF_RECV(K+2,IL)).GT.ABS(V3(II,J)))     V3(II,J)=BUF_RECV(K+2,IL)
              K=K+3
         ENDDO
    ENDDO
ENDIF
```

In the nested loop, the value of K is always updated and the newest value of K will be used by the array BUF_RECV(:,:). Therefore data dependence should be avoided if openMP is used to parallelise the nested loop.

## Code Developments

In the nested loop, an additional variable KK is introduced to compute the correct value of K, compatible for all the openMP threads.

The statement K=KK+(J-1)*IKA*3 between the first nested loop and the second nested loop can ensure that the value of K in every individual thread of openMP is independent.

## Code Developments

The transformed code segment with the openMP directives is shown below.

```
K = 1
!$ KK = K
IF (IAN.EQ.3) THEN
!$OMP PARALLEL DEFAULT(shared), PRIVATE(I,J,II,K)
!$OMP DO
  DO J=1,NPLAN
!$ K=KK+(J-1)*IKA*3
     DO I=1,IKA
        II=NH_COM(I,IL)          !!!! STILL INDIRECT ADDRESSING !!!!
        IF (ABS(BUF_RECV(K   ,IL)).GT.ABS(V1(II,J)))    V1(II,J)=BUF_RECV(K   ,IL)
        IF (ABS(BUF_RECV(K+1,IL)).GT.ABS(V2(II,J)))    V2(II,J)=BUF_RECV(K+1,IL)
        IF (ABS(BUF_RECV(K+2,IL)).GT.ABS(V3(II,J)))    V3(II,J)=BUF_RECV(K+2,IL)
        K=K+3
     ENDDO
  ENDDO
!$OMP END DO NOWAIT
!$OMP END PARALLEL
ENDIF
```

## Data Reordering Strategy

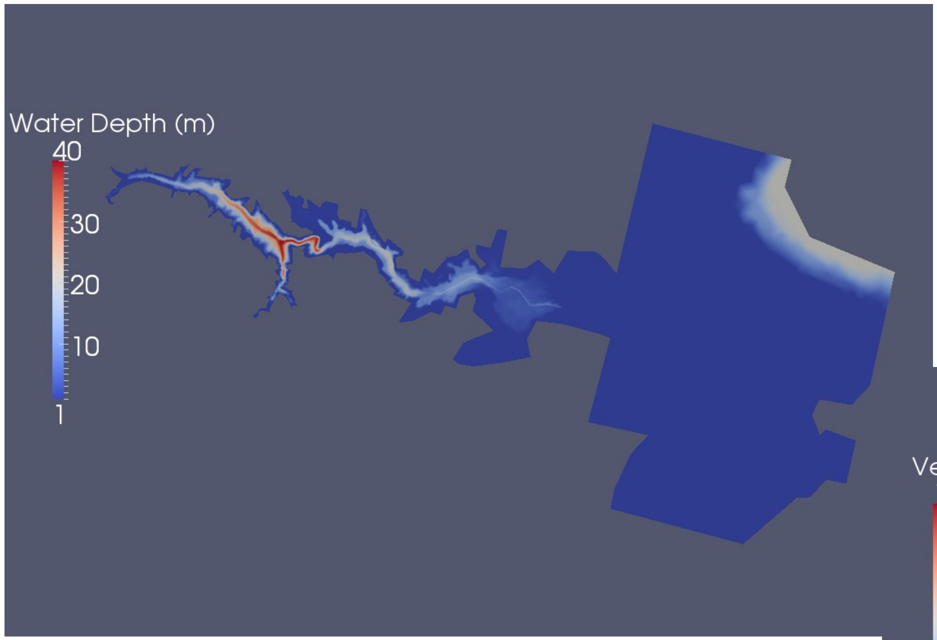
The benchmark case corresponds to the Malpasset Dam Break simulation, which is one of the examples provided by the distribution.

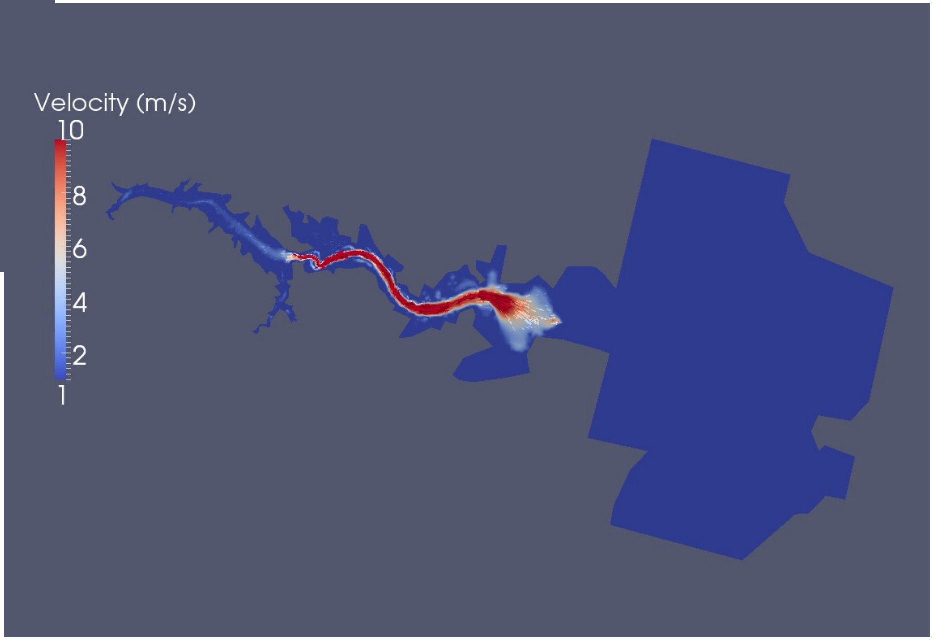
A fine mesh is built up by refining several times a coarse mesh. The fine mesh (denoted 1.6M) contains 1,664,000 elements for 836,321 nodes.

As the fine 1.6M grid is obtained by global refinement, it is difficult to ensure that the assembled matrix of the finite element solver will have a low bandwidth.

A data reordering strategy has to be used to fix this issue. The reverse Cuthill-McKee (RCM) method is employed to re-order the elements to ensure a lower bandwidth.

## Data Reordering Strategy





## Data Reordering Strategy

The Cuthill-Mckee (CM) algorithm and the reverse Cuthill-Mckee (RCM) algorithm are used to reorder a sparse matrix in order to lower its bandwidth.

Through this reordering, the storage of the matrix can be optimized and contiguous memory can be accessed.

It is expected that openMP implementation will benefit from reordering the node indeces of the grid by using RCM.

## Data Reordering Strategy

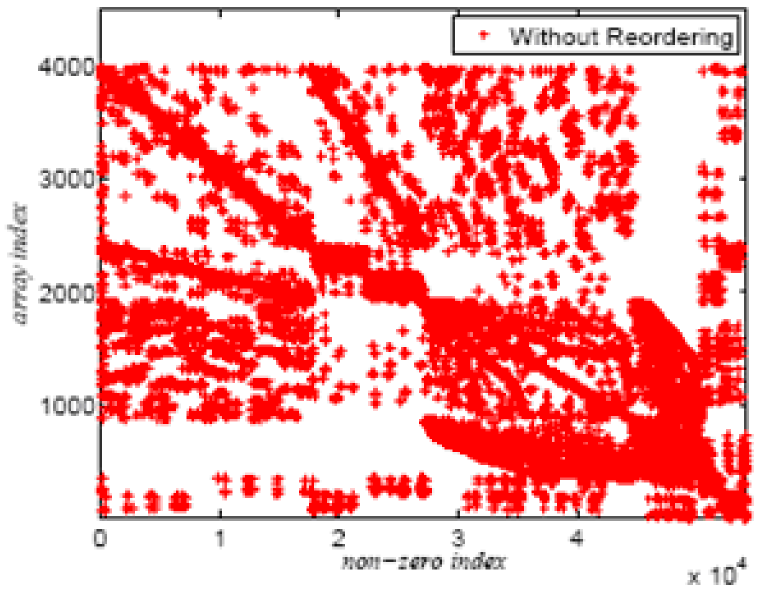
RCM algorithm reads:

1. start with any node from the grid and store its index in an array;
2. find the neighbouring nodes of that node;
3. sort the neighbouring node indeces in ascending order;
4. choose the first of the neighbouring nodes of the array. If it has not been previously inserted, add its index up, as well as it neighbour node indeces, which are not already in the array;
5. repeat procedure 2 - 4 until all the node indeces are stored into the array;
6. reverse the order of the elements of the array.

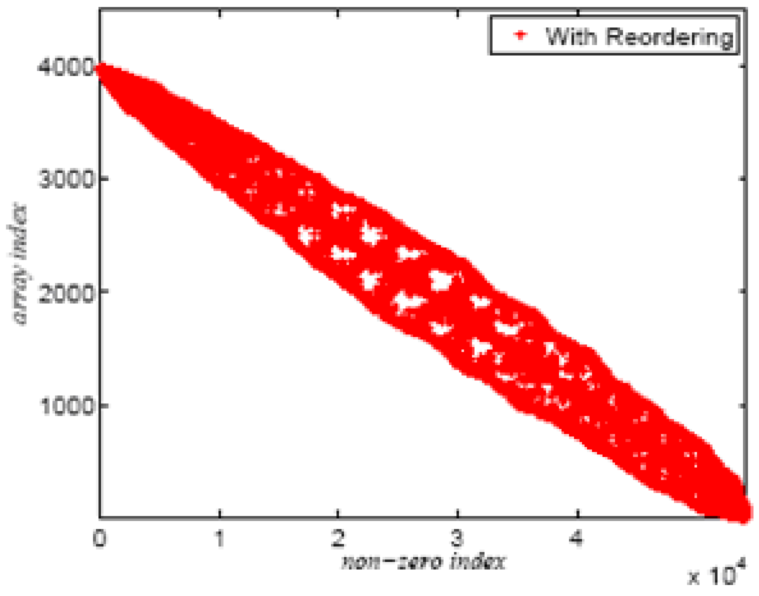
The first 5 steps of this procedure correspond to the Cuthill-McKee algorithm; when all the steps are performed, the method is called Reverse Cuthill-McKee.

## Data Reordering Strategy

Tests show that reordering can help speeding-up TELEMAC-2D by about 20%. Consequently, the mesh reordered by RCM is used for the next parallel tests and the time for running a serial simulation on that mesh is used as the reference for comparison with the time spent by the parallel simulations.

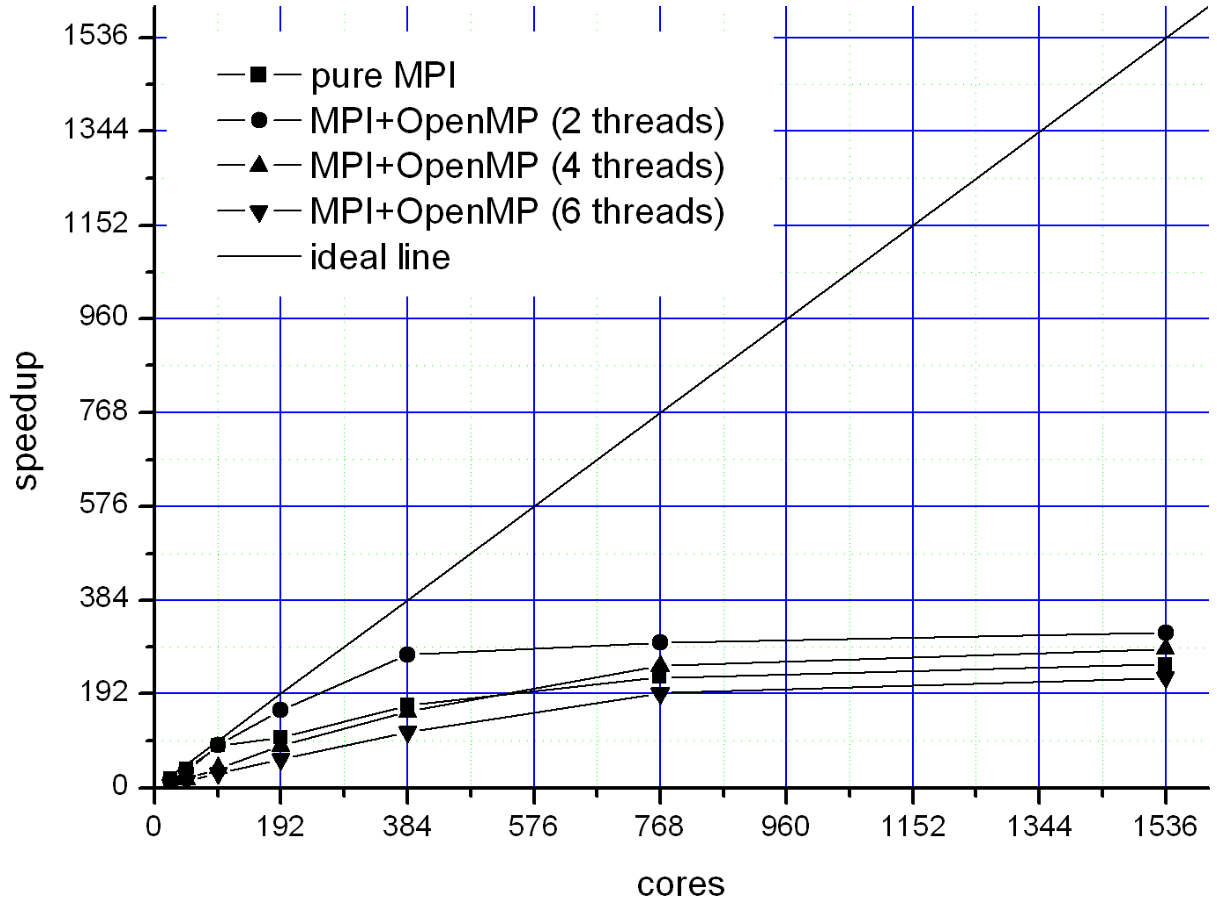


(a) Without data reordering



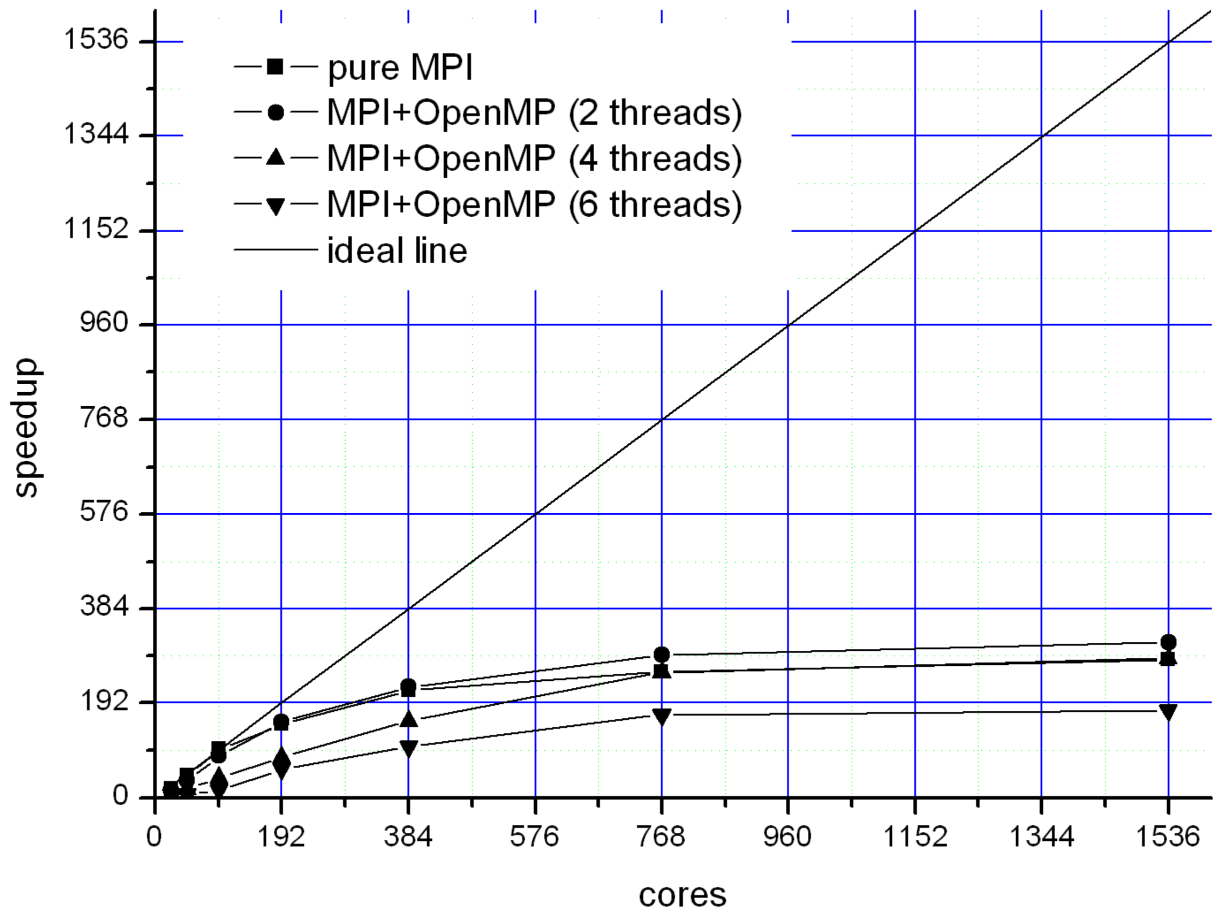
(b) With data reordering

## Results



CG solver

## Results



PSI scheme

## Conclusions

The use of a hybrid MPI/openMP programming approach can be used to overcome some of the drawbacks of pure MPI and pure openMP.

First benchmarking tests have shown that a simulation using MPI/openMP (384 cores and 2 threads) in the Conjugate Gradient solver is 80% faster than a simulation using pure MPI with the same number of cores.

In addition, the reverse Cuthill-McKee (RCM) method has been employed to re-order the mesh elements of the assembled finite-element matrix to reduce its bandwidth.

## Future Work

Generalise the openMP strategy to other subroutines, for instance to the other linear solvers present in the code, to the other advection schemes, with a special focus on the method of characteristics.

Test larger grids, for instance above 20 million elements.

Extend openMP to other parts of TELEMAC-2D, before adding it up to the other codes of the TELEMAC system.

## THANK YOU VERY MUCH