Efficiently solving large sparse linear systems on a distributed and heterogeneous grid by using the multisplitting-direct method

S. Contassot-Vivier, R. Couturier, C. Denis and F. Jézéquel

Université de Franche-Comté - LIFC
University Pierre et Marie Curie-Paris 6
LIP6 - Deparment of Scientific Computing

7 September 2006

Outline

- Background
- Direct-multisplitting method
- Experiments on GRID'5000
- Load balancing
- Conclusion
- Perspectives

Background

- Solving linear systems on grid may be slow down by :
  - Frequent synchronizations
  - Heterogeneity of machines and networks
  - Variations of bandwidth and latencies

Designing an efficient solver on a grid

- Objectives of the GREMLINS project :
  - Designing a simple algorithm
  - Using a coarse-grained algorithm

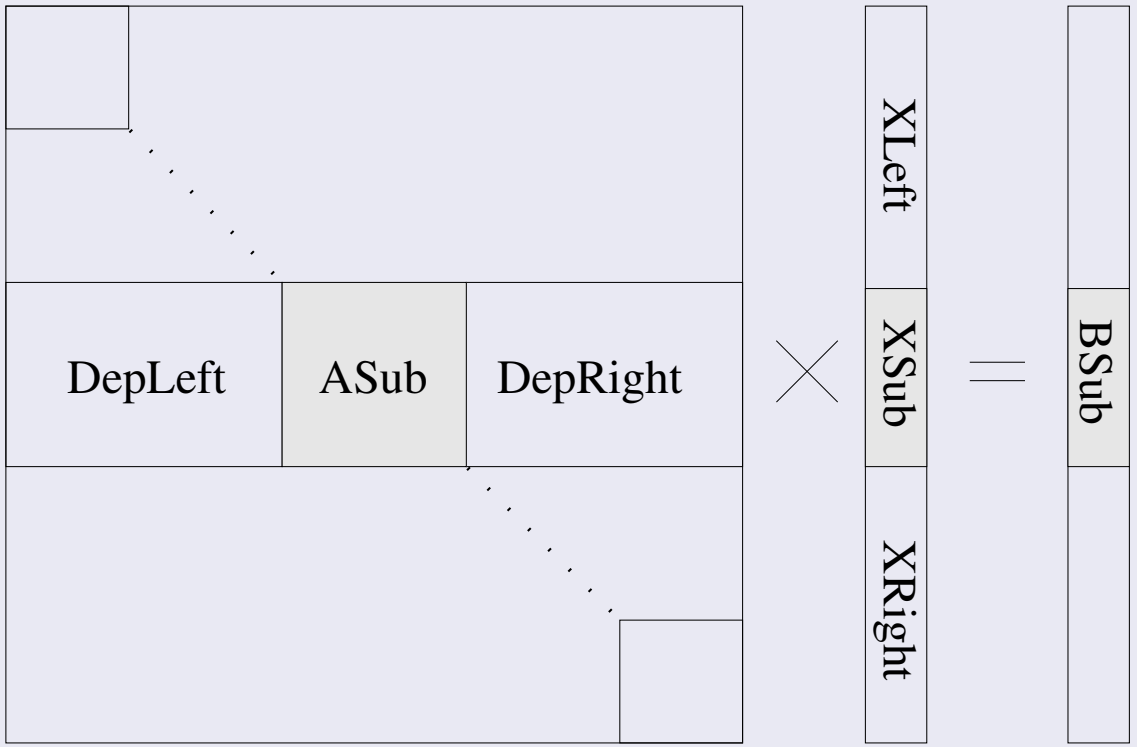
    $\Rightarrow$ the multisplitting method
  - Avoiding synchronizations

    $\Rightarrow$ with asynchronous algorithm

Idea of the Multisplitting method

- Splitting the system into rectangle matrices and iterating on :
  - Solve each subsystem independently
  - Exchange dependencies between systems

Decomposition of the system



- At each iteration a processor solves :

  $ASub * XSub = BSub - DepLeft * XLeft - DepRight * XRight$

Core of the algorithm

`repeat`
Compute right-hand side into Bloc
XSub = Solve(ASub,BLoc)
Send dependencies to all processors who need it
Receive dependencies from all processors which it depends on
Convergence detection
`until` *Global convergence is achieved*

- Iterations may be desynchronized, in this case some receptions may be ignored

Characteristics of the method

- Each submatrix is solved using a sequential algorithm
- If a direct method is used, the factorization is achieved out only once
- Accuracy of an iterative method may vary dynamically according to the residual
- Different solvers may be used (even simultaneously)

CRAC library

- Developped by Stéphane Domas in Belfort
- Multithreaded library dedicated to asynchronous computation
- Same performance as MPI using a synchronous algorithm

Experimentations on GRID'5000

- Experimentations with 120 and 190 machines with MUMPS without load balancing
  - Use of generated matrices
- Experimentations of load balancing with SuperLu
  - Use of matrices from the sparse matrix florida collection

Experimentations 1/2

120 machines : 40 rennes, 40 orsay, 25 nancy and 15 lille

Size of the matrix	Number of diagonals	Synchronous		Asynchronous	
		exec. time (s)	nb. iter.	exec. time (s)	nb. iter.
1,000,000	23	10.055	72	4.550	[303-475]
2,000,000	23	14.989	69	5.394	[195-229]
4,000,000	23	19.332	68	12.312	[204-268]
6,000,000	23	24.194	69	13.348	[146-176]
8,000,000	23	27.871	68	18.188	[142-143]
10,000,000	23	28.102	67	24.220	[136-144]

Experimentations 2/2

190 machines : 30 rennes, 30 sophia, 70 orsay, 30 lyon and 30 lille

Size of the matrices	Number of diagonals	Synchronous		Asynchronous	
		exec. time (s)	nb. iter.	exec. time (s)	nb. iter.
10,000,000	13	16.415	30	10.984	[138-159]
10,000,000 (superlu)	13	19.830	30	14.778	[85-109]
20,000,000	13	15.917	22	15.489	[74-79]

- If the ratio $\frac{computation\ time}{communication\ time}$ is large then the synchronous version is rather preferred
- The asynchronous version is more robust in contexts where network parameters are fluctuating
- The synchronous version is preferred when the communication graph is not dense

Load balancing (LB) : motivation and objective

- The partial factorization computing of submatrices issued from classic matrix partition is not balanced
- Our objective : Designing an automatic static load balancing algorithm of the multisplitting method to be used on a grid computing environment

Principle of the load balancing method

- Input parameters
  - An initial partition $\mathcal{P}_i$ issued from an external program (here provided by the multilevel graph partitioning program METIS)
  - A model returning an estimated computing time for a given amount of operations
- Algorithm
  1. Compute the estimated partial factorization computing times for each diagonal bloc
  2. Repeat
     1. Select the diagonal block A^{max} having the maximum estimated computing time
     2. Transfer n_t rows from A^{max} to its neighbor diagonal bloc -> new partition $\mathcal{P}$
     3. Reorder in parallel each diagonal bloc with the given reordering method
     4. Compute the estimated partial factorization computing times for each diagonal bloc
     5. Save the partition $\mathcal{P}$ -> $\mathcal{P}_c$ if the result is better
  3. Until the automatic termination is reached
  4. Return $\mathcal{P}_c$

Experimentations

The global computing time T_{glob} (including the LB computing time), the maximum partial factorization time $MaxT_{fac}$ and the number of iterations of the LB method

Matrix	# proc.	without LB		with LB		
		T_{glob}	$MaxT_{fac}$	T_{glob}	$MaxT_{fac}$	*iter*
all nodes from lille						
cage12	8	671	651	102	58	32
cage12	16	75	67	33	15	31
cage13	16	2304	2254	929	628	107
cage13	32	728	698	198	110	62
$\frac{1}{2}$ nodes from lille and $\frac{1}{2}$ from rennes						
cage12	8	672	650	110	73	31
cage12	16	82	71	39	18	31
cage13	16	2402	2253	1021	674	96
cage13	32	1057	1007	261	135	24
cage13	64	134	74	86	38	26

Analysis and future works of LB with multisplitting

- The LB method permits to decrease the global computing time in a grid computing environment
- The number of iterations is dynamically chosen during the run of the LB method
- The mean of relative errors between the estimated computing time and the measured computing time is less than 20 %.
- Experiments in progress show that the LB method is successfully applied to the MUMPS software
- Testing the LB method on large generated matrices and on many more distributed processors

Conclusion

- Our algorithms are suitable for solving large linear systems on grids
- According to the context, synchronous or asynchronous versions allow to obtain good performances
- Load balancing with superlu drastically reduces execution times (in some cases)

Perspectives

- Use more processors (CRAC is currently limited to mono processor)
- Implement more solvers with our algorithms (essentially iterative ones)
- Generalize the approach of LB with direct solver
- Compare the behavior of different solvers
- Study how to use preconditionners to widen the spectrum of usable matrices