

ADAPT solver coupled with efficient linear solver

Imad KISSAMI

2014

Plan

- 1 Introduction
- 2 ADAPT
- 3 Solving a linear system
- 4 MUMPS
- 5 Some experiments with the Streamer code
- 6 Other work related to the parallelization
- 7 Conclusion

Plan

- 1 Introduction
 - Context of our work
- 2 ADAPT
- 3 Solving a linear system
- 4 MUMPS
- 5 Some experiments with the Streamer code
- 6 Other work related to the parallelization
- 7 Conclusion

- Gathering competencies from numerical analysis, computer science and physics to develop an integrated platform(ADAPT).
- Many physical phenomena are treated(combustion, plasma discharge, wave propagation, etc.).
- Using an object oriented code(C++).
- Simulation codes with a dynamic mesh adaptation strategy.
- Coupling between finite volume and finite element method.

Plan

1 Introduction

2 ADAPT

- State of the art
- Description
- Important parts of the code
- Work Environment
- Profiling

3 Solving a linear system

4 MUMPS

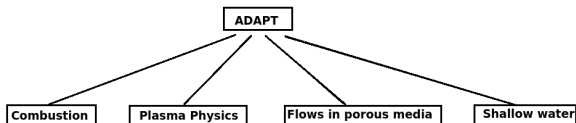
5 Some experiments with the Streamer code

6 Other work related to the parallelization

7 Conclusion

Contribution of ADAPT over existing solvers:

Library of scientific computing	FV method	EF method	Mesh adaptation	Mesh type
FreeFem++	No	Yes	No	Structured
Mélinea	No	Yes	No	Structured
ADAPT	Yes	Yes	Yes	Unstructured



Existing solution:

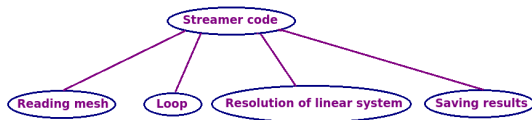
Sequential C++ code for each phenomenon.

Problem:

Execution time(Streamer code make 30 days to give results).

Solution:

Parallelization of code



The numerical simulation of the propagation of the streamer:

- based on a mathematical model.
- convection-diffusion equation, coupled with the Poisson equation.

We worked on the "magi cluster" which is located in the University Paris 13, this cluster consists of two partitions:

- B500 Partition : dedicated to distributed computing, it contains 12 nodes; on each node there are two Xeon X5670, with 6 cores each and 24 GB of RAM per node.
- Partition SMP for calculating shared memory; 1 single node, 4 Xeon E7-4850 to 10 cores each 512 GB of RAM total.

100.00	0.12	1	main
42.68	2.58	4 592	void loop<>(Cell_field<>&, Cell_field<>&, Cell_fi
34.99	0.00	4 615	UMFPACKSolver::solve()
34.99	0.00	4 615	umfpack_di_solve
34.99	12.51	4 615	umfdi_solve
24.21	4.82	4 666	void compute_el_field<>(Grid&, Cell_field<>&, G
13.34	5.92	490 761 056	std::vector<>::operator=(std::vector<> const&)
10.82	10.82	13 845	umfdi_usolve
10.31	10.31	13 845	umfdi_lsolve
8.48	8.48	117 559 816	weight_parameters(Node const&, Grid&, double
8.30	1.98	4 592	void compute_b0<>(Grid&, std::vector<>&, Ghos

The resolution of the linear system takes a large part of time in the execution of the Streamer code.

Plan

- 1 Introduction
- 2 ADAPT
- 3 Solving a linear system**
 - Methods for solving a linear system
 - Direct method versus iterative one
 - Direct method using LU factorization
- 4 MUMPS
- 5 Some experiments with the Streamer code
- 6 Other work related to the parallelization
- 7 Conclusion

We will consider the assembled system $Ax=b$.

When

A is large sparse matrix.

x is vector of unknowns.

b is right hand side.

The methods for solving linear systems can be divided into :

- Direct Methods
- Iterative Methods

method	advantage	disadvantage
Direct	precision less execution time	large memory
Iterative	small memory	less precise than direct method

In the present work we are interested in the direct method using the LU factorization.

A is unsymmetric matrix and factorized as : $A = LU$.

L is a lower triangular matrix.

U is an upper triangular matrix.

Suppose that :

$$A = \begin{pmatrix} 7 & 0 & 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 2 & 0 & 0 & 5 & 0 & 0 & 11 \\ 4 & 0 & 3 & 0 & 0 & 0 & 9 & 16 \\ 0 & 0 & 0 & 4 & 0 & 7 & 0 & 0 \\ 0 & -3 & 0 & 0 & 10 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 3 & -4 & 0 & 0 \\ 11 & 0 & 0 & 2 & 3 & 0 & 9 & 0 \\ 0 & 6 & 0 & 15 & 0 & 3 & 0 & 15 \end{pmatrix}$$

LU Decomposition is given by

$$L = \begin{pmatrix} 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 2.42857 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & -3 & 0 & 0 & 17.5 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 3 & -3.05882 & 0 & 0 \\ 11 & 0 & -1.57143 & 2 & 3 & -7.38235 & 32.9538 & 0 \\ 0 & 6 & 0 & 15 & -15 & -23.25 & 58.4967 & 35.8718 \end{pmatrix}$$

$$U = \begin{pmatrix} 1 & 0 & 0.142857 & 0 & 0 & 0.285714 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2.5 & 0 & 0 & 5.5 \\ 0 & 0 & 1 & 0 & 0 & -0.470588 & 3.70588 & 6.58824 \\ 0 & 0 & 0 & 1 & 0 & 1.75 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0.0571429 & 0.942857 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2.47912 & 5.23242 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1.4005 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Forward-backward substitution : $Ly=b$ then $Ux=y$.

but the big concern is:

-Time cost.

-Memory cost.

Hence the use of parallel solvers as MUMPS, pastix etc...

Introduction
ADAPT
Solving a linear system
MUMPS
Some experiments with the Streamer code
Other work related to the parallelization
Conclusion

Description
Symbolic factorization and elimination tree
Multifrontal approach
Pivot order
Importance of the shape of the tree
Mapping of assembly tree
Parameters of MUMPS

Plan

1 Introduction

2 ADAPT

3 Solving a linear system

4 MUMPS

- Description
- Symbolic factorization and elimination tree
- Multifrontal approach
- Pivot order
- Importance of the shape of the tree
- Mapping of assembly tree
- Parameters of MUMPS

5 Some experiments with the

ADAPT solver coupled with efficient linear solver

The MUMPS (MULTifrontal Massively Parallel Solver) is one of parallel distributed solver using a multifrontal method for the solution of sparse linear equations.

The solution of sparse system with the MUMPS solver is achieved in three phases:

- Analysis
- Factorization
- Solution

The analysis phase:

- Define pivot order.
- Symbolic factorization (this step aims at computing the non-zeros structure of the factors before the actual numeric factorization).
- Mapping of the assembly tree

Symbolic information is transferred from the host to the other processors. Using this information, the processors estimate the memory necessary for factorization and solution.

The factorization phase:

- Elimination tree.
- Frontal matrices.
- Multifrontal approach.

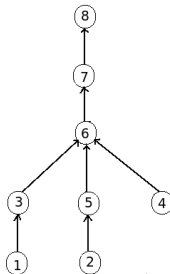
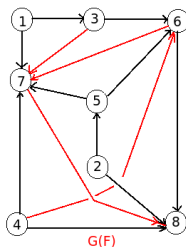
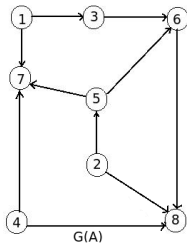
The solution phase:

- The right-hand side is broadcast from the host to the other processors. These processors compute the solution using the (distributed) factors compute during step 2, and the solution is assembled on the host.

$$A = \begin{pmatrix} x & 0 & x & 0 & 0 & x & 0 & 0 \\ 0 & x & 0 & 0 & x & 0 & 0 & x \\ x & 0 & x & 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & x & 0 & x & 0 & 0 \\ 0 & x & 0 & 0 & x & 0 & x & 0 \\ 0 & 0 & x & 0 & x & x & 0 & 0 \\ x & 0 & 0 & x & x & 0 & x & 0 \\ 0 & x & 0 & x & 0 & x & 0 & x \end{pmatrix}$$

$$F = \begin{pmatrix} x & 0 & x & 0 & 0 & x & 0 & 0 \\ 0 & x & 0 & 0 & x & 0 & 0 & x \\ x & 0 & x & 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & x & 0 & x & 0 & 0 \\ 0 & x & 0 & 0 & x & 0 & x & 0 \\ 0 & 0 & x & F & x & x & F & 0 \\ x & 0 & F & x & x & F & x & F \\ 0 & x & 0 & x & 0 & x & F & x \end{pmatrix}$$

x : nonzero element
 F: Fill-in



$$A = \begin{pmatrix} 7 & 0 & 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 2 & 0 & 0 & 5 & 0 & 0 & 11 \\ 4 & 0 & 3 & 0 & 0 & 0 & 9 & 16 \\ 0 & 0 & 0 & 4 & 0 & 7 & 0 & 0 \\ 0 & -3 & 0 & 0 & 10 & 0 & 1 & 0 \\ 0 & 0 & 2 & \textcircled{F} & 3 & -4 & \textcircled{F} & 0 \\ 11 & 0 & \textcircled{F} & 2 & 3 & \textcircled{F} & 9 & \textcircled{F} \\ 0 & 6 & 0 & 15 & 0 & 3 & \textcircled{F} & 15 \end{pmatrix}$$

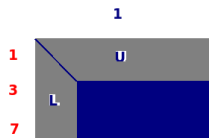
Pivot 1

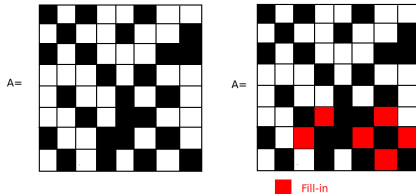
$$\begin{pmatrix} \textcircled{7} & 0 & 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & \textcircled{2} & 0 & 0 & 5 & 0 & 0 & 11 \\ 0 & 2 & \textcircled{17/7} & 0 & 0 & -8/7 & 9 & 16 \\ 0 & 0 & 0 & \textcircled{4} & 0 & 7 & 0 & 0 \\ 0 & -3 & 0 & 0 & \textcircled{10} & 0 & 1 & 0 \\ 0 & 0 & 2 & \textcircled{F} & 3 & \textcircled{-4} & \textcircled{F} & 0 \\ 0 & 0 & -11/7 & 2 & 3 & -22/7 & \textcircled{9} & \textcircled{F} \\ 0 & 6 & 0 & 15 & 0 & 3 & \textcircled{F} & \textcircled{15} \end{pmatrix}$$

L3->L3-(4/7)L1

L7->L7-(11/7)L1

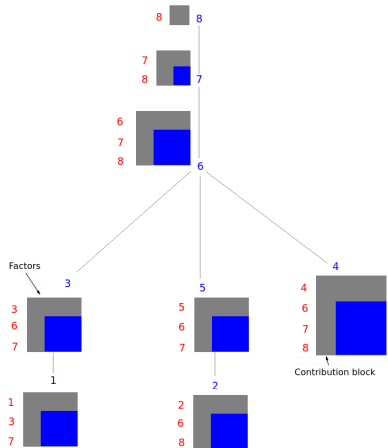
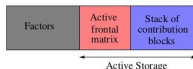
modified rows





Storage divided into two parts :

- Factors;
- Active Storage.

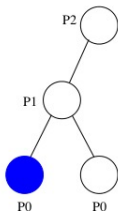
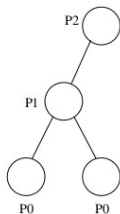


The parallel algorithm is characterized by :

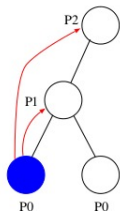
- Computational graph dependency
- Communication graph

Three classical approaches

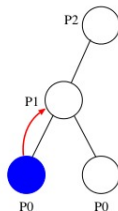
- 1 "Fan-in"
- 2 "Fan-out"
- 3 "Multifrontal"



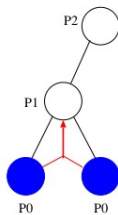
Fan-in.



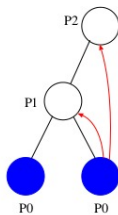
Fan-out.



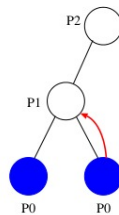
Multifrontal.



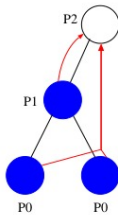
Fan-in.



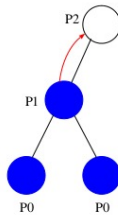
Fan-out.



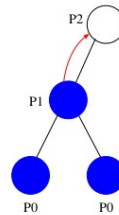
Multifrontal.






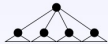

Fan-in.



Fan-out.



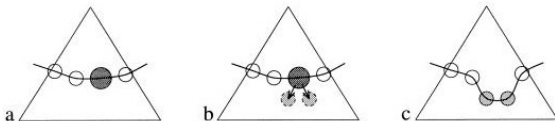
Multifrontal.

Reordering technique	Shape of the tree	observations
AMD		<ul style="list-style-type: none"> * Deep well-balanced * Large frontal matrices on top
AMF		<ul style="list-style-type: none"> * Very deep unbalanced * Small frontal matrices
PORD		<ul style="list-style-type: none"> * deep unbalanced * Small frontal matrices
SCOTCH		<ul style="list-style-type: none"> * Very wide well-balanced * Large frontal matrices
METIS		<ul style="list-style-type: none"> * Wide well-balanced * Smaller frontal matrices (than SCOTCH)

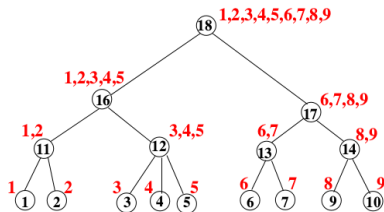
Suppose that each node in the tree corresponds to a task that :

- consumes temporary data from the children,
 - produces temporary data, that is passed to the parent node.
- 1 The width of the tree is large
 - Good parallelism
 - Many temporary blocks to store
 - Large memory usage
 - 2 The deep of the tree is large
 - Less parallelism
 - Smaller memory usage

A mapping of the assembly tree to the processors is performed statically as part of the analysis phase.
The main objectives of this phase are to control the communication costs, and to balance the memory used and the computation done by each processor.



Example :



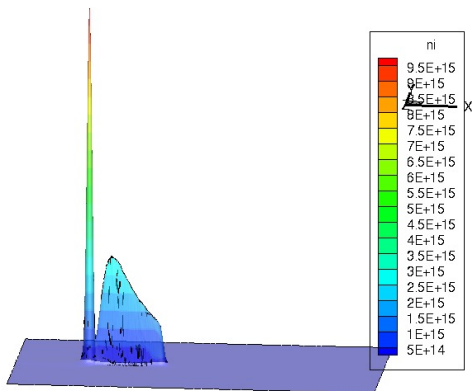
- The main parameters which will be used are JOB , SYM and the ICNTL array.
- We remind that the solution of sparse linear systems is done in three steps : analysis, factorization, and solution.

JOB	steps to be done
SYM	symmetry of the matrix
PAR	does the host processor take part to the computations ?
N	order of the matrix
NZ	number of non-zero entries
A, IRN, JCN	sparse matrix description
RHS	right-hand side
ICNTL	control parameters
INFO/INFOG, RINFO/RINFOG	informations returned by the solver

Plan

- 1 Introduction
- 2 ADAPT
- 3 Solving a linear system
- 4 MUMPS
- 5 Some experiments with the Streamer code
 - Comparing results of two direct solvers
- 6 Other work related to the parallelization
- 7 Conclusion

	cells
reference mesh	43674
mesh at $t = 5.25 \times 10^{-8}$	102092



Similarity:

- The results for MUMPS and UMFPACK are equivalent.
- The number of adaptation is the same.

The important difference is in the execution time :

solver	time (s)
UMFPACK	1.737×10^5
MUMPS	1.436×10^5

17.34% is the gain.

Plan

- 1 Introduction
- 2 ADAPT
- 3 Solving a linear system
- 4 MUMPS
- 5 Some experiments with the Streamer code
- 6 Other work related to the parallelization
 - Profiling using Valgrind
 - Test of MUMPS solver
 - Petsc
 - Parallel GMRES
 - Test of Petsc solver

UMFPACK

1 main			
100.00	0.03		
79.80	1.58	8 992	void loop<>(Cell_field<>&, Cell_field<>&, Cell_field<...
42.10	5.80	26 976	void calculate_rez<>(Cell_field<> const&, Face_field<...
26.31	12.63	26 976	void deriv_xy<>(Cell_field<> const&, Ghost_field<> c...
18.30	2.41	26 976	void calculate_rez_dissipative<>(Cell_field<> const&,...
13.64	5.16	26 976	void dissipation<>(Cell_field<> const&, Ghost_field<...
11.06	0.00	9 037	UMFPACKSolver::solve()

MUMPS

1 main			
100.00	0.03		
89.98	2.07	8 992	void loop<>(Cell_field<>&, Cell_field<>&, Cell_field<...
48.31	4.98	26 976	void calculate_rez<>(Cell_field<> const&, Face_field<...
31.28	13.81	26 976	void deriv_xy<>(Cell_field<> const&, Ghost_field<> co...
18.31	3.11	26 976	void calculate_rez_dissipative<>(Cell_field<> const&, ...
13.04	3.72	26 976	void dissipation<>(Cell_field<> const&, Ghost_field<> ...
12.93	1.60	9 088	void compute_el_field<>(Grid&, Cell_field<>&, Ghost_...
10.85	10.85	13 062 786 728	Array<>::operator[](unsigned long) const
9.78	9.40	763 157 377	<cycle 1>
9.50	2.29	930 901 279	std::vector<>::operator=(std::vector<> const&)
9.29	0.32	381 136 320	operator new(unsigned long)
5.51	5.51	4 519 102 512	operator+(Streamer const&, Streamer const&)
5.47	5.47	4 295 024 113	operator+=(Streamer&, Streamer const&)
5.33	5.33	4 377 853 344	operator-(Streamer const&, Streamer const&)
5.33	0.00	9 217	dmumps_c

Solving the linear system decreased by 6 %.

Matrix extracted from the code in a given iteration:

size of matrix	70932
number of nonzero	935050

We choose AMF to define pivot order.

We use a dynamic mapping of the assembly tree.

Process number	ratio
1	1
2	0.57
4	0.47
6	0.42
7	0.40

ratio = Exec time parallel code / Exec time Seq code

- The Portable, Extensible Toolkit for Scientific Computation (PETSc) is a package to develop large scale scientific application codes in Fortran, C and C++.
- Supports parallel computations by employing Message Passing Interface (MPI) to communicate with multi-processor.

In the next example we will look at the krylov subspace method(GMRES).

GMRES is an iterative method used for solutions of large sparse nonsymmetric linear systems based on the Arnoldi process for computing the eigenvalues of a matrix. GMRES minimize norm of residual vector $\|r^k\| = \|b - Ax^k\|$.

Guess x_0 then approximate solution x_n .

Compute $r_0 = b - Ax_0$.

Apply Arnoldi process (It constructs the basis and orthogonalizes it at the same time; as soon as a new vector is added, it is orthogonalized relative to the previous vectors and normalized).

Matrix extracted from the code in a given iteration:

size of matrix	70932
number of nonzero	935050

We choose GMRES to define iterative method.

Process number	ratio	iteration number
1	1	898
2	0.75	1271
4	0.63	1937
6	0.41	1403
7	0.47	1834

ratio = Exec time parallel code / Exec time Seq code

Introduction

ADAPT

Solving a linear system

MUMPS

Some experiments with the Streamer code

Other work related to the parallelization

Conclusion

Plan

- 1 Introduction
- 2 ADAPT
- 3 Solving a linear system
- 4 MUMPS
- 5 Some experiments with the Streamer code
- 6 Other work related to the parallelization
- 7 Conclusion

Contributions:

- Parallelize of a significant part of the ADAPT code.
- Same performance as the sequential solver like UMFPACK.
- Experiments are promising.

Future work:

- Introduce more parallel code (replace basic sequential chunks by parallel ones).
- Revise more in depth the code to introduce parallelism at a larger grain.