Free software for scientific computing

F. Varas

Departamento de Matemática Aplicada II
Universidad de Vigo, Spain

Sevilla Numérica
Seville, 13-17 June 2011

Acknowledgments

- to the Organizing Commmittee
  especially T. Chacón and M. Gómez
- and to the other promoters of this series of meetings
  S. Meddahi and J. Sayas

... and disclaimers

- I'm not an expert in scientific computing

  decidely I'm neither Pep Mulet nor Manolo Castro!

- My main interest is in the numerical simulation of industrial problems

  and not in numerical analysis itself

- This presentation reflects my own experience

  and can then exhibit a biased focus in some aspects

Outline

Outline

On the use of the term *Scientic Computing*

Scientific Computing will be understood in its broad sense:

A field of study concerning a number of techniques

- construction of mathematical models
- construction of numerical algorithms
- implementation of numerical algorithms
- analysis of results

applied to solve problems in Engineering and Science

Remark

This includes (and somehow emphasises) large–scale problems solved by truly multidisciplinary teams in fields like climate, astrophysics or life sciences

Outline

From algorithms to implementation. A trivial task?

A. Iserles. A First Course in the Numerical Analysis of Differential Equations, 2nd ed. Cambridge Univ. Press, 2009

There comes a point in every exposition of numerical analysis when the theme shifts from the familiar mathematical progression of definitions, theorems and proofs to the actual ways and means whereby computational algorithms are implemented. This point is sometimes accompanied by an air of angluish and perhaps disdain: we abandon the Palace of the Queen of Sciences for the lowly shop floor of a software engineer. **Nothing could be further from the truth!** *Devising an algorithm that fulfils its goal accurately, robustly and economically is an intellectual challenge equal to the best in mathematical research.*

Education in software development

G. Wilson, Scientific Software: Get More Done with Less Pain, SIAM News, June 2010

[S]cientists are almost never taught how to use computers effectively. After a generic first-year programming course, most science students are expected to figure out the rest of software engineering on their own. This is as unfair and ineffective as teaching students addition and long division, then expecting them to figure out calculus whitout any more help.

A trivial example: Gaussian elimination

A naive (but sound) implementation

Fortran code `gauss.f90` implementing Gaussian elimination with scaled row pivoting taken from

- D.R. Kincaid & E.W. Cheney. Numerical Mathematics and Computing, Fifth Edition. Brooks/Cole Publ. Co. 2003.

available at
`http://www.netlib.org/netlib/kincaid-cheney/`
`http://www.ma.utexas.edu/CNA/cheney-kincaid/`

Intel Core 2 Duo T6570 @ 2.10 GHz

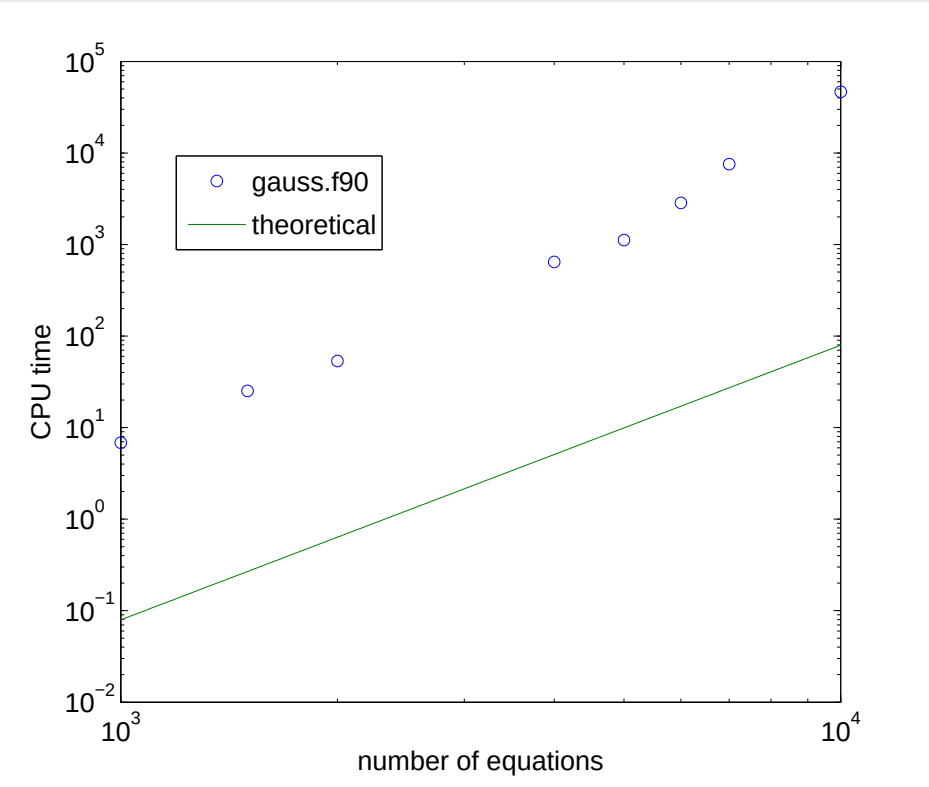
Peak FLOPs (double precision/sequential):

- 1 (core) $\times$ 2 (fp per cycle) $\times$ 2 (fp units: 1 mul + 1 add) $\times$ $2.1 \cdot 10^9$ (cycles/sec) = 8.4 GFLOPs (per core)

A trivial example: Gaussian elimination (cont.)

Computation time

Actual computation time vs theoretical computation time



A trivial example: Gaussian elimination (cont.)

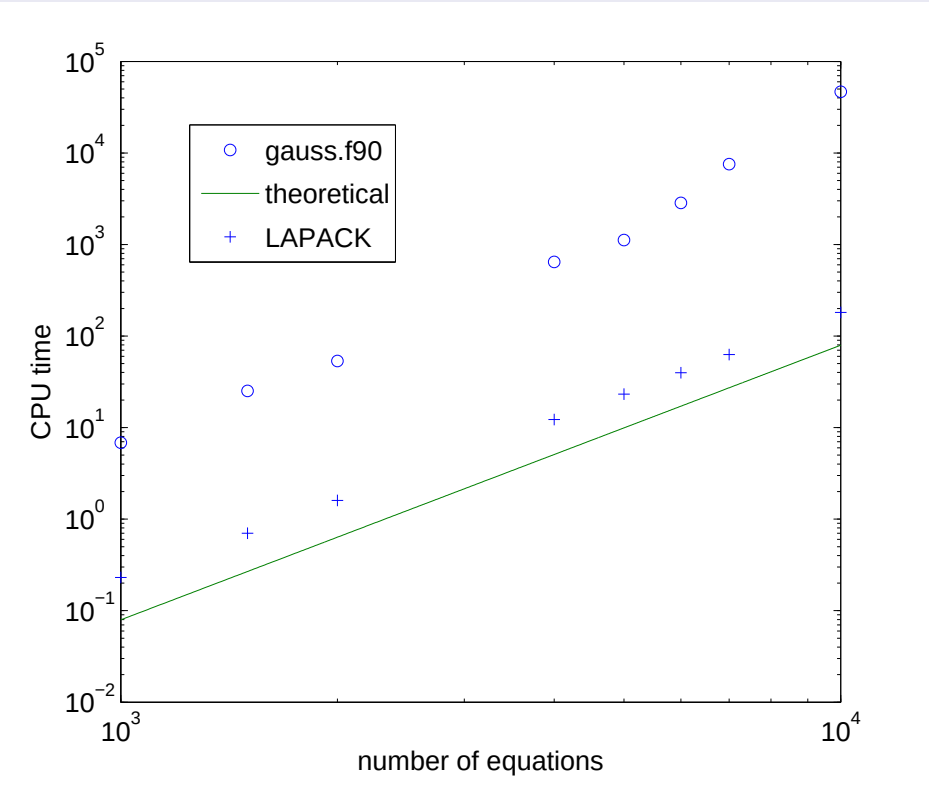
A second implementation

- Subroutine `dgesv` implementing Gaussian elimination in LAPACK available at
  `http://www.netlib.org/lapack/`
  that uses Basic Linear Algebra Subprograms (BLAS)
  `http://www.netlib.org/blas/`
  as *building blocks*.
- LAPACK libray and standard BLAS (whithout use of ATLAS) from repositories in Ubuntu will be used

A trivial example: Gaussian elimination (cont.)

Computation time

Actual computation time vs theoretical computation time



A trivial example: Gaussian elimination (cont.)

Why such a large difference?

- *Memory wall* makes instruction level parallelism crucial
- attention should be paid to some technical aspects

  cache memory size (Level1 and shared Level2), latency, bandwidth, speculative execution . . .

To know more

- Pep Mulet course *High performance linear algebra methods for PDE* in Sevilla Numérica 2011
- V. Eijkhout, E. Chow, R. van de Geijn, *Introduction to High Performance Computing*. Lulu, 2010.
- J. Doweck, *Inside Intel Core Microarchitecture and Smart Memory Acces*. Intel White Paper, 2006.

Evolution of Gaussian elimination algorithms

Architecture–related evolution of Gaussian implementations

- Introduction of vector computers with pipelining in mid-1970s:
  - attention to efficient vector operations (LINPACK and Level-1 Basic Linear Algebra Subprograms, BLAS)
  - also increasing attention to matrix-vector operations (Level-2 BLAS) to facilitate compiler in minimizing data movement
- Introduction of RISC-type microprocessors and other machines with cache memories in the late 1980s and early 1990s:
  - implementation of blocked algorithm, in terms of matrix multiplications (LAPACK and Level-3 BLAS)

Evolution of Gaussian elimination algorithms (cont.)

Architecture–related evolution of Gaussian implementation

- Increasing use of Multiple Instruction Multiple Data (MIMD) distributed-memory machines in the late 1990s:
  - extension of blocked algorithm to MIMD computers through distributed-memory versions of Level-2 and Level-3 BLAS and subprograms for communication (ScaLAPACK)

Further reading

J. Dongarra, P. Luszczek, *How Elegant Code Evolves With Hardware: The Case Of Gaussian Elimination* in *Beautiful Code Leading Programmers Explain How They Think*. O'Reilly, 2007.

Evolution of Gaussian elimination algorithms (cont.)

New architectures

- New paradigms to improve computer performance
  - Multicore and GPU (Graphics Processing Unit)
  - Cell BE (Cell Broadband Engine Architecture)
  - FPGA (Field Programable Gate Arrays), ASIC (Application-Specific Integrated Circuit)
- result in a (hard) challenge for scientific computing
  - the end of *free lunch* (associated to deeper pipelines and increases in clock speeds, ILP)
  - from weak parallelism to strong parallelism (extracting additional, fine grain parallelism from the algorithm) to avoid bus bottleneck in a purely BLAS–based parallel approach
- A. Buttari et al. *The Impact of Multicore on Math Software*, Lectures Notes in Computer Science 4699, 2007

Evolution of Gaussian elimination algorithms (cont.)

... and new algorithms

- Gaussian elimination to exploit thread level parallelism
  - tiled algorithm (block tasks broken in several small tasks)
  - dynamic scheduling
  - out–of–order execution
- implemented in
  - PLASMA (Parallel Linear Algebra for Scalable Multicore Architectures)
  - MAGMA (Matrix Algebra on GPU and Multicore Architectures)
  - FLAME (Formal Linear Algebra Methods Environment)

Evolution of Gaussian elimination algorithms (cont.)

Further reading on *new algorithms*

- A. Buttari et al. *A class of parallel tiled linear algebra algorithms for multicore architectures*, Parallel Computing 35, 2009.
- G. Quintana-Ortí et al. *Programming Matrix Algorithms–by–Blocks for Thread–Level Parallelism*, ACM Trans. Math. Software 36(3), 2009.

Outline

Quality of Scientific Computing Software

Key aspects concerning quality of Scientific Computing Software

- Efficiency
- Correctness
- Flexibility and reusability
- Maintainability and portability

A good guide

S. Oliveira and D. Stewart, *Writing Scientific Software. A Guide to Good Style*, SIAM, 2006.

Efficiency of Scientific Computing Software

Efficiency of Scientific Computing Software

- Indeed a core concern in scientific computing!
- **partly** determined by properties of involved numerical methods **but** strongly related to implementation
- requires a good knowledge of computer science:
  - at a sequential level: data locality, speculative execution, caches, bandwidth, pipelines....
  - at a parallel level: multithreading, load balance, communications...
  - awareness of emerging architectures and languages
- and some empirical self–adaption/tuning

Correctness of Scientific Computing Software

Correctness of Scientific Computing Software

- Scientific Software requires careful and extensive validation
- **but** validation of Scientific Software is difficult!
  - unknown solutions (in *real* problems)
  - many sources of approximation
  - error propagation

Flexibility and reusability of Scientific Software

Flexibility and reusability of Scientific Computing Software

- Scientific computing is algorithm–dominated:
  - algorithm is the natural unit of reuse **but** complex data structures can make it difficult
- Abstraction, modular design and use of libraries
- Software engineering/programming techniques:
  - generic programming
    - encapsulation of data structures with abstract interfaces (templates)
  - software component technology
    - create applications by composing components together in *plug and play* mode

Software Engineering techniques

Generic programming

- allows development of classes/functions without specifying data types during implementation
- avoids maintenance difficulties and performance penalties in OOP approach

Some examples of reusing/developing

- F. Cirak, J.C. Cummings, *Generic programming techniques for parallelizing and extending procedural finite element programs*, Engineering with Computers 24, 2008.
- P. Bastian et al. *A generic grid interface for parallel and adaptive scientific computing. I: abstract framework, II: implementation and tests in DUNE*, Computing 82, 2008.

Software Engineering techniques (cont.)

Component–Based Software Engineering

- a means of addressing software complexity
- units of programming functionality that can be composed to build an application
- Common Component Architecture project based on components, ports and framework: `http://www.cca-forum.org/`

Introductory reading material

- D.E. Bernholdt et al. *Managing Complexity in Modern High End Computing Through Component-Based Software Engineering*. Proc. of the First Workshop on Productivity and Performance in High–End Compunting, 2004.

Maintainability, portability and other issues

Maintainability, portability and software engineering issues

- Aspects not always properly emphasized in Scientific Computing
  - Difficulties with legay code:
    `HARRY READ ME.txt` case in the Climatic Research Unit email controversy is an (extreme) example
  - The risk of *adandonware*:
    research code thrown away after a grad student's move

Maintainability, portability and other issues

Maintainability, portability and software engineering issues

- Software Engineering methodologies not commonly used in Scientific Computing
  - direct use of these methodologies is **not** always possible: Scientific Computing is (often) explorative and (for instance) specifications of software requirements are difficult/impossible
  - scientific software often start small and only grow large with time (as it proves its usefulness): the need for Software Engineering arrives too late
- J. Tang, *Developing Scientific Software: Current Processes and Future Directions*, McMaster University, 2008.

Outline

Current practices in Scientific Computing Software development

A recent survey on current practices

- Extensive survey with almost 2000 responses (from 40 countries) carried out in October–December 2008
- J.E. Hannay, C. MacLeod, J. Singer, H.P. Langtangen, D. Pfahl, G. Wilson, *How do scientists develop and use scientific software?* Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering.

Current practices

Main conclusions from a recent survey

- knowledge required to develop and use scientific software is primarily acquired from peers and through self–study, rather than from formal education and training
- number of scientists using supercomputers is small compared to the number using desktop or intermediate computers
- most scientist rely primarily on software with a large user base

Current practices

Main conclusions from a recent survey (cont.)

- while the scientists believe that software testing is important, a smaller number believe they have sufficient understanding about testing concepts
- there is a tendency for scientists to rank standard software engineering concepts higher if they work in large software development projects and teams

Current challenges in Scientific Computing Software development

A roadmap activity in the UK

- effort to identify challenges and barriers in the development of HPC algorithms and software
- following similar experiences in US and France

Sources

- A. Trefethen, N. Higham, I. Duff, P. Coveney, *Developing a high-performance computing/numerical analysis roadmap*, International Journal of High Performance Computing Applications, 23(4) 2009.
- Full report at `http://www.oerc.ox.ac.uk/research/hpc-na`

Current challenges

Main conclusions from the roadmap activity

- Concerning software
  - Efficiency and performance
  - Validation of software and models
  - Software engineering
  - Portability and reusability
  - Language issues
- Concerning developers community

Current challenges

Efficiency and performance

- Ability to manage locality and data movement and to schedule for latency hiding (in the context of new and heterogeneous architectures)
- H. Sutter, *The Free Lunch is Over. A Fundamental Turn Toward Concurrency in Software* Dr. Dobb's Journal 30(3) 2005.

Current challenges

Validation of software and models

- There are no well-defined methods and techniques for validating scientific software and the underlying models
- D.E. Post, L.G. Votta, *Computational Science Demands a New Paradigm* Physics Today 58(1) 2005.

Software engineering

- Guidance on best practices for software development would be a step to assist the community
- G. Wilson, *Scientific Software: Get More Done with Less Pain*, SIAM News, June 22, 2010.

Current challenges

Portability and reusability

- More effective code reuse could be achieve by supporting software library development and framework for reuse.
- To be able to move from one platform to another it would be beneficial to have underlying libraries that *do the right thing* for any given platform. This is becoming increasingly important with the plethora of new architectures that need to be considered.
- Higher–level abstractions should allow applications developers an easier development environment. The provision of efficient, portable, *plug-and-play* libraries would also simplify the application developer's tasks.

Current challenges

Portability and reusability (cont.)

- T. Epperly et al. *Component Technology for High–Performance Scientific Simulation Software*, Proc. of the IFIP TC2/WG2.5 Working Conference on the Architecture of Scientific Software, 2000.
- R. Armstrong, *High–Performance Scientific Component Research: Accomplishments and Future Directions*, White paper SciDAC Center for Component Technology for Terascale Simulation Software, 2005.
- Active projects at the Center for Applied Scientific Computing (Lawrence Livermore National Laboratory) `https://computation.llnl.gov/casc`

Current challenges

Language issues

- Need for mixed–language support as a variety of languages are used for application development.
- BABEL: a High Performance Language Interoperability Tool. Center for Applied Scientific Computing, Lawrence Livermore National Laboratory. `https://computation.llnl.gov/casc/components`

Current challenges

concerning (Scientific Computing Software) developers community:

- Cultural issues around sharing:
  - Some application domain scientists are used to sharing models and codes and reusing other people's software. For other domains this approach is completely alien.
- Lack of awareness of existing libraries:
  - Patchy awareness of what is already available.
- Skills and training:
  - Insufficient students being trained with the required skills in mathematics, software engineering and high-performance computing.

The way forward in Scientific Computing Software development

The way forward

- Scientific Computing Software developers should gain a good knowledge on
  - programming
  - computer science
  - software engineering
  - high–performance computing
  - emerging architectures
  - ...

  to avoid software performance falling well behind hardware capabilities and to produce a reliable and reusable scientific software.

The way forward

A good starting point (in this long travel)

Course materials from G. Wilson and others (freely available) at: `http://software-carpentry.org/`

But ...

- This will put heavy stress on developers!
- (Non computer) scientists after all wouldn't want to spend a lot of time on software issues that do not directly and **visibly** contribute to their doing science

The way forward

Two final advices

- Vision from *Developing a HPC/numerical analysis roadmap* identified the Grand Challenge to provide :
  - Algorithms and software that applications developers can reuse in the form of high–quality, high–performance, sustained software components, libraries and modules
  - A community environment that allows the sharing of software, communication of interdisciplinary knowledge and the development of appropriate skills
- Encouragement of scientific software sharing by the Climate Code Foundation
  - N. Barnes, *Publish your computer code: it is good enough*, Nature 467(753) 2010.

Outline

Outline

A definition of free software

A rough definition

Software that

- can be used, studied, and modified without restriction, and
- can be copied and redistributed in modified or unmodified form either without restriction, or with minimal restrictions only to ensure that further recipients can also do these things.

On software ownership

M. Puckette, *Who owns our software? A first–persone case study*, Proceedings of ISEA 2004.

Free Software Foundation

http://www.fsf.org

Every (free) software user must have four freedoms

0. The freedom to run the program for any purpose.
1. The freedom to study and modify the program.
2. The freedom to copy the program so you can help your neighbor.
3. The freedom to improve the program, and release your improvements to the public, so that the whole community benefits.

Remark

Access to source code is a precondition for freedoms 1 and 3

Open Source Initiative

http://www.opensource.org

The Open Source Definition

- Free Redistribution

  The license shall not restrict any party from selling or giving away the software. The license shall not require a royalty or other fee for such sale.

- Source code

  The program must include source code, and must allow distribution in source code as well as compiled form.

- Derived works

  The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

More on free software definition

Free software and open source software

Open source is a development methodology; free software is a social movement. (R. Stallman)

What free software is **not**: some kinds of *disguised* proprietary software

- freeware (software in binary form available at no cost)
  - free software is a matter of liberty not price
- shareware (*try-before-you-buy* software)
  - temporal/functional limitations
- software available for restricted use
  - ex. educational or non-profit research

Some interesting reading materials

Books on Free/Open Source Software

- R.M. Stallman, *Free Software, Free Society* 2nd ed., Free Software Foundation, 2010.
- S. Williams, *Free as in Freedom (2.0): Richard Stallman and the Free Software Revolution*, Free Software Foundation, 2010.
- J. Feller, B. Fitzgerald, S.A. Hissan, K.R. Lakhani (Eds), *Perspectives on Free and Open Source Software*, The MIT Press, 2005.
- L. Lessig, *The Future of Ideas. The fate of the commons in a connected world*, Random House, 2001.

(Most of) Free software is copyrighted

Why?

- To clarify granted rights, restrictions and disclaimers.
- Restrictions are intended
  - to ensure author recognition
  - to preserve future users rights

How?

Through software licences under wich software is released

Remark

Software with sorce code in the public domain is an special case (but under the Berne Convention, every written work is automatically copyrighted)

Free software licences

Available free software licences

- Many available licences (*licence proliferation*)
- Possible compatibility problems (merging different programs)
- Lists of FSF/OSI approved licences

Permissive versus Copyleft licences

- Copyleft licences

  any derivative work must be distributed under the same terms (viral feature)
- Permissive licences

  no restriction on licences for derivative works (merging into proprietary software allowed)

Most used free software licences

GNU General Public License (GPL)

Most widely used copyleft licence

BSD licenses (Berkeley Software Distribution)

Family of permissive licenses (based on BSD license)

GNU Less General Public License (LGPL)

Permissive version of the GPL intended for libraries

Multiple licensing

An abuse of language

Use of **Free software** instead of **Software released under a free software licence**

- free/non-free is NOT an attribute of a piece of software
- free/non-free is an attribute of the software licence

Dual licensing

- Software can be released under multiple licences.
- For instance, the authors of a program decide:
  - release the software under GPL
  - sell licences to include it in a proprietary software

More on free software licensing

Some interesting reading materials

- A.M. St Laurent, *Understanding Open Source and Free Software Licensing*, O'Reilly, 2004.
- S.L. Chen, *Free/Open Source Software Licensing*, UN Development Programme, 2006.

and some links

- Free Software Foundation on Free Software Licensing:

  `http://www.fsf.org/licensing`
- *A Legal Issues Primer for Open Source and Free Software Projects* by the Software Freedom Law Center:

  `http://www.softwarefreedom.org/...`
  `resources/2008/foss-primer.html`

Outline

Developing a scientific computing programme

Developing a scientific computing programme

Three main phases:

1. Model development
2. Numerical methods/algorithms development
3. Software development

with extensive validation (at each phase)

A common practice in scientific computing

Modelling and numerical methods (phases 1 and 2)

- Step 1: State-of-the-art review (through available literature)
- Step 2: Adaption/extension of previous developments
- Step 3: Communication/dissemination of original results (encouraging other researchers to use them!)

Software (phase 3)

- Step 1: Code development from scratch (apart from some home–made pieces of software and low–level subroutines)

A common practice in scientific computing (cont.)

Some flaws of this model

- Development of efficient and reliable software is difficult and highly time–consuming
- Exploiting modern arquitectures capabilities is challenging
- Extensive validation could not be achievable
- Real risk of *abandonware* exists (unless attention is paid to software engineering issues)
- Development of in–house software is not a recognized merit in academia

An immediate, undesired consequence

A practice deterring (large scale) challenging scientific computing projects

An alternative practice in scientific computing

Software development (imitating phases 1 and 2)

- Step 1: State of the art review (through available free software)
- Step 2: Adaption/extension of existing (free) software
- Step 3: Release of derivative works as free software (encouraging other researchers to use them!)

Remark

A (relatively) large body of free software must exist

An alternative practice in scientific computing (cont.)

What would this practice bring about?

An approach meeting the *Great Challenges* (UK HPC/NA roadmap)

- Algorithms and software that applications developers can reuse in the form of high–quality, high–performance, sustained software components, libraries and modules
- A community environment that allows the sharing of software, communication of interdisciplinary knowledge and the development of appropriate skills

and likely contributing (more than HPC does) to make computing a fundamental tool of Science

- G. Wilson, *Those Who Will Not Learn From History...*, Computing in Science & Engineering. May/June 2008.

Free software in Scientific Computing

Anytime and everywhere

- Since the very early times (and before!)

  Ex. LINPACK/EISPACK
- Absolutely everywhere

  Ex. LAPACK and LaTeX

Many kinds of contributors

- individuals
- universities
- research centres
- companies

France is a very good example of this!

Free software in Scientific Computing (cont.)

Free software development models

- the cathedral model
  - source code is available with each software release, but code developed between releases is restricted to a small group of developers
- the bazaar model
  - code is openly developed (over the Internet) by a large number of developers (*given enough eyeballs, all bugs are shallow*)

Further reading

E.S. Raymond, *The Cathedral and the Bazaar*, O'Reilly, 1999.

Some interesting links

Information on scientific computing software

- `http://www.netlib.org`

  a repository of scientific computing software
- `http://www.ann.jussieu.fr/free.htm`

  a (comprehensive) list by A. Le Hyaric, Univ. Paris VI
- `http://www.netlib.org/utk/...`
  `people/JackDongarra/la-sw.html`

  a list of Linear Algebra software by J. Dongarra, UTK

Outline

Review of free software in Scientific Computing

Scientific Software to be considered in this tutorial

Using a partial differential equations approach:

- Linear Algebra
- Preprocessing and postprocessing
  - Computer–Aided Design (CAD)
  - Meshing
  - Visualization
- PDE solvers
  - general solvers (multiphysics)
  - dedicated solvers (solids/fluids)

Outline

Linear Algebra software

Linear Algebra related software categories

- Support routines
- Dense matrices linear algebra
- Sparse matrices linear algebra
  - direct methods
  - iterative methods
- Eigenvalue solvers
- Graph partitioning

Linear Algebra support routines

Basic Linear Algebra Subprograms (BLAS)

- A *de facto* standard interface for basic (dense) vector and matrix operations:
  - Level 1 BLAS: scalar, vector and vector-vector operations
  - Level 2 BLAS: matrix-vector operations
  - Level 3 BLAS: matrix-matrix operations
- Efficient (machine-dependent) implementations makes higher-level linear algebra both efficient and portable

Linear Algebra support routines (cont.)

BLAS implementations

- Non–optimized reference implementation
  - `http://www.netlib.org/blas`
- Optimized (for some architectures) implementation in Goto BLAS
  - `http://www.tacc.utexas.edu/...tacc-projects/gotoblas2/`
  - developed by K. Goto and no longer under active development
- Other optimized implementations can be generated with ATLAS (see below)
- **Non–free** BLAS implementations provided by (some) computer vendors

ATLAS

http://math-atlas.sourceforge.net/

General features

- released under a BSD-style license
- developed by C. Whaley et al. Univ. of Texas at San Antonio, USA
- (empirical) Tuning system to produce a BLAS library specifically optimized (for the platform ATLAS is installed on)
- Prebuilt ATLAS libraries exist for some selected architectures/configurations
- Tend to be competitive with the machine-specific versions...

  ...and tries to be optimal for unknown machines

Basic Linear Algebra Subprograms

Other BLAS–related software

- Level 3 BLAS designed to easily represent algorithms
  - Formal Linear Algebra Methods Environment (FLAME)
    `http://www.cs.utexas.edu/users/flame/`
- Parallel version BLAS
  - Basic Linear Algebra Communication Subprograms (BLACS)
    `http://www.netlib.org/blacs/`
- Wrapper/templates to BLAS
  - to use different kinds of matrices (sparse CRS, for instance)
  - to have unified calling (C++)
  - some examples:
    `Trilinos`, uBLAS, `Armadillo`, `Blitz++` ...

Dense Linear Algebra

Dense (matrices) Linear Algebra

Name	Comments	License
LAPACK	shared memory	BSD–style
ScaLAPACK	MIMD version	BSD–style
libflame	multicore/GPU	LGPL
PLASMA	multicore	BSD–style
MAGMA	hybrid	BSD–style

LAPACK

http://www.netlib.org/lapack/

Main features

- written in Fortran 90 (originally in FORTRAN 77)
- Use block matrix operations (to seize multi-layered memory hierarchies) and BLAS
- Efficiency depends on BLAS optimization
- Includes a large number of routines (solvers, factorizations and eigenvalue computations)

Developers

Jack Dongarra and many others, Univ. of Tennessee, Univ. of California Berkeley, Univ. of Colorado Denver, and Numerical Algorithms Group (NAG), USA

ScaLAPACK

http://www.netlib.org/scalapack/

Main features

- A LAPACK version for distributed-memory message-passing MIMD computers and networks of workstations
- uses a parallel BLAS implementation (PBLAS) and BLACS (Basic Linear Algebra Communication Subprograms, an abstraction layer over MPI)
- ScaLAPACK only exploits parallelism at the BLAS level!

Developers

University of Tennessee and Oak Ridge National Laboratory (USA)

libflame

http://z.cs.utexas.edu/wiki/flame.wiki/FrontPage

Main features

- Library providing some factorizations (with GPU support)
- FLAME approach to seize abundant high-level parallelism:
  - FLAME algorithms are expressed (and coded) in terms of smaller operations on sub-partitions of the matrix operands
  - a runtime system, SuperMatrix, detects and analyzes dependencies found within FLAME algorithms-by-blocks
  - the system schedules sub-operations to independent threads of execution.
- it doesn't currently offer distributed memory parallelism!

Developers

F.G. Van Zee et al. University of Texas at Austin (USA) and Univ. Jaime I de Castellón

PLASMA

http://icl.cs.utk.edu/plasma/

Main features

- Parallel Linear Algebra for Scalable Multi–core Architectures
- Same approach as `libflame`:
  - tiled algorithms
  - asynchronous, out–of–order scheduling

Developers

J. Dongarra, J. Kurzak et al. University of Tennessee (USA)

MAGMA

http://icl.cs.utk.edu/magma/

Main features

- Matrix Algebra on GPU and Multicore Architectures
- Development of linear algebra algorithms (and frameworks) for hybrid (multicore and GPUs) systems

Developers

J. Dongarra, J. Kurzak et al. University of Tennessee (USA)

Sparse Linear Algebra: direct solvers

Some direct solvers

Name	Comments	License
MUMPS	parallel	public domain
SuperLU	parallel	BSD–style
UMFPACK	serial	GPL

Other (discontinued) direct solvers

Name	Last release	License
TAUCS	2003	BSD–style
SPOOLES	1999	public domain

MUMPS

http://graal.ens-lyon.fr/MUMPS/

Main features

- MUltifrontal Massively Parallel Solver
- Unsymmetric, symmetric positive definite, or general symmetric
- Uses MPI, BLAS, BLACS and ScaLAPACK
- written in Fortran 90, with a C interface
- several built–in ordering algorithms and interfaces (SCOTCH, METIS ...)

Developers

Patrick Amestoy et al. CERFACS, ENSEEIHT, CNRS, and INRIA (France)

SuperLU

http://crd.lbl.gov/ xiaoye/SuperLU/

Main features

- implementation of supernodal elimination
- three C libraries (with Fortran interfaces):
  - `SuperLU` for sequential machines
  - `SuperLU_MT` (multithreaded) for shared memory parallel machines
  - `SuperLU_DIST` (with MPI) for distributed memory
- preordering through library or user supplied routines
- uses BLAS and CBLAS

Developers

X.S. Li et al. Lawrence Berkeley National Laboratory (USA)

UMFPACK

http://www.cise.ufl.edu/research/sparse/umfpack/

Main features

- Unsymmetric MultiFrontal method implementation
- companion code (`CHOLMOD`) for symmetric positive definite matrices
- Written in C, provides Fortran-callable interfaces
- Uses BLAS and `AMD` (GNU LGPL) for ordering

Developers

Tim Davis (and others), University of Florida (USA)

Sparse Linear Algebra: iterative solvers

Iterative solvers and preconditioners

Name	Comments	License
PETSc	parallel	BSD–style
HYPRE	parallel	LGPL
Trilinos	parallel	LGPL

Iterative Solvers Templates

Name	Comments	License
ISTL	Template Library	GNU GPL
Netlib Templates	Template Guide	public domain

PETSc

http://www.mcs.anl.gov/petsc/petsc-as/

Main features

- A large suite of parallel linear/nonlinear solvers (and ODE integrators)
- Includes support for GPU
- Usable from C, C++, Fortran 77/90, and Python
- Two alternatives concerning internal data structures:
  - maintaining duplicate matrices (storage overhead)
  - using PETSc throught the code
- Many low-level routines to implement new methods

Developers

S. Balay et al. Argonne National Laboratory (USA)

HYPRE

https://computation.llnl.gov/casc/linear_solvers/sls_hypre.html

Main features

- Library of solvers (Krylov and multigrid) and preconditioners (AMG and ILU)
- Written in C with interfaces to Fortran 77/90, C++, Python, and Java
- Uses conceptual interfaces (to data structures):
  - structured/semi-structured grids
  - finite elements
  - linear–algebraic system

Developers

Robert Falgout et al. Lawrence Livermore National Laboratory (USA)

Trilinos

http://trilinos.sandia.gov/

Main features

- **huge** object-oriented software library incl. linear solvers
- Library includes many **self–contained** packages:
  - basic linear algebra routines (and BLAS/LAPACK wrappers)
  - preconditioners (ILU, multigrid)
  - direct solvers (and classes to SuperLU, UMFPACK...)
  - Krylov iterative solvers (using generic programming)
  - partition and load balancing tools
  - domain decomposition preconditioners and solvers
  - abstract interfaces, wrappers and adapaters

Developers

Sandia National Laboratories (USA)

ISTL

http://www.dune-project.org

Main features

- Example of a template library in a FE code (DUNE)
- Supports block recursive matrix and vector classes (natural structures in FE discretizations)
- Parallel capabilities (parallel communication interface)
- M. Blatt, P. Bastian, *The Iterative Solver Template Library* in *Applied Parallel Computing. State of the Art in Scientific Computing*. Lectures Notes in Computer Science. 2007.

Developers

P. Bastian, M. Blatt. Universität Heidelberg (Germany)

Templates Guide

http://www.netlib.org/templates/index.html

Main features

- NIST Templates provide:
  - mathematical description of algorithms, tips on parallel implementation and advice for preconditioning
- as well as some (old) example programs:
  - `IML++`, `SparseLib++`, `NIST Sparse BLAS`...
- R. Barret et al. *Templates for the Solution of Linear Systems. Building Blocks for Iterative Methods* 2nd ed. SIAM, 2006.

Developers

R. Barret et al. Oak Ridge National Lab., Univ. of Tennessee, Univ. of California (USA), and Utrecht Univ. (The Netherlands)

Eigenvalue solvers and tools

Eigenvalue solvers

Name	Comments	License
ARPACK	serial/parallel	BSD–style
SLEPc	parallel	GPL

Graph/Mesh partitioning and reordering

Name	Comments	License
Scotch	serial/parallel	CeCILL-C
Zoltan	serial/parallel	LGPL
Mondrian	serial	LGPL

ARPACK

http://www.caam.rice.edu/software/ARPACK/

Main features

- Based on implicitly restarted Arnoldi/Lanczos methods
- A parallel version exists (P_ARPACK) based on BLACS and MPI
- collection of Fortran77 subroutines
- also an object–oriented (C++) version: ARPACK++

Developers

D.C. Sorensen et al. Rice University (USA)

SLEPc

http://www.grycap.upv.es/slepc/

Main features

- Several eigensolvers: Krylov-Schur, Arnoldi, Lanczos, Jacobi-Davidson...
- built on top of PETSc and uses MPI
- Other related problems: SVD and quadratic eigenvalue problem, QEP
- Usable from C, C++ and Fortran

Developers

J.E. Román et al. Universidad Politécnica de Valencia (Spain)

Scotch

http://www.labri.fr/perso/pelegrin/scotch/

Main features

- Two versions:
  - Sequential: `Scotch`
  - Parallel: `PT-Scotch` and `libScotch`
- Graph/mesh partitioning, matrix block ordering

Developers

Laboratoire Bordelais de Recherche en Informatique (France)

Zoltan

http://www.cs.sandia.gov/Zoltan/

Main features

- Part of Trilinos suite: `http://trilinos.sandia.gov`
- Perform partition and dynamic load-balancing (among other tasks)
- Its serial predecessor (`Chaco`) is also available
- Incorporating Zoltan into applications requires:
  - Writing query functions returning information to Zoltan
  - Initializing Zoltan, creating a Zoltan object
  - Calling Zoltan tools

Developers

E. Boman et al. Sandia National Laboratories (USA)

Mondrian

http://www.staff.science.uu.nl/ bisse101/Mondriaan/

Main features

- Sparse matrix partitioning
- Provides a MATLAB interface
- Only serial version

Developers

Rob Bisseling et al. Utrecht University (the Netherlands)

Outline

CAD, meshing and visualization

Some CAD, meshing and visualization tools

Name	Comments	License
`Gmsh`	2D/3D mesh generator	GPL
`MayaVi2`	data visualizer	BSD–style
`Netgen`	2D/3D mesh generator	LGPL
`ParaView`	data visualizer	BSD–style
`Salome`	2D/3D CAD software	GPL

Salome

http://www.salome-platform.org

General features

- intended to provide a generic platform for Pre- and Post-Processing in numerical simulation
- developed by a consortium of
  - companies: EDF R&D, Bureau Veritas, Open Cascade (formerly an EADS division), Principia, CEDRAT
  - laboratories (private and public): EADS CCR, CEA, Lab. d'Informatique Paris VI, Lab. d'Electrotechnique de Grenoble

Salome

http://www.salome-platform.org

Some technical features

- modelling and meshing utilities
  - geometry reparation
  - mesh quality control
- meshing tools
  - Mephisto (2D) + Netgen(2D/3D)
  - plugin mechanisms for others
- integration of analysis tools
  - solid mechanics analysis: `SALOME-Meca`
  - see also `http://www.caelinux.com`
- allows Python scripting

Other Computer–Aided Design tools

Some other free software CAD programs

Name	*Comments*
BRLCAD	Constructive Solid Geometry (CSG) kernel
gCAD3D	poorly documented and meshing programs incompatibility
QCad	2D CAD program

Remarks

- Modern CAD programs use Boundary Representation (BREP) kernels
- QCad is a limited functionality version of a proprietary code

NETGEN

http://www.hpfem.jku.at/netgen

Main features

- automatic 2D/3D mesh generator
- accepts input from constructive solid geometry (CSG) or boundary representation (BRep)
- can mesh geometries from IGES and STEP CAD-files
- contains modules for mesh optimization and hierarchical mesh refinement

Developers

J. Schöberl, H. Gerstmayr and R. Gaisbauer

Johannes Kepler University, Austria

GMSH

http://www.geuz.org/gmsh

Main features

- automatic 2D/3D mesh generator
- built-in (BRep) CAD engine and post-processor
- parametric input (own scripting language)
- can mesh geometries from IGES and STEP CAD files
- easily interfaced with GetDP solver

Developers

- Ch. Geuzaine, Université de Liège, Belgium
- J.F. Remacle, Université catholique de Louvain, Belgium

Other tools

Other non–free utilities

- `TetGen`, a 3D tetrahedral mesh generator free for research and non–commercial use

  `http://tetgen.berlios.de/`

- `METIS` and `ParMETIS`, a family of programs for partitioning graphs and hypergraphs and computing fill–reducing orderings of sparse matrices

  `http://glaros.dtc.umn.edu/gkhome/views/metis`

  `METIS` and `ParMETIS` can be freely used but redistributed **only** under certain conditions

Mayavi

http://code.enthought.com/projects/mayavi/

Main features

- Uses Visualization Toolkit (VTK)
- Allows Python scripting
- Can be used as a library (embeded into other applications)

Developers

Enthought Scientific Computing Solutions, USA

Paraview

http://www.paraview.org

Main features

- Uses Visualization Toolkit (VTK)
- Allows Python scripting
- Able to process large data sets (distributed computing)

Developers

Collaborative effort between

- companies: Kitware and CSimSoft
- and laboratories: Sandia National Lab., Los Alamos National Lab. and Army Research Lab.

Other visualization tools

Visit

- similar to `Paraview` and `MayaVi`
- developed by Lawrence Livermore National Laboratory
- `https://wci.llnl.gov/codes/visit/`

OpenDX

- general scientific visualization tool
- developed by IBM (low activity since 2007)
- `http://www.opendx.org/`

Outline

PDE numerical simulation tools

A rough clasification of PDE numerical simulation tools

- end–user programs
- numerical simulation environments (metalanguages)
- numerical simulation libraries

Remark

Fuzzy boundary since:

object–oriented programming library + good interface = numerical simulation environment

PDE numerical simulation tools

End–user programs

Name	*Comments*	*License*
`Elmer`	Parallel	GPL

Finite element environments

Name	*Comments*	*License*
`FEniCS`	Sequential (so far)	LGPL
`FreeFEM`	Parallel	GPL
`Rheolef`	Sequential	GPL

PDE numerical simulation tools

Finite element libraries

Name	*Comments*	*License*
ALBERTA	Sequential	GPL
Deal.II	Parallel	QPL
DUNE	Parallel	GPL
GetDP	Parallel	GPL
GetFEM++	Sequential (so far)	LGPL
LibMesh	Parallel	LGPL
OFELI	Sequential	GPL
OOFEM	Parallel	GPL

Alberta

http://www.alberta-fem.de/

Main features

- Adaptive multiLevel finite element toolbox
- written in C
- Provides appropriate data structures holding geometrical, finite element, and algebraic information
- Allows dimension-independent development and programming

Developers

A. Schmidt (Universität Bremen), K.G. Siebert and C.J. Heine (Universität Duisburg-Essen), D. Köster (ANSYS) and O. Kriessl (BadenIT)

Deal.II

http://www.dealii.org/

Main features

- Libray for adaptive finite element techniques
- Written in C++
- Support for a variety of finite elements
- Stand–alone linear algebra library + interface to other packages such as Trilinos, PETSc and METIS

Developers

W. Bangerth and G. Kanschat (Texas A&M Univ.), and R. Hartmann (Deutsches Zentrum für Luft- und Raumfahrt)

DUNE

http://www.dune-project.org

Main features

- modular (C++) toolbox for solving partial differential equations (PDEs) with grid–based methods
- based on a separation of data structures and algoritms
- supports easy implementation of methods like Finite Elements (FE), Finite Volumes (FV), and also Finite Differences (FD)
- slim interfaces allowing an efficient use of legacy and/or new libraries

Developers

P. Bastian, M. Blatt (Univ. Heidelberg) and many others

Elmer

http://www.csc.fi/elmer

Main features

- a relatively large set of physical models
- coupling between physical models is easy
- new solvers can be easily added (segregated scheme)
- some tools to implement advanded numerical alghoritms
- written in Fortran 90
- parallel capabilities

Developers

- Developed by CSC, the Finnish IT center for science

FEniCS
http://www.fenics.org

Main features

- project to develop a numerical simulation environment
  - object–oriented progr. FE library + Python interface
  - tools for automatic/efficient evaluation of variational forms
- large family of finite elements
- (experimental) parallel capabilities
- limited input/output compatibility support
- poorly documented (for the time being)

Developers

Univ. of Chicago, Argonne National Lab., Delft Univ. of Techn., Royal Institute of Techn. KTH, Simula Research Lab., Finite Element Center and Univ. of Cambridge

FreeFEM

http://www.freefem.org/

Main features

- Implementation of a language dedicated to the finite element method
- Two branches of development: FreeFEM++ and FreeFEM3D
  - `FreeFEM++`: 2D/3D code with a dedicated IDE (`Freefem++-cs`)
  - `FreeFEM3D`: solver based on FEM or Fictious Domain approach
- Quite popular around here!

Developers

O. Pironneau, F. Hetch, A. Le Hyaric et al. (Univ. Paris VI)

GetDP

http://www.geuz.org/getdp/

Main features

- GetDP: **G**eneral **e**nvironment for the **t**reatment of **D**iscrete **P**roblems
- Open to different numerical methods (FEM, BEM)
- Programming (in C) try to follow symbolic mathematical expressions
- Many examples in electromagnetics (with thermal coupling)
- Easily linked to Gmsh (also developed by C. Geuzaine)

Developers

P. Dular and C. Geuzaine (Univ. of Liège)

GetFEM++

http://home.gna.org/getfem/

Main features

- A generic C++ finite element library
- offers some models (specially from structural mechanics) in the form of resusable bricks
- Can deal with discontinuities by fictitious domain methods of XFEM type
- Includes a dedicated library (`Gmm++`) for basic linear algebra
- Provides interfaces with Scilab, Matlab and Python

Developers

Y. Renard and J. Pommier (INSA Lyon)

LibMesh

http://libmesh.sourceforge.net/

Main features

- A framework (C++) for the numerical simulation of PDEs using arbitrary unstructured discretizations:
  - Classes and functions for writing parallel adaptive finite element applications
  - An interface to linear algebra, meshing, partitioning, etc. libraries.
- Interface from GNU Octave (by R. Rodríguez-Galván)
- Uses PETSc and SLEPc.

Developers

B.S. Kirk (NASA), J.W. Peterson and D. Gaston (Univ. of Texas at Austin) and others

OFELI

http://www.ofeli.net/

Main features

- A framework of C++ classes for the development of finite element programs
- Includes classes for the solution of common problems in solid mechanics heat transfer, fluid dynamics...
- Also contains some prototype codes and utility programs (mesh generation/conversion)

Developers

R. Touzani (Univ. Blaise Pascal)

OOFEM

http://www.oofem.org/en/oofem.html

Main features

- C++ FE code with object oriented architecture including
  - Modular & extensible FEM kernel (`OOFEMlib`)
  - Dedicated modules (structural mech., fluid dynamics, etc)
  - Utilities (postprocessing, meshing, etc)
- Interfaces to many codes, including solvers (SPOOLES, PETSc, SLEPc), partitioning (METIS) or visualization (Mayavi, Paraview)

Developers

B. Patzák (Czech Technical University) and many others

Rheolef

http://ljk.imag.fr/membres/Pierre.Saramito/rheolef/

Main features

- C++ finite element environment
- clear definition of (discrete) problems (FreeFEM-like)
- simple CAD, meshing and (VTK) postprocessing tool
- some simple models/techniques pre-programmed:
  - elasticity, Stokes, convection-diffusion, Navier-Stokes
  - mesh adaption

Developers

- P. Saramito, N. Roquet and J. Etienne

  Lab. Jean Kuntzmann, Grenoble

An overview of Elmer

Physical models in Elmer

- fluid dynamics and heat transfer
- solid mechanics
- electromagnetics
- electrokinetics, acoustics...

Numerical methods in Elmer

- finite element library
- stabilization techniques
- adaptive techniques
- ALE formulation, level sets...

Physical models in Elmer

Fluid dynamics

- compressible/incompressible Navier-Stokes
  with free surface and turbulence models
- Richards equation (unsaturated porous media)
- Reynolds equation

Heat transfer

- convection–conduction–reaction
  with phase change, radiation

Physical models in Elmer

Solid mechanics

- linear/non-linear elasticity
- plate and shell models

Acoustics

- Helmholtz Equation
- BEM solver

Physical models in Elmer

Electromagnetism

- electrostatics
- static current conduction
- low frequency models (potential/magnetid field)

Electrokinetics

- Poisson-Boltzmann Equation
- slip conditions and Joule effect

Numerical methods in Elmer

Finite element techniques

- arbitrary order finite elements
- stabilization techniques:
  - SUPG
  - residual–free bubbles
  - discontinuous Galerkin
- adaptive techniques:
  - preprogrammed error estimator
  - user–defined error estimators

Numerical methods in Elmer (cont.)

Other implemented techniques

- multiple meshes supported (segregated solution)
- level sets
- moving meshes (ALE)
  - elastic deformation
  - easy to implement user–defined *smoothers*
- matrix manipulation tools
  - system reduction (displacement solvers)
  - boundary conditions
  - flux computations

Numerical methods in Elmer (cont.)

Basic numerical methods

- linear systems solvers:
  - multifrontal (UMFPACK)
  - Krylov (BiPCG, GMRES)
  - multigrid (geometric/algebraic)

  and possibility to use HYPRE, MUMPS or SuperLU
- nonlinear solver: Newton-Picard
- time integration:
  - BDF (step adaption)
  - Newmark
- eigensolvers: Arnoldi (ARPACK)

Numerical simulation tools in solid mechanics

Free software in solid mechanics analysis

Name	*Comments*	*License*
`Calculix`	structural analysis	GPL
`Code_Aster`	thermomechanical analysis	GPL
`FELyX`	structural analysis	GPL
`Impact`	dynamic analysis	GPL
`Tahoe`	thermomechanical analysis	BSD-style
`WARP3D`	structural analysis	GPL

Calculix

http://www.calculix.de

Main features

- linear and non-linear calculations
- static, dynamic and thermal solvers
- pre- and postprocessing capabilities
- some import/export compatibilities

Developers

- Guido Dhondt and Klaus Wittig

  MTU Aero Engines GmbH, Germany

FELyX

http://felyx.sourceforge.net

Main features

- object–oriented Finite Element code
- linear/modal analysis
- basic structural elements
- input/output compatibility with proprietary codes
  ANSYS (CATIA, MSC)
- beta version and low activity project

Developers

- EVEN Evolutionary Engineering AG, Switzerland

WARP3D

http://cern49.cee.uiuc.edu/cfm/warp3d.html

Main features

- Static/Dynamic Nonlinear Analysis of Fracture
  - finite strain formulation
  - crack growth modelling
  - stress intensity factor computation
- Parallel capabilities

Developers

- Department of Civil Engineering

  University of Illinois at Urbana-Champaign, USA

Impact

http://impact.sourceforge.net

Main features

- small (Java) implementation of an explicit dynamic FE code based on DYNA3D by J.O. Hallquist at the LLNL
- pre- and postprocessing capabilities
- parallel capabilities
- poorly documented
- beta version and low activity project

Developers

- J. Forssell (Sweden), Y. Mikhaylovskiy (Russia) and many others

Tahoe

http://sourceforge.net/projects/tahoe/

Main features

- analysis of non–standard problems in solid mechanics:

  fracture/failure, interfacial adhesion and debonding, shear banding, length-scale dependent elasticity and plasticity.
- includes meshfree simulation and particle methods
- support parallel execution
- documentation not updated
- project home `http://tahoe.ca.sandia.gov/` discontinued

Developers

- Sandia National Laboratory, USA

Code_Aster

http://www.code-aster.org

Main features

- complete structural and termomechanical code:
  - static, dynamic, modal and harmonic analysis
  - fracture, damage and fatigue
  - wide range of constitutive laws
- some multiphysics analysis
  - drying/hydratation, metallurgy analysis, fluid-structure ...
- large library of elements (including X–FEM)
- Python scripting
- a CAD–integrated version `Salome-Meca` available

Developers

- Electricité de France R & D

Numerical simulation tools in fluid dynamics

Free software in fluid dynamics analysis

Name	*Comments*	*License*
`Code_Saturne`	general purpose	GPL
`FEATFlow`	incompr. NS	BSD–style
`Gerris`	incompr. NS	GPL
`OpenFOAM`	general purpose	GPL

FEATFlow

http://www.featflow.de

Main features

- incompressible Navier-Stokes 2D/3D
- parallel capabilities
- superseded by FEAST, not released as free software (as yet?)

Developers

- S. Turek and co-workers

  Department of Applied Mathematics, University of Dortmund, Germany

Gerris Flow Solver

http://gfs.sourceforge.net

Main features

- incompressible 2D/3D Navier-Stokes solver

  with (advection-diffusion) transport equations
- adaptive mesh refinement
- automatic mesh generation
- parallel capabilities

Developers

- Stéphane Popinet

  National Institute of Water and Atmospheric Research, Wellington, New Zealand

Code_Saturne

http://rd.edf.com

Main features

- complete fluid dynamics with a large number of models
  - turbulence, radiation, combustion, magnetohydrodynamics, multiphase flow
- parallel capabilities
- input/output capabilities
  - I-DEAS, GMSH, Gambit, Salomé ...
  - EnSight, MED files ...
- easy to link to Code_Aster (notably by Salomé)

Developers

- Electricité de France R & D

OpenFOAM

http://www.opencfd.co.uk/openfoam

Main features

- finite volume C++ toolbox
- large set of models and solvers
- pre- and postprocessing capabilities
- parallel compunting capabilities
- very active community of users/contributors

Developers

- OpenCFD Ltd, UK

Other numerical simulation tools

Many other free software tools such as...

- Puma–EM: Method of Moments for Electromagnetics using the Multilevel Fast Multipole Method

  `http://sourceforge.net/projects/puma-em/`
- Fire Dynamics Simulator: Computational fluid dynamics (CFD) model of fire–driven fluid flow

  `http://www.fire.nist.gov/fds/`
- FiPy: Object oriented, partial differential equation FV solver, written in Python

  `http://www.ctcms.nist.gov/fipy/`
- and **many** others!

Outline

Outline

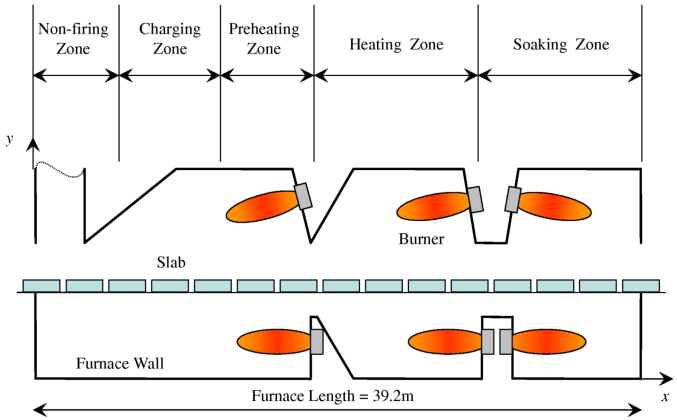
Steel products heating in industrial furnaces

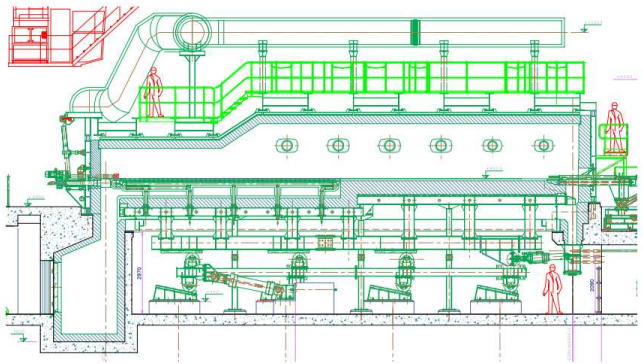
Heat treatment of hot forged axles

austenizing furnace in quench hardening

Reheating of billets in hot rolling plants

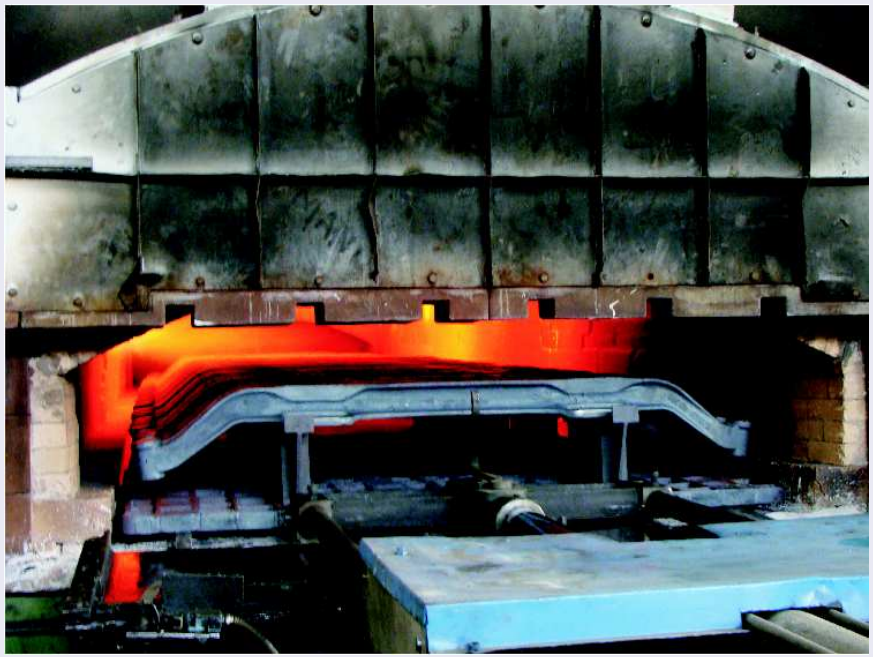
reheating (with austenizing) before hot rolling

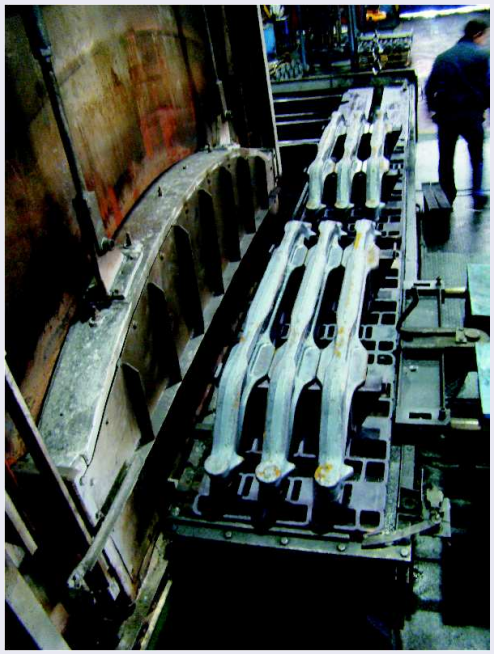




Steel products heating in industrial furnaces

Heat treatment of hot forged axles





CIE Automotive

Steel products heating in industrial furnaces (II)

Goal I: Design of furnace reconfiguration

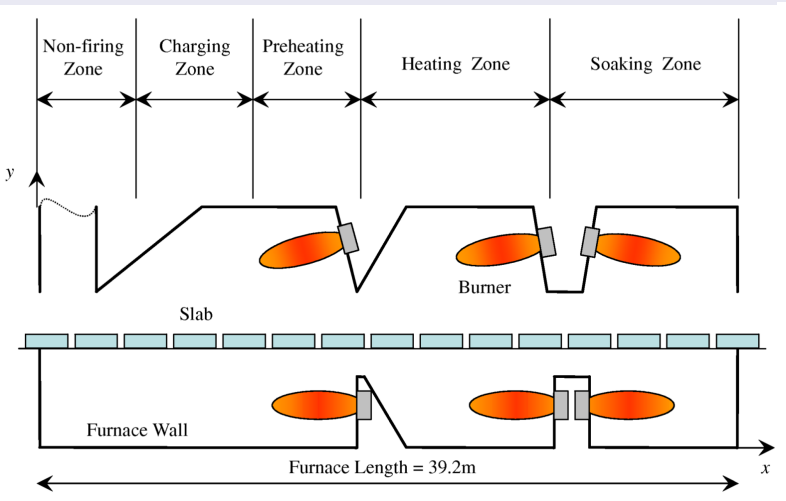
Prediction of steel pieces heating for a given furnace design (and operation conditions) in the framework of the modification of an existing furnace

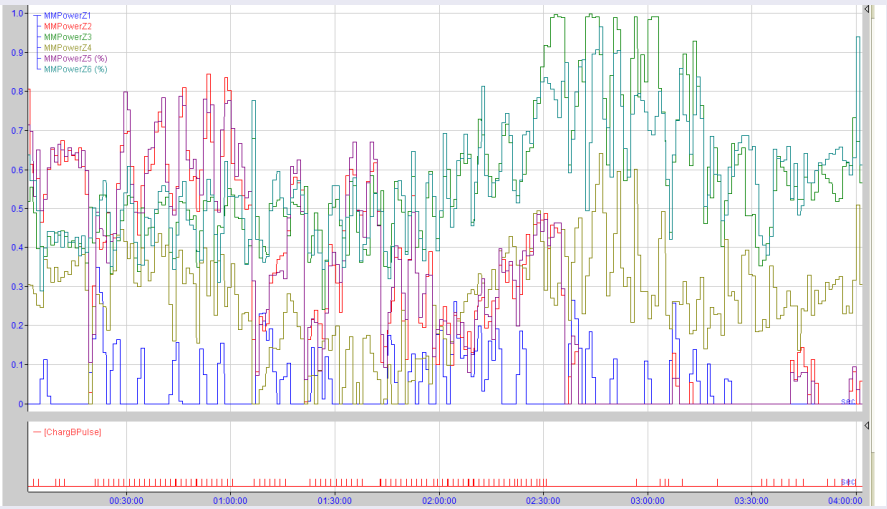
Goal II: Design of operation conditions

- Direct design: prediction of axles heating for given operation conditions
- Inverse design: determine optimal operation conditions (minimizing some objetive function)

Reheating of billets in hot rolling plants

Billet reheating furnace operation



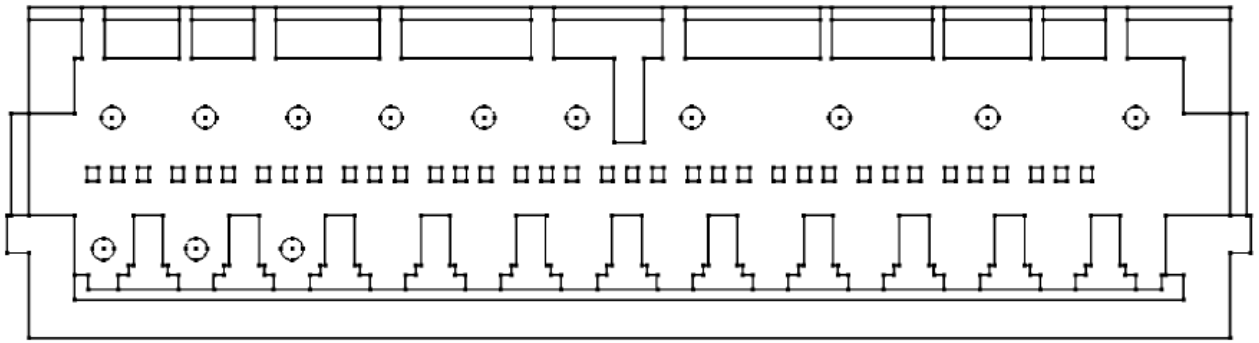


Reheating of billets in hot rolling plants (II)

Goal: Fast prediction of billet heating

- billet heating prediction under dynamical conditions
  - time–dependent power of (group of) burners
  - non–steady operation of walking beam system (resulting in variable residence times)
  - non–regular billets feeding (presence of *gaps*)
- *real time* implementation of a simulation tool
  - to be used in control strategy

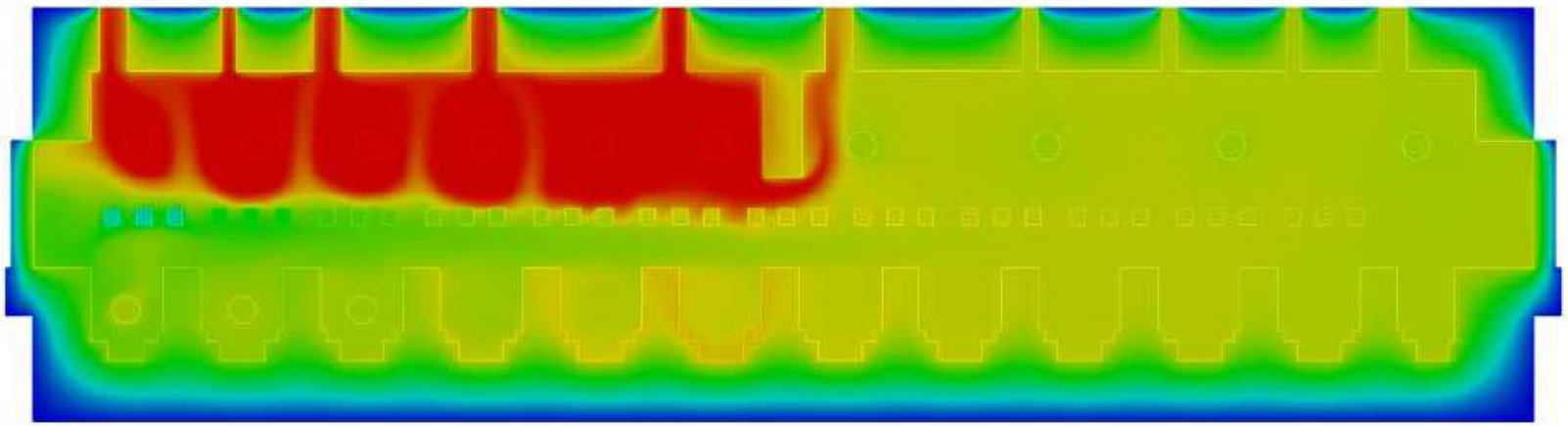
Basic building block:2D quasi–steady model



Involved phenomena

- combustion in burners
- thermofluid dynamics of combustion products
- heat transfer in pieces and furnace walls
- thermal radiation in furnace chamber

Basic aspects of the model



Simplifying hypotheses

- complete combustion on a known flame surface
- gases (except flame) not participating in thermal radiation
- steady temperatures on furnace walls and gases
- 2D model (over mean vertical section)

Derivation of a global model: segregated approach

Submodel 1: furnace walls

- Thermal radiation: refractory–flames–axles
- Convection from/to gases (chamber and ambient)

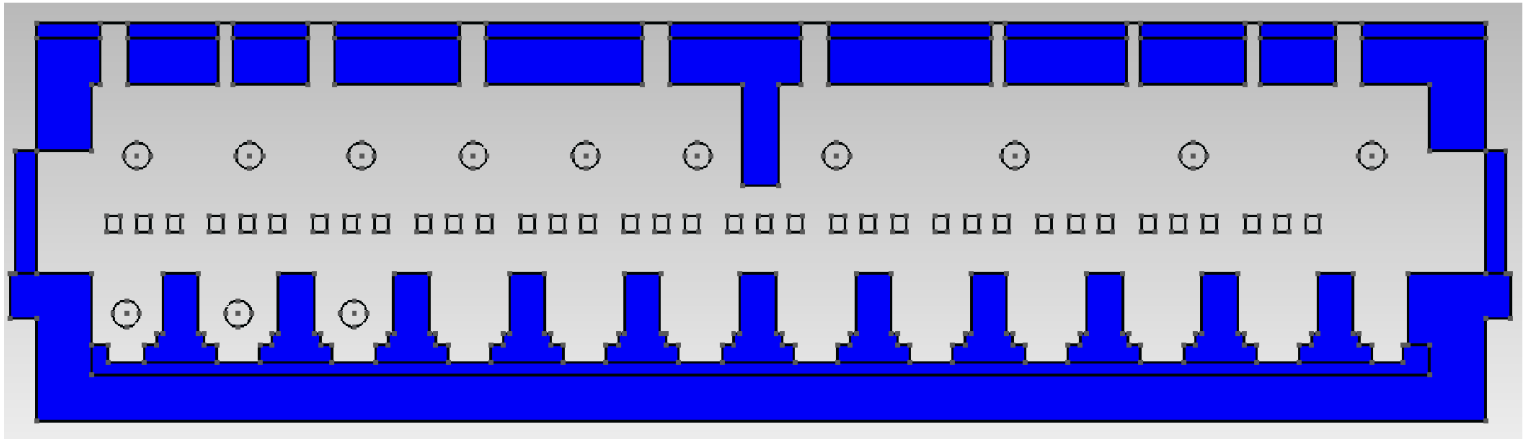
Submodel 2: thermofluid dynamics of combustion products

- Convection from/to furnace walls and axles

Submodel 3: axles heating

- Evolution problem
- Thermal radiation and convection from/to gases

Furnace walls submodel



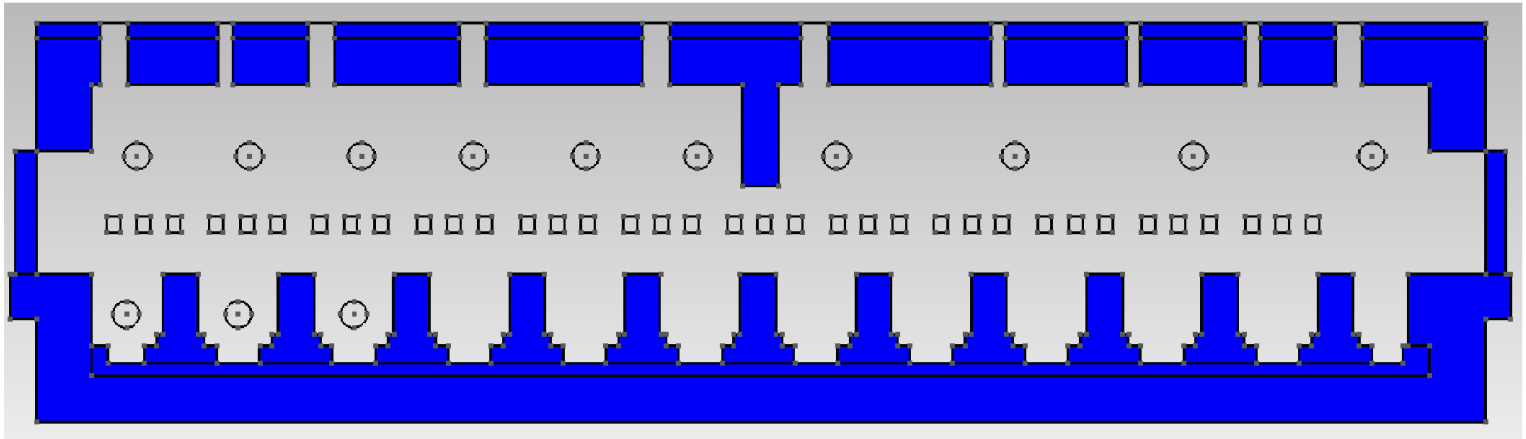
On furnace walls

$$-\mathrm{div}(k_e \vec{\nabla} T_e) = 0$$

Boundary conditions

- Outer boundary: $-k_e \frac{\partial T_e}{\partial n} = h(T_e - T_\infty)$
- Inner boundary: $-k_e \frac{\partial T_e}{\partial n} = q_{conv} + q_{rad}$

Furnace walls submodel (II)



Surface–to–surface thermal radiation

For k–th surface element:

$$q_{in}^{k} = \frac{1}{A_k} \sum_{j=1}^{N_{rad}} A_j F_{jk} q_{out}^{j}$$

$$q_{out}^{k} = \rho_k q_{in}^{k} + \epsilon_k \sigma T_k^4$$

Furnace walls submodel (III)

Flux computation on participating surfaces: Gebhardt factors

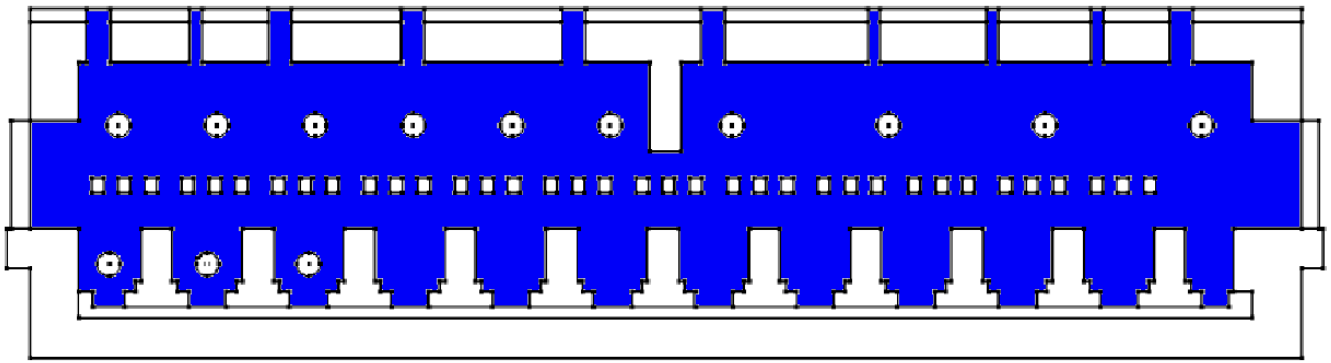
$$q^k_{out} - \rho_k \sum_{j=1}^{N_{rad}} F_{jk} q^j_{out} = \epsilon_k \sigma T_k^4$$

$$q^k_{net} = \sigma \epsilon_k T_k^4 - \frac{\sigma \epsilon_k}{A_k} \sum_{j=1}^{N_{rad}} G_{jk} T_j^4$$

Conditions on burners: energy balance on flame surface

$$\dot{W}_{burner} = \sum_{k=1}^{N_{burner}} (q^k_{net} + \rho \Delta h \vec{v} \cdot \vec{n} A_k)$$

Thermofluid dynamics submodel



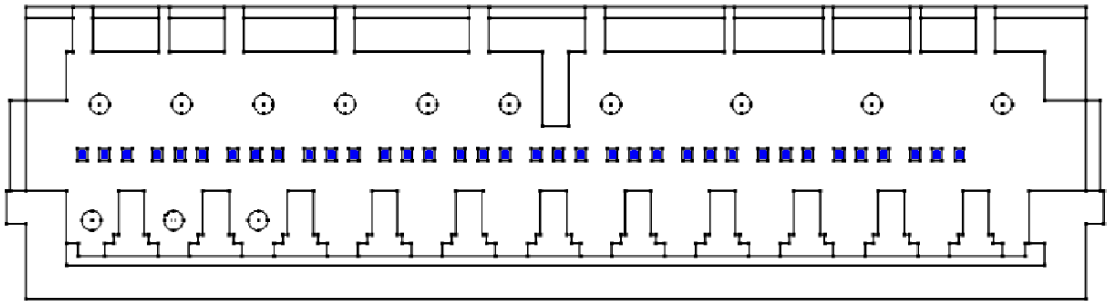
Turbulent, steady, compressible flow

$$\mathrm{div}(\rho_g \vec{U}) = 0$$

$$\mathrm{div}(\rho_g \vec{U} \otimes \vec{U}) + \vec{\nabla} P - \mathrm{div}(\mu(\vec{\nabla}\vec{U} + (\vec{\nabla}\vec{U})^T) = \mathrm{div}\tau^R$$

$$\mathrm{div}(\rho_g \vec{U} T_g) - \mathrm{div}((k + k_T)\nabla T_g) = 0$$

Axles heating submodel



Axles heating

$$\rho_p c_p \frac{\partial T_p}{\partial t} - \mathrm{div}(k_p \vec{\nabla} T_p) = 0$$

Boundary conditions

$$-k_p \frac{\partial T_p}{\partial n} = q_{rad} + q_{conv}$$

Global algorithm

Algorithm outline

- Initialize axles heating curve
- Initialize convective fluxes
- Iteration loop on submodels:
  - Solve (steady) model (radiation–conduction) on furnace
  - Solve (steady) model (thermofluid dynamics) on gases
  - Solve (evolution) model for axles heating
  - Convergence test

Remark

Some relaxation is needed to avoid numerical instabilities

Numerical discretization

Furnace (walls) submodel

- non–linearity: fixed point
- spatial discretization: $P1$ finite element
- Gebhardt (factor) matrix is stored

Gases submodel

- segregated solver (N–S, $k - \epsilon$, energy) with fixed point
- non–linearities: fixed point and Newton
- spatial discretizacion: $P2/P1$+SUPG and $P1 - b$

Axles submodel

- time integration: BDF–1 with time step adaption
- spatial discretizacion: $P1$ finite element

Implementation with free software tools

CAD and meshing using Gmsh

`http://geuz.org/gmsh/`

Model solvers using Elmer

`http://www.csc.fi/english/pages/elmer`

Global algorithm programming using Python

`http://www.python.org/`

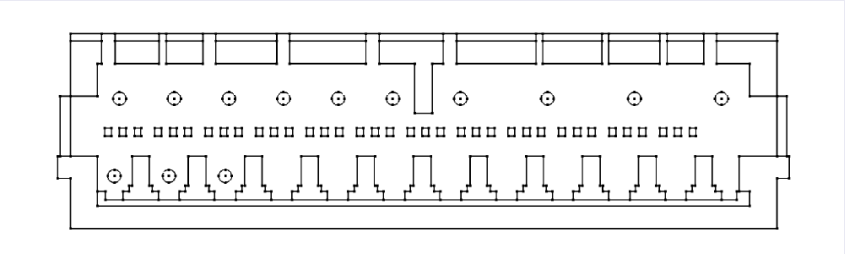
`http://www.scipy.org/`

Postprocessing using Paraview

`http://www.paraview.org/`

Validation

Heat treatment (austenizing) furnace



Operation conditions: power

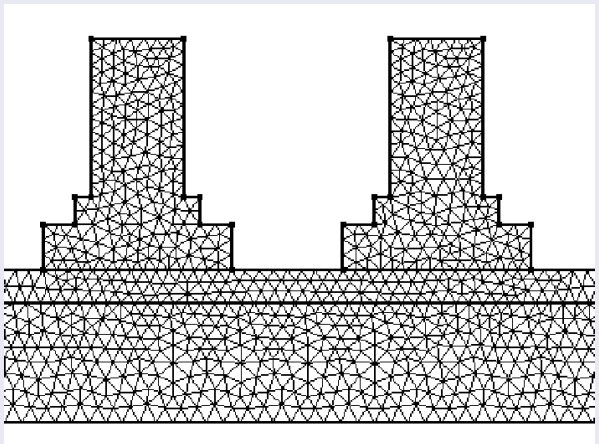
- Total furnace power: 1.81 MW
- Group I nominal power (6 burners): 1.15 MW
- Group II nominal power (7 burners): 0.66 MW

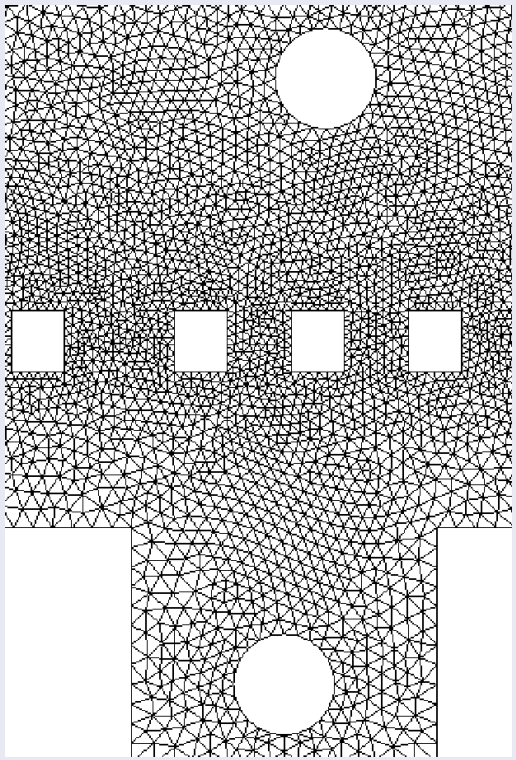
Operation conditions: feeding

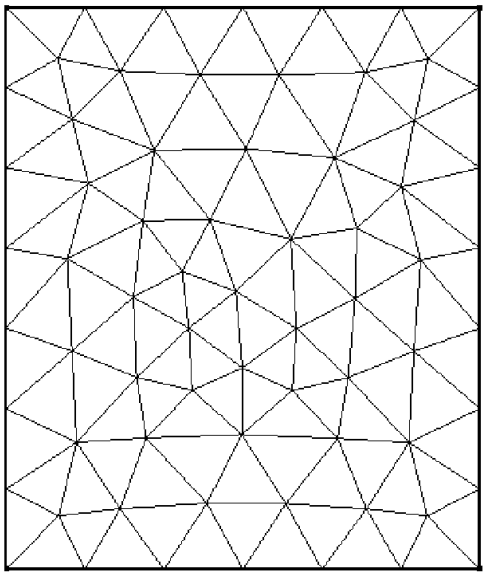
- Residence time: 720 s.

Heat treatment furnace simulation

Details of meshes







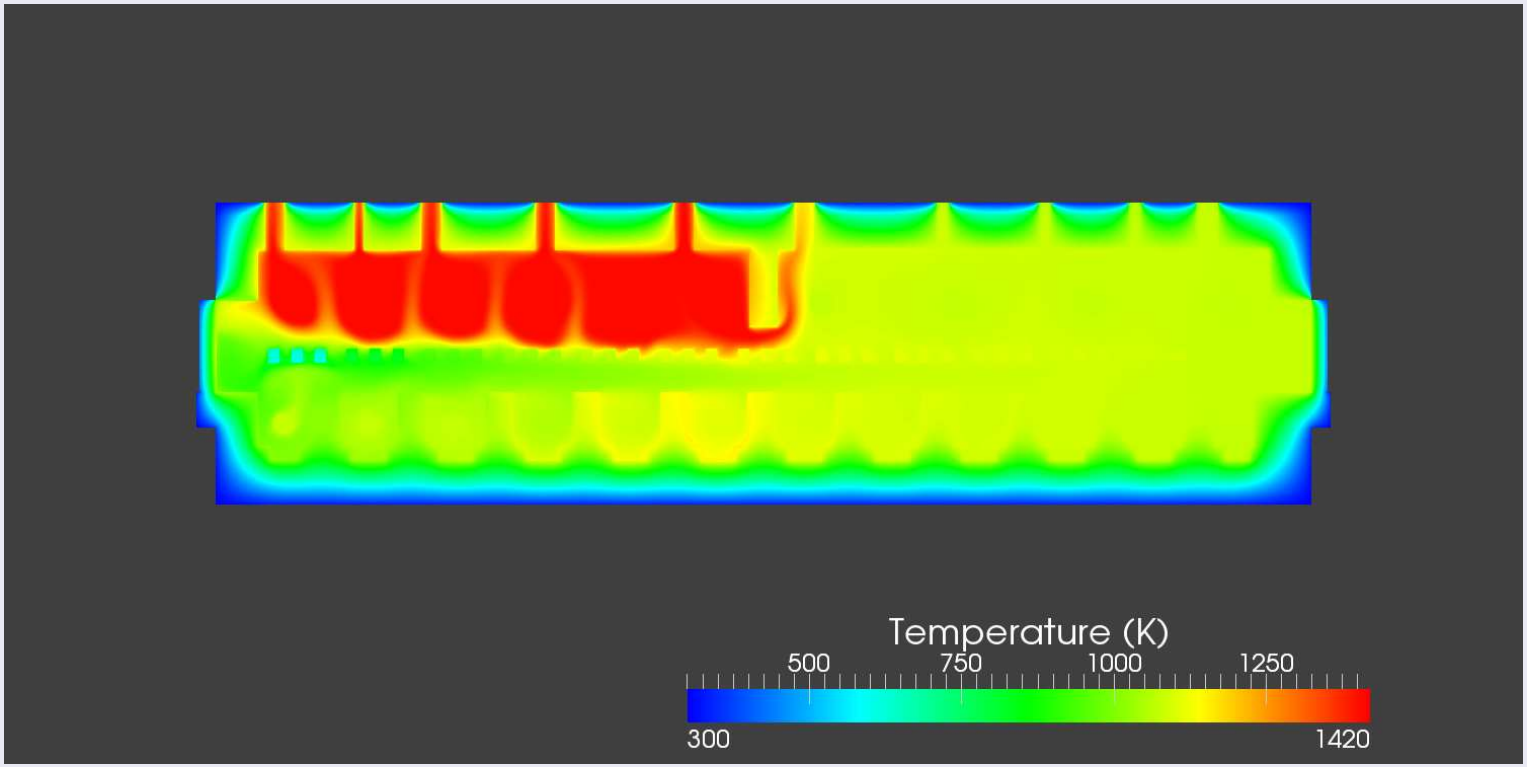
Heat treatment furnace simulation (II)

Some computational figures

- Mesh size
  - furnace (walls): 24000 nodes
  - gases: 38500 nodes
  - axles: 2300 nodes
  - radiating surfaces: 30000 edges
- Global algorithm iterations: 9
- Tolerance in convergence test: $1K$
- Computational cost:
  - 3.5 h. of computation (Core2Duo 2.5 GHz, 1 core)
  - memory peak below 1 GB

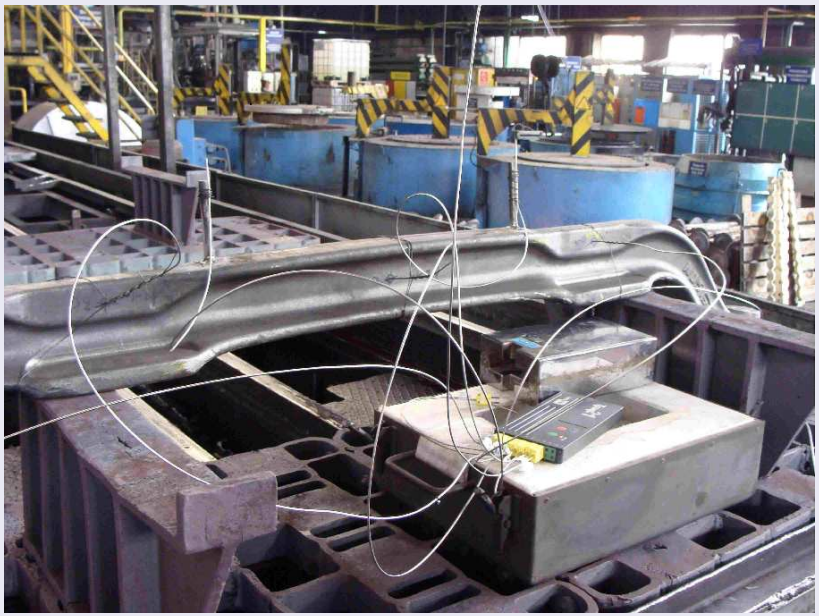
Heat treatment furnace simulation (IIO)

Numerical simulation results



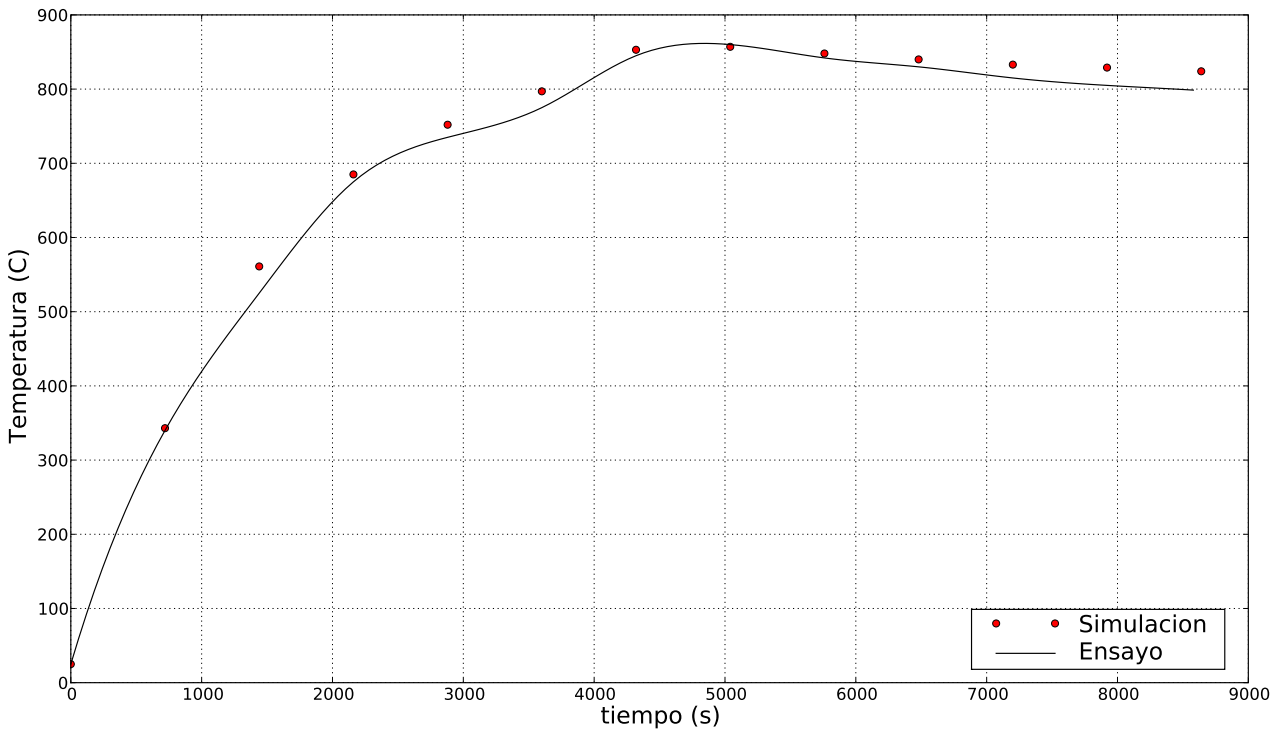
Validation

Thermocouples (Datapaq) test at CIE–Galfor



Validation (II)

Experimental and numerical results



On–going work

(Fast) direct design under steady conditions

- building numerical simulation database (tensor form)
- compression using Higher–Order Singular Value Decomp.
- interpolation exploiting HOSVD properties

(Fast) inverse design under steady conditions

- minimization of (heating related) cost function
- heating prediction using HOSVD + interpolation
- Steihaug–Toint (truncated CG) trust region algorithm
  - exact derivatives (based on single variable interpolation)
  - initial guess using *gappy Proper Orthogonal Decomposition*

On–going work (2)

(Fast) prediction under dynamical conditions

- derivation of (evolution) reduced order models (using Galerkin–Proper Orthogonal Decomposition) for
  - billets heating
  - thermofluid dynamics of gases (not yet!)
  - furnace walls heating
- preprocessing of some information
  - Gebhardt matrices
  - fundamental matrices of (linearized) ode systems
- *real time* implementation

Conclusions drawn from this experience

Concerning flexibility, free software...

- offers huge modelling/solving flexibility
  for instance *ad hoc* wall laws
- eases technology transfer
  avoiding licence dependencies

Concerning productivity, free software...

- makes possible a small team to face industrial problems
- allows early concentration on *less standard* tasks

Outline

Concluding remarks

Scientific Computing Software Development

- developing scientific computing software is **hard**
- exploiting new/emerging architectures is challenging
- validation of scientific computing software requires careful attention
- software engineering issues are important when maintenance, portability and reusability are considered

Concluding remarks (cont.)

Free software approach

- free software model addresses main identified challenge in scientific computing:
  - Algorithms and software that applications developers can reuse in the form of high–quality, high–performance, sustained software components, libraries and modules
  - A community environment that allows the sharing of software, communication of interdisciplinary knowledge and the development of appropriate skills
- possibility to build free state–of–the–art scientific computing software based on usual *full disclosure* principles for modelling and numerical analysis applied to **software** as well
- scientific computing free software is already quite mature!