Software Component Architecture

Rainer Niekamp

Institute for Scientific Computing

TU Braunschweig

Contents:

- About Software Components
  - What is it?
  - Why use them?
- Software Component Architectures
- The Component Template Library (CTL)
  - central idea: the generalisation of linkage
  - main functionalities and their usage

Contents:

- Examples of Coupled Simulations
  - a Simulation Interface
  - algebraic solvers for coupled systems
  - the mapping of a (multi) physical system to an running distributed application
  - multiscale simulation
  - high dimensional integration

What is a Software Component ?

- A piece of software offering (via an interface) a predefined service and which is able to communicate with other components.
- Criteria (by Clemens Szyperski and David Messerschmitt) for software components:
  - Multiple-use => parallel execution
  - Non-context-specific => exchangeable
  - Composable with other components
  - Encapsulated i.e., non-investigable through its interfaces
  - A unit of independent deployment and versioning

History of Software Components

- So-called software crisis in the sixties
- The idea to componentize prefabricated software first published at the NATO conference on software engineering in Garmisch, Germany, 1968 titled Mass Produced Software Components by Douglas McIlroy
- Subsequent inclusion of pipes and filters into the Unix operating system
- The modern concept of a software component largely defined by Brad Cox of Stepstone, => Objective-C programming language
- IBM: System Object Model, SOM in the early 1990s.
- Microsoft: OLE and COM
- Today, several successful software component models exist

Reasons to use Components

- Growing number of (freely) available libraries and programs worth to be re-used
- Exchangeable software units
- Support of distributed parallel run time systems
- Avoids linkage of may be incompatible libraries
- Longer lifetime of implementations

Explicit Message Passing versus Remote Method Invocation

Explicit Message Passing using MPI, PVM, MPCCI:

- Coupling programs by explicit message passing needs inserting of communication points into source code => source and expertise for each program needed
- No seperation of communication and algorithm => difficult maintenance of code
- Each new pair of coupling produces amount of programming

Remote Method Invocation based on Components:

- Needs component framework
- Keeps functional programming style
- Exchangeability of coupled components
- Type safe communication

Differences from object-oriented programming

- Idea in object-oriented programming (OOP): software represents a mental model of real world objects and interactions in terms of ’nouns’ and ’verbs’
- Software componentry accepts that the definitions of components build from existing software, unlike objects, can be counter-intuitive (but useful!)

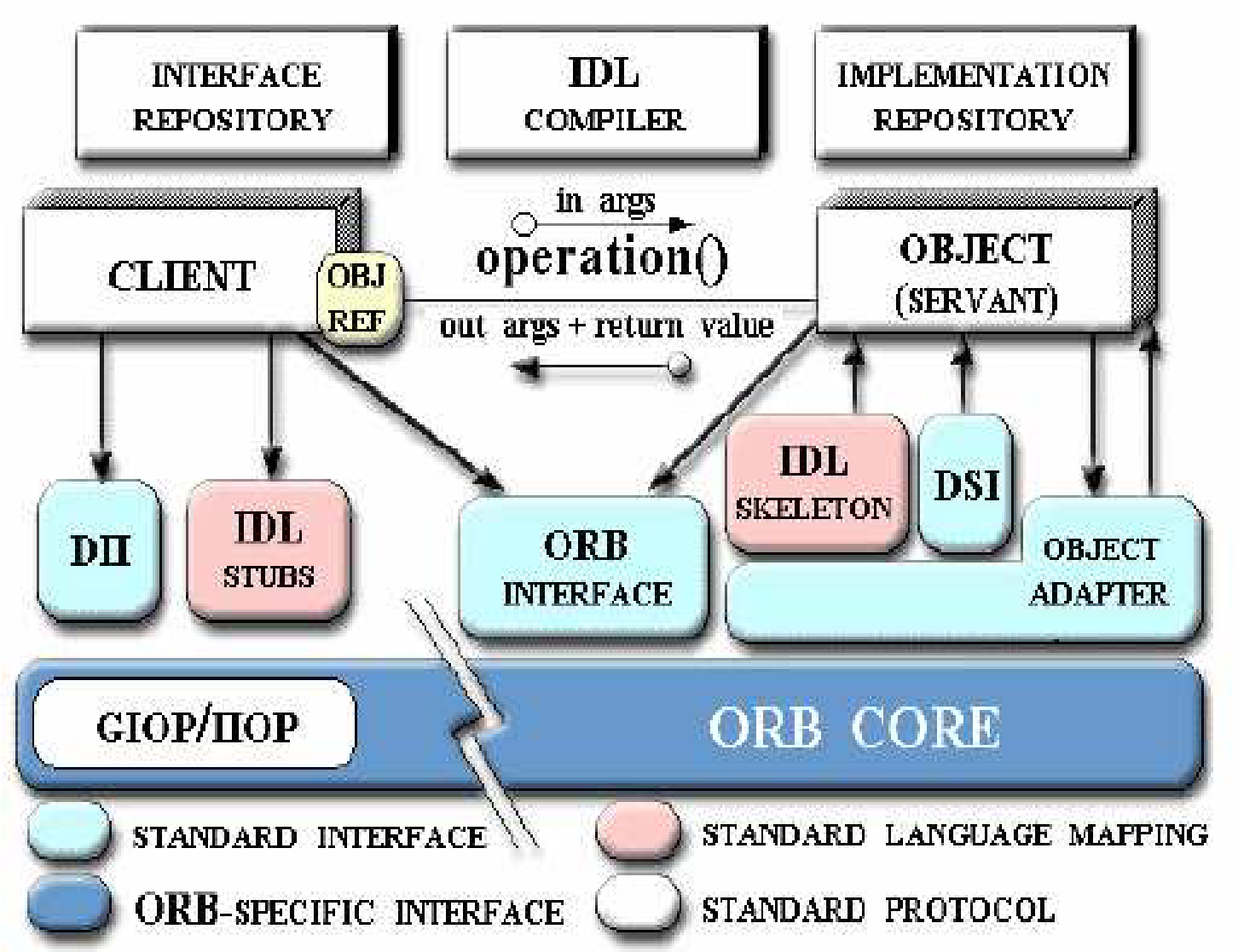
Software Component Architectures

Common Object Request Broker Architecture(CORBA)

- Distributed object specification by the Object Management Group (OMG).
- Own communication protocol (IIOP)
- Special languages for defining interfaces, (IDL).
- Implementations available in many programming languages.

```
interface Perf
{
 typedef sequence<double> doublearr;
 doublearr send (in doublearr a);
};
```

Architecture of CORBA

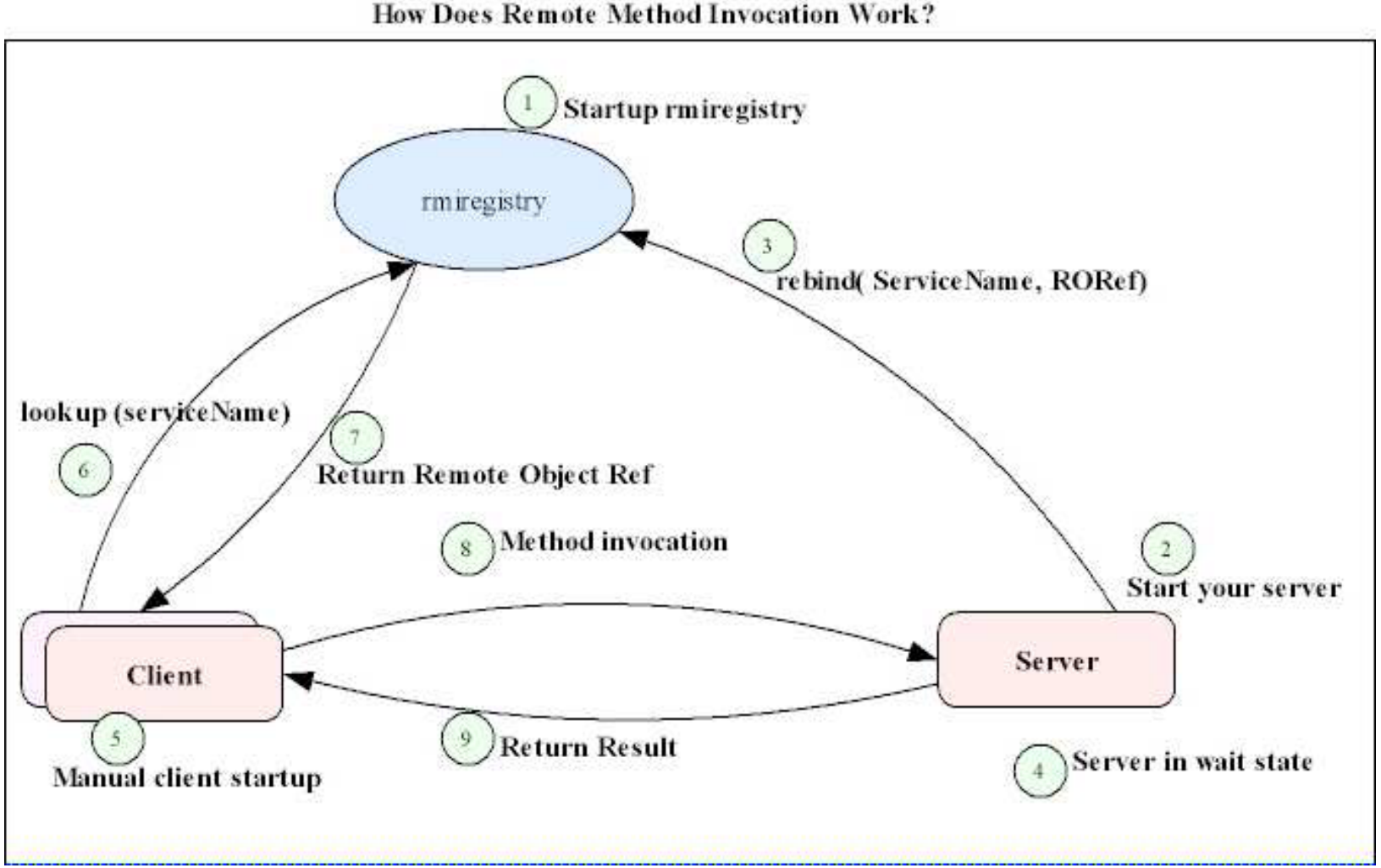


Java RMI

- Part of the standard Java SDK by Sun.
- Uses Java interfaces as definition language of the remote interfaces => limited to be used by Java applications only
- Available since JDK 1.02
- Similar to CORBA in its complexity.

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface Perf extends Remote
{
  double[] send (double[] a)throws RemoteExcep-
tion;
}
```

Architecture of Java-RMI



Common Component Architecture

- A fairly new high-performance and distributed computing framework.
- Developed in December 1998 by US national energy laboratories and research laboratories from the Universities of Utah and Indiana, sponsored by the Department of Energy (DOE).
- CCA is focused on scientific computing
- Provides prebuilt components
- Defines a standard for communication and component retrieval, as well as the Scientific Interface Definition (SIDL) for defining component interfaces
- Does not enforce a specific implementation.
- The CCA forum provides a reference implementation with SIDL language bindings for C, C++, Fortran, Java and Python

Microsoft .NET

- Framework offers two ways for distributed components.
  - using SOAP as communication protocol
    - For writing web services with .NET
    - Compatible with SOAP implementations in other programming languages.
  - The so-called Remoting
    - Using a binary stream for communication.
    - Providing a complete infrastructur for distributed objects.
    - Interfaces for Remoting applications written in C# => limited to be used by programming languages ported to the .NET runtime.
- .NET will replace the formerly used DCOM on the Windows platform.
- No explicit interface declaration, but given implicitly by the implementation.

XML-RPC

- Standard for simple remote procedure calls over TCP/IP networks using HTTP as transport and XML as encoding
- No support for using distributed components
- No naming services
- Neither location nor access transparency

Further Related Technologies

- pipes and filters Unix operating system
- Component-oriented programming Visual Basic Extensions,
- OCX/ActiveX/COM and DCOM from Microsoft
- XPCOM from Mozilla Foundation
- VCL and CLX from Borland and similar free LCL library.
- Enterprise Java Beans from Sun Microsystems
- UNO from the OpenOffice.org office suite
- Eiffel programming language
- Oberon programming language
- BlackBox Compound document technologies Bonobo (a part of GNOME)
- Object linking and embedding (OLE)
- OpenDoc Fresco Business object technologies
- 9P distributed protocol developed for Plan 9, and used by Inferno and other systems.
-

- D-BUS from the freedesktop.org organization
- DCOM and later versions of COM (and COM+) from Microsoft
- DCOP from KDE
- DSOM and SOM from IBM
- Java EE from Sun
- Universal Network Objects (UNO) from OpenOffice.org

The CTL is

- first partially realised as part of the parallel FE-code ParaFep at the Institute of Structural and Numerical Mechanics in Hanover, 1995
- originally thought as an enhancement of C++
- a template (header) library like the STL $\Longrightarrow$ no further libraries or tools are needed
- an implementation of the component concept with an RMI semantic similar to CORBA or Java-RMI.
- a tool for component building from existing libraries
- suitable for High Performance computing on parallel hardware
- also usable by C or FORTRAN programs
- compared with CORBA very easy to use

```
#define CTL_Class Perf
#include CTL_ClassBegin
#define CTL_Method1\
     array<real8>,send,(const array<real8>/*a*/),1
#include CTL_ClassEnd
```

The CTL supports the concepts of processes, process groups with intra- and inter-communication, point to point communication in stream semantic as well as the following C++ features in a remote sense:

- remote libraries, a remote library is a C++ namespace
- remote class, as a C++ class
- remote template classes, as C++ class templates
- static remote methods, as static methods
- remote methods, as virtual methods
- remote construction, as class constructors
- remote functions, as global functions
- remote template functions, as global function templates
- function and operator overloading

STL types like `std::string, std::list, std::vector, std::set, std::map` can be used as arguments of remote methods and functions. Arbitrary user defined types can be used by defining their IO-operators.

Any type checking is done at compile time.

The CTL uses the C-preprocessor and template-instantiation for the code generation.

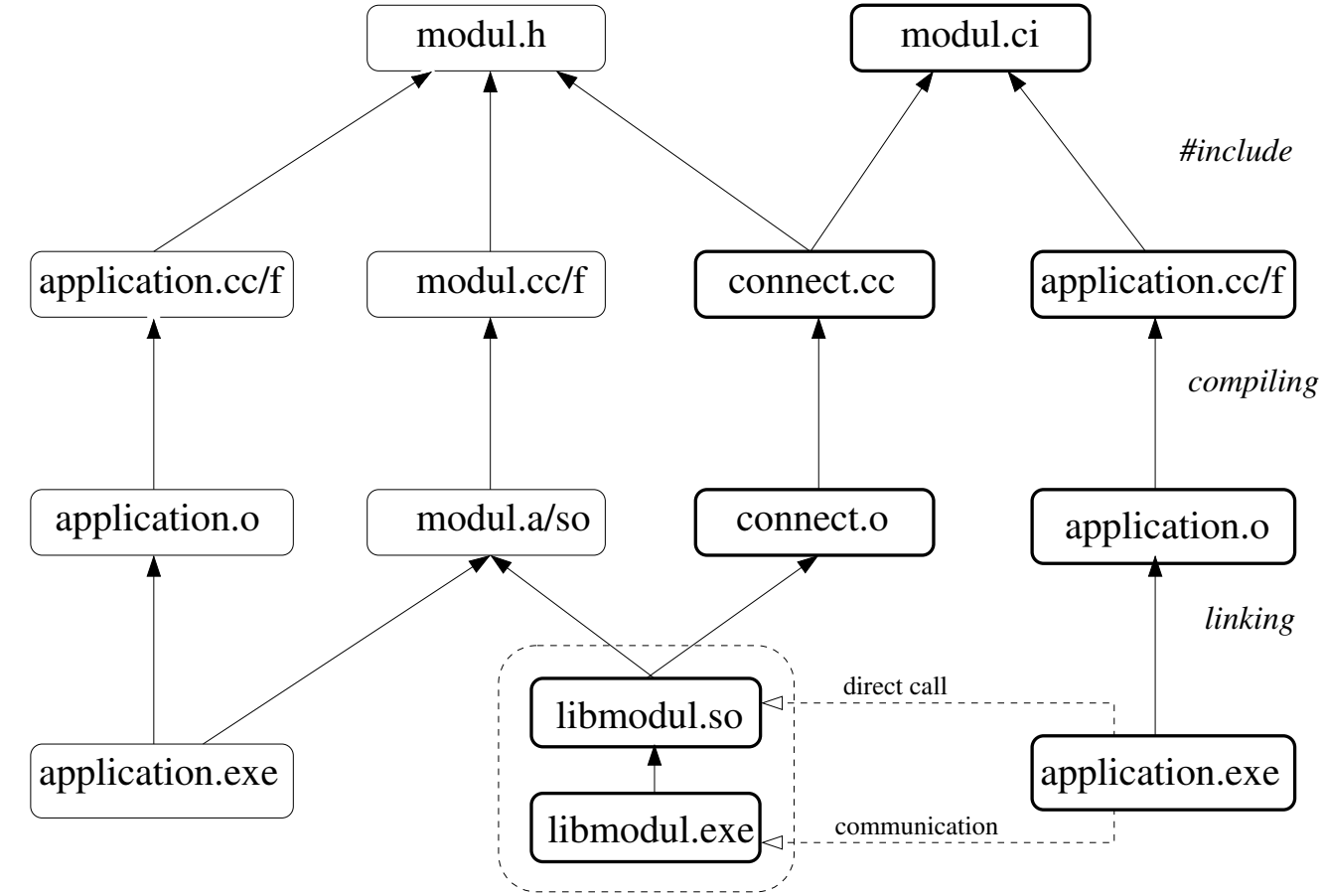
The following communication protocols/linkage types are supported:

- MPI (Message Passing Interface),
- PVM (Parallel Virtual Machine),
- TCP/IP (directly via sockets).
- dynamic linkage
- threads
- pipes (to get through a firewall via ssh)
- daemon (connect to a running process)
- file (reasonable for dump of data to disc)

Further implemented features:

- high/low endian detection, 32bit and 64bit compatibility.
- IO of polymorphic types.
- IO of cyclic data structures.
- exception handling
- automatic control of the life-time of shared remote objects by reference counting.
- automatic notification of the owner of a component (the component which created the other one) of failure/exceptions.
- automatic control of the life-time of components by ownership.
- components are convertable to python moduls (using swig)
- automatic selection of components by a resource manager

Linkage and Dependencies (Classical Linkage versus Component Linkage)



Function Call Overhead/Performance

The following software was used to compile and run the test applications:

CTL	Intel compiler	8.1
PVM	pvm3	3.4.3
MPI	LAM-MPI	6.5.1
.NET remoting	Mono	1.1.8.2
CORBA	ORBit2	2.12.2
Java RMI	Sun JDK	1.5.0 04
SOAP	gSOAP	2.7.3 38

The measurement was done on a PC-Cluster with 18 Pentium4 (2.4GHz) connected via GigaBit-Ethernet.

Function Call Overhead/Performance

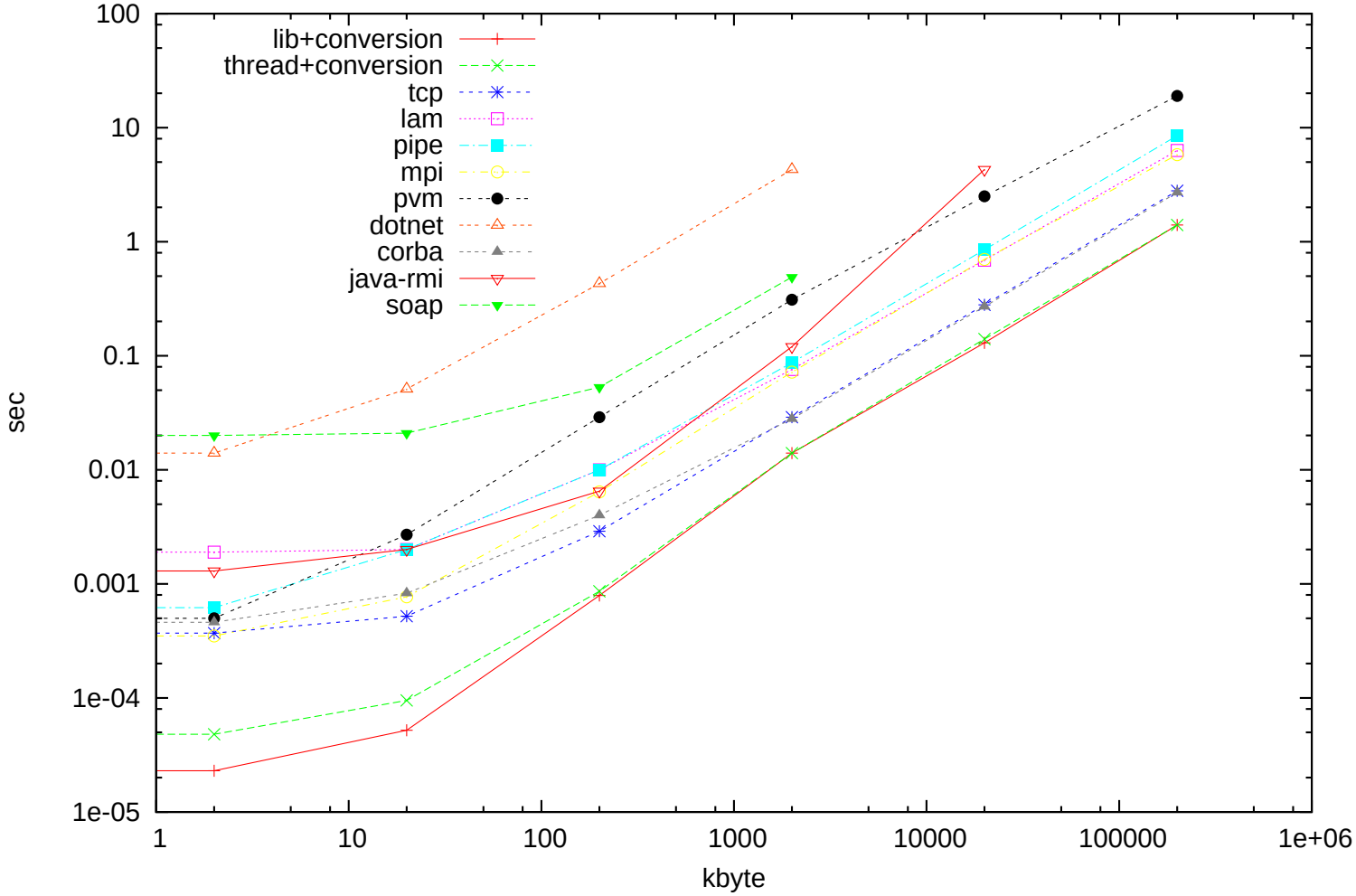
call a function with the signature:

```
array<real8> f(const array<real8>);
```

(corresponds to the pingpong test)

	0 Kbyte	2 Kbyte	20 Kbyte	200 Kbyte	2 Mbyte	20Mbyte	200 MByte
ctl/lib	$2.3 * 10^{-6}$	$2.3 * 10^{-6}$	$2.3 * 10^{-6}$	$2.3 * 10^{-6}$	$2.3 * 10^{-6}$	$2.3 * 10^{-6}$	$2.3 * 10^{-6}$
ctl/thread	$1.0 * 10^{-5}$	$1.0 * 10^{-5}$	$1.0 * 10^{-5}$	$1.0 * 10^{-5}$	$1.0 * 10^{-5}$	$1.0 * 10^{-5}$	$1.0 * 10^{-5}$
ctl/tcp	$2.5 * 10^{-4}$	$3.7 * 10^{-4}$	$5.2 * 10^{-4}$	$2.9 * 10^{-3}$	$2.9 * 10^{-2}$	$2.8 * 10^{-1}$	$2.8 * 10^{0}$
ctl/lam	$2.0 * 10^{-3}$	$1.9 * 10^{-3}$	$2.0 * 10^{-3}$	$1.0 * 10^{-2}$	$7.6 * 10^{-2}$	$6.9 * 10^{-1}$	$6.3 * 10^{0}$
ctl/pipe	$3.8 * 10^{-4}$	$6.2 * 10^{-4}$	$2.0 * 10^{-3}$	$1.0 * 10^{-2}$	$8.7 * 10^{-2}$	$8.5 * 10^{-1}$	$8.5 * 10^{1}$
ctl/mpi	$3.0 * 10^{-4}$	$3.5 * 10^{-4}$	$7.7 * 10^{-4}$	$6.4 * 10^{-3}$	$7.2 * 10^{-2}$	$7.0 * 10^{-1}$	$5.8 * 10^{0}$
ctl/pvm	$4.0 * 10^{-4}$	$5.0 * 10^{-4}$	$2.7 * 10^{-3}$	$2.9 * 10^{-2}$	$3.1 * 10^{-1}$	$2.5 * 10^{0}$	$1.9 * 10^{1}$
.net	$4.8 * 10^{-2}$	$1.4 * 10^{-2}$	$5.1 * 10^{-2}$	$4.3 * 10^{-1}$	$4.3 * 10^{0}$	none	none
corba	$1.7 * 10^{-3}$	$4.6 * 10^{-4}$	$8.3 * 10^{-4}$	$4.0 * 10^{-3}$	$2.8 * 10^{-2}$	$2.7 * 10^{-1}$	$2.7 * 10^{0}$
java rmi	$1.5 * 10^{-3}$	$1.3 * 10^{-3}$	$2.0 * 10^{-3}$	$6.5 * 10^{-3}$	$1.2 * 10^{-1}$	$4.3 * 10^{0}$	none
soap	$2.1 * 10^{-2}$	$2.0 * 10^{-2}$	$2.1 * 10^{-2}$	$5.3 * 10^{-2}$	$4.9 * 10^{-1}$	none	none

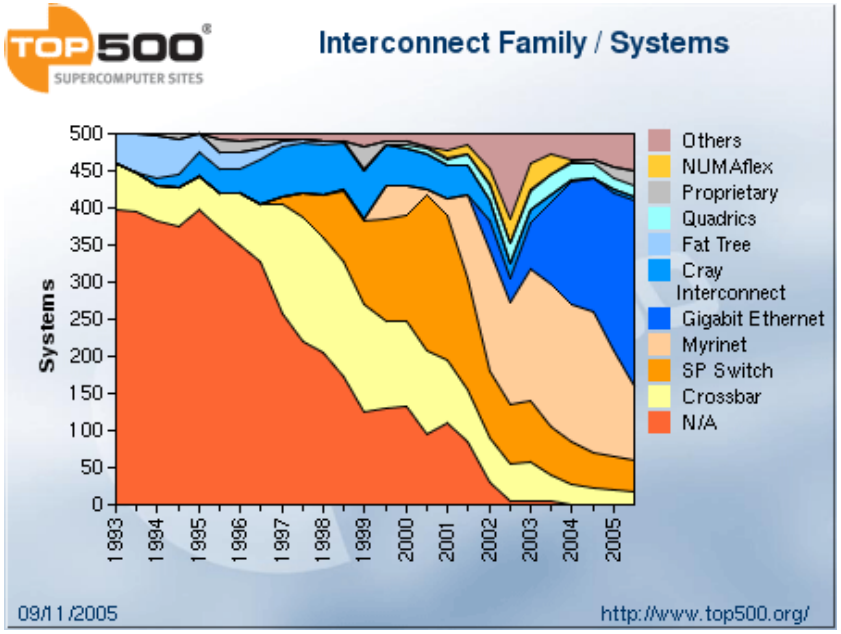
Function Call Overhead/Performance

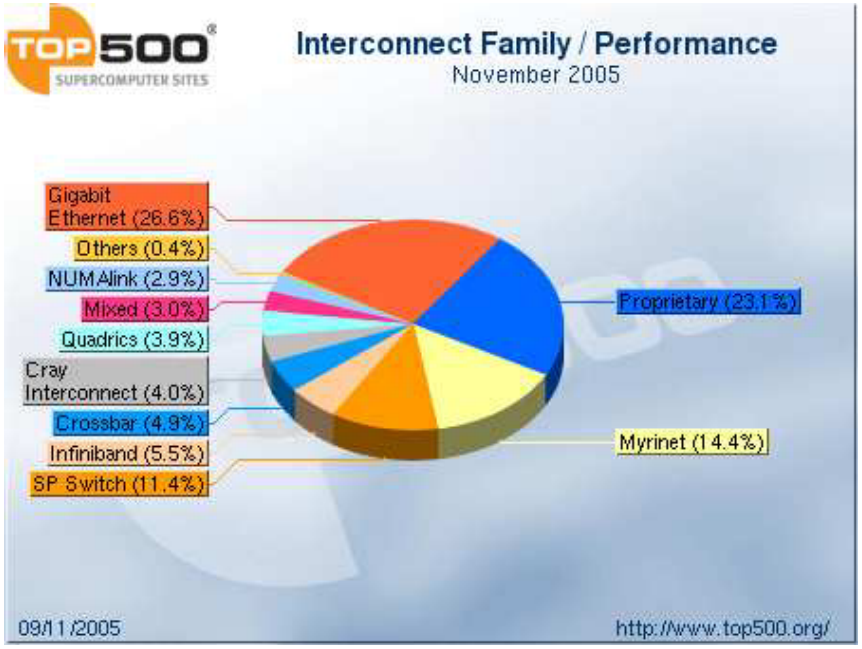


GigaBit in the Top 500

The GigaBit technology is predominantly used but has non optimal performance.

Percentage of GigaBit Nets





Percentage of GigaBit Power

Data Serialisation

If two software-components have to exchange structured data types without sharing any data type declaration an abstraction concerning binary representation is needed. The mean ideas are

- any structured type is a composition of simpler types
- there are only a few compositions
- to read a structured data only its binary representation is needed

fundamentals

The fundamental types defined in the CTL are:

the integral types

```
char = bool with values in {0,1}
char=int1, int2, int4, int8
unsigned char =uchar=uint1, uint2, uint4, uint8
```

and the float types

```
real4, real8
```

ctl::array

```
template<class T> class array;
```

serialisation as

```
os<<int8(vec.size());
for(int8 i=0; i<vec.size; i++)
  os<<vec[i];
```

examples

```
std::vector<T,Alloc>: array<T>
std::set<T, Cmp, Alloc>: array<T>
std::list<T, Alloc>: array<T>
```

ctl::empty

```
struct empty {};
oStream &operator<<(oStream &os,const empty&)
{ return os; }
```

ctl::tupel

```
template<class T0, class T1=empty,..., class Tmax=empty>
    class tupel;
T0 t0;
T1 t1;
...
Tmax tmax;
os<<t0<<t1<<...<<tmax;
```

examples

```
std::complex<T> : tupel<T,T>
std::pair<S,T> : tupel<S,T>
struct info
{
  int n;
  float x,y;
  CTL_Type(info, tupel, (n,x,y), 3)
}; : tupel<int,float,float>

map<key,val,Cmp,Alloc> : array<tupel<key,val> >
```

ctl::cstring

```
template<class T> class cstring;
while(!!str[i])
  os<<str[i++];
os<<T();
```

examples

```
char *
std::string
```

ctl::reference

```
if(!t)
  return os << true << int4(-1);
int4 log = os.getStreamId(t);
if(log>0) // t is already in the stream
  return os << true << log;
os.addReference(t);
os<<false;
const char *typeName=typeName<T>(*t);
if(!typeName)
  return os<<std::string();
os << std::string(typeName);
os << binarySize(*t) << *t;
```

examples

```
std::auto_ptr<T>: reference<T>
T*: reference<T>
```

A Simulation Interface

```
#define CTL_Class simu
#include CTL_ClassBegin

     //// must be implemented ////

// init simulation with "filename" as starter file
#define CTL_Method1 void, init, (const string /*filename*/), 1

// get parameter
#define CTL_Method2 void, getparam, (array<real8> /*p*/), 1

// set parameter
#define CTL_Method3 void, setparam, (const array<real8> /*p*/), 1

// get state variables into x
# define CTL_Method4 void, getstate, (array<real8> /*x*/) const, 1

// set state variables to x
#define CTL_Method5 void, setstate, (const array<real8> /*x*/), 1

// set load of system
#define CTL_Method6 void, setload, (const array<real8> /*load*/), 1
```

```
// get coupling indices i and values y (boundary cond. or load or. ...)
#define CTL_Method7 void, getcoupling, (array<int4> /*i*/,\
     array<real8> /*y*/) const, 2

// set coupling values y (boundary cond. or load or. ...)
#define CTL_Method8 void, setcoupling, (const array<real8> /*y*/), 1

// compute residuum at given state and write it into r
#define CTL_Method9 void, residual, (array<real8> /*r*/) const, 1

// solve with given param, load, coupling values with at least accuracy
// and set new state; write the new state into x
# define CTL_Method10 void, solve,\
     (const real8 /*accuracy*/, array<real8> /*x*/), 2

     //// might be implemented (but define at leat empty functions) ////

// compute a timestep and set new state; write new state into x_new
# define CTL_Method11 void, timestep,\
     (const real8 /*dt*/, array<real8> /*x_new*/), 2

// compute a preconditioning step on vector r and write result into pr
# define CTL_Method12 void, precond, (const array<real8> /*r*/,\
     array<real8> /*pr*/) const, 2
```

```
// compute the directional derivative in x0
     // in direction dx and write it into dr
# define CTL_Method13 void, dirderivative, (const array<real8> /*x0*/,\
     const array<real8> /*dx*/, array<real8> /*dr*/) const, 3

# include CTL_ClassEnd
```

- formulated in the interface description format of the Component Template Library CTL.
- Usable by C, Fortran and C++ codes

Requirements to a simulation code

- its functionality can be split into the functions listed in the interface above
- a method invocation has no more side-effects than those given in the comments to this method above.
- the simulation components have the information in which manner which degrees of freedom are coupled.

Algebraic solver for coupled systems

Given the global residual function:

Dirichlet	**Neumann**
recv(ζ_Γ)	send(ζ_Γ)
grid adapt(ξ, ζ_Γ)	
res=residual(ξ)	r_{Int}=residual(ζ)
send(p_Γ)	recv(p_Γ)
	res=r_{Int} + $r_{Ext}(p_\Gamma)$

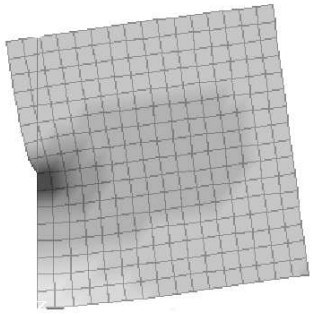
one can apply a nonlinear solver directly to the equilibrium equations like

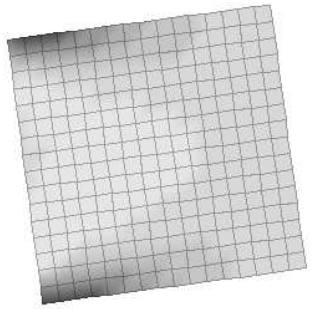
- the quasi–Newton Broyden–Fletcher–Goldfarb–Shanno (BFGS) method
- inexact–Newton method using numerical differentiation

Having also access to the local solvers one can formulate

- the Jacobi– and Gauss–Seidel iterations
- block Newton methods solving the fixpoint equation given by the Newton method

Limits of Weak Coupling Algorithms





Material StVenant:$\eta = 0.2, E = E_l$ Navier–Lame:$\eta = 0.2, E = E_r$

System might arise in the simulation of a gasket consisting of plastic ring as the left and a metallic ring as the right part.

The lower left half of the left part is fixed by a homogeneous Dirichlet condition, the right side of the right part is uniformly loaded.

$E_l = 10^5, E_r = 5 * 10^4$			$E_l = 10^5, E_r = 10^6$			$E_l = 5 * 10^4, E_r = 10^6$		
	iter	cpu[t]		iter	cpu[t]		iter	cpu[t]
Jakobi	18	10.9	Jakobi	∞	-	Jakobi	∞	-
Gauss-Seidel	14	7.9	GS	28	16.0	GS	∞	-
BFGS	56	6.8	BFGS	44	4.8	BFGS	110	13.4
Newton	3	13.5	Newton	3	13.7	Newton	12	52.0

Due to the boundary conditions the left partial structure is much more distorted and was simulated with a geometrically nonlinear formulation.

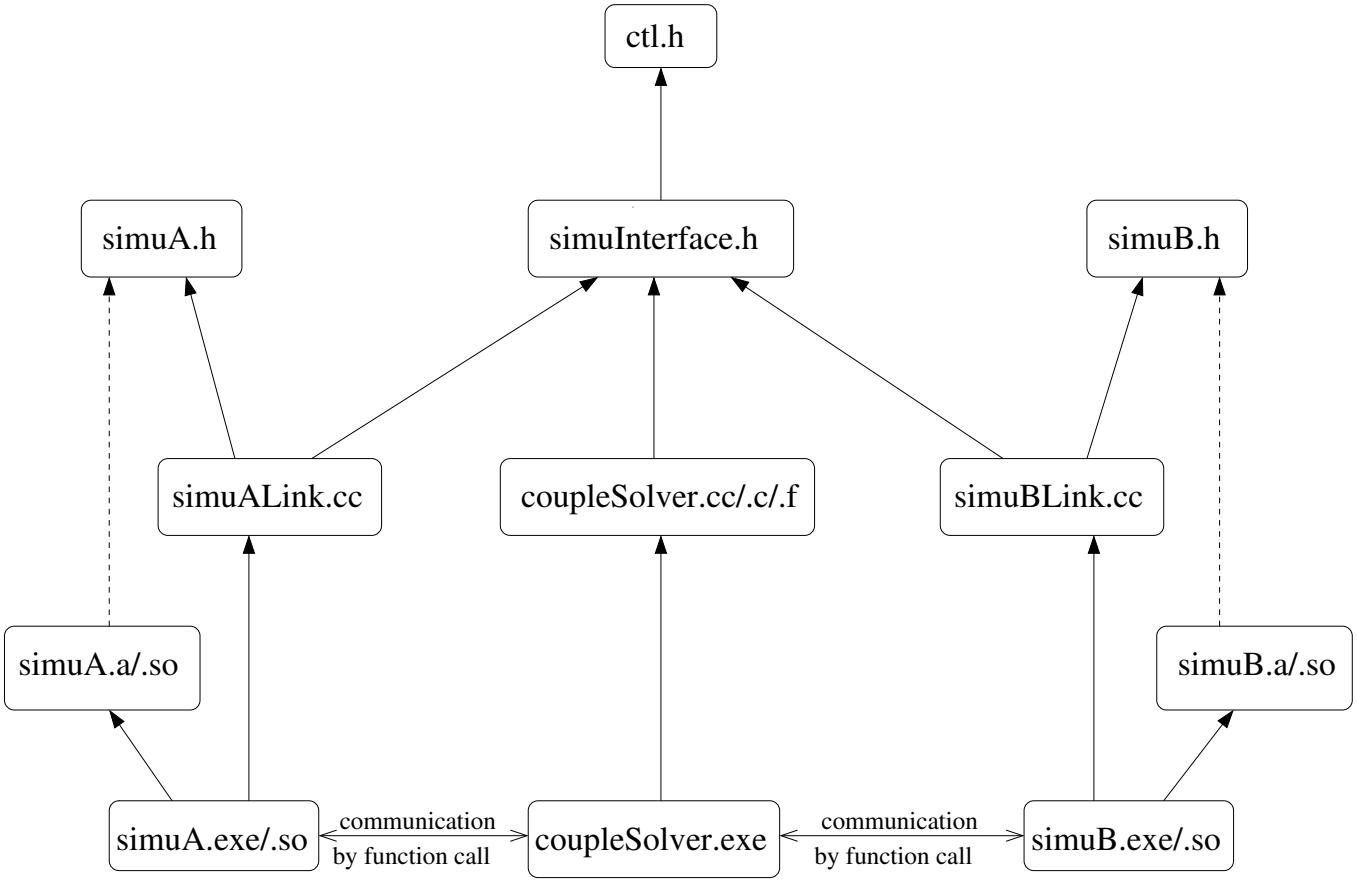
For the right part a linear formulation was chosen.

Assumptions of the Different Solvers

- The BFGS method needs only the residual function (via `residu`al) and optionally a preconditioning.
- The inexact Newton–method needs the residual function, optionally a preconditioning and also optionally the directional derivatives (via `directionalDerivation`).
- The Gauss–Seidel and Jacobi–Iterations need only the simulation internal solver (via `solve`).
- The block–Newton method needs the solver, the residual and for the linear iterative solver optionally directional derivatives of the residual

In the case that the method precond is not implemented, preconditioning is not performed. In the case the method `directionalDerivation` is not implemented, numerical differentiation is used.

Dependencies of the Sources of the Components using block Newton



Multi Physics

Offshore Wind Turbine

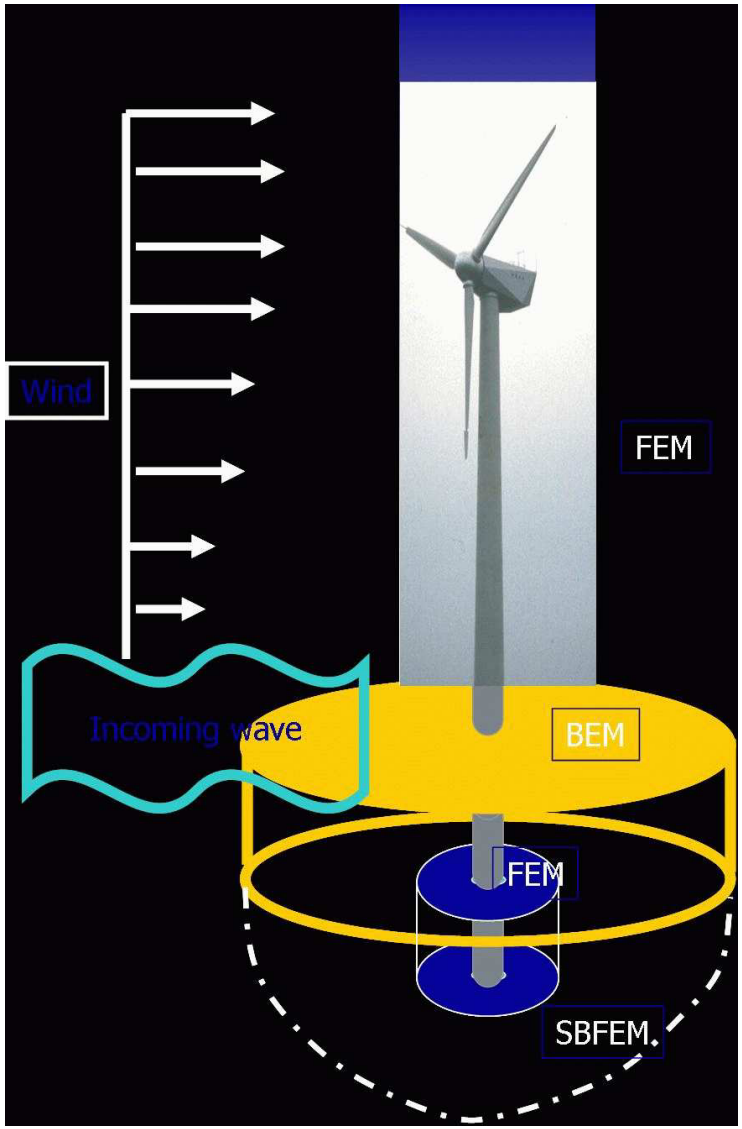
wind(WKA, MatLab)

rotor + tower(beam3D, C)

fluid with waves(fastlap, Fortran)

soil:near–field(felt, C)

soil:far–field(similar,Fortran)



Offshore Wind Turbine/Components of the system

The Multi-Physical System and Control Units

- Wind
- Water/Waves
- Structure(Tower+Blades)
- Soil
- Interaction by exchange of energy, momentum
- Control

Mathematical Models and Formulations

- Stochastic Wind
- Linear Formulation in the far field, nonlinear formulation in the near field + transition zone
- Tower as solids, blades as beams
- Linear Formulation in far field, non linear formulation of porous medium in the near field
- In each timestep continuity constraints of displacement and velocity, equilibrium of forces

The Numerical Methods

- Stochastic Wind Load
- Multipol method for the potential problem
- Finite Elements 3D for tower, beam(1D) for blades
- Scaled Finite Elements in the far field, non-linear 3D Finite Element formulation in the near field
- Coupling by boundary conditions

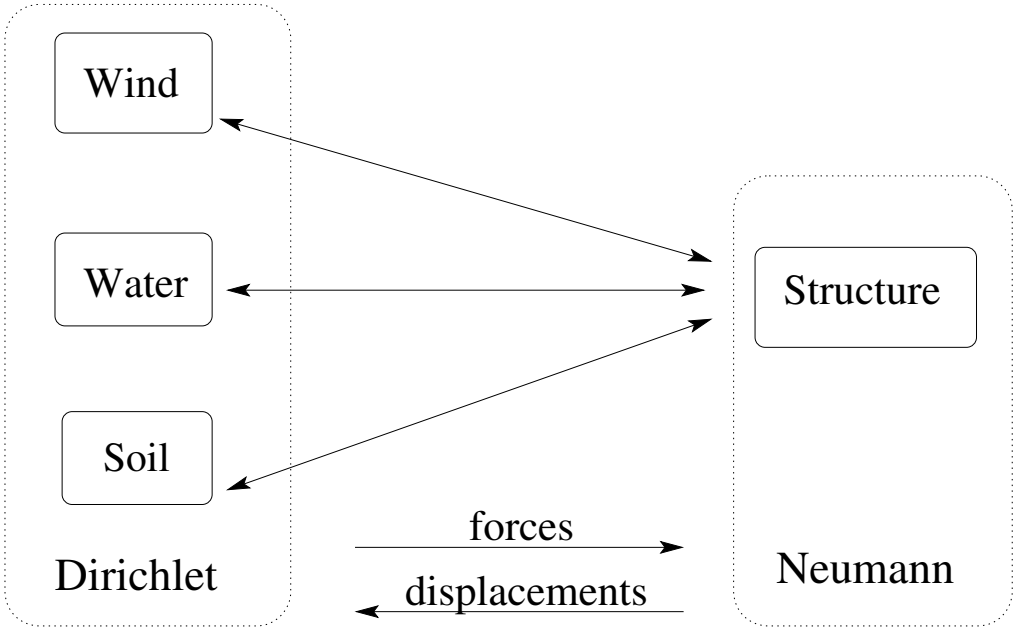
Software Components

- MatLab generates random wind load
- Multipol solver FastLap solves the potential problem
- C-Code (WKA) computes the structural response
- Far field -> SIMILIAR , near field FELT2
- Remote Method Invocation via CTL-Interfaces

Distributed Application/Hardware-Mapping

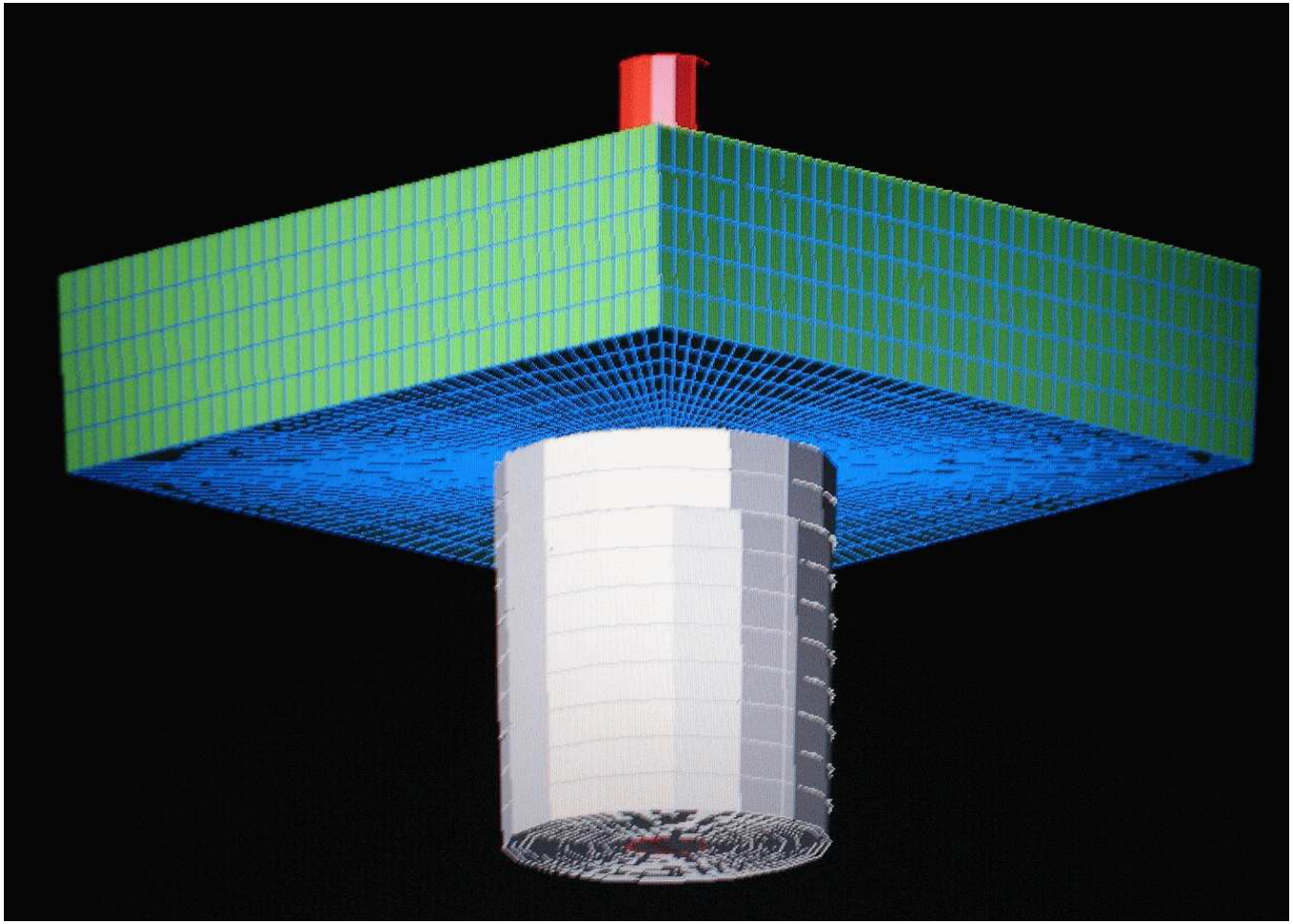
- Localhost/linux: Comp. Time <1%
- Server of wire/linux : Comp. Time ca. 40%
- Localhost/linux: Comp. Time <2%
- SGI at Institute of Applied Mechanics (TU-BS): Comp. Time ca. 55%
- Data Exchange using the TCP/IP protocol and Pipes: Comp. Time <3%

The Topology of Coupling



Compared with the parts on the left hand side high density/stiffness of the structural part ==> classical iteration schemes work

Discretisation of the Offshore Windkraft System

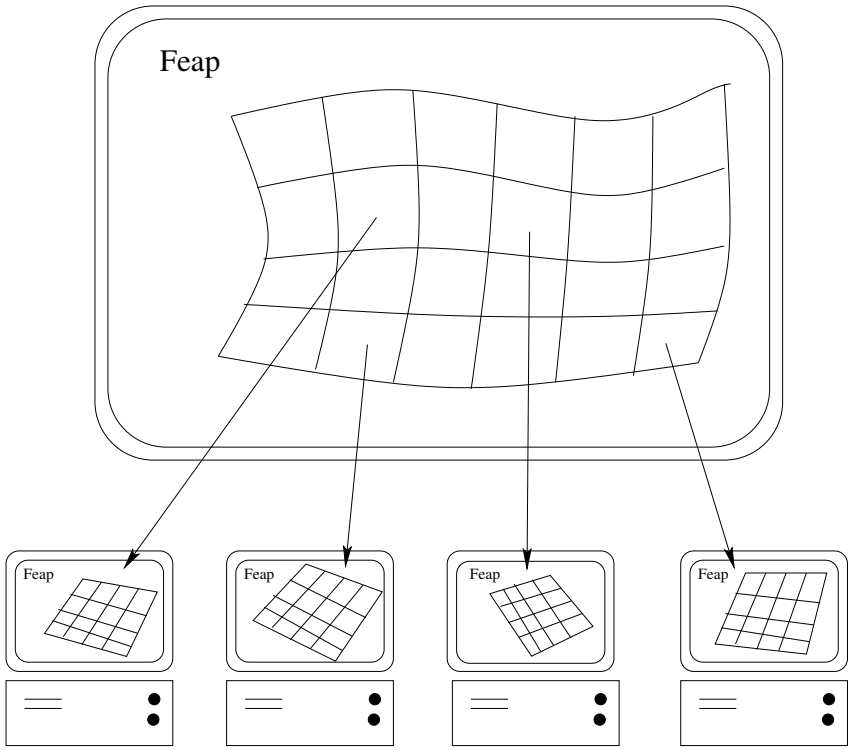


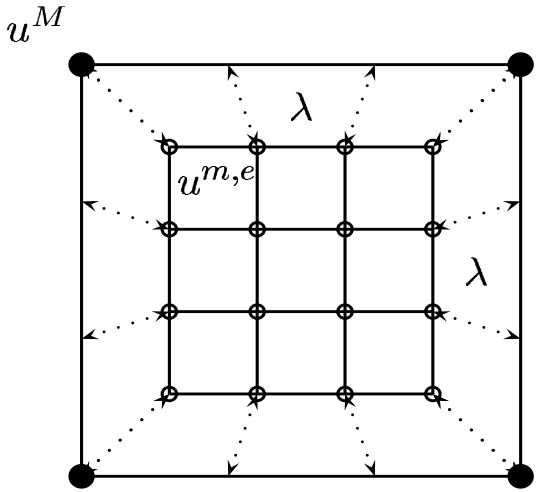
Meso–Macro coupling

used by Damijan Markovic and Adnan Ibrahimbegovic ,

at the Ecole Normale Superieure de Cachan, France.

using the Feap by Bob Taylor





coupling of the displacements using Lagrange–multipliers

High-dimensional Integration

Given a function

$$f: \ R^d \to R$$

we want to approximate the integral

$$I(f) := \int_\Omega f(\overline{x}) d\overline{x}, \ \Omega \subseteq R^d$$

by a quadrature formular

$$Q(f) := \sum_{k=1}^{n} w_k f(\overline{x_k})$$

The error is given by

$$e_Q(f) := |I(f) - Q(f)|$$

Tensor product Formulars

The tensor product of one dimensional quadrature formulars is given by:

$$Q^P(f) := (Q_1 \otimes Q_2 \otimes \cdots \otimes Q_d)(f) := \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} w_{1,i_1} \cdots w_{d,i_d} f(x_{1,i_1}, x_{2,i_2}, \cdots, x_{d,i_d})$$

with given one dimensional quadrature formulars

$$Q_i(f) := \sum_{k=1}^{n_i} w_{i,k} f(x_{i,k})$$

The Smolyak formular of level l is given by:

$$Q^S(f) := \sum_{|\overline{k}| \leq l+d-1} (\Delta_{k_1} \otimes \Delta_{k_2} \otimes \cdots \otimes \Delta_{k_d})(f)$$

using the notations

$$|\overline{k}| := \sum_{i=1}^{d} k_i,\ \overline{k} \in \{1, \cdots, l\}^d,\ \Delta_k(f) := Q_k(f) - Q_{k-1}(f)\ k = 2 \cdots l,\ \Delta_1 := Q_1$$

Monte Carlo Methods

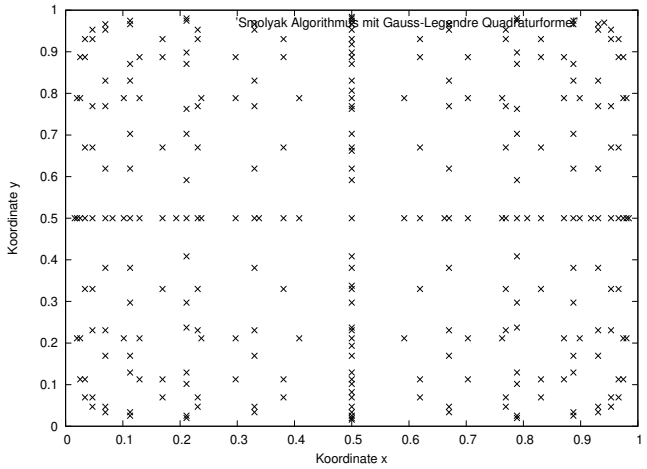
The Monte Carlo Quadrature formulars are of the form:

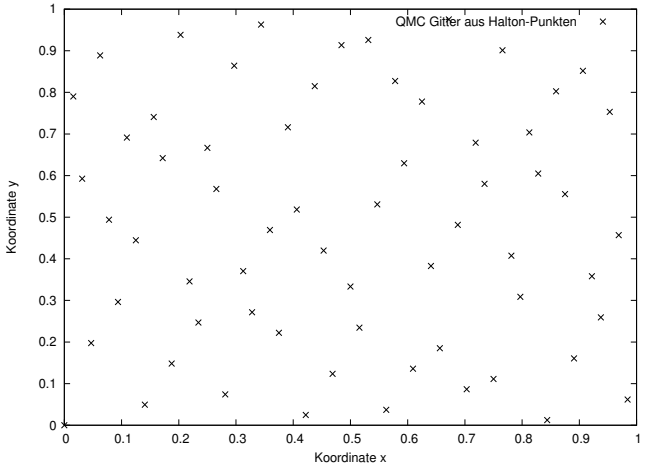
$$Q^{MC}(f) := \sum_{k=1}^{n} \frac{1}{n} f(\overline{x_k})$$

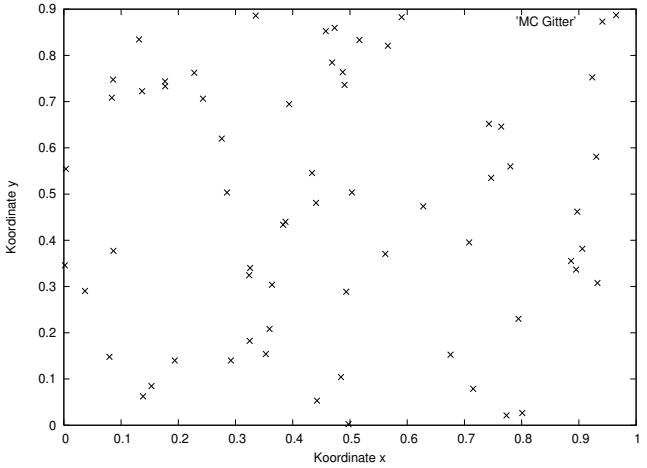
A Quasi Monte Carlo method uses a deterministic sequences $\overline{x}_k$ with low discrepance.

The Monte Carlo method uses uniform distributed (pseudo-) random numbers to generate the sequence $\overline{x}_k$.

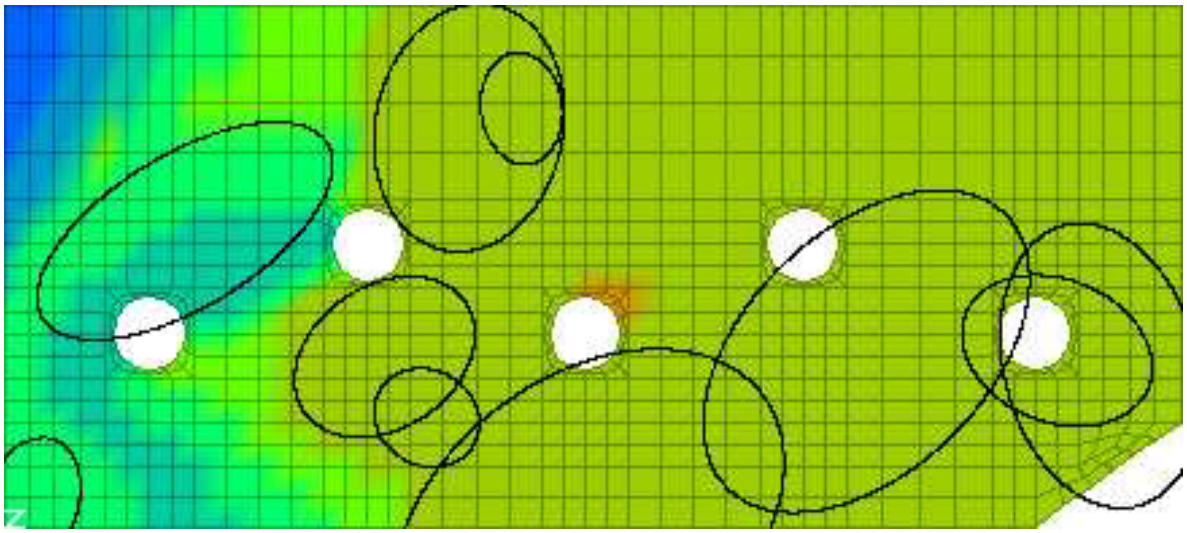
Generated Grids





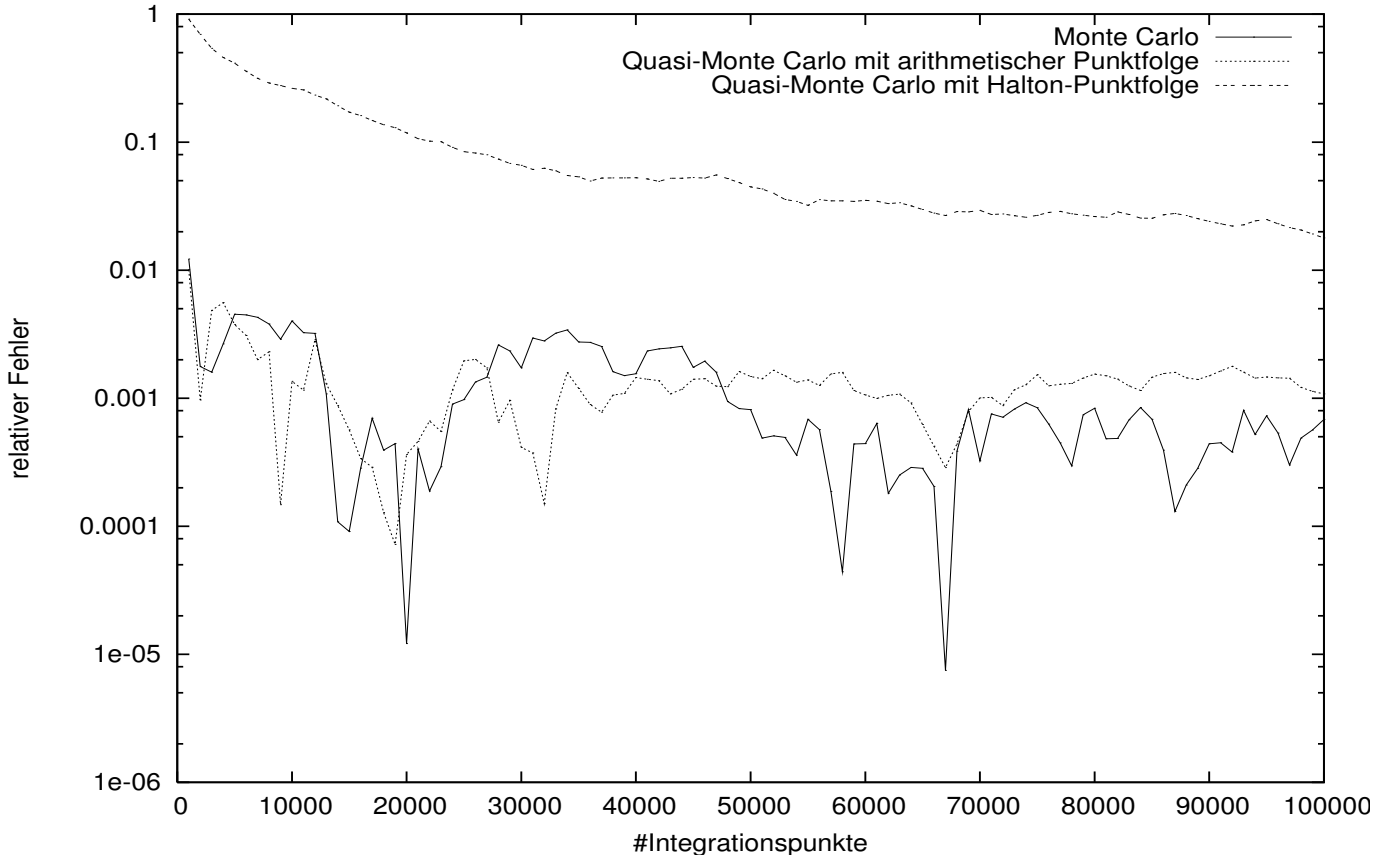


Example: Butt Plate with Elliptic Inclusions



$E_{matrix} : E_{inclusion} = 1 : 10^4$

Convergence Comparison (50 elliptic Inclusions)

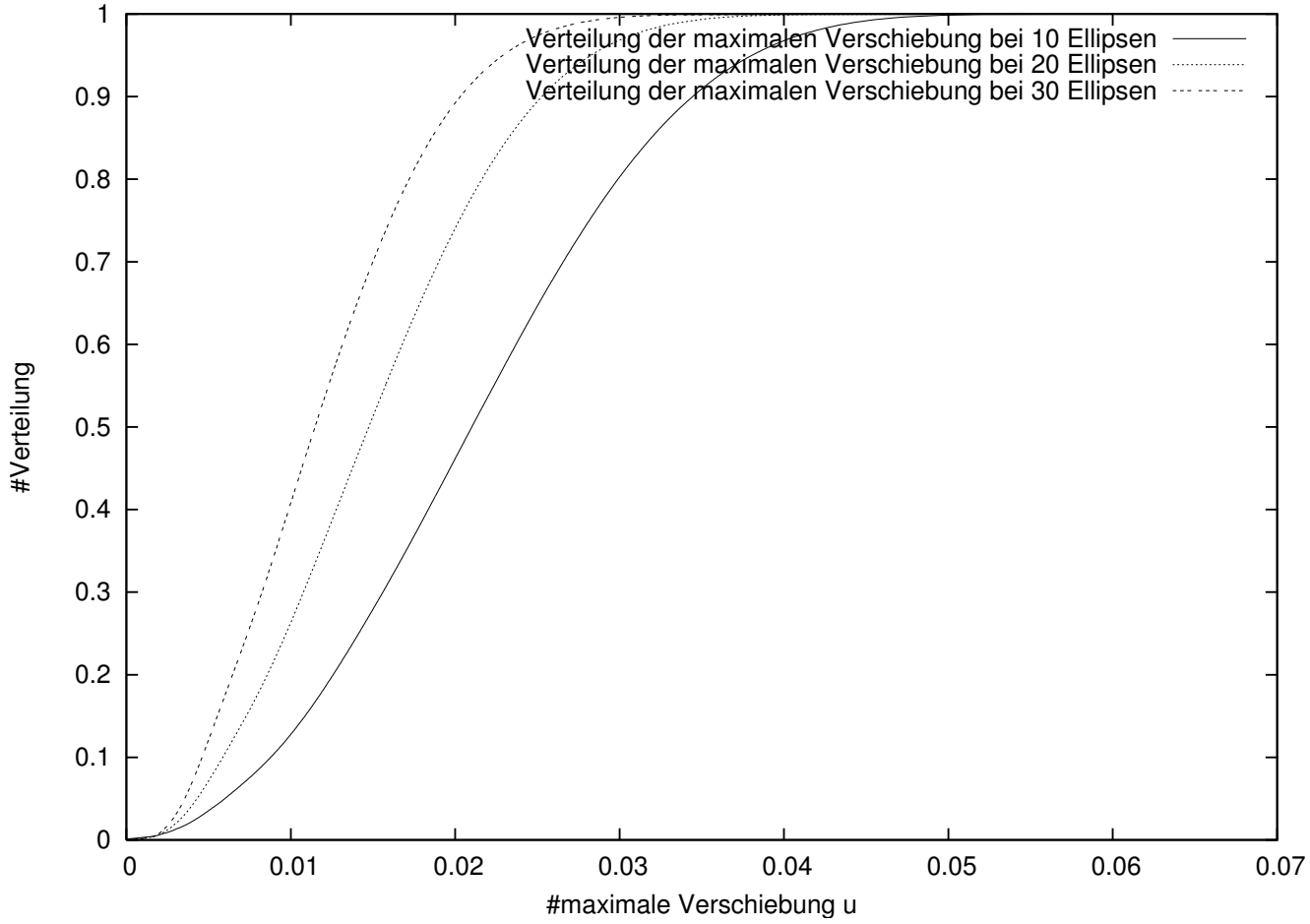


Convergence of the relative error of $expectation(\max ||u||)$

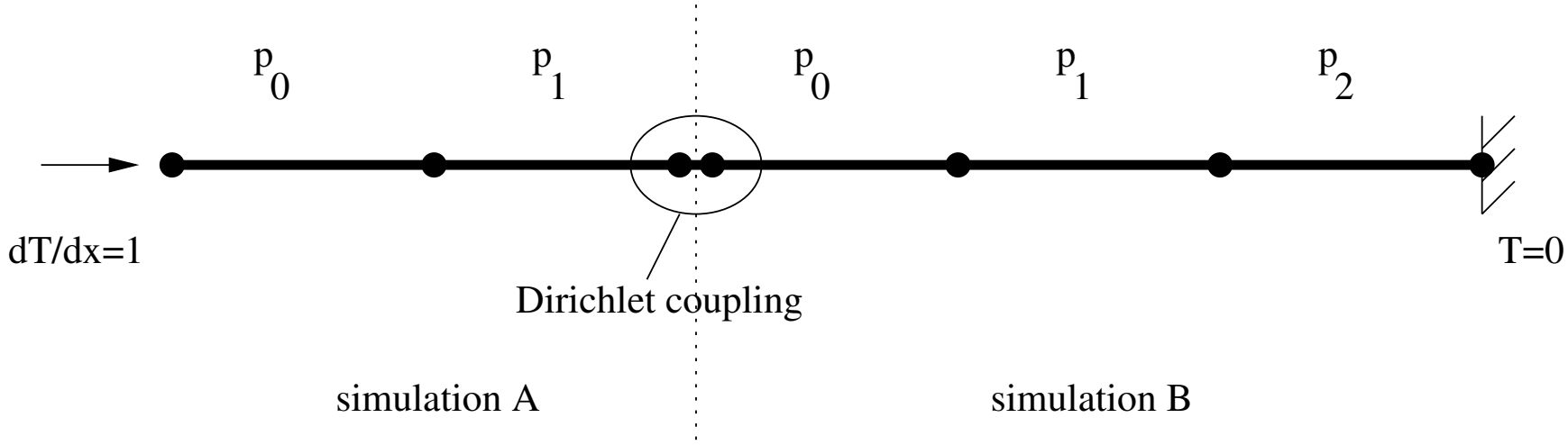
The reference solution was computed by MC using 4.400.000 evaluations.

(Smolyak integration gives no reasonable results)

Distribution of $\max \|u\|$



Demo of Simple Coupled System (see Disc)




```
######## Simulation Selection File rsc.txt #########


## simulator A for the left part
../C/simuC.so -l thread -f log/simuC.log
#../../feap/linux-gcc/libfeap.exe -l tcp -d ../../feap/run1/ -x

## simulator B for the right part
../../feap/linux-gcc/libfeap.exe -x -l tcp -d ../../feap/run2
#../fortran/simuF.exe -l tcp
#pare3:~/ctl/examples3/feap/linux-gcc/libfeap.exe -d ../run2/
```

For further information and downloads see

www.wire.tu-bs.de/forschung/projekte/ctl/e_ctl.html

Demo of Simple Coupled System (also on Disc)

```
>scp -r nocosoflume@134.169.77.134:~/ctl .
(passwd: NoCoSoFluMe270406)
>ssh nocosoflume@134.169.77.134
>cp -r ctl <individual-name>
>cd ~/<individual-name>/lib/linux-gcc
>make D
>cd ~/<individual-name>/examples3/feap/linux-gcc
>make
>cd ~/<individual-name>/examples3/simu/fortran
>make
>cd ~/<individual-name>/examples3/simu/C
>make
>cd ~/<individual-name>/examples3/simu/solver
>make
>nedit rsc.txt
...
>./solver.exe
```

References

[1] Kristopher Johnson. Remoting.CORBA homepage. http:// remoting-corba. source-forge.net/.

[2] Microsoft. Component Object Model Technologies. http://www.microsoft.com/com/.

[3] Marcus Meyer and Hermann G. Matthies: Non-linear Galerkin methods in the simulation of the aeroelastic response of wind turbines, in: K.-J. Bathe (ed.): Proc. First MIT Conf. on Computational Fluid and Solid Mechanics. Elsevier, Amsterdam, 2001.

[4] Hermann G. Matthies and Jan Steindorf: Strong Coupling Methods, in: W. L. Wendland and M. Efendiev (Eds.), Analysis and Simulation of Multifield Problems. Lecture Notes in Applied and Computational Mechanics, Vol 12. Springer-Verlag, Berlin, 2003.

[5] Markus Krosche and Rainer Niekamp and Hermann G. Matthies: A compo-nent based architecture for coupling, optimisation, and simulation software in a distributed environment, pp. 20 23 in: W. Dosch and R. Y. Lee (eds.), Proc. ACIS Fourth Int. Conf. on Software Engrng., Artificial Intelligence, 9 Networking, and Parallel/Distributed Computing (SPND 03), 16 18 October 2003, Lübeck. ACIS, 2003. ISBN 0-9700776-7-X.

[6] Markus Krosche and Rainer Niekamp and Hermann G. Matthies: PLATON A problem solving environment for computational steering of evolutionary optimisation on the GRID, in: G. Bugeda and J. A. Désidéri and J. Periaux and M. Schoenauer and G. Winter (eds.), Proc. Int. Conf. on Evolutionary Methods for Design, Optimisation, and Control with

Application to Industrial Problems (EUROGEN 2003). CIMNE, Barcelona, 2003.

[7] Damijan Markovič and Adnan Ibrahimbegović and Rainer Niekamp and Hermann G. Matthies: A multi-scale finite element model for inelas-tic behaviour of heterogeneous structures and its parallel computing implementation, in: A. Ibrahimbegović and B. Brank (eds.), Multi-physics and multi-scale computer models in non-linear analysis and optimal design of engineering structures under extreme conditions, NATO Science Series. IOS Press, Amsterdam, 2005. http://arw-bled2004.scix. net/Files/acceptedpapers/Accepted/Markovic.pdf

[8] Damijan Markovič and Rainer Niekamp and Adnan Ibrahimbegović and Her-mann G. Matthies: Multi-scale modeling of heterogeneous structures with inelastic constitutive behavior: Part I Physical and Mathematical aspects, Engrng. Computations 22 (2005) 664 683.

[9] Hermann G. Matthies and Rainer Niekamp and Jan Steindorf: Algorithms for strong coupling procedures, accepted for publication, Comp. Meth. Appl. Mech. Engrng. (2006)

[10] Tarin Srisupattarawanit and Rainer Niekamp and Hermann G. Matthies: Simulation of nonlinear random finite depth waves coupled with an elastic structure, accepted for publication, Comp. Meth. Appl. Mech. Engrng. (2006)

[11] T. Korsmeyer and K. Nabors and J. White: FastLap version 2: User s Guide and Reference Manual, MIT, U.S.A. (1996)

[12] J.I. Gobat and D.C. Atkinson: the FElt System: User s Guide and Reference Manual,

University of California, San Diego, U.S.A. (1997)

[13] J.P. Wolf and C. Song: Finite Element Modelling of Unbounded Media, John Wiley and Sons, Chichester (1996) 1