

# The solver-G project

M. Garbey and H.Ltaief \*

(\* ) Dept. of Computer Science - Univ. of  
Houston,

Thanks NSF & DOE

Thanks B.Hadri, F.Pacull, C.Picard, and  
J.Baranger, R.Keller, F.Oudin, M.Resch,  
W.Shyy.

# **Plan for a grid computing solver**

PART 1: Motivation and Definitions

PART 2: Fault Tolerant Algorithm

PART 3: Solution Verification

# Part 1: Grid Computing: The Challenge!

- the resources of the grid are shared with no centralized control.
- the grid is a highly complex system with a very large number of distributed hardware and software components.
- un-predictable reliability of resources.
- unreliability of computing nodes.
- communication latency between some pairs of nodes might be high, variable, and unpredictable.
- heterogeneity of computers with variable performances.
- we may have (many) more computing nodes than with ordinary parallel computers.
- Remark: some of these items may apply to ... ASCII machines.

# Part 1: Goal

Fast and Robust solvers for

$$\begin{aligned} -\operatorname{div}(\rho \nabla p) &= F, \quad x \in \Omega \subset \mathbf{R}^3, \\ \rho &\equiv \rho(x) \in \mathbf{R}^+, \quad p \equiv p(x) \in \mathbf{R}^+, \end{aligned}$$

$$\frac{\partial C}{\partial t} = \nabla \cdot (K \nabla C) + (\vec{a} \cdot \nabla) C + F(t, x, C),$$

$$C \equiv C(x, t) \in \mathbf{R}^m, \quad K \equiv K(x, t) \in \mathbf{R}^{m \times m}, \quad x \in \Omega \subset \mathbf{R}^3, \quad t > 0,$$

and

$$\frac{\partial^2 C}{\partial t^2} = \nabla \cdot (K \nabla C) + F(t, x, C),$$

$$C \equiv C(x, t) \in \mathbf{R}^m, \quad K \equiv K(x, t) \in \mathbf{R}^{m \times m}, \quad x \in \Omega \subset \mathbf{R}^3, \quad t > 0,$$

with appropriate boundary and initial conditions.

that can work efficiently on the grid.

# Part 1: Goal

## Fast and Robust solvers ?

- of arithmetic complexity of order  $(N \log(N))$  at most, where  $N$  is the number of unknowns (unless problem is highly anisotropic).
- memory scalable on a grid of computers, i.e the elapsed time, as the number of nodes increases linearly with  $N$ , follows the same arithmetic complexity law.
- robust, i.e for a wide variety of data (BC and/or IC), provide a numerical solution with some a posteriori estimate on the error (in the asymptotic range of a priori estimate), no matter large is  $N$ .
- tolerant to high latency and low bandwidth common in network computing.
- with data distribution highly flexible to allow load balance on heterogeneous resources.
- fault tolerant to failures of computing nodes.

**Remark:** Explicit time stepping and Block-Jacobi is not the solution.

**Question:** What about FFT, Multigrid, Krylov methods ?

# Part 1: Goal

## A critical choice to start with

Working with regular data structure such as grid topologically equivalent to Cartesian grids.

- Domain Decomposition and Chimera Methods to deal with complex geometry.
- C.Peskin's Immersed boundary Method or boundary layer fitted subdomain for Fluid-Structure interaction.

Some of the key advantages:

- optimized utilization of the cache.
- easy mesh generation.
- Load balancing.

However it is more difficult to have the numerical approximation accurate.

# Part 1: Specificity of a grid solver ?

- Metacomputing :algorithm should be tolerant to high latency and low bandwidth.
- Load balancing: expect heterogeneity of computers.
- Fault Tolerant: expect unreliability of processing units and network links.
- Multi-algorithm aware: best algorithm is architecture dependent.
- Solution Verification included: expect numerical failures for complex nonlinear problems.
- Efficient Security: use of global internet for industrial applications.

# Part 1: Software Architecture

## Solver-G uses spare processors to companion application's processors

### Inquiries in asynchronous mode

Spare processors,

1. probe the (local) pool of processing units to detect early signs of failure.
2. retrieve data  $I_j^n, v_{j,k}^{\hat{n}}$  from (distanced) processing units with a merry-go-round algorithm.
3. get performance of block solver and interface solver.

### Few remarks

- a priori, the number of spare processors is much less than the number of application processors.
- a priori each hypernode has roughly the same ratio of application processors versus spare processors.

- Our asynchronous local check pointing should scale and be much more efficient than global checkpointing.
- However the numerical difficulty is to reconstruct a uniform initial condition in time to restart the computation from piece wise data  $u_j^n$  where  $n$  is subdomain dependent.
- Communications to spare processors may use a slower network than the network used for message passing in the application.

For example fast network such as Myrinet should be used for the interface solver (=many communications of low dimension arrays) in the application, while Gigabit ethernet might be more cost effective to move complete subdomain's data (=few communications of large dimension arrays) .

- The mapping of data to spare processors should minimize the overhead on the application while probability of unrecoverable failures, i.e a processing units and its dual spare processor both fails, is above a given tolerance level.

# Part 1: Software Architecture

## Solver-G uses spare processors to companion application's processors

### Processing activity of Spare nodes

- Once a processor unit (that is about to fail) is deactivated, a spare processor replaces it. The pool of active processors keep the same number of active units while the number of spare processors decays. If the number of spare units reach a minimum critical level, the partitioning between application processing units and spare processors might be dynamically redefined at the expense of slowing down the application.
- Since one uses asynchronous local check pointing, one has to reconstruct a consistent global solution  $u^n$  from a distributed set of spare data  $u_j^{n(j)}$ , where time step  $n$  is subdomain's dependent. This is a new interesting mathematical problem that is operator/application dependent.

- Proceed with dynamic load balancing: spare processors run an optimization procedure to reallocate subdomains and/or move interfaces between subdomains.
- Apply verification procedures to detect numerical failures such as
  - ◇ detect possible abnormal growth of Fourier modes,
  - ◇ detect possible abnormal growth of the number of iterations (or Krylov space dimensions) in block solvers /interface solvers,
  - ◇ compute a different subset of approximations and/or a posteriori estimators to detect accuracy failures,
  - ◇ detect lack of monotonicity and/or local unstabilities using the computation of total variations.
  - ◇ detect lack of conservation (or positivity) of physical quantities .....etc....

# Part 2: Fault Tolerant Algorithms

## Problem:

- We assume that we have detected a failure and need to restart the computation of the time dependent problem.
- We assume that spare processors have stored on disk copies of subdomain data  $u_j^{n(j)}$ ,  $j = 1..N$ , with no missing block.
- We want to reconstruct in parallel  $u$  at time step  $M$ .

Remark: the first step of the algorithm is to allocate to the spare processor(s), that will become the application processor(s), the subdomain's data corresponding to the processor(s) that fail(s).

# Part 2: Fault Tolerant Algorithms

Model problem

$$\frac{\partial u}{\partial t} = \Delta u + F$$

Implicit First Order Euler Scheme

$$dt \sim h$$

# Part 2: Fault Tolerant Algorithms

## First solution

- Let  $M = \frac{1}{N} \sum_{j=1..N} n(j)$ .

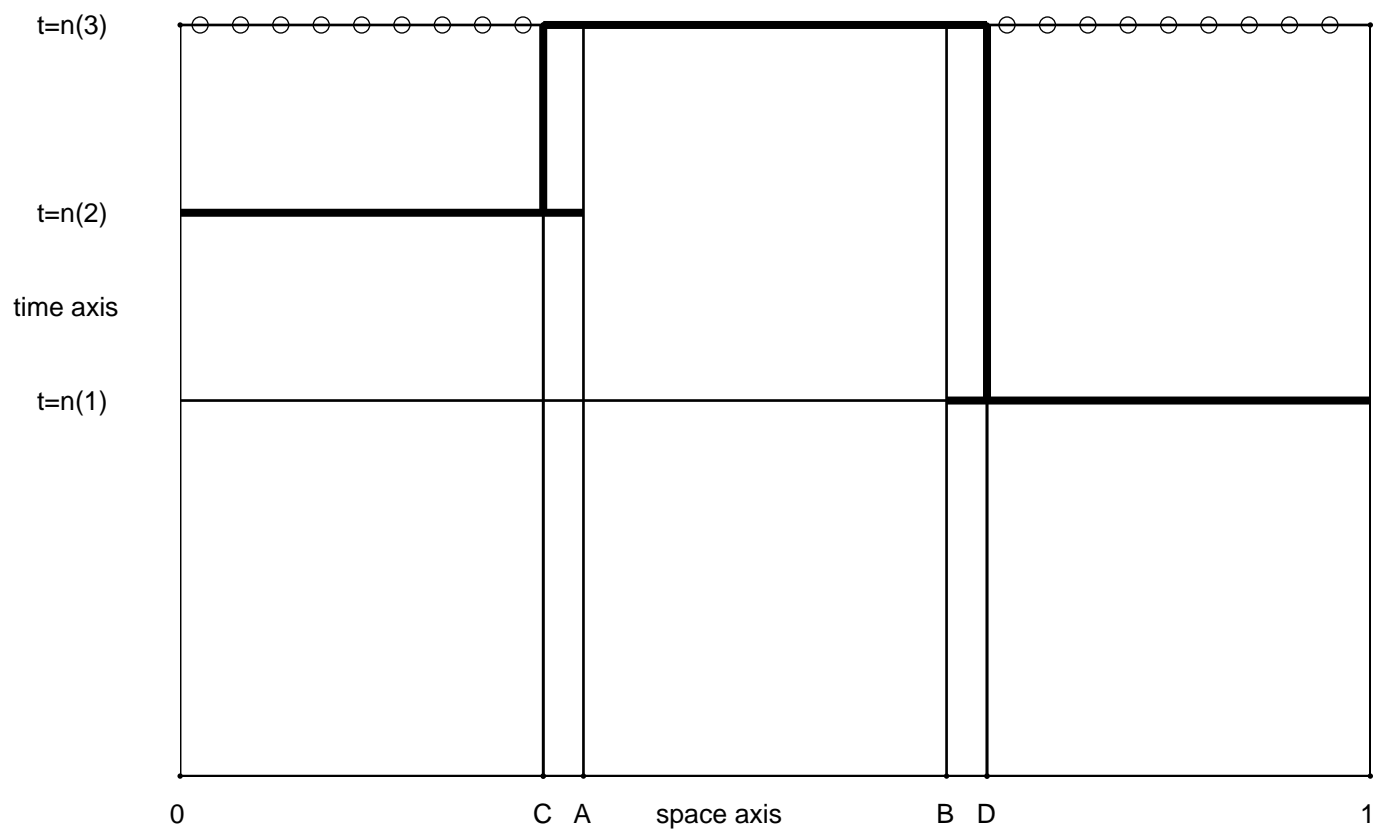
We look for an approximation of  $u_j^M$ .

- We keep in spare processors two copies  $u_j^{n(j)}$  and  $u_j^{m(j)}$  obtained at different time steps.
- If  $\|u_j^n(j) - u_j^m(j)\|_{\Omega_j}$  is below some tolerance number, one can use interpolation/extrapolation in time to get an approximation of  $u_j^M$ .
- However this process is a priori inconsistent and non conservative.
- This procedure might be used to provide an initial guess for (non linear) problems.

# Part 4: Fault Tolerant Algorithms

## Second solution

solution 2



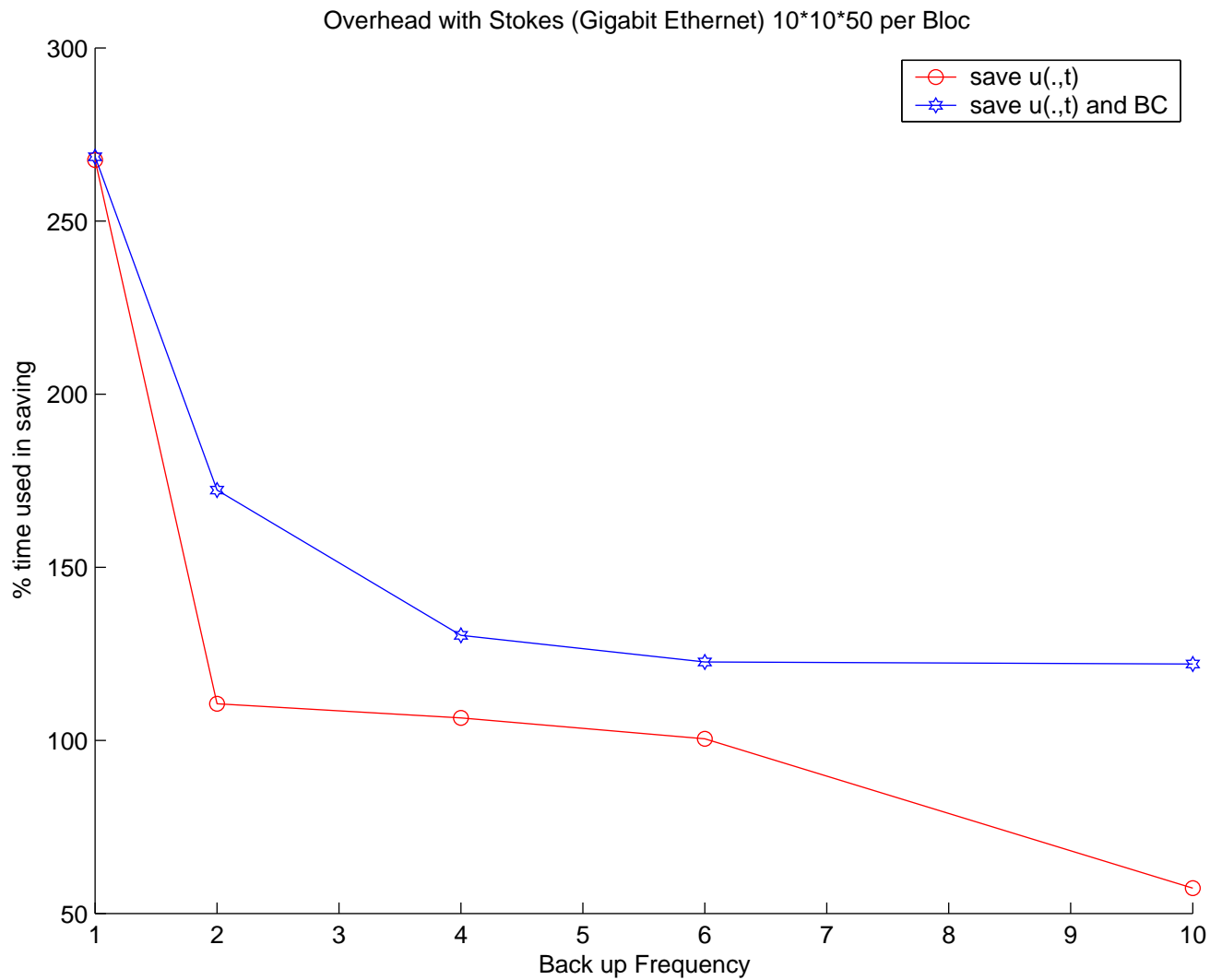
## **Part 4: Fault Tolerant Algorithms**

The difficulty is to accumulate the history of artificial boundary conditions for enough time steps.

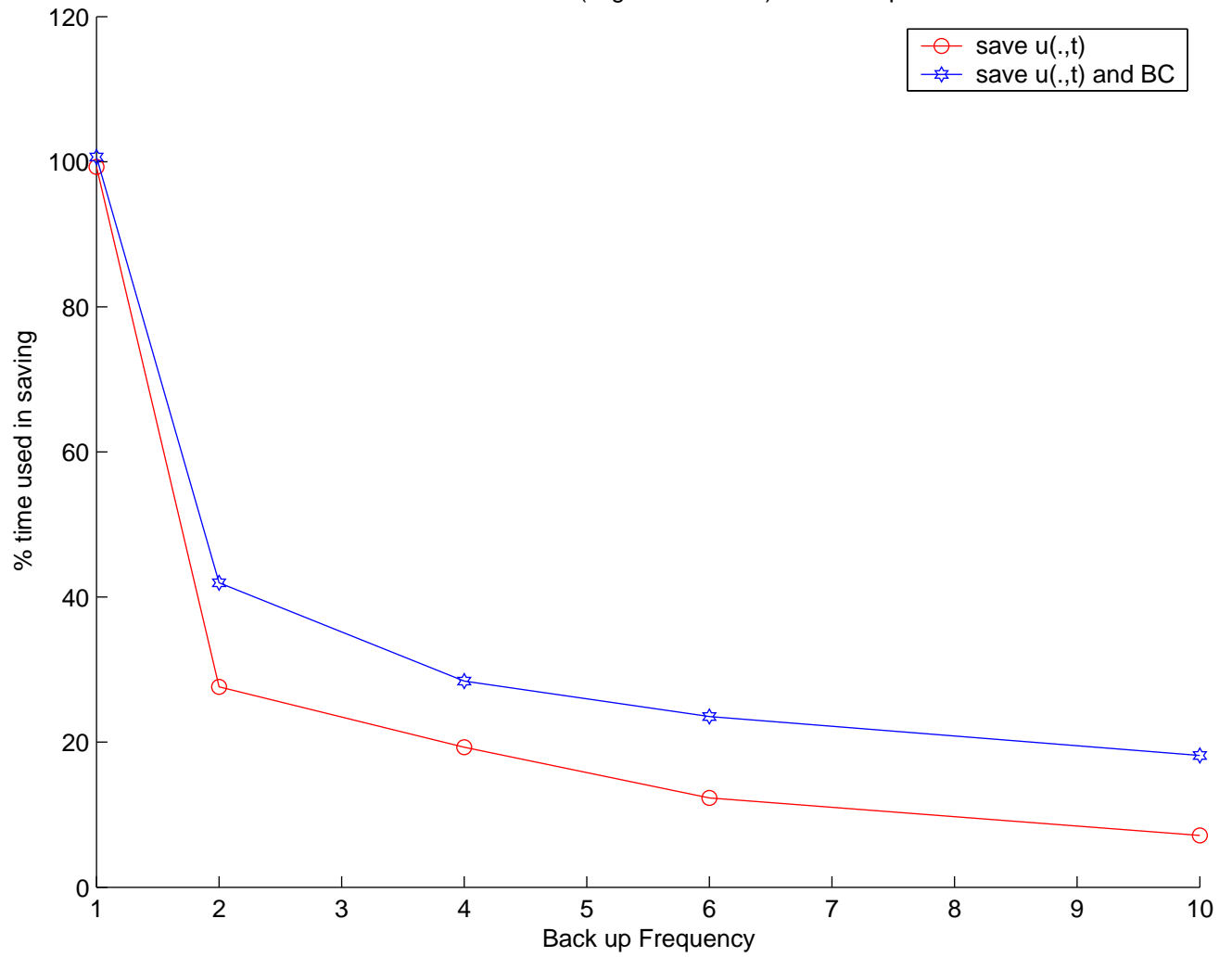
While interface conditions are still one dimension lower than the solution itself, message passing might be time consuming and may slow down the application.

# Part 2: Fault Tolerant Algorithms

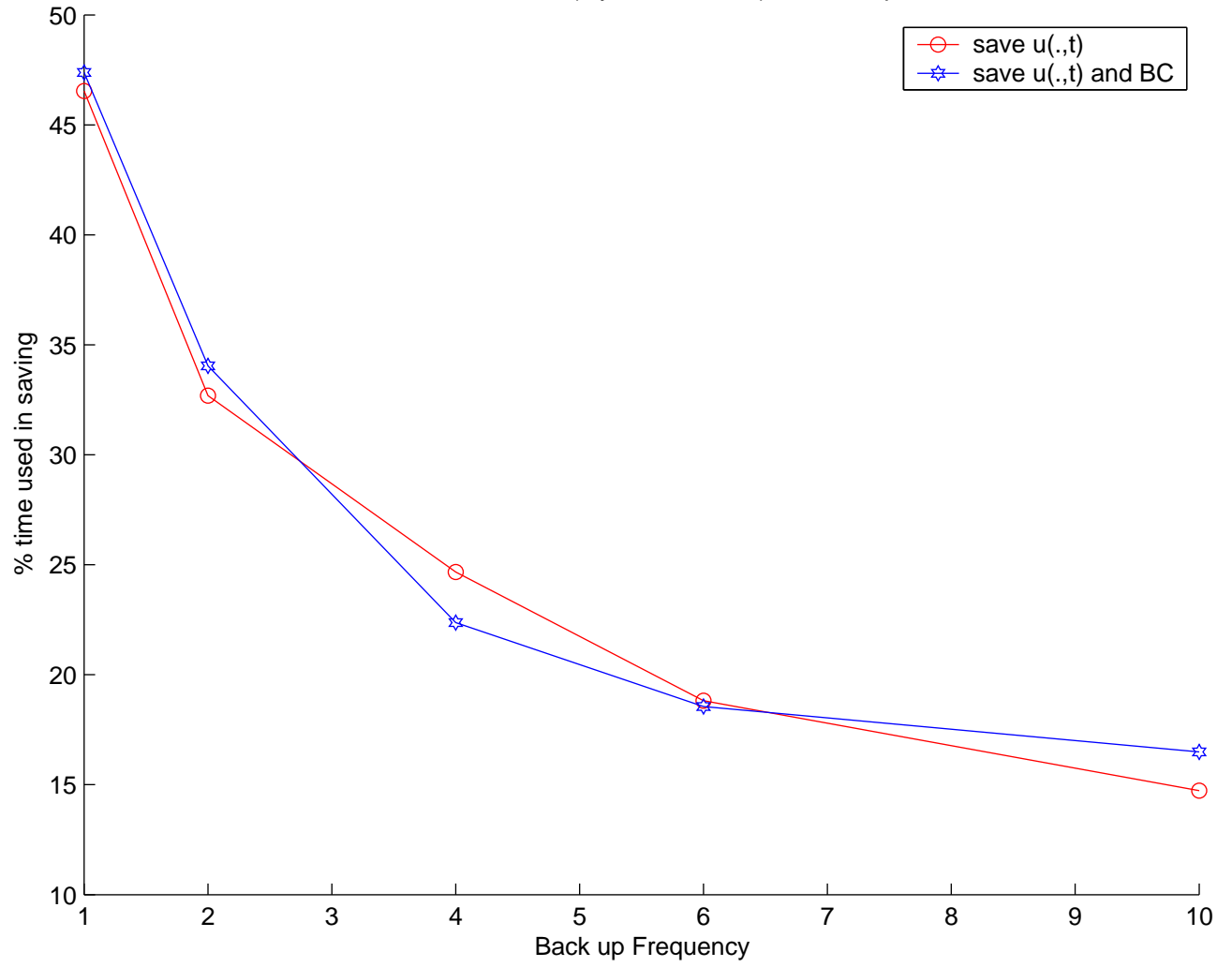
## Performance's issue



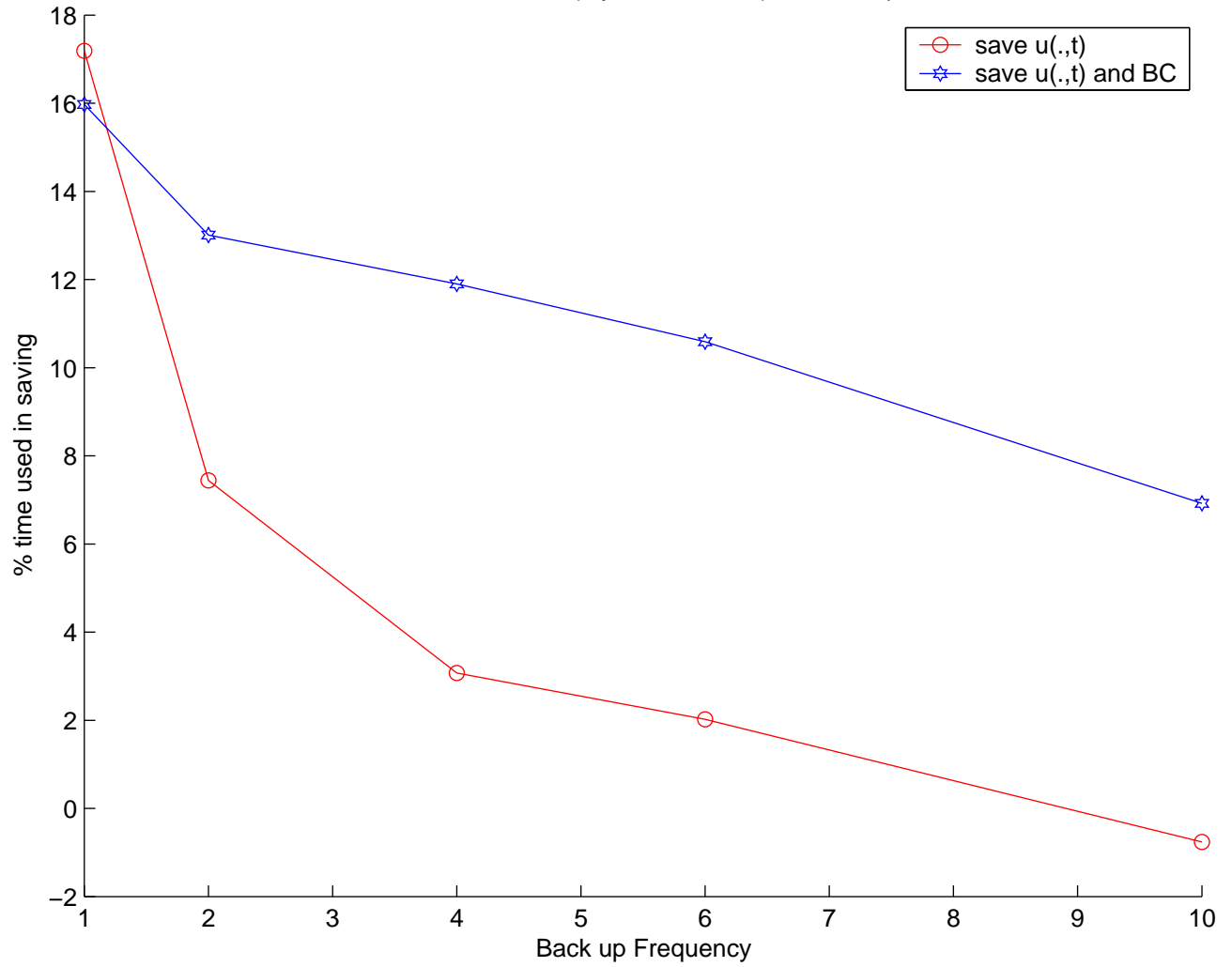
Overhead with Stokes (Gigabit Ethernet) 18\*18\*98 per Bloc



Overhead with Atlantis (Myrinet Network) 10\*10\*50 per Bloc



Overhead with Atlantis (Myrinet Network) 18\*18\*98 per Bloc

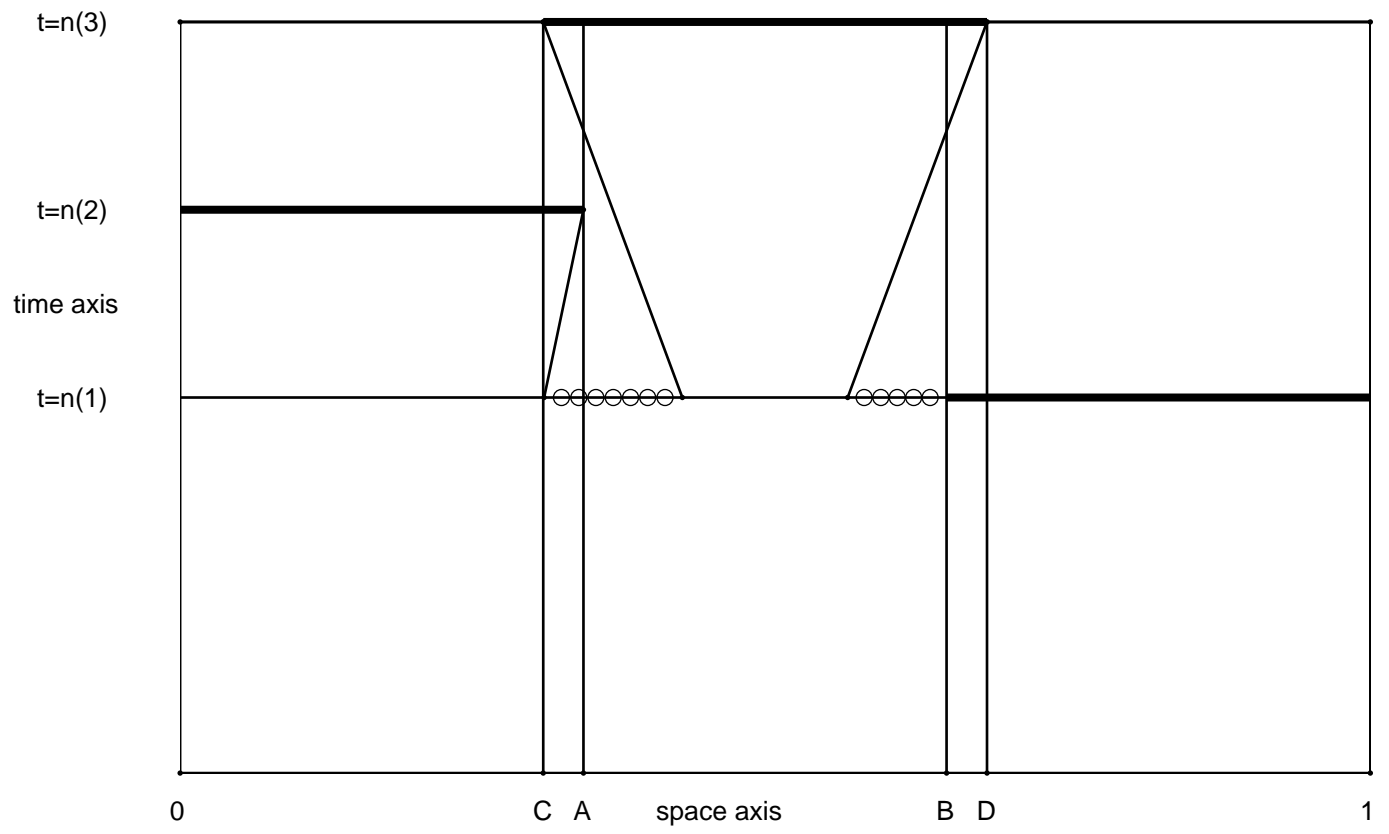


# Part 2: Fault Tolerant Algorithms

## Third solution

- Backward integration in time with an **explicit** scheme for a limited number of time steps ??????
- Existence of the solution is granted by the forward integration in time.
- Remark: the implicit backward integration is in general not defined.
- Information propagates at a finite speed, i.e data influence neighbor cells only.

solution 4



# Part 2: Fault Tolerant Algorithms

Some ideas

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + f(x, t), x \in (0, 1), t \in (0, T)$$

$$u(x, 0) = v(x), x \in (0, 1)$$

$$u(x, T) = w(x), x \in (0, 1)$$

**Problem:**

$u(0, t)?, u(1, t)?$  using an explicit scheme ???

**Assume:**  $T = Kdt$ , with  $1 < K \leq 10$ .

# Part 2: Fault Tolerant Algorithms

First component of the solution

$$\frac{v_j^{n+1} - v_j^n}{dt} = \frac{v_{j+1}^{n+1} - 2v_j^{n+1} + v_{j-1}^{n+1}}{h^2}, \quad h = 1/N.$$

$$\hat{v}_k^n = \delta_k \hat{v}_k^{n+1},$$

with

$$\delta_k \sim -\frac{2}{h}(\cos(k 2 \pi h) - 1), \quad |k| \leq \frac{N}{2}.$$

**Conclusion:** expect error

$$\frac{\nu}{h^K}$$

in inverse subdomain of dependency where  $\nu$  is machine precision.

i.e it may work for few time steps.

# Part 2: Fault Tolerant Algorithms

A better idea?

$$\epsilon \frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} + \frac{\partial u}{\partial t} = f(x, t), x \in (0, 1), t \in (0, T)$$

$$u(x, T) = w(x), x \in (0, 1)$$

$$u(x, T - h) = w_2(x), x \in (0, 1)$$

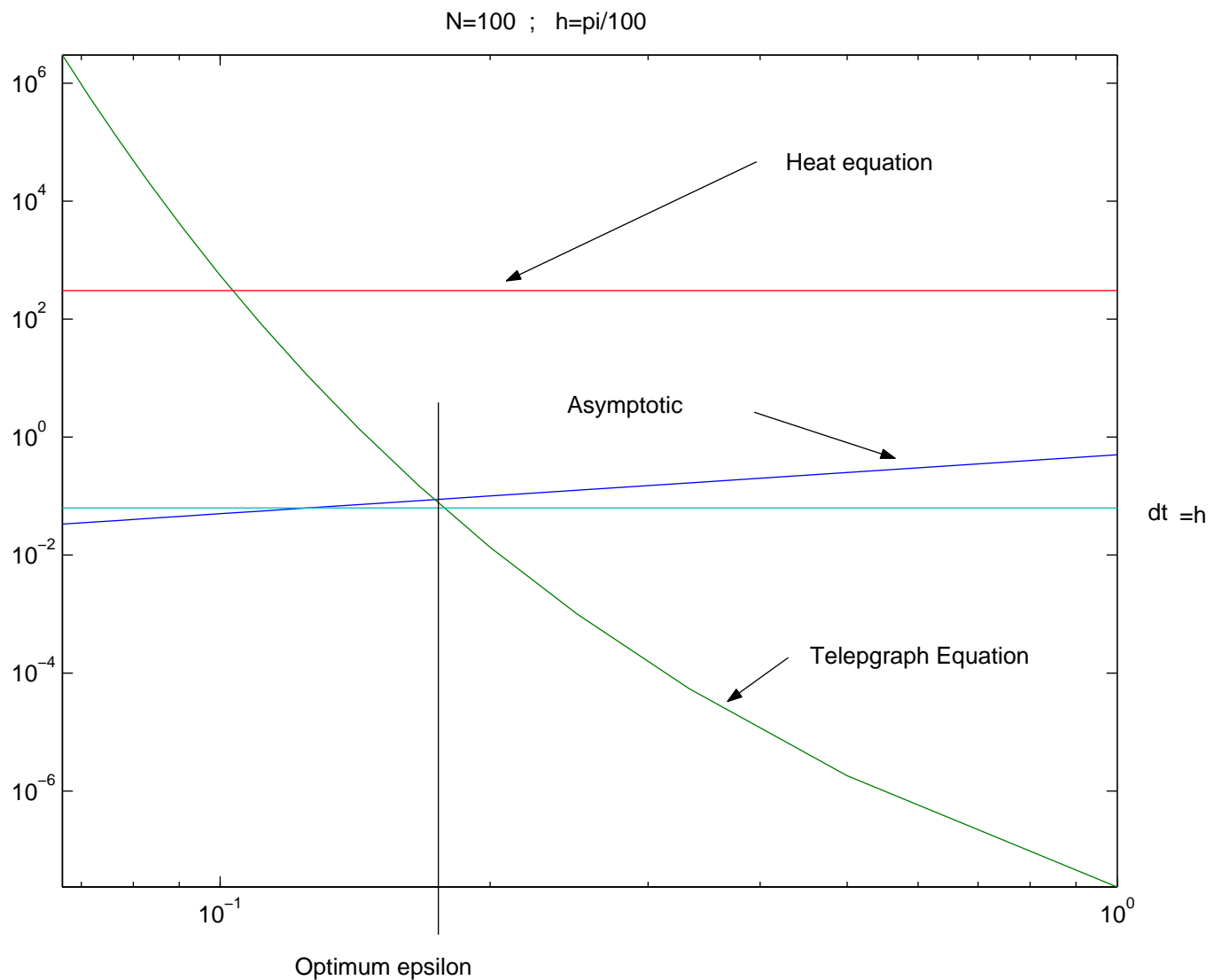
- Requires to save the solution on two consecutive time levels.
- Asymptotic theory of the Telegraph equation that converges to the heat equation for large time - see W.Eckhaus and MG SIAM J. Math. Anal. 1990.
- The smaller is  $\epsilon$  the more unstable is the explicit backward scheme:

$$\epsilon \frac{v_j^{n+1} - 2v_j^n + v_j^{n-1}}{dt^2} - \frac{v_{j+1}^n - 2v_j^n + v_{j-1}^n}{h^2} + \frac{v_j^{n+1} - v_j^n}{dt} = f_j^n.$$

- The smaller is  $\epsilon$  the better is the (non numeric) approximation.

- The slope of the characteristic for the backward integration is of the order of  $\epsilon^{1/2}$  !

Illustration of the stability and error analysis with Fourier for **10 time steps**:



Conclusion: we may retrieve part of the solution in an inverse cone with first order accuracy in time.

# Part 2: Fault Tolerant Algorithms

## Second component of the solution

A standard procedure in **inverse heat problem**: the space marching method.

reference the Mollification method see D.A.Murio's book Wiley 93.

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + f(x, t), x \in (0, 1), t \in (0, T)$$

$$u(x, 0) = v(x), x \in (0, 1)$$

$$u(x, T) = w(x), x \in (0, 1)$$

$u(x, t)$  given in the narrow strip  $(-d, d)$ ,  $d < \frac{1}{2}$ .

- This method (may) requires a regularization procedure of the data obtained by the Telegraph approximation:

$$\rho_\delta * u\left(\frac{1}{2} \pm h, t\right), t \in (0, K dt),$$

where

$$\rho_\delta = \frac{1}{\delta\sqrt{\pi}} \exp\left(-\frac{t^2}{\delta^2}\right).$$

- The space marching scheme

$$\frac{v_{j+1}^n - 2v_j^n + v_{j-1}^n}{h^2} = \frac{v_j^{n+1} - v_j^{n-1}}{2dt} + f_j^n,$$

is unconditionally stable, provided  $\delta \geq \sqrt{\frac{2dt}{\pi}}$ .

Conclusion: we may avoid the solution of the inverse problem with a gradient method or use this technique to get an initial guess.

Remark: this procedure should be easy to extend to higher dimensions and may survive to mild non linearities

## **Part 3 and Conclusion:**

**One more job to do for spare processors is  
solution verification:**

- AIAA Guide for the Verification and Validation of Computational Fluid Dynamics Simulations.
- the ZZ recovery method - see Zienkiewicz et Al, and ref.
- Equilibrated residual method for FE .- see Ainsworth & Oden and ref.
- A posteriori Finite-Element free constant output bounds - see Patera and ref.
- Stochastic method in the Bayesian framework - ref Glimm et Al..
- More recently the least square extrapolation method - ref. M.Garbey and W.Shyy.