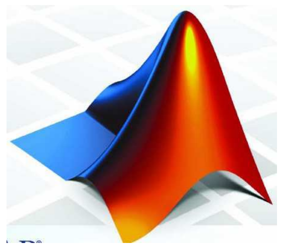
Parallel Matlab – an Introduction

Ondřej Jakl, Tomáš Musil

Institute of Geonics, Czech Academy of Sci.
VŠB – Technical University Ostrava

SNA 2008

MATLAB



- MATrix LABoratory: One of the most widely used mathematical computing environments in technical computing
- An interactive environment providing
  - high performance computational routines
  - an easy-to-use, C-like scripting language
- In the 1970s started out as an interactive interface to EISPACK and LINPACK (sequential) linear system solution routines
- Commercially produced by The Mathworks since 1984
  - founders Jack Little and Cleve Moler
- A serial program
- 1995: Cleve Moler argued that there was no market at the time for a parallel Matlab

Main competitors:
- Maple (Maplesoft)
- Mathematica (Wolfram Research)

Motivation for parallel Matlab

- Modern scientific and engineering problems grow in complexity
  - the computation time and memory requirements increase
  - parallel computation becomes a necessity
- Multiple Matlab instances running on a parallel computer can be used to solve embarrassingly parallel problems
  - without any change to Matlab itself
- Increase of problem sizes and processor speed have reduced the portion of time spent in non-computation related routines
  - e.g. in the parser, interpreter and graphics routines, where parallelism is difficult to find
- Dramatic changes in hardware: parallel systems entered mainstream
  - clusters since late nineties
  - multicore processors recently

Going parallel

- November 2003 [Choy2005]: 27 parallel Matlab projects found on the Internet
  - varying in their scope (one-man projects, university lab research projects, commercial projects, etc.)
  - varying in status (defunct, developed)
  - generally not supported by The MathWorks
- Approaches
  - compile Matlab scripts into parallel native code
  - provide a parallel backend to Matlab, using Matlab as a graphical frontend
  - coordinate multiple MATLAB processes to work in parallel

MathWorks’ approach

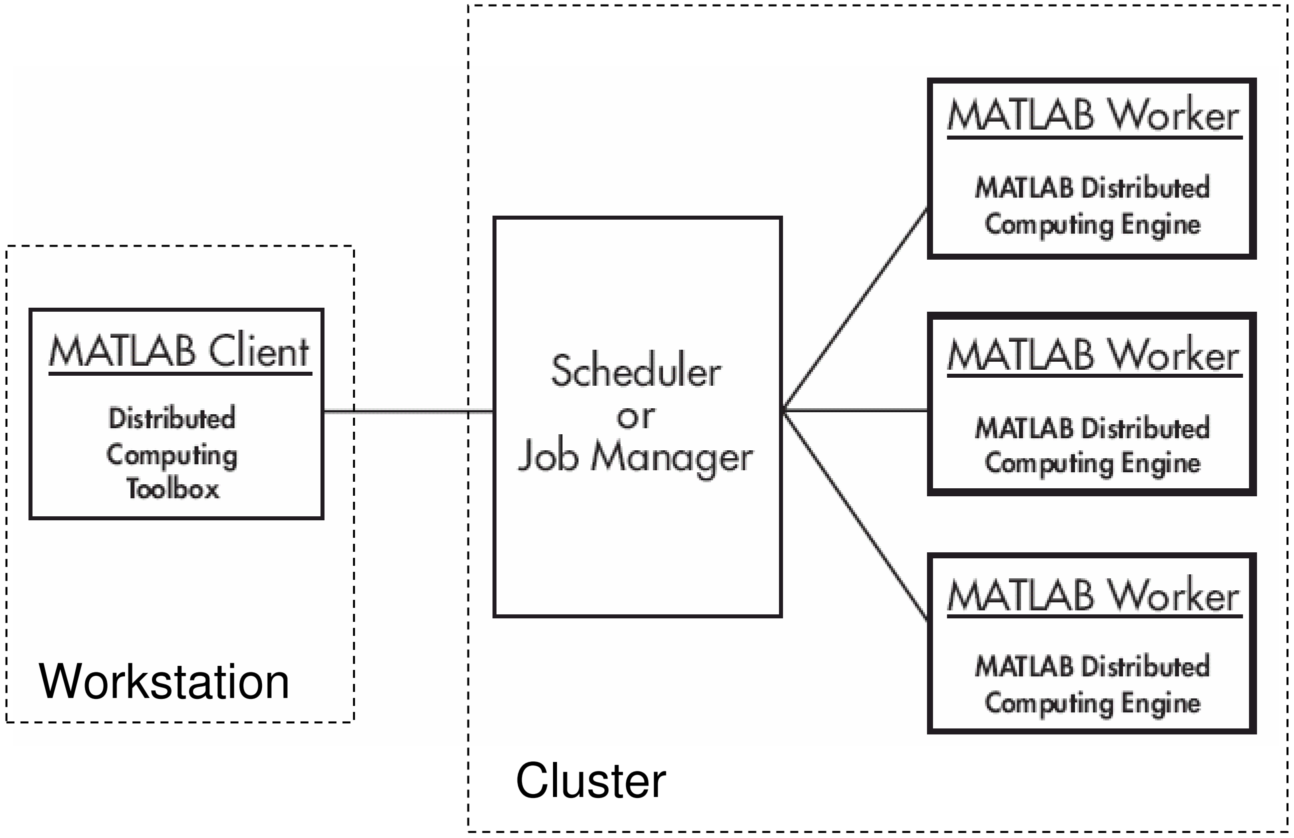
DCT/DCE

- Distributed computing products (pM)
  - Distributed Computing Toolbox (DCT): adds parallel constructs / processing to Matlab
  - Distributed Computing Engine (DCE): necessary to take advantage of DCT on a cluster or on more than 4 processors
- Initial version 1.0 in 2004 (part of R14 SP1)
  - enabled to execute coarse-grained MATLAB algorithms divided into independent tasks in a cluster of computers (= distributed computing)
- Supported on Windows, UNIX (Linux, Solaris), Macintosh platforms
- Can be tested on a one-processor machines
- Generally very little can be found on pM internals

Rapid development

- Each year substantial improvements of DCT (DCE)
- Version 2.0 (part of R14 SP3 – Nov 2005) - 224
  - support for communication among interdependent tasks, based on the industry-standard Message Passing Interface (MPI) (= parallel computing)
  - support for various schedulers (LSF, MPIExec)
- Version 3.0 (part of Release 2006b – Sep 2006) - 374
  - interactive parallel mode (pmode)
  - distributed arrays & parfor loop
  - support for Windows Compute Cluster Server (CCS)
- Current version 3.2 (part of Release 2007b – Sep 2007) - 487
  - local scheduler and workers (ver. 3.1)
  - new parfor loops
  - parallel profiler

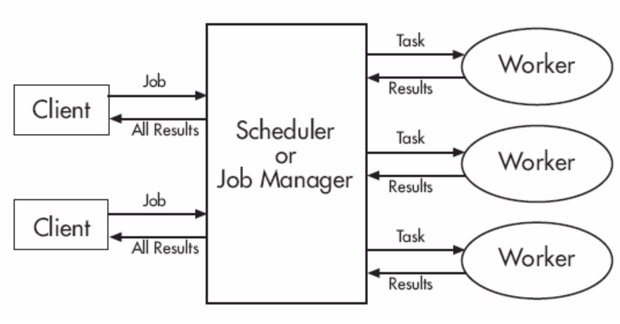
Basic pM setup


MATLAB Client
Distributed Computing Toolbox
Workstation
Scheduler or Job Manager
MATLAB Worker
MATLAB Distributed Computing Engine
MATLAB Worker
MATLAB Distributed Computing Engine
MATLAB Worker
MATLAB Distributed Computing Engine
Cluster

Note: for testing purposes, all components can share a single computer

Terminology

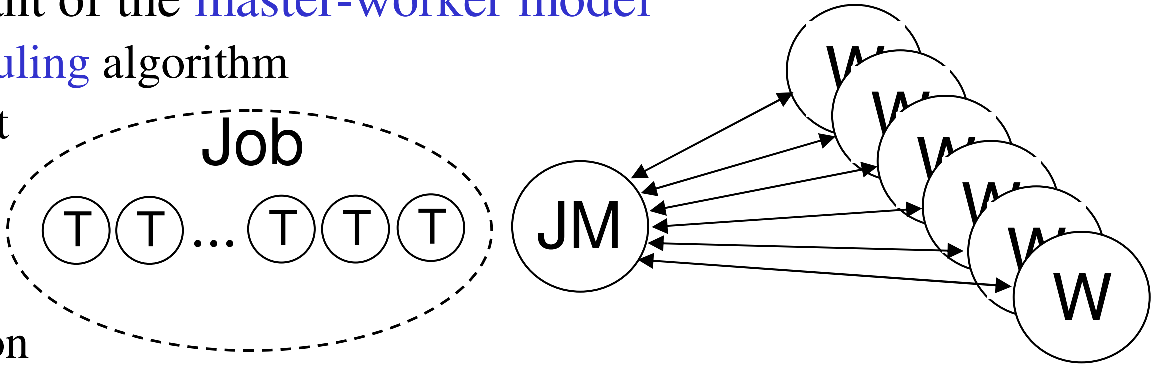
- A job is some large operation that you need to perform in MATLAB
- A job is decomposed into tasks
  - the user defines the decomposition
  - tasks are supposed to run concurrently
  - tasks do not necessarily have to be identical
- The MATLAB session in which the job and its tasks are defined is called the client session
  - needs DCT



- DCE is a run-time environment (engine) that executes the job by evaluating each of its tasks and returns the result to the client session
- The job manager is the central part of the DCE
  - coordinates the execution of jobs and the evaluation of their tasks
  - distributes the tasks for evaluation to the engine's individual Matlab sessions called workers (labs)
  - pM can also collaborate with some third-party job schedulers (programs imposing some job priority scheme on the cluster)
  - DCT can also run a local scheduler and up to 4 workers on the client machine

pM's distributed computing

- Distributed jobs: composed of independent tasks
  - tasks do not directly communicate with each other
  - tasks do not need to run simultaneously
  - not important which worker executes a specific task
- pM implements a variant of the master-worker model
  - a kind of task-scheduling algorithm
  - load balancing effect
  - data and functional decomposition
  - pM takes care of all the organization of the computation
- Applicable for coarse-grained embarrassingly parallel problems
  - e.g. Monte Carlo simulations
- Java RMI (Remote Method Invocation) behind the scene?



Running a distributed job

- Find a job manager
  - sched = findResource('scheduler','type','local')
- Create a job
  - job = createJob(sched);
- Create tasks of the job (compute the products 2×3, 2×4, 2×5 in parallel)
  - createTask(job, @prod, 1, {[2 3]});
  - createTask(job, @prod, 1, {[2 4]});
  - createTask(job, @prod, 1, {[2 5]});
- Submit a job for execution
  - submit(job);
- The job manager distributes the tasks to the workers for evaluation
- Retrieve the job's results
  - waitForState(job)
  - getAllOutputArguments(job)   ans = [6] [8] [10]
- Destroy the job
  - destroy(job)

Distributed evaluation of functions

- Straightforward function evaluation on a set of arguments concurrently
- Alleviates from having to define individual tasks and jobs
- feval – evaluates a function handle
  - prod([2,3]) ans = 6
  - feval(@prod, [2,3]) ans = 6
- dfeval – distributed version of feval
  - dfeval(@prod, {[2,3] [2,4] [2,5]}) ans = [6] [8] [10]
- Number of task equals to the number of elements in the cell array
- Some natural restrictions
- pM: finds job a job manager, creates a job, tasks in that job, submits the job, retrieves results
- Synchronous (blocking) and asynchronous (non-blocking) versions (dfevalasync)

@ - function handle for passing functions as arguments

{} – cell array

pM's parallel computing

- Parallel job: composed of several instances of a single task
  - the task is duplicated on each worker (called lab in pM)
  - the duplicated tasks run simultaneously
  - each worker can perform its task on a different set of data
  - the task instances can communicate with each other during execution
  - the task instances are enumerated (labindex)
- Makes pM to a true message passing system
  - built on MPI (Message Passing Interface) implementation
    - supplied with MPICH2 runtime
  - message passing through labSend, labReceive, labSendReceive, labProbe, labBarrier, labBroadcast, gop, gplus, ...
    - those functions have direct counterparts in MPI
    - labSend analog to MPI's standard send (MPI_Send) – synchronous blocking send (may deadlock for large data!)
    - in MPI much richer collection of message passing functions
  - tags for data identification

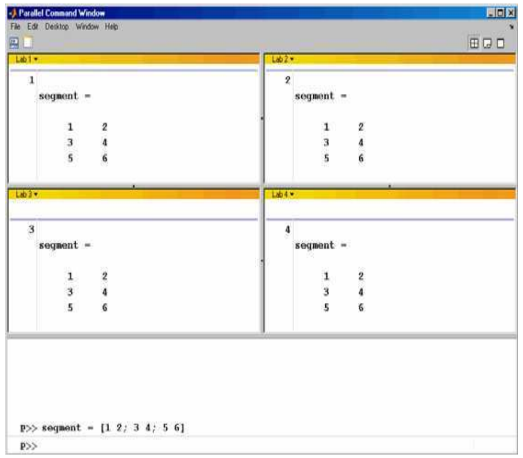
Feel & taste of pM’s message passing

```
% Message ping-pong
A = rand(10);                  % a simulated message
tic;                           % start time measuring
tag = 1;
for i = 1:100
  if  (labindex == 1)          % master
    labSend(A,2,tag);          % send the message and wait for reply
    A=labReceive(2,tag);
  else                         % slave
    A = labReceive(1,tag);     % get the message and return immediately
    labSend(A,1,tag);
  end
end
tm = toc;                      % resulting time
```

Interactive parallel mode

- pmode – an interactive interface to the parallel job mechanism
- pM client session interacts directly with the labs participating in the interactive session.
  - commands are executed immediately on all the labs
  - results are returned immediately to the client session
- Useful for debugging purposes, working with distributed arrays, etc.
- Multi-window user interface in DCT 3.2

```
>> pmode start
P>> ! hostname
1: thea01
2: thea02
3: thea03
4: thea04
5: thea05
6: thea06
7: thea07
8: thea08
P>> pmode exit
>>
```



```
1
  segment =

      1     2
      3     4
      5     6
```

```
2
  segment =

      1     2
      3     4
      5     6
```

```
3
  segment =

      1     2
      3     4
      5     6
```

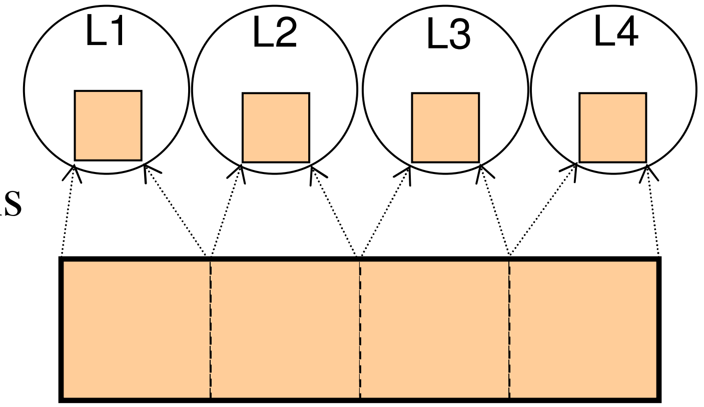
```
4
  segment =

      1     2
      3     4
      5     6
```

```
P>> segment = [1 2; 3 4; 5 6]
P>>
```

Distributed arrays

- Distributed arrays: arrays partitioned into segments, each of which resides in the workspace of a different lab
- Allow to handle larger data sets than in a single Matlab session
- Support for more than 150 Matlab functions (e.g. finding eigenvalues)
  - in a very similar way as with regular arrays



- Parallel processing transparent to the user
  - without having to manage low-level details of message passing
- Coheres to the data parallel model of parallel computations
  - based on collective operations on arrays, with these arrays distributed over a number of processors
  - distribution of data and communication is done by the compiler with guidance from the programmer
  - HPF – the best known representative

More on distributed arrays

- Construction:
  - partitioning a larger array (distribute)
  - building from smaller arrays (darray)
  - using constructor functions (e.g. rand(m,n,darray()))
- Decomposition options:
  - 1D block decomposition along the selected distribution dimension
  - 2D decomposition for 2-dimensional arrays only
- Quite a lot of other auxiliary functions for e.g.
  - creating local arrays from distributed ones and vice versa
  - obtaining information about (distributed) arrays
    - e.g. if and how they are partitioned
  - changing the dimension of the distribution
  - providing indices in the distribution segments

```
P>> D = ones(250, 10, darray())
1: local(D) is 250-by-3
2: local(D) is 250-by-3
3: local(D) is 250-by-2
4: local(D) is 250-by-2
```

Parallel FOR loop

- Performs loop iterations without enforcing their particular ordering
- Allows for fine-grained parallelism, interleaving serial and parallel code
- Parfor-loop distributes loop iterations over a set of workers
  - iterations must be independent of each other
  - no communication can occur between workers during the execution
  - part of the iterations is executed on the client (where the parfor was issued), part is executed in parallel on the workers
- pM takes care for the necessary communications
  - distributes parfor-loop data to the workers
  - gathers results back to the client and pieces them together
- May be counterproductive with a small number of simple calculations
- Data parallel construct – analogues in other languages:
  - FORALL in HPF Ex.: `FORALL (I=1:N, J=1:M)  A(I,J) = 1.0 / REAL(I+J)`
  - PARALLEL FOR in OpenMP Ex. C language:

```
#pragma omp parallel for
  for (i = 1; i <= n; i++)
  b[i] = (a[i]-a[i-1]) * 0.5;
```

More on parfor-loop

- Changed in DCT ver. 3.2 – even syntax!
  - originally just for use with distributed arrays in a parallel job
- matlabpool: reserves/start (some) workers for executing subsequent parfor-loops
- Parfor-loops are not fully equivalent to their for-loop counterparts, e.g.
  - restrictions on using some statements in the loop body (e.g. break)
  - value of the loop variable at the end of the loop is unchanged
- Classification of variables referred to in the parfor-loop (5 types)
  - automatic – no explicit clauses
    - OpenMP: private, shared, reduction, lastprivate, etc.

```
% Parallel Pi calculation
matlabpool open 4
nsteps = 100000;
step = 1/nsteps;
s = 0;
parfor i = (1:(nsteps-1))
    x = (i - 0.5) * step;
    s = s + (4 /(1 + x^2));
end
s * step
ans = 3.1416
matlabpool close
```

Practical pM computations

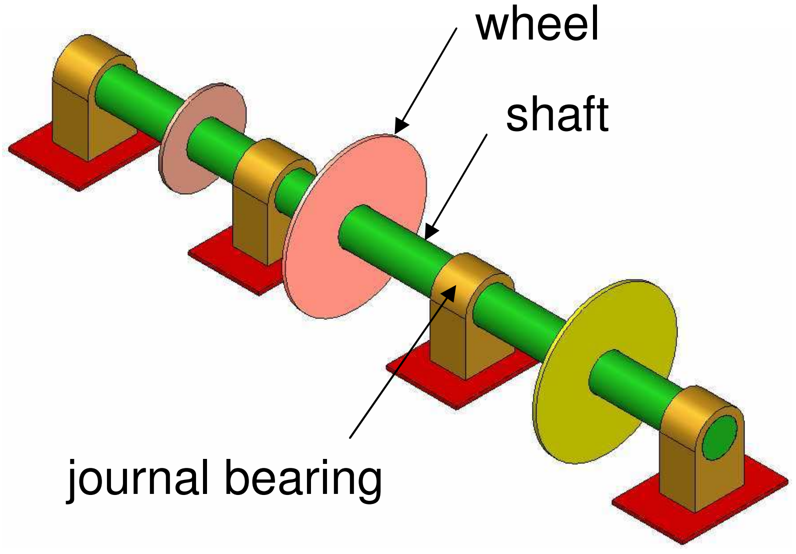
Nonlinear rotordynamics

Equation of motion of a rotor with journal bearings

$$\mathbf{M}\ddot{\mathbf{x}} + (\mathbf{B} + \nu\mathbf{K}_\mathbf{H} + \Omega\mathbf{G})\dot{\mathbf{x}} + (\mathbf{K} + \Omega\mathbf{K}_\mathbf{C})\mathbf{x} = \mathbf{f}(t) + \mathbf{f}_\mathbf{L}(\mathbf{x}, \dot{\mathbf{x}})$$

Features:

- rank of matrices is quite low, shaft is beam-like body (DOF about 100)
- dependence of left-hand side on revolutions Ω of the rotor
- nonlinear couple vector on right-hand side



Elementary rotor calculations

Static calculations involve

- calculation of equilibrium position and its stability judgment
- assembling of Campbell's diagram – dependence of eigenvalues of the linearized system on revolutions

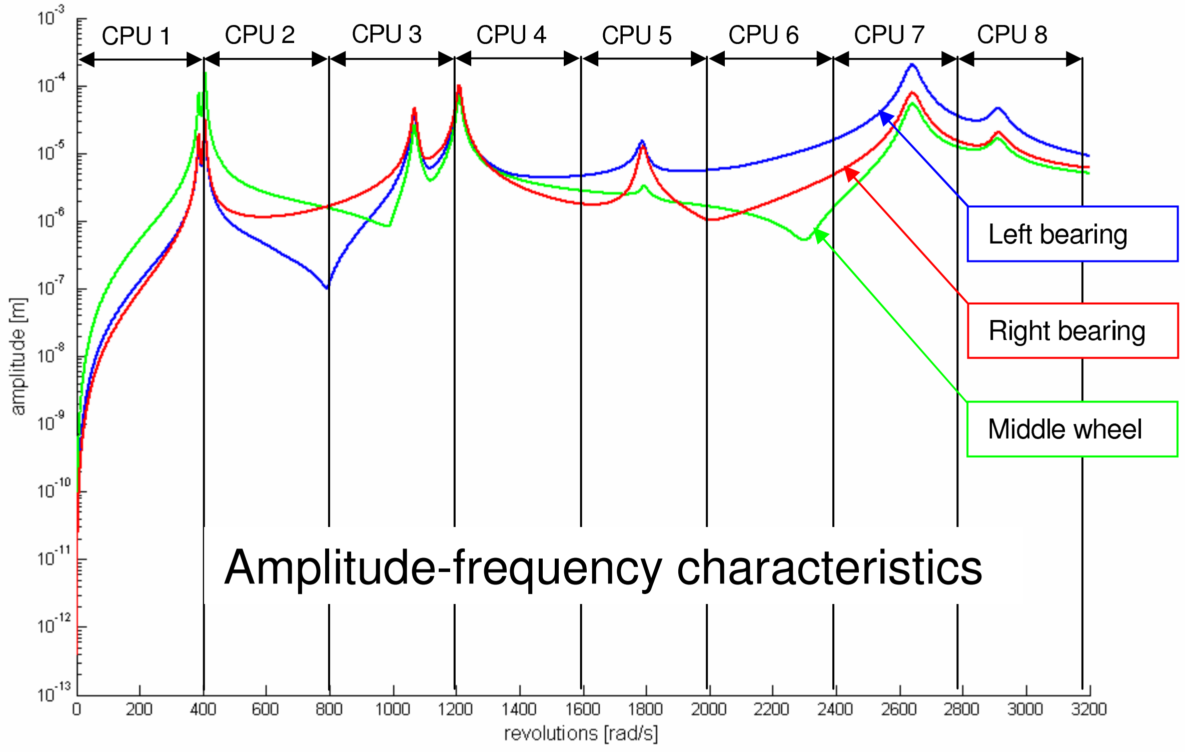
Dynamic calculations involve

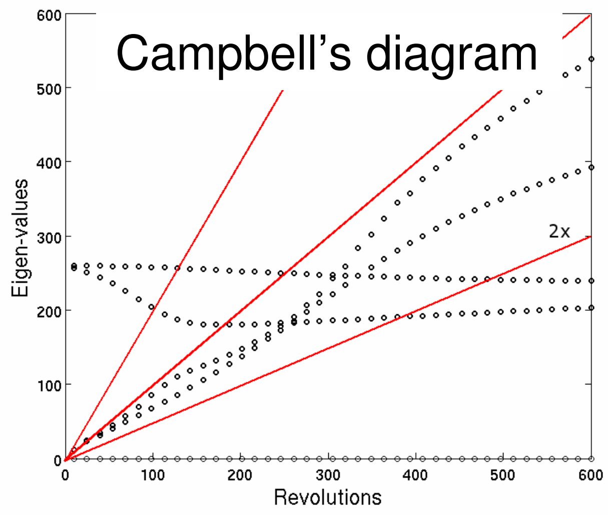
- determination of dynamic coefficients dependence on revolutions
- calculation of steady state response of a rotor on centrifugal forces
- stability judgment of a periodical response
- calculation of amplitude-frequency characteristic
- transient analysis – response on general time dependent forces

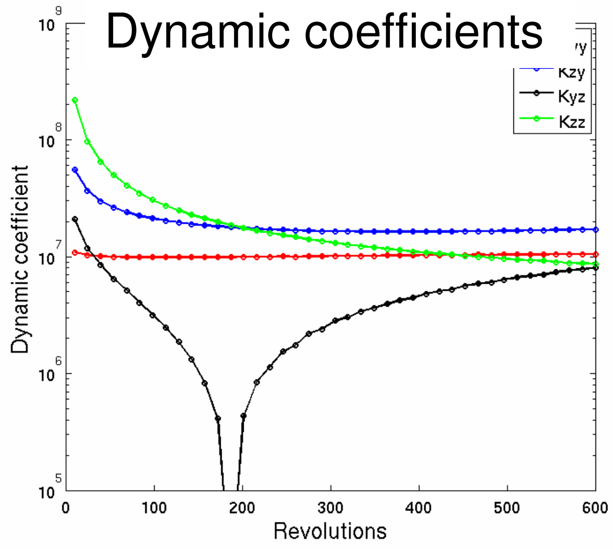
Calculations dependent on revolutions

Appropriate for distributed computations

- revolution range is divided into intervals
- each task calculates independently one interval
- efficiency is almost absolute (1)





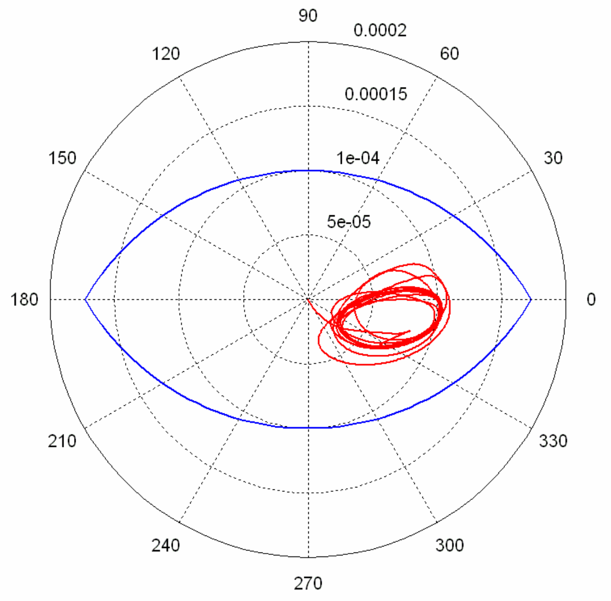


Time dependent problems

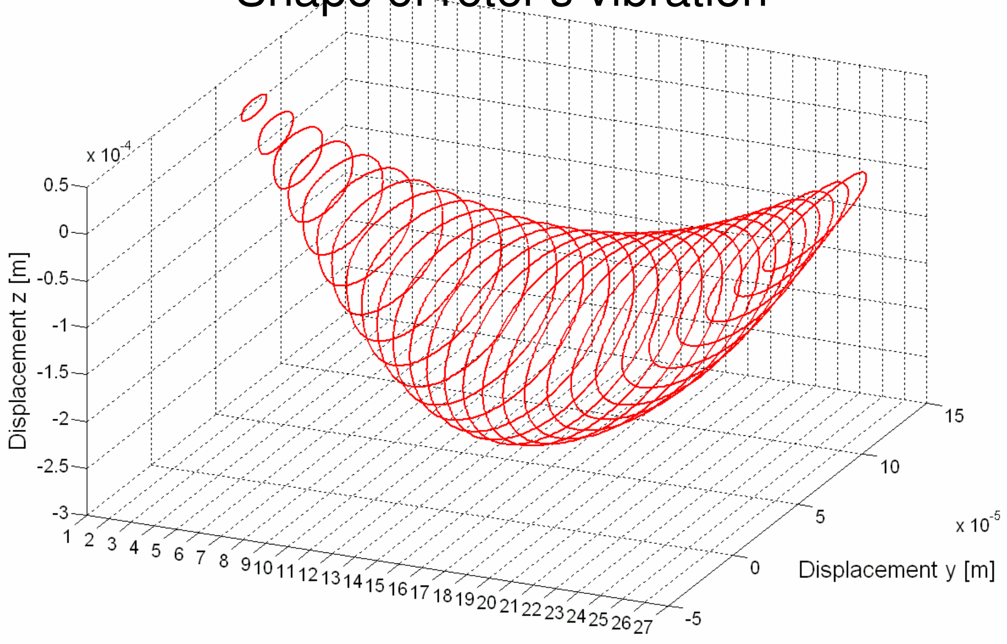
Implicit algorithms for time dependent problems often require calculations of Jacobi matrices

– e.g. Modified Newmark's method, Trigonometric collocation method

Journal centre trajectory



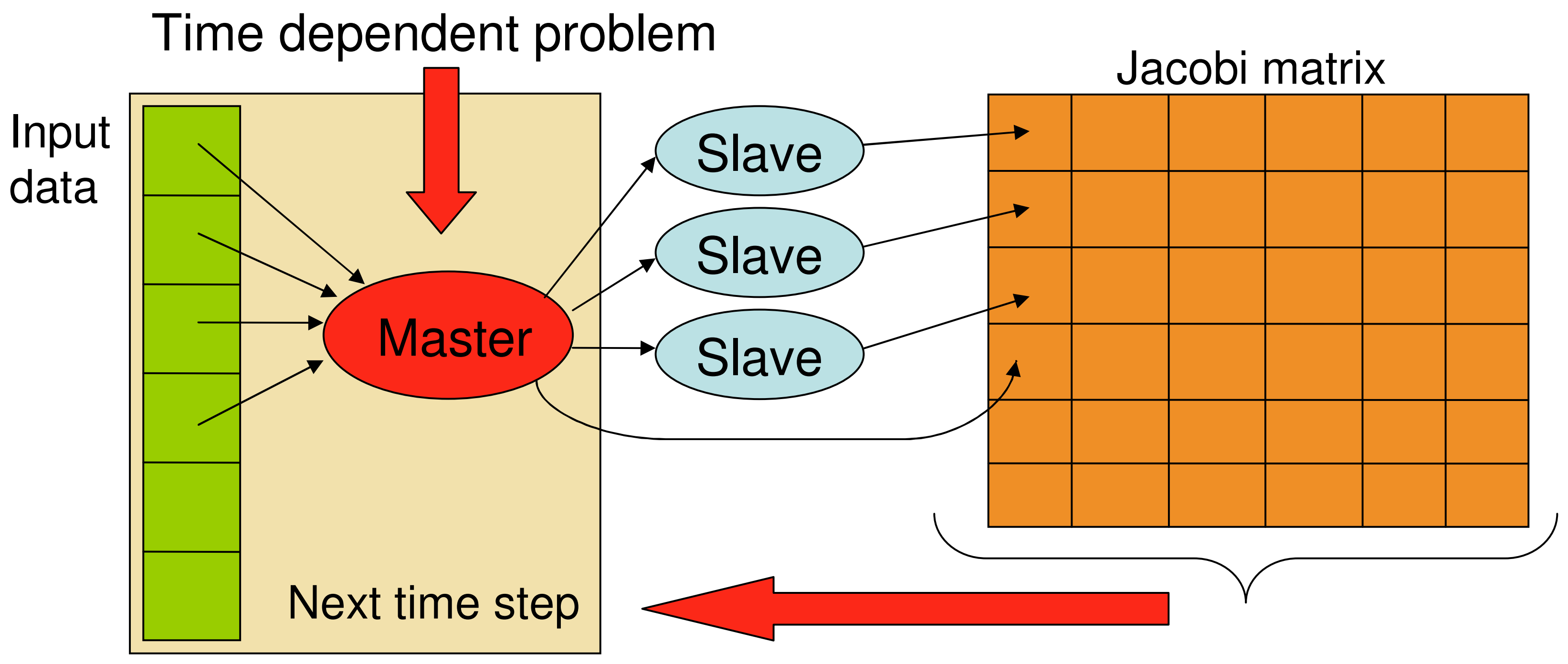
Shape of rotor's vibration



Scheme of the computation

Parallel computation applied

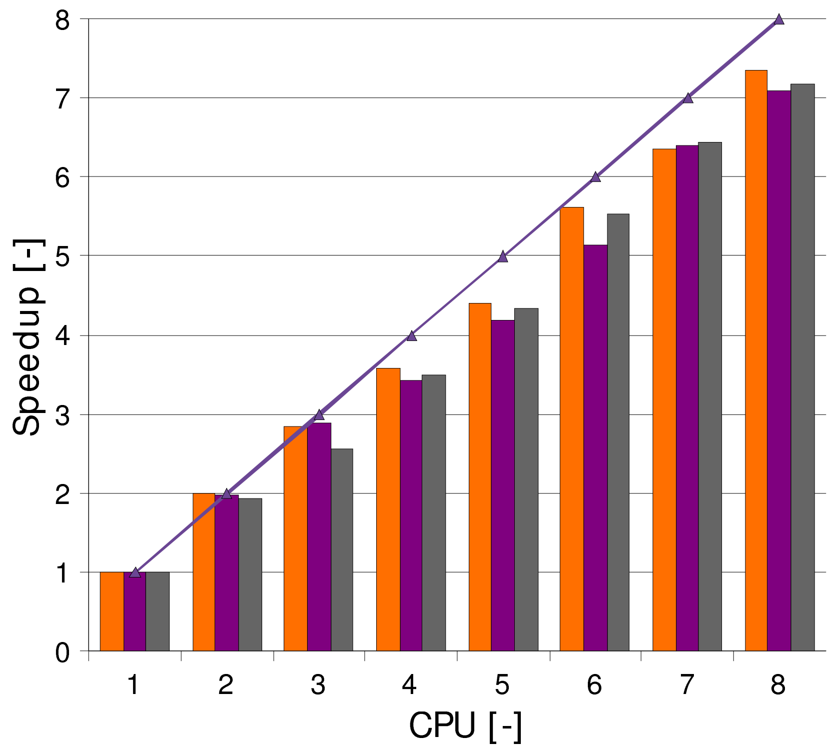
- elements of Jacobi matrix are computed simultaneously
- Master-Slave scheme with active message passing was most efficient

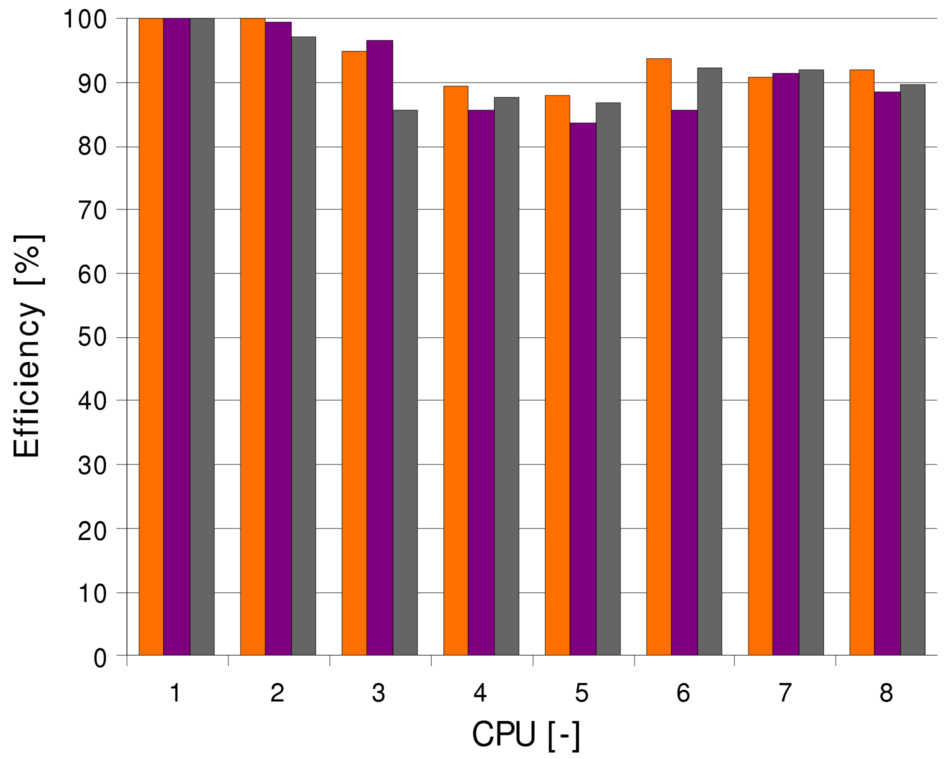


Performance characteristics

Achieved speedup and efficiency on the Thea cluster (Institute of Geonics) for different configurations (number of bearings) of the rotor

– Thea: a Beowulf cluster composed of 8 PC's (AMD Athlon 1.4 GHz, 1.5 GB RAM), Fast Ethernet interconnect; DCT/DCE version 3.0





Conclusions

“Think Matrices, Not Messages”

(Cleve Moler)