

**FORTISSIMO**

D10.10

Final Compendium of General Training Material

Workpackage:	10	Dissemination and Training
Author(s):	Ullrich Becker-Lemgau	INTEL
Authorized by	Ullrich Becker-Lemgau	INTEL
Reviewer	Carlo Cavazzoni	CINECA
Reviewer	Jochen Buchholz	HLRS
Reviewer	George Beckett	UEDIN
Dissemination Level	PU	

Date	Author	Comments	Version	Status
2017-03-09	U. Becker-Lemgau	Initial draft	V0.1	Draft
2017-03-21	U. Becker-Lemgau	Incorporating reviewers feedback	V0.2	Draft
2017-03-21	U. Becker-Lemgau	Including comment on BMI and presentation skills training	V0.3	Draft
2017-03-21	U. Becker-Lemgau	References corrected	V0.4	Draft
2017-03-22	U. Becker-Lemgau	Cleaning up	V1.0	Final

Table of Contents

1	Introduction	1
1.1	Hands-on Introduction to High-Performance Computing	1
1.2	HPC Computer Aided Engineering	1
1.3	HPC Cloud.....	1
2	Training Material “Hands-on Introduction to High Performance Computing (EPCC)”	2
3	Training Material “HPC Computer Aided Engineering (CINECA)”	25
4	Training Material “HPC Cloud (SURFsara)”	39
5	References	41

1 Introduction

The work package WP10 “Dissemination and Training” of the Fortissimo project involves 13 tasks of which one task is concerned with training for the partners involved in the experiments [1]. This document includes the training material for three courses that make up the final training compendium.

All of the training courses are targeted at HPC beginners. No special pre-requisites are necessary although a general understanding of computing and programming might be useful.

As described in the deliverable “The Month 42 Training Report” [4] Fortissimo has organized a training on Business Model Innovation and two training sessions on Communication and Presentation Skills for all Fortissimo partners. For all three training events external speakers have been contracted. Due to copyright restrictions, material used for these events cannot be included into the collection of training material.

All of the material presented here is available for download from the projects shared document server under <https://fortissimo.4pm.si/login.jsf> [6]. The material given here has been reduced in resolution to reduce the size of this document. The material in original resolution is available upon request (info@fortissimo-project.eu).

All courses are free to be reused and there is no restriction. No fee or any other payment applies.

The following sections give a brief introduction into the training courses.

1.1 Hands-on Introduction to High-Performance Computing

This training – developed by EPCC – provides an introduction to HPC and how to use an HPC resource like the ARCHER system at EPCC. Basic building blocks like CPU and memory are explained and their importance for using an HPC systems. Additionally the basics of Parallel Computing, Libraries and Compilers are introduced. Several examples are given that can be used to try-out the HPC system (examples are not included in this document but available through download from the project shared folder 4PM [6]).

1.2 HPC Computer Aided Engineering

This training has been developed by CINECA and gives an introduction to Computer Aided Design with an emphasis on HPC and CFD. As an example, open source applications and tools are used to demonstrate how to solve a typical CAE problem. Additionally an introduction to HPC and HPC tools is given.

1.3 HPC Cloud

This training has been developed by SURFsara and introduces how to access an HPC resource in the cloud. Several aspects and concepts of HPC clouds are introduced.

2 Training Material “Hands-on Introduction to High Performance Computing (EPCC)”

This training has been developed by EPCC and was held June 30th + July 1st 2014. The material is available through the shared document server [6].

<h3>Introduction to High Performance Computing</h3> <p>Toni Collis t.collis@epcc.ed.ac.uk</p>	<h3>Reusing this material</h3> <p>This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en_US</p> <p><small>This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.</small></p> <p><small>Note that this presentation contains images owned by others. Please seek their permits or before reusing these images.</small></p>																								
<h3>Course Parameters</h3> <ul style="list-style-type: none"> • Pre-requisites <ul style="list-style-type: none"> - None, this course is designed for everyone, from computing novices upwards, to be able to participate in and complete • Hands-on practicals form an integral part of the course. <ul style="list-style-type: none"> - We will help with these, and do not expect any programming experience of attendees (although you're free to dive into the programs if you have more computing experience) 	<h3>Aims</h3> <ul style="list-style-type: none"> • Why do people use HPC? • What do people use HPC for? • Understanding of computer hardware <ul style="list-style-type: none"> - Which parts matter for performance in modelling and simulation? • Understanding of processes and threads • Understanding of parallel programming models • How to interact with a HPC resource • Knowledge of current HPC architectures • Knowledge of current parallel programming libraries • Appreciation of the future of HPC 																								
<h3>Timetable</h3> <table border="0"> <thead> <tr> <th>Day 1</th> <th>Day 2</th> </tr> </thead> <tbody> <tr> <td>09:30-09:45 Welcome</td> <td>09:30-10:30 Current HPC Architectures</td> </tr> <tr> <td>09:45-10:15 Why learn about HPC?</td> <td>10:30-10:30 Parallel Programming Libraries and Implementations</td> </tr> <tr> <td>10:15-11:45 Using HPC machines</td> <td>10:30-11:00 Practical</td> </tr> <tr> <td>13:45-11:15 Coffee Break</td> <td>11:00-11:30 (Coffee Break)</td> </tr> <tr> <td>11:15-12:30 Practical</td> <td>11:30-12:00 Practical</td> </tr> <tr> <td>12:30-13:30 Building Blocks 1: CPUs, Memory, and Accelerators</td> <td>12:00-12:30 Software support</td> </tr> <tr> <td>13:30-14:30 Lunch</td> <td>12:30-13:30 The Future of HPC and Wrap Up</td> </tr> <tr> <td>14:30-14:30 Building Blocks 2: Operating Systems</td> <td>12:00-14:30 Lunch with W-PC lunch</td> </tr> <tr> <td>14:30-15:30 What does parallel computing mean?</td> <td></td> </tr> <tr> <td>15:30-15:30 Coffee Break</td> <td></td> </tr> <tr> <td>15:30-17:30 Practical</td> <td></td> </tr> </tbody> </table>	Day 1	Day 2	09:30-09:45 Welcome	09:30-10:30 Current HPC Architectures	09:45-10:15 Why learn about HPC?	10:30-10:30 Parallel Programming Libraries and Implementations	10:15-11:45 Using HPC machines	10:30-11:00 Practical	13:45-11:15 Coffee Break	11:00-11:30 (Coffee Break)	11:15-12:30 Practical	11:30-12:00 Practical	12:30-13:30 Building Blocks 1: CPUs, Memory, and Accelerators	12:00-12:30 Software support	13:30-14:30 Lunch	12:30-13:30 The Future of HPC and Wrap Up	14:30-14:30 Building Blocks 2: Operating Systems	12:00-14:30 Lunch with W-PC lunch	14:30-15:30 What does parallel computing mean?		15:30-15:30 Coffee Break		15:30-17:30 Practical		<h3>Support</h3> <ul style="list-style-type: none"> • Helpdesk <ul style="list-style-type: none"> - Email support@archer.ac.uk - via ARCHER SAFE http://www.archer.ac.uk/SAFE - phone: +44 (0) 131 550 5300 • By post, to: Liz Sim EPCC University of Edinburgh JCMB The King's Buildings Mayfield Road EDINBURGH EH9 3JZ • http://www.archer.ac.uk/community/techforum/
Day 1	Day 2																								
09:30-09:45 Welcome	09:30-10:30 Current HPC Architectures																								
09:45-10:15 Why learn about HPC?	10:30-10:30 Parallel Programming Libraries and Implementations																								
10:15-11:45 Using HPC machines	10:30-11:00 Practical																								
13:45-11:15 Coffee Break	11:00-11:30 (Coffee Break)																								
11:15-12:30 Practical	11:30-12:00 Practical																								
12:30-13:30 Building Blocks 1: CPUs, Memory, and Accelerators	12:00-12:30 Software support																								
13:30-14:30 Lunch	12:30-13:30 The Future of HPC and Wrap Up																								
14:30-14:30 Building Blocks 2: Operating Systems	12:00-14:30 Lunch with W-PC lunch																								
14:30-15:30 What does parallel computing mean?																									
15:30-15:30 Coffee Break																									
15:30-17:30 Practical																									

Training opportunities

- ARCHER Training (free to academics):
• <http://www.archer.ac.uk/training/>
- EPCC M.Sc. in HPC
• <http://www.epcc.ed.ac.uk/m-sc/>

ARCHER Training Schedule

Tools for Large-Scale Parallel Debugging and Profiling	29 April – 1 May	EPCC, Edinburgh
Advanced OpenMP	6-8 May	Oxford
Introduction to F95	12-13 May	Daresbury
Advanced OpenMP	6-8 May	Oxford
Programming the Xeon Phi	29-30 May	Bristol
Message Passing with MPI	2-4 July	EPCC, Edinburgh

Funding calls

- Embedded CSE support
 - Through a series of regular calls, Embedded CSE (eCSE) support provides funding to the ARCHER user community to develop software in a sustainable manner for running on ARCHER. Funding will enable the employment of a researcher or code developer to work specifically on the relevant software to enable new features or improve the performance of the code.
 - Apply for funding for development effort
 - Second call opened 1st April 2014
 - Closes in May 2014
 - Planned every 4 months
- See <http://www.archer.ac.uk> for details

Feedback and follow-up

- <http://www.archer.ac.uk/training/feedback/>
- Virtual Tutorial
 - Experts available
 - Likely to be 14th May 2014

High Performance Computing

What is it used for and why?

Overview

- What is it used for?
 - Drivers for HPC
 - Examples of usage
- Why do you need to learn the basics?
 - Hardware layout and structure matters
 - Serial computing is required for parallel computing
 - Appreciation of fundamentals will help you get more from HPC and scientific computing

What is HPC used for?

Drivers and examples

Why HPC?

- Scientific simulation and modelling drive the need for greater computing power.
- Single systems could not be made that had enough resource for the simulations needed.
 - Making faster single chip is difficult due to both physical limitations and cost.
 - Adding more memory to single chip is expensive and leads to complexity.
- Solution: parallel computing – divide up the work among numerous linked systems.

Modeling dinosaur gait
Dr Bill Sellers, University of Manchester

Dye-sensitized solar cells
F. Schiffrin and J. VandoVondele
University of Zurich

Fractal-based models of turbulent flows
Christos Vasiliadis & Sylvain Laizet,
Imperial College

ARCHER

ARCHER in a nutshell

- UK National Supercomputing Service
 - Peak performance of 1.65 PF
 - ARCHER is *not* a Linpack engine
 - #19 in November 2013 top 500 list fastest (known) computer in the UK
 - Designed to provide 3-4 times scientific throughput of HECToR
 - HECToR #50 in top 500 with 830 TFlop/s
- Cray XC30 Hardware
 - Nodes based on 2x Intel Ivy Bridge 12-core processors
 - 64GB (or 128GB) memory per node
 - 3008 nodes in total (72,162 cores)
 - Linked by Cray Aries interconnect (dragonfly topology)
- Cray Application Development Environment
 - Cray, Intel, GNU Compilers
 - Cray Parallel Libraries (MPI, SHMEM, PGAS)
 - DDT Debugger, Cray Performance Analysis Tools

What is it used for?

Category	Percentage
Chemistry/Materials Science	53%
Other/Unknown	28%
Earth Science/Climate	11%
Engineering	6%
Physics	2%

What is it used for?

Category	Percentage
Chemistry	40%
Environment	25%
Engineering	19%
Physics	1%
Support	1%
Life Sciences	3%
Materials	1%
European	1%
External	1%

Simulation software

Software	Percentage
GROMACS	5%
CASTEP	4%
DL_POLY	4%
MFTsim	6%
SANGA	2%
ChemShell	2%
NAMD	2%
CP2K	7%

The Fundamentals

Why do I need to know this?

Hardware Layout

- Understanding the different types of HPC hardware allows you to understand why some things better on one resource than another
- Allows you to choose the appropriate resource for your application
- Allows you to understand the ways to parallelise your serial application
- Gives you an appreciation of the parts that are important for performance

Serial Computing

- Without an understanding of how serial computing operates it is difficult to understand parallel computing
 - What are the factors that matter for serial computation
 - How does the compiler produce executable code?
 - Which bits are automatic and which parts do I have to worry about

Parallel Computing

- Parallel computing and HPC are intimately related
- Understanding the different parallel programming models allows you to understand how to use HPC resources effectively

Differences from Cloud computing

- Performance
 - Clouds usually use virtual machines which add an extra layer of software.
 - In cloud you often share hardware resource with other users – HPC access is usually exclusive.
- Tight-coupling
 - HPC parallel programming usually assumes that the separate processes are tightly coupled
 - Requires a low-latency, high-bandwidth communication system between tasks
 - Cloud usually does not usually have this
- Programming models
 - HPC use high-level compiled languages with extensive optimisation.
 - Cloud often based on interpreted/JIT.

HPC Layout and Use

Starting concepts

Typical HPC system layout

Typical Software Usage Flow

Batch Systems

Running calculations on HPC resources

Outline

- What is a batch system?
- How do I interact with the batch system
 - Job submission scripts
 - Interactive jobs
- Common batch systems
- Converting between different batch systems

Batch Systems

What are they and why are they used?

What is a batch system?

- A batch system controls access to the resources on a machine
- Used to ensure all users get a fair share of resources
 - As machine is usually oversubscribed
- Allows user to setup computational job, place it into batch queue and then log off machine
 - Job will be processed when there is space and time
 - Do not need to be continually logged-in for simulations to run
- Usually assumed that jobs are non-interactive
 - It runs for a time and produces results without intervention from the user
 - (Unlike interactive programs on a laptop.)

Reservation and Execution

- When you submit a job to a batch system you specify the resources you require:
 - Number of cores, job time.
- The batch system reserves a block of resources for you to use
- You can then use that block as you want, for example:
 - For a single job that spans all cores and full time
 - For multiple shorter jobs in sequence
 - For multiple smaller jobs running in parallel

Batch system flow

```

    graph LR
      A[Write Job Script] -- Job Submit Command --> B[Job Queued]
      B -- Job Delete Command --> C[Job Executes]
      C --> D[Job Finished]
      B --> E[Allocated Job ID]
      C --> F[Output Files]
      D --> G[Status]
    
```

Running calculations

Interacting with the batch system

Batch and interactive jobs

- Most resources allow both batch and interactive jobs to be run through the batch system
- Batch jobs are non-interactive
 - They run without user intervention and you collect the results at the end
 - Write a job submission script to run your job
- Interactive jobs allow you to use the resources interactively
 - For debugging/profiling
 - For visualization and data analysis
- How you run these types of jobs differs with batch system and site

Job submission scripts

- Contain:
 - Batch system options
 - Commands to run
- Example:

```

#!/bin/bash -login
PBS -N weather1
PBS -l select=171
PBS -l walltime=1:00:00
cd $PBS_O_WORKDIR
aprun -n 4096 ./weathersim
    
```

Annotations:

- select=171: how many nodes
- walltime=1:00:00: how long
- \$PBS_O_WORKDIR: which directory
- aprun: Parallel job launcher
- weathersim: Program name

Generic example
















```











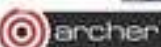
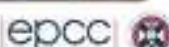
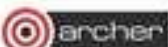





#!/bin/bash
set -v
set -l h_rt=10:
set -cd
set -pe mpi 4

mpirun -n $NSLOTS ./myprogram
    
```

Annotations:

- h_rt=10: how long
- l: which directory
- pe mpi 4: how many processors
- mpirun: Parallel job launcher
- myprogram: Program name

<h2>Common batch systems</h2>  	<h2>Batch systems</h2> <ul style="list-style-type: none"> - PBS, Torque - Grid Engine - SLURM - LSF – IBM Systems - LoadLeveler – IBM Systems  
<h2>Common concepts</h2> <ul style="list-style-type: none"> - Queues <ul style="list-style-type: none"> - Portions of machine and time constraints - Generally small numbers of defined queues - Generally specify: <ul style="list-style-type: none"> - Executable name - Account name - Maximum run time - Number of CPUs - Output file names/ directories   	<h2>Control programs</h2> <ul style="list-style-type: none"> - Monitor, submit, and delete programs: <ul style="list-style-type: none"> - qsub - qdel - qstat  
<h2>Migrating</h2> <p>Changing your scripts from one batch system to another</p>  	<h2>Conversion</h2> <ul style="list-style-type: none"> - Usually need to change the batch system options - Sometimes need to change the commands in the script <ul style="list-style-type: none"> - Particularly to different paths - Usually the order (logic) of the commands remains the same - There are some utilities that can help <ul style="list-style-type: none"> - Bolt – from EPCC, generates job submission scripts for a variety of batch systems/IPC resources: https://github.com/atumel-epcc/bolt  
<h2>Best practice</h2> <ul style="list-style-type: none"> - Run short tests using interactive jobs if possible - Once you are happy the setup works write a short test job script and run it - Finally, produce scripts for full production runs - Remember you have the full functionality of the Linux command line available in scripts <ul style="list-style-type: none"> - This allows for sophisticated scripts if you need them - Can automate a lot of tedious data analysis and transformation - ...be careful to test when moving, copying deleting important data – it is very easy to lose the results of a large simulation due to a typo (or unforeseen error) in a script.  	

<h2 style="text-align: center;">Building Blocks</h2> <p style="text-align: center;">CPUs, Memory and Accelerators</p> <div style="display: flex; justify-content: space-around; align-items: center;">   </div> <div style="display: flex; justify-content: space-around; align-items: center;">    </div>	<h3>Outline</h3> <ul style="list-style-type: none"> - Computer layout <ul style="list-style-type: none"> - CPU and Memory - What does performance depend on? - Limits to performance - Silicon-level parallelism <ul style="list-style-type: none"> - Single Instruction Multiple Data (SIMD/Vector) - Multicore - Symmetric Multi-threading (SMT) - Accelerators (GPGPU and Xeon Phi) <ul style="list-style-type: none"> - What are they good for? <div style="display: flex; justify-content: space-around; align-items: center;">   </div>
<h3>Computer Layout</h3> <p>How do all the bits interact and which ones matter?</p> <div style="display: flex; justify-content: space-around; align-items: center;">   </div>	<h3>Anatomy of a computer</h3>  <div style="display: flex; justify-content: space-around; align-items: center;">   </div>
<div style="display: flex; justify-content: space-around; align-items: center;">   </div> <div style="display: flex; justify-content: space-around; align-items: center;">   </div>	<h3>Performance</h3> <ul style="list-style-type: none"> - The performance (time to solution) on a single computer can depend on: <ul style="list-style-type: none"> - Clock speed – how fast the processor is - Floating point unit – how many operands can be operated on and what operations can be performed? - Memory latency – how fast can we access the data? - Memory bandwidth – how much data can we access in one go? - Input/Output (IO) to storage – how quickly can we access persistent data (files)? <div style="display: flex; justify-content: space-around; align-items: center;">   </div>
<h3>Performance (cont.)</h3> <ul style="list-style-type: none"> - Application performance often described as: <ul style="list-style-type: none"> - Compute bound - Memory bound - IO bound - (Communication bound – more on this later...) <div style="display: flex; justify-content: space-around; align-items: center;">   </div>	<h3>Limits to performance</h3> <ul style="list-style-type: none"> - Scientific simulation and modeling drive the need for greater computing power. - Single systems can not be made that had enough resource for the simulations needed. <ul style="list-style-type: none"> - Making faster single chip is difficult due to both physical limitations and cost - Adding more memory to single chip is expensive and leads to complexity. - Solution: parallel computing – divide up the work among numerous linked systems. <ul style="list-style-type: none"> - HPC has become synonymous with parallel computing <div style="display: flex; justify-content: space-around; align-items: center;">   </div>

Silicon-level parallelism

What does Moore's Law mean anyway?

Moore's Law

Microprocessor Transistor Counts 1971-2011 & Moore's Law

- Number of transistors doubles every 18 months
- What to do with all the extra silicon real estate?
 - More FPU, multicore and cache

Single Instruction Multiple Data (SIMD)

- For example, vector addition:





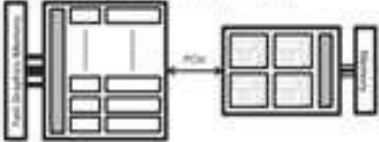











$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}$$

Symmetric Multi-threading (SMT)


















- Some hardware supports running more processes than there are physical cores
- Known as Symmetric Multi-threading (SMT) or hyperthreading
- Threading in this case can be a misnomer as it can refer to processes as well as threads
 - These are hardware threads, not software threads.
 - Intel Xeon supports 2-way SMT
 - IBM BlueGene/Q 4-way SMT

Multicore

Intel Xeon E5-2600 – 8 cores HT

<h3>Chip types and manufacturers</h3> <ul style="list-style-type: none"> - x86 – Intel and AMD <ul style="list-style-type: none"> - "PC" commodity processors, SIMD (SSE, AVX) FPU, multicore, SMT (rare), Intel currently dominate the HPC space. - Power – IBM <ul style="list-style-type: none"> - Used in high-end HPC, high clock speed (direct water cooled), SIMD FPU, multicore, SMT, not as important anymore. - PowerPC – IBM BlueGene <ul style="list-style-type: none"> - Low clock speed, SIMD FPU, multicore, high level of SMT. - SPARC – Fujitsu - ARM – Lots of manufacturers <ul style="list-style-type: none"> - Not yet relevant to HPC (weak FP Unit)  	<h3>Accelerators</h3> <p>Go-faster stripes</p>  
<h3>Anatomy</h3> <ul style="list-style-type: none"> - An Accelerator is a additional resource that can be used to off-load heavy floating-point calculation <ul style="list-style-type: none"> - It is an additional processing engine that is attached to the standard processor - It has its own floating point units and memory   	<h3>Summary - What is automatic?</h3> <ul style="list-style-type: none"> - Which features are managed by hardware/software and which does the user/programmer control? <ul style="list-style-type: none"> - Cache and memory – automatically managed - SIMD/Vector parallelism – automatically produced by compiler - SMT – automatically managed - Multicore parallelism – manually specified by the user - Use of accelerators – manually specified by the user  
<h2>Building Blocks</h2> <p>Operating Systems, Processes, Threads</p> 	<h3>Outline</h3> <ul style="list-style-type: none"> - What does an Operating System (OS) do? <ul style="list-style-type: none"> - OS types in HPC - The Command Line - Processes - Threads <ul style="list-style-type: none"> - Threads on accelerators - OS performance optimisation <ul style="list-style-type: none"> - Why is the OS bad for performance? - Approaches to improving OS performance  
<h3>Operating Systems</h3> <p>What do they do? Which ones are used for HPC?</p>  	<h3>Operating System (OS)</h3> <ul style="list-style-type: none"> - The OS is responsible for orchestrating access to the hardware by applications. <ul style="list-style-type: none"> - Which cores is an application running on? - How is the memory allocated and de-allocated? - How is the file-system accessed? - Who has authority to access which resources? - How do we deal with oversubscription (e.g. more applications running than cores available) - Running applications are controlled through the concepts of processes and threads.  

<h3>OS's for HPC</h3> <ul style="list-style-type: none"> - HPC sector is dominated by Linux (of various flavours) <ul style="list-style-type: none"> - Most HPC vendors modify a commercial Linux distro (RedHat or SUSE) and tailor to their own system. - Many commodity clusters run a free Linux distro (Scientific Linux is particularly popular). - Only IBM Power systems still use UNIX (AIX) <ul style="list-style-type: none"> - 11 HPC systems in the November 2013 Top500 list use UNIX - Windows HPC is used on a small number of HPC systems <ul style="list-style-type: none"> - 2 HPC systems in the November 2013 Top500 list use Windows 	<h3>The Command Line</h3> <ul style="list-style-type: none"> - HPC sector is dominated by Linux - Interaction is almost always through the Linux command line <ul style="list-style-type: none"> - Often a reasonably large barrier to new people adopting HPC. - For any serious use of HPC you will have to learn to use the command line. <ul style="list-style-type: none"> - Knowledge is often useful for using the command line on your own laptop/PC - You should also learn the basic operation of an in-terminal text editing program – 'vi' is probably the simplest to learn and is available everywhere.
<h3>Processes</h3>	<h3>Processes</h3> <ul style="list-style-type: none"> - Each application is a separate process in the OS <ul style="list-style-type: none"> - A process has its own memory space which is not accessible by other running process. - Each process is scheduled to run by the OS – it can be tied to a particular core or can be migrated between cores
<h3>Process Scheduling</h3> <ul style="list-style-type: none"> - The OS has responsibility for interrupting a process and granting the core to another process <ul style="list-style-type: none"> - Which process is granted access is determined by the scheduling policy - Interrupt happens at regular intervals (every 0.01seconds is typical) - Process selected should have processing work to do - Hardware can support scheduling of multiple processes <ul style="list-style-type: none"> - Known as Symmetric Multi-threading (SMT) - Usually appears to the OS as an additional core to use for scheduling - Process scheduling can be a hindrance to performance 	<h3>Threads</h3> <p>Sharing memory</p>
<h3>Threads</h3> <ul style="list-style-type: none"> - For many applications each process has a single thread... - ... but with the advent of multicore processors it is becoming more common for a process to contain multiple threads 	<h3>Threads (cont.)</h3> <ul style="list-style-type: none"> - All the threads in a process have access to the same memory <ul style="list-style-type: none"> - Can operate in parallel on the same data to speed up applications - Can have threads operating asynchronously (often used in GUIs) - OS scheduling policy is aware of threads <ul style="list-style-type: none"> - Usually scheduled as one thread per core but not a requirement - Switching between threads is usually a bit quicker than switching between processes

<h3>Threads and Accelerators</h3> <ul style="list-style-type: none"> - The Accelerator programming model generally requires a huge number of threads to provide efficient usage <ul style="list-style-type: none"> - Oversubscription of the accelerator by threads is encouraged - Hardware supports fast switching of execution of threads - As GPUs can have 1000's of computing elements, oversubscription can be difficult! - Threading is becoming more and more important on modern HPC machines  	<h3>OS Optimisation</h3> <p>How do vendors get performance?</p>  
<h3>Compute node OS</h3> <ul style="list-style-type: none"> - On the largest supercomputers the compute nodes often run an optimised OS to improve performance <ul style="list-style-type: none"> - Interactive (front-end) nodes usually run a full OS - Often means that you are cross-compiling - How is the OS optimised? <ul style="list-style-type: none"> - Remove features that are not needed (e.g. USB support) - Restrict scheduling flexibility and increase interrupt period - Remove support for virtual memory (paging)  	
<h2>Parallel Programming</h2> <p>Overview and Concepts</p>     	<h3>Outline</h3> <ul style="list-style-type: none"> - Why use parallel programming? - Parallel models for HPC <ul style="list-style-type: none"> - Shared memory (thread-based) - Message-passing (process-based) - Other models - Assessing parallel performance: scaling <ul style="list-style-type: none"> - Strong scaling - Weak scaling - Limits to parallelism <ul style="list-style-type: none"> - Amdahl's Law - Gustafson's Law  
<h3>Why use parallel programming?</h3> <p>It is harder than serial so why bother?</p>  	<h3>Drivers for parallel programming</h3> <ul style="list-style-type: none"> - Traditionally, the driver for parallel programming was that a single core alone could not provide the time-to-solution required for complex simulations <ul style="list-style-type: none"> - Multiple cores were tied together as a HPC machine - This is the origin of HPC and explains the symbiosis of HPC and parallel programming - Recently, due to the physical limits on the increase in power of single cores, the driver is due to the fact that all modern processors are parallel <ul style="list-style-type: none"> - In effect, parallel programming is required for all computing, not just HPC.  

Focus on HPC

- In HPC, the driver is the same as always
 - Need to run complex simulations with a reasonable time to solution
 - Single core or even single/multiple processors in a workstation do not provide the compute/memory/I/O performance required
- Solution is to harness the power of multiple cores/ memory/storage simultaneously
- In order to do this we require concepts to allow us to exploit the resources in a parallel manner
 - Hence, parallel programming
- Over time a number of different parallel programming models have emerged.

Parallel models

How can we write parallel programs

Embarrassingly Parallel/Task Farm

- Use large computers to run many small programs at once
 - run multiple copies of a program with different input parameters
 - Monte Carlo simulations
- Simple structure
 - Source: passes out tasks to workers
 - Workers: repeatedly process tasks
 - Sink: collates results
- No parallel programming required
- Cannot exploit multiple resources for individual programs

Shared-memory programming

- Shared memory programming is usually based on threads
 - Although some hardware/software allows processes to be programmed as if they share memory
 - Sometimes known as Symmetric Multi-processing (SMP) although this term is now a little old-fashioned
- Most often used for Data Parallelism
 - Each thread operates the same set of instructions on a separate portion of the data
- More difficult to use for Task Parallelism
 - Each thread performs a different set of instructions

Shared-memory concepts

- Threads "communicate" by having access to the same memory space
 - Any thread can alter any bit of data
 - No explicit communications between the parallel tasks

Advantages and disadvantages


- Advantages:
 - Conceptually simple
 - Usually minor modifications to existing code
 - Often very portable to different architectures
- Disadvantages:
 - Difficult to implement task-based parallelism – lack of flexibility
 - Often does not scale very well
 - Requires a large amount of inherent data parallelism (e.g. large arrays) to be effective
 - Can be surprisingly difficult to get good performance

Message-passing programming

- Message passing programming is process-based
- Processes running simultaneously communicate by exchanging messages
 - Messages can be 2-sided – both sender and receiver are involved in the process
 - Or they can be 1-sided – only the sender or receiver is involved
- Used for both data and task parallelism
 - In fact, most message passing programs employ a mixture of data and task parallelism

Message-passing concepts

- Each process does not have access to another process's memory
- Communication is usually explicit

<h3>Advantages and disadvantages</h3> <ul style="list-style-type: none"> - Advantages <ul style="list-style-type: none"> - Flexible – almost any parallel algorithm imaginable can be implemented - Scaling usually only limited by your choice of algorithm - Portable – MPI library is provided on all HPC platforms - Disadvantages <ul style="list-style-type: none"> - Parallel routines usually become part of the program due to explicit nature of communications <ul style="list-style-type: none"> - Can be a large task to retrofit into existing code - May not give optimum performance on shared-memory machines - Can be difficult to scale to very large numbers of processes (>100,000) due to overheads  	<h3>Scaling</h3> <p>Assessing parallel performance</p>  
<h3>Scaling</h3> <ul style="list-style-type: none"> - Scaling is how the performance of a parallel application changes as the number of parallel processes/threads is increased - There are two different types of scaling: <ul style="list-style-type: none"> - Strong Scaling – total problem size stays the same as the number of parallel elements increases - Weak Scaling – the problem size increases at the same rate as the number of parallel elements, keeping the amount of work per element the same - Strong scaling is generally more useful and more difficult to achieve than weak scaling  	<h3>Limits to parallel performance</h3> <p>How much can you gain from parallelism</p>  
<h3>Performance improvement</h3> <ul style="list-style-type: none"> - Two theoretical descriptions of the limits to parallel performance improvement are useful to consider: <ul style="list-style-type: none"> - Amdahl's Law – how much improvement is possible for a fixed problem size given more cores - Gustafson's Law – how much improvement is possible given a fixed amount of time and given more cores  	<h3>Amdahl's Law</h3> <ul style="list-style-type: none"> - Performance improvement from parallelisation is strongly limited by serial portion of the code <ul style="list-style-type: none"> - As the serial part's performance is not increased by adding more processes/threads - Based on having a fixed problem size $S(N) = \frac{1}{(1 - P) + \frac{P}{N}}$ <ul style="list-style-type: none"> - For example, 90% parallelisable (P=0.9): <ul style="list-style-type: none"> - S(10) = 0.4 - S(100) = 0.2  



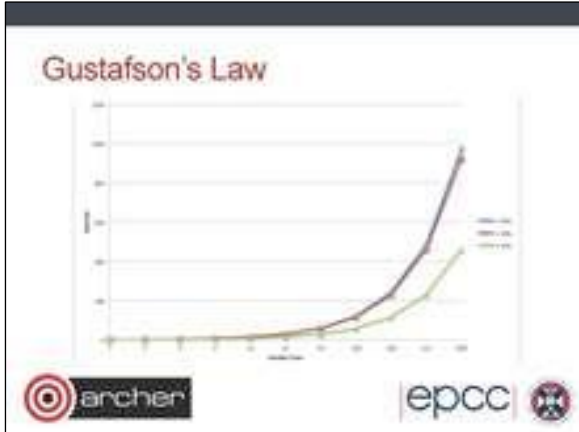
Gustafson's Law

- If you can increase the amount of work done by each process/task then the serial component will not dominate
- Increase the problem size to maintain scaling
- This can be in terms of adding extra complexity or increasing the overall problem size.

$$S(N) = N - (1 - P)(N - 1)$$

- For example, 90% parallelisable ($P=0.9$):
- $S(16) = 14.5$
- $S(1024) = 925.7$

archer epcc



HPC Architectures

Types of resource currently in use

EPSRC NERC COES OF THE ENVIRONMENT

archer CRAY THE SUPERCOMPUTER COMPANY epcc

Outline

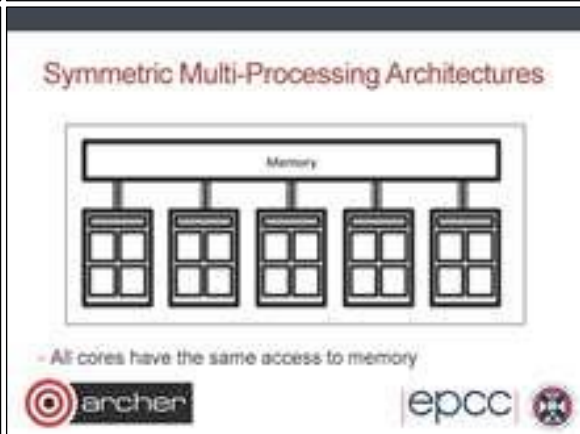
- Shared memory architectures
- Distributed memory architectures
- Distributed memory with shared-memory nodes
- Accelerators
- What is the difference between different Tiers?
 - Interconnect
 - Software
 - Job-size bias (capability)

archer epcc

Shared memory architectures

Simplest to use, hardest to build

archer epcc



Non-Uniform Memory Access Architectures

- Cores have faster/wider access to local memory

Shared-memory architectures

- Most computers are now shared memory machines due to multicore
- Some true SMP architectures...
 - e.g. BlueGene/Q nodes
- ...but most are NUMA
 - Program NUMA as if they are SMP – details are hidden from the user
- Difficult to build shared-memory systems with large core numbers (> 1024 cores)
 - Expensive and power hungry
 - Some systems manage by using software to provide shared-memory capability

Distributed memory architectures

Clusters and interconnects

Distributed-Memory Architectures

Distributed-memory architectures

- Each self-contained part is called a node.
- Almost all HPC machines are distributed memory in some way
 - Although they all tend to be shared-memory within a node.
- The performance of parallel programs often depends on the interconnect performance
 - Although once it is of a certain (high) quality, applications usually reveal themselves to be CPU, memory or IO bound
 - Low quality interconnects (e.g. 10Mbps – 1Gbps Ethernet) do not usually provide the performance required
 - Specialist interconnects are required to produce the largest supercomputers, e.g. Cray Aries, IBM BlueGene/Q

start on smaller systems





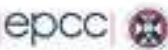



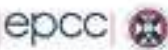



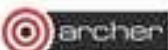

Distributed/shared memory hybrids

Almost everything now falls into this class





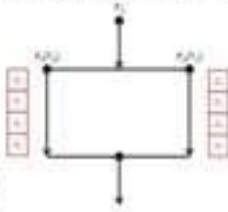





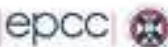






Hybrid Architectures











Hybrid architectures

- Almost all HPC machines fall in this class
- Most applications use a message-passing (MPI) model for programming
 - Usually use a single process per core
- Increased use of hybrid message-passing + shared memory (MPI+OpenMP) programming
 - Usually use 1 or more processes per NUMA region and then the appropriate number of shared-memory threads to occupy all the cores
- Placement of processes and threads can become complicated on these machines

<h3>Example: ARCHER</h3> <ul style="list-style-type: none"> - ARCHER has two 12-way multicore processors per node - Each 12-way processor is made up of two 6-core dies - Each node is a 24-core, shared-memory, NUMA machine   	<h3>Accelerators</h3> <p>How are they incorporated?</p>  
<h3>Including accelerators</h3> <ul style="list-style-type: none"> - Accelerators are usually incorporated into HPC machines using the hybrid architecture model <ul style="list-style-type: none"> - A number of accelerators per node - Nodes connected using interconnects - Communication from accelerator to accelerator depends on the hardware: <ul style="list-style-type: none"> - NVIDIA GPU support direct communication - AMD GPU have to communicate via CPU memory - Intel Xeon Phi communication via CPU memory - Communicating via CPU memory involves lots of extra copy operations and is usually very slow  	<h3>Comparison of types</h3> <p>What is the difference between different tiers?</p>  
<h3>HPC Facility Tiers</h3> <p>- HPC facilities are often spoken about as belonging to Tiers</p>   	<h3>Summary</h3> <ul style="list-style-type: none"> - Vast majority of HPC machines are shared-memory nodes linked by an interconnect. <ul style="list-style-type: none"> - Hybrid HPC architectures – combination of shared and distributed memory - Most are programmed using a pure MPI model (more later on MPI). <ul style="list-style-type: none"> - Does not really reflect the hardware layout - Shared HPC machines span a wide range of sizes: <ul style="list-style-type: none"> - From Tier 0 – Multi-petaflops (1 million cores) - To workstations with multiple CPUs (+ Accelerators)  

<h2 style="text-align: center;">Parallel Programming</h2> <p style="text-align: center;">Libraries and implementations</p> <div style="display: flex; justify-content: space-around; align-items: center;"> </div> <div style="display: flex; justify-content: space-around; align-items: center;"> </div>	<h3>Outline</h3> <ul style="list-style-type: none"> - MPI – distributed memory de-facto standard <ul style="list-style-type: none"> - Using MPI - OpenMP – shared memory de-facto standard <ul style="list-style-type: none"> - Using OpenMP - CUDA – GPGPU de-facto standard <ul style="list-style-type: none"> - Using CUDA - Others <ul style="list-style-type: none"> - Hybrid programming - Xeon Phi Programming - SHMEM - PGAS <div style="display: flex; justify-content: space-around; align-items: center;"> </div>
<h3>MPI Library</h3> <p>Distributed, message-passing programming</p> <div style="display: flex; justify-content: space-around; align-items: center;"> </div>	<h3>Message-passing concepts</h3> <div style="display: flex; justify-content: space-around; align-items: center;"> </div>
<h3>Explicit Parallelism</h3> <ul style="list-style-type: none"> - In message-passing all the parallelism is explicit <ul style="list-style-type: none"> - The program includes specific instructions for each communication - What to send or receive - When to send or receive - Synchronisation - It is up to the developer to design the parallel decomposition and implement it <ul style="list-style-type: none"> - How will you divide up the problem? - When will you need to communicate between processes? <div style="display: flex; justify-content: space-around; align-items: center;"> </div>	<h3>Message Passing Interface (MPI)</h3> <ul style="list-style-type: none"> - MPI is a portable library used for writing parallel programs using the message passing model <ul style="list-style-type: none"> - You can expect MPI to be available on any HPC platform you use - Based on a number of processes running independently in parallel <ul style="list-style-type: none"> - HPC resource provides a command to launch multiple processes simultaneously (e.g. mpirun, aprun) - There are a number of different implementations but all should support the MPI 2 standard <ul style="list-style-type: none"> - As with different compilers, there will be variations between implementations but all the features specified in the standard should work - Examples: MPICH2, OpenMPI <div style="display: flex; justify-content: space-around; align-items: center;"> </div>
<h3>Point-to-point communications</h3> <ul style="list-style-type: none"> - A message sent by one process and received by another - Both processes are actively involved in the communication – not necessarily at the same time - Wide variety of semantics provided: <ul style="list-style-type: none"> - Blocking vs. non-blocking - Ready vs. synchronous vs. buffered - Tags, communicators, wild-cards - Built-in and custom data-types - Can be used to implement any communication pattern <ul style="list-style-type: none"> - Collective operations, if applicable, can be more efficient <div style="display: flex; justify-content: space-around; align-items: center;"> </div>	<h3>Collective communications</h3> <ul style="list-style-type: none"> - A communication that involves all processes <ul style="list-style-type: none"> - "all" within a communicator, i.e. a defined sub-set of all processes - Each collective operation implements a particular communication pattern <ul style="list-style-type: none"> - Easier to program than lots of point-to-point messages - Should be more efficient than lots of point-to-point messages - Commonly used examples: <ul style="list-style-type: none"> - Broadcast - Gather - Reduce - AllToAll <div style="display: flex; justify-content: space-around; align-items: center;"> </div>

<h3>Example: MPI HelloWorld</h3> <pre>int main(int argc, char* argv[]) { int size,rank; MPI_Init(&argc, &argv); MPI_Comm_size(MPI_COMM_WORLD, &size); MPI_Comm_rank(MPI_COMM_WORLD, &rank); printf("Hello world - I'm rank %d of %d\n", rank, size); MPI_Finalize(); return 0; }</pre>  	<h3>OpenMP</h3> <p>Shared-memory parallelism using directives</p>  
<h3>Shared-memory concepts</h3> <ul style="list-style-type: none"> - Threads "communicate" by having access to the same memory space <ul style="list-style-type: none"> - Any thread can alter any bit of data - No explicit communications between the parallel tasks   	<h3>OpenMP</h3> <ul style="list-style-type: none"> - OpenMP is an Application Program Interface (API) for shared memory programming <ul style="list-style-type: none"> - You can expect OpenMP to be supported by all compilers on all HPC platforms - Not a library interface like MPI <ul style="list-style-type: none"> - You interact through directives in your program source rather than calling functions/subroutines - Parallelism is less explicit than MPI <ul style="list-style-type: none"> - You specify which parts of the program you want to parallelise and the compiler produces a parallel executable  
<h3>Loop-based parallelism</h3> <ul style="list-style-type: none"> - The most common form of OpenMP parallelism is to parallelise the work in a loop <ul style="list-style-type: none"> - The OpenMP directives tell the compiler to divide the iterations of the loop between the threads <pre>#pragma omp parallel shared(a,b,c,chunk) private(i) { #pragma omp for schedule(dynamic,chunk) nowait for (i=0; i < N; i++) { c[i] = a[i] + b[i]; } }</pre>  	<h3>CUDA</h3> <p>Programming GPGPU Accelerators</p>  
<h3>CUDA</h3> <ul style="list-style-type: none"> - CUDA is an Application Program Interface (API) for programming NVIDIA GPU accelerators <ul style="list-style-type: none"> - Proprietary software provided by NVIDIA. Should be available on all systems with NVIDIA GPU accelerators - Write GPU specific functions called kernels - Launch kernels using syntax within standard C programs - Includes functions to shift data between CPU and GPU memory - Similar to OpenMP programming in many ways in that the parallelism is implicit in the kernel design and launch - More recent versions of CUDA include ways to communicate directly between multiple GPU accelerators (GPUdirect)  	<h3>Example:</h3> <pre>// CUDA kernel. Each thread takes care of one element of c __global__ void vecAdd(double *a, double *b, double *c, int n) { // set our global thread ID int id = blockIdx.x*blockDim.x+threadIdx.x; // Make sure we do not go out of bounds if (id < n) c[id] = a[id] + b[id]; } // Called with vecAdd(<gridDim, blockDim>>(A, B, C, N);</pre>  

<h2>Others</h2> <p>Niche and future implementations</p>  	<h2>Other parallel implementations</h2> <ul style="list-style-type: none"> - Partitioned Global Address Space (PGAS) <ul style="list-style-type: none"> - Coarray Fortran, Unified Parallel C, Chapel - Cray SHMEM, OpenSHMEM <ul style="list-style-type: none"> - Single-sided communication library - OpenACC <ul style="list-style-type: none"> - Directive-based approach for programming accelerators  
<h2>Example: Running Parallel Programs</h2> <ul style="list-style-type: none"> - Here is a simple example of a PBS script for that: <ul style="list-style-type: none"> - sets the shell to /bin/bash - names the job "Weather1" - limits the run time of the job to one wall hour - and then runs the MPI executable ./weather1m on 4096 cores <pre> #!/bin/bash -login PBS -N weather1 PBS -l select=171 PBS -l walltime=1:00:00 if \$PBS_0_MKDIR aprun -n 4096 ./weather1m </pre>  	<h2>Summary</h2>  
<h2>Parallel Implementations</h2> <ul style="list-style-type: none"> - Distributed memory programmed using MPI - Shared memory programmed using OpenMP - GPU accelerators programmed using CUDA - Hybrid programming approaches (e.g. MPI/OpenMP) are becoming more common <ul style="list-style-type: none"> - They match the hardware layout more closely - A number of other, more experimental approaches are available  	

<h2 style="text-align: center;">Compilers</h2> <p style="text-align: center;">Algorithms to executables</p> <div style="display: flex; justify-content: space-around; align-items: center;"> </div> <div style="display: flex; justify-content: space-around; align-items: center;"> </div>	<h3>Outline</h3> <ul style="list-style-type: none"> - What does compiling mean? <ul style="list-style-type: none"> - Libraries - Anatomy of a compiler - Compiler "optimisations" - Can the compiler parallelise my code? - Why are there differences in compilers? <div style="display: flex; justify-content: space-between; align-items: center;"> </div>
<h3>Compiling</h3> <p>What does compiling mean?</p> <div style="display: flex; justify-content: space-between; align-items: center;"> </div>	<h3>Compiling Overview</h3> <ul style="list-style-type: none"> - HPC programs are usually written in a high-level, human-readable language. <ul style="list-style-type: none"> - Almost always Fortran or C (99% of all HPC applications) - Occasionally C++, rarely something else. - Processors execute machine code (via instruction sets) - Compilers convert high-level source code into machine code. <ul style="list-style-type: none"> - Also incorporate functionality from external libraries - Usually try to optimise the code produced so that it runs as fast as possible on the processors <div style="display: flex; justify-content: space-between; align-items: center;"> </div>
<h3>Libraries</h3> <ul style="list-style-type: none"> - Libraries provide functionality that is common across multiple programs. <ul style="list-style-type: none"> - Low level – e.g. filesystem access. Usually not interesting to users - Optimised numerical operations – e.g. linear algebra, Fourier transformations - Communications and parallelism – e.g. Message Passing Interface (MPI), OpenMP - The compiler combines the code in these libraries with the code generated from the user's program to produce the final executable. <ul style="list-style-type: none"> - Linking at run time is also possible – known as dynamic linking (or shared libraries). <div style="display: flex; justify-content: space-between; align-items: center;"> </div>	<h3>Anatomy of a compiler</h3> <p>How does it actually work?</p> <div style="display: flex; justify-content: space-between; align-items: center;"> </div>
<h3>Compiler Flow</h3> <pre> graph TD SC[Source Code Files] --> SC_box[Source Code] L[Libraries] --> MC[Machine Code] SC_box -- Compile --> OF[Object Files (*.o)] OF -- Link --> FA[Full Application] FA --> EB[Executable Binary File] </pre> <div style="display: flex; justify-content: space-between; align-items: center;"> </div>	<h3>Compile Stage</h3> <ul style="list-style-type: none"> - Transforms high level source to machine code <ul style="list-style-type: none"> - Produces object files – usually one object file per source file - Actually consists of a number of sub-stages <ul style="list-style-type: none"> - Details are beyond this course - Optimisations are performed at this stage <ul style="list-style-type: none"> - Move on optimisations later <div style="display: flex; justify-content: space-between; align-items: center;"> </div>

<h2>Link Stage</h2> <ul style="list-style-type: none"> - Object files are combined (linked) to produce the actual application <ul style="list-style-type: none"> - Application is an executable binary file - Any library code required by the application is also linked at this stage - Two forms of linking: <ul style="list-style-type: none"> - Static - All code is combined into a single executable file - Dynamic - Code from libraries is not combined into executable file, instead this code is dynamically include when the executable is run.  	<h2>Compiler optimisations</h2> <p>What do they do? When should/shouldn't I use them?</p>  
<h2>Optimisation</h2> <ul style="list-style-type: none"> - Compiler will try to alter produced code so it runs more quickly <ul style="list-style-type: none"> - This can be done at a number of levels and can include the reordering of operators - Note: although these are called optimisations, this is a misnomer <ul style="list-style-type: none"> - Resulting code is never optimal - Seldom any iterative process - Seldom any attempt to quantify effect of any transformations - Usually a predetermined sequence of transformations that is known to produce performance gains for some codes.  	<h2>Optimisation strategies</h2> <ul style="list-style-type: none"> - Loop index reordering (to match memory layout) - Loop unrolling - Use of fast mathematical operators - Function inlining (avoiding a function call) - Operation reordering to allow for cache reuse  
<h2>When to use optimisation</h2> <ul style="list-style-type: none"> - Simple answer: always - You should always use the performance gains given by optimisation - If you are debugging then you usually switch optimisation off to ensure that the statements are being executed in the order you specified - If you suspect that compiler optimisations are causing a problem you can turn them off gradually <ul style="list-style-type: none"> - All good compilers allow the specification of a range of optimisation levels so you can turn it off gradually  	<h2>Compilers and parallelisation</h2> <p>Can compilers parallelise my code?</p>  
<h2>Compiler parallelisation</h2> <ul style="list-style-type: none"> - Compilers can produce parallel (or vector) instructions <ul style="list-style-type: none"> - Makes use of the SIMD instructions on the core's floating point unit. - However, they cannot produce the general, high-level parallelism required for scaling on multiple cores <ul style="list-style-type: none"> - Compilers do not have the holistic view required to produce this level of parallelism - Data parallelism is usually easier to produce automatically than task parallelism - Attempts have been made to automate this but with limited success so far.  	<h2>Different compilers</h2> <p>Why are there differences between compilers?</p>  

Standards and implementations

- Although standards exist they cannot cover all cases and contain ambiguities
- When the standard is not clear then it is up to the compiler architect to select the behaviour
 - Differences exist between compiler implementations




HPC Future Look

Exascale and Challenges







Outline

- Future architectures
 - Exascale initiatives
 - Processors
 - Memory
 - Impacts on performance
- Software challenges
 - Parallelism and scaling
 - New algorithms
 - What about software that does not scale?
 - Impact for standard computing




Future architectures

What will HPC machines look like?




What will future systems look like?

	2014	2018	2022
System Perf	35 PFlops	100-200 PFlops	1 EFlops
Memory	1 PB	5 PB	10 PB
Node Perf	200 GFlops	400 GFlops	1-10 TFlops
Concurrency	32	O(100)	O(1000)
Interconnect BW	40 GB/s	100 GB/s	200-400 GB/s
Nodes	100,000	500,000	O(MBcm)
IO	2 TB/s	10 TB/s	20 TB/s
MTTl	Days	Days	O(1 Day)
Power	10 MW	10 MW	20 MW




Processors














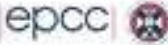
- More Floating-Point compute power per processor
 - Only exploit this power via parallelism
 - Lots of low power compute elements combined in some way




Memory

- Will be packaged with processor
 - Increases power efficiency, speed and bandwidth
 - ... at the cost of smaller memory per core




<h3>System on a chip</h3> <ul style="list-style-type: none"> - Instead of separate: <ul style="list-style-type: none"> - Processor - Memory - Network interface - Combined system package where all these things are included in one manufactured part <ul style="list-style-type: none"> - This is the only way to improve power efficiency - Less scope for customisation - If you need more memory than in package you will have to have levels of memory hierarchies  	<h3>Software challenges</h3> <p>What does software need to do to exploit future HPC?</p>  
<h3>What does this mean for applications?</h3> <ul style="list-style-type: none"> - The future of HPC (as for everyone else): <ul style="list-style-type: none"> - Lots of cores per node (CPU + co-processor) - Little memory per core - Lots of compute power per network interface - The balance of compute to communication power and compute to memory are both radically different to now - Must exploit parallelism at all levels - Must exploit memory hierarchy efficiently  	<h3>Algorithms</h3> <ul style="list-style-type: none"> - For many problems new algorithms will be needed - May not be optimal but contain more scope for parallelisation - Mixed-precision will become more important  
<h3>Applications that do not scale</h3> <ul style="list-style-type: none"> - The good news is that if you do not need to be able to treat larger/more-complex problems then you can access more of current resource size <ul style="list-style-type: none"> - May be caught out by decrease in memory per core - Options to scale in trivial-parallel way: increase sampling, use more sophisticated statistical techniques - This may well be the best route for many simulations  	<h3>Impact on standard computing</h3> <p>What does this mean for my workstation?</p>  
<h3>Parallel everywhere</h3> <ul style="list-style-type: none"> - All current computers are parallel <ul style="list-style-type: none"> - From supercomputers all the way down to mobile phones - Most parallelism is task-based on 4-8 cores – each application (task) runs on an individual core - In the future: <ul style="list-style-type: none"> - More parallelism per device – 10s to 100s cores running at lower clock speeds - All applications will have to be parallel - Parallel programming skills will be required for all application development - More system on a chip – more things will be packaged together  	

3 Training Material “HPC Computer Aided Engineering (CINECA)”

This training was developed by CINECA and was held June 16th -18th 2014. The material is available through the shared document server [6].

Slide 1: Title Slide

SCAI

HPC Computer Aided Engineering @ CINECA

Raffaele Ponzini Ph.D.
CINECA
SuperComputing Applications
and Innovation Department – SCAI

16-18 June 2014
Segrate (MI), Italy

PRACE CINECA

Slide 2: Outline

SCAI

Outline

- Analysis of student background
- Overview and timing of the school
- Timing of the day

PRACE CINECA

Slide 3: Students background

SCAI

Students background

About 5 minutes per student:

- Who/Where/Why
- CAE tools
- CAE background

PRACE CINECA

Slide 4: Overview and timing of the school

SCAI

Overview and timing of the school

Day	Time	Activity
16 June	09:00 - 10:30	Registration and Welcome
	10:30 - 12:00	Analysis of student background
17 June	09:00 - 10:30	Overview and timing of the school
	10:30 - 12:00	Timing of the day
18 June	09:00 - 10:30	Final remarks and closing
	10:30 - 12:00	Free time

PRACE CINECA

<p>SCAI</p> <h3>Overview and timing of the school</h3> <p>School Contents</p> <ul style="list-style-type: none"> Lectures Tutorials Invited Lectures <p>PRACE CINECA</p>	<p>SCAI</p> <h3>Overview and timing of the school</h3> <p>Raffaele Porzini (CINECA). Raffaele Porzini has a PhD (sum. Laude) and a master's degree in Bioengineering from the Politecnico di Milano. His research interests include computational models in aerodynamics, and scientific visualization. Since 2003 he worked as a member of the High Performance Computing group of CINECA for the management of fluid dynamics computational codes. His working domain includes the teaching C++ and Python programming for scientific applications. Starting from September 2012 he's working at CINECA within the Supercomputing Applications and Innovation Department.</p> <p>Roberto Punt (SCS), B.Sc. in Aerospace Engineering at Politecnico di Milano; M.Sc. in Aeronautical Engineering at Politecnico di Milano with specialization in Aerodynamics. From 2014 working at SCS as CFD application specialist.</p> <p>Francesco Pasqua (CINECA), B.Sc. in Aerospace Engineering at Politecnico di Milano; M.Sc. in Aeronautical Engineering at Politecnico di Milano with specialization in Aerodynamics. LISA Scholarship holder at CINECA.</p> <p>PRACE CINECA</p>
<p>SCAI</p> <h3>Overview and timing of the school</h3> <p>Alessandro Chiorini (SCS), Responsible for High Performance Computing (HPC) services for industrial customers: applications in manufacturing, plasma and big data.</p> <p>Umberto Martignetti (POLITEC), Associate Professor at Politecnico di Torino, CINECA's Department of Mechanical and Aerospace and member of the Board of Biomedical Engineering. His working domain includes CFD analysis of medical and implantable devices.</p> <p>PRACE CINECA</p>	<p>SCAI</p> <h3>Timing of the day</h3> <p>PRACE CINECA</p>
<p>SCAI</p> <h2>HPC Computer Aided Engineering @ CINECA</h2> <p>Raffaele Porzini Ph.D. CINECA SuperComputing Applications and innovation Department - SCAI</p> <p>16-18 and 20-21</p> <p>PRACE CINECA</p>	<p>SCAI</p> <h3>Outline</h3> <ul style="list-style-type: none"> Computer Aided Engineering Engineering tools: Experimental vs Numerical HPC Platforms and CAE applications CAE applications at CINECA <p>PRACE CINECA</p>
<p>SCAI</p> <h3>Computer Aided Engineering</h3> <p>Computer-aided engineering (CAE) is the broad usage of computer softwares to aid in engineering analysis tasks. It includes:</p> <ul style="list-style-type: none"> Finite Element Analysis (FEA) Computational Fluid Dynamics (CFD) Multi-body dynamics (MBD) Optimization <p>Software tools that have been developed to support these activities are considered CAE tools.</p> <p>CAE tools are being used, for example, to analyze the robustness and performance of components and assemblies. The term encompasses simulation, validation, and optimization of products and manufacturing tools. In the future, CAE systems will be major providers of information to help support design teams in decision making [...]</p> <p>PRACE CINECA</p>	<p>SCAI</p> <h3>Computer Aided Engineering</h3> <p>CAE areas covered include:</p> <ul style="list-style-type: none"> Stress analysis on components and assemblies using FEA (Finite Element Analysis); Thermal and fluid flow analysis Computational fluid dynamics (CFD); Multi-body Dynamics (MBD) & Kinematics; Analysis tools for process simulation for operations such as casting, molding, and die press forming; Optimization of the product or process <p>PRACE CINECA</p>

SCAI

Computer Aided Engineering

In general, there are three phases in any computer-aided engineering task:



- **Pre-processing** - defining the geometry model, the physical model and the boundary conditions
- **Computing** (usually performed on high-powered computers (HPC))
- **Post-processing** of results (using scientific visualization tools & techniques)

This cycle is iterated, often many times, either manually or with the use of automation techniques or using optimization software.




SCAI

Computer Aided Engineering

SCAI

Engineering tools: Experimental & Numerical

- Wind-tunnel
- Towing tank
- Biological systems




SCAI

Engineering tools: Experimental & Numerical

Wind tunnels are large tubes with air moving inside. The tunnels are used to copy the actions of an object in flight. Researchers use wind tunnels to learn more about how an aircraft will fly. NASA uses wind tunnels to test scale models of aircraft and spacecraft. Some wind tunnels are big enough to hold full-size versions of vehicles. The wind tunnel moves air around an object, making it seem like the object is really flying.



How Do Wind Tunnels Work?

Most of the time, powerful fans move air through the tube. The object to be tested is balanced in the tunnel so that it will not move. The object can be a small model of a vehicle. It can be just a piece of a vehicle. It can be a full-size aircraft or spacecraft. It can even be a common object like a tennis ball. The air moving around the still object shows what would happen if the object were moving through the air. How the air moves can be studied in different ways. Smoke or dye can be placed in the air and can be seen as it moves. Threads can be attached to the object to show how the air is moving. Special instruments are often used to measure the force of the air on the object.




SCAI

Engineering tools: Experimental

SCAI

Engineering tools: Experimental

[NACA PLATE](#) SCAI Home - available content / Publications - publications / NACA PLATE

[Turbulence](#) SCAI Home - available content / Publications - publications / Turbulence

[The](#) SCAI Home - available content / Publications - publications / The




SCAI

Engineering tools: Experimental




SCAI

Engineering tools: Experimental

[Flow visualization](#) SCAI Home - available content / Publications - publications / Flow visualization

[Hydrofoil](#) SCAI Home - available content / Publications - publications / Hydrofoil

[The](#) SCAI Home - available content / Publications - publications / The




SCAI

HPC Platforms and CAE applications

The American Food and Drug Administration has promoted a wide interlaboratory study to assess the usability of CFD for implantable design in hemodynamics (2012 Assessment of CFD Performance in Simulations of an Idealized Medical Device: Results of FDA's First Computational Inter-laboratory Study)

SCAI

HPC Platforms and CAE applications

Once virtualization is recognized to be cost-effective and reliable new trends are incoming for R&D in engineering applications:

- Several CAE software vendor is now pushing towards CFD analysis (HyperWorks and Abaqus included CFD add-on in their application platform)
- Visualization is a key turn point thanks to very rich datasets 3D+D and is now of great interest for all our vendor
- Automation for app production
- Data analysis MKL by Ansys and Tecplot360 for advanced data analysis
- Optimization: Design Of Experiment, adjoint solver
- Cloud computing

Computational platforms today must be considered a commodity, always available, scalable as needed in order to face CAE issues in a proper way

SCAI

CAE application at CINECA

CINECA is the largest computing center in Italy, as a part for joint Consortium, made up of 60 Super-Computers, three national institutions and the Ministry of Education and Research.

SCAI Super-Computing Applications and Secondary in the High Performance Computing Department of CINECA. The Mission of SCAI is to accelerate the scientific discovery by providing high-performance computing resources with integrated and storage systems and tools and HPC services and expertise all along, aiming to develop and provide industrial and scientific services related to high-performance computing for the Italian and European research community.

SCAI

CAE application at CINECA

- Top500 ranked HPC infrastructure**
Paves a record #13 at the latest Top500 list (Nov. 13)
- Excellent ranked HPC infrastructure**
Cineca is ranked #10 in the latest Green500 list (Nov. 13)
- Advanced CAE applications**
Advanced CAE applications are supported by the infrastructure, including:
 - External aerodynamics
 - Marine hydrodynamics
 - Internal aerodynamics
 - Structural analysis
 - Fluid-structure interaction
 - Biomechanics
 - Hemodynamics

SCAI

CAE application at CINECA

- Network capabilities**
Performance based, bandwidth aware and adaptive network for specific needs and settings
- Cloud capabilities**
Highly automated, flexible, scalable and virtualized cloud computing infrastructure, supporting distributed and multi-user software
- Advanced CAE applications**
Advanced CAE applications are supported by the infrastructure, including:
 - External aerodynamics
 - Marine hydrodynamics
 - Internal aerodynamics
 - Structural analysis
 - Fluid-structure interaction
 - Biomechanics
 - Hemodynamics
- SCAI Super-Computing Applications and Secondary in the High Performance Computing Department of CINECA**

SCAI

Main scope of the course

During the course we will focus on:

- CFD applications: external aerodynamics, marine hydrodynamics
- Open-source tools: SnappyHexMesh, OpenFOAM (OpenCFD Ltd), ParaView (Kitware Inc.)
- HPC platforms: overview of distributed infrastructure and added value to speed-up the design process and time-to-result
- Productivity: automation, Design Of Experiment, Optimization

Intent of the course is to give a correct, technical and scientific definition of the topics but using a plain speak (without oversimplifying) in order to give to the students a taste of the practical issues encountered when dealing with this kind of problem in day by day work.

<h2>HPC Computer Aided Engineering @ CINECA</h2> <p>Raffaello Porzini Ph.D. CINECA SuperComputing Applications and Innovation Department - SCAI</p>	<h2>Outline</h2> <ul style="list-style-type: none"> • What is CFD • CFD main concepts • CFD main definitions • How should I use it 			
<h2>Computational Fluid Dynamics: CFD</h2> <ul style="list-style-type: none"> • Fluid dynamics: physics of fluids • Computational: numerical engine involved in solving the equation describing the motion of the fluid by means of iterative methods <p>There is a strong interplay between math concepts, physics knowledge and technological tools and environments used to implement a CFD model.</p> <p>Demanding task:</p> <ul style="list-style-type: none"> • No general rules for specific CFD model setup • Need of a priori knowledge of the fluid behavior • Need of a priori knowledge of the physics involved by the problem • If possible experimental (or theoretical) data to validate CFD results 	<h2>CFD workflow</h2>			
<h2>CFD: general approach</h2> <p>A very rough definition to explain how CFD works can be as follow:</p> <p>"CFD applies the principles of conservation of mass and momentum according to the geometry and the mechanical properties of the fluid involved by the problem to be able to get information on instantaneous velocity and pressure distributions."</p> <p>Three pylons:</p> <ol style="list-style-type: none"> 1. Geometry description 2. Fluid properties 3. Fluid conditions 	<h2>CFD: general idea</h2> <p>Analytical (Exact) solution (integrals) is not available for the specific geometry (CAD) but thanks to iterative (Automated) methods a numerical (Approximated) solution of the problem can be found</p> <p>Continuum \rightarrow Discrete Math \rightarrow Numeric Numeric \leftrightarrow Technology (HW / SW)</p> <p>When moving from math to numeric we need to understand that there is a counterpart version of each concept and that once we move to numeric also technical issues comes in to play</p>			
<h2>CFD: general approach</h2> <p>When moving from math to numeric we need to understand that there is a counterpart version of each concept and that once we move to numeric also technical issues comes in to play</p> <table border="0"> <tr> <td> Math <input type="checkbox"/> Convergence <input type="checkbox"/> Consistency <input type="checkbox"/> Stability </td> <td style="text-align: center;"> </td> <td> Numeric <input type="checkbox"/> Boundedness <input type="checkbox"/> Conservativeness <input type="checkbox"/> Transportiveness </td> </tr> </table>	Math <input type="checkbox"/> Convergence <input type="checkbox"/> Consistency <input type="checkbox"/> Stability		Numeric <input type="checkbox"/> Boundedness <input type="checkbox"/> Conservativeness <input type="checkbox"/> Transportiveness	<h2>Problem Discretization</h2> <p>Theory \leftrightarrow Practice</p> <p>Theory if #cells (used to discretize the continuum) $\rightarrow \infty$ THEN the numerical solution \rightarrow "exact" and this will be independent from the numerical system chosen</p> <p>Practice if #cells is finite THEN the numerical solution \rightarrow "OK" and this will be dependent from the properties of the numerical system chosen</p> <p>This part of the workflow is the so-called pre-processing and/or meshing process</p>
Math <input type="checkbox"/> Convergence <input type="checkbox"/> Consistency <input type="checkbox"/> Stability		Numeric <input type="checkbox"/> Boundedness <input type="checkbox"/> Conservativeness <input type="checkbox"/> Transportiveness		

SCAI

CAD-Mesh-Cell

Computational domain Meshly discretized domain Cell

FRACS

SCAI

CAD-Mesh-Cell

- Domain is discretized into a finite set of control volumes or cells. The discretized domain is called the "grid" or the "mesh"
- General conservation equations for mass and momentum are discretized into algebraic equations and solved for each and all cells in the discretization
- (physic → math → numeric → SW)

FRACS

SCAI

Finite volume method

The finite-volume method (FVM) is a method for representing and evaluating partial differential equations in the form of algebraic equations [LeVeque, 2002; Toro, 1999]. Similar to the finite difference method or finite element method, values are calculated at discrete places on a meshed geometry. "Finite volume" refers to the small volume surrounding each node point on a mesh. In the finite volume method, volume integrals in a partial differential equation that contain a divergence term are converted to surface integrals, using the divergence theorem. These terms are then evaluated as fluxes at the surfaces of each finite volume. Because the flux entering a given volume is identical to that leaving the adjacent volume, these methods are conservative. Another advantage of the finite volume method is that it is easily formulated to allow for unstructured meshes. The method is used in many computational fluid dynamics packages. [Wikipedia]

FRACS

SCAI

Finite volume method

- In order to build a CFD model of a physical problem only the portion of your problem where the fluid is present need to be discretized (meshed).
- CFD models usually are used to study local phenomena related to a 3D problem so usually only a certain amount of your physical domain need to be included in the CFD model ("nearby")
- All other parts (solid or far away) are not of interest and are not included in the meshing process.
- Example Wind Tunnel Application: the solid object surrounded by the flowing air is not meshed except for his surface.
- All other surfaces are called numerical boundaries (or just boundaries) and can be described efficiently within the CFD model by understanding their main characteristics.

FRACS

SCAI

Finite volume method

In order to build a CFD model correctly you must select, according to your available experimental data or your a-priori knowledge, a coherent set of boundaries (so that you solve a problem that results in a unique solution)

Activities involved are:

1. identify the position of the boundaries in your problem
2. recognize what kind of boundary you are dealing with (inlets, outlets, walls, symmetry, cyclic...)
3. retrieve the information available from your physical profile for that boundary.

FRACS

SCAI

Finite volume method

Boundary conditions are a necessary part of the mathematical model. In fact Navier-Stokes equations and continuity equation in order to be solved need:

- initial conditions (starting point for the iterative process)
- boundary conditions (define the flow region problem)

In order to build a CFD model correctly you must select, according to your available experimental data or your a-priori knowledge, a coherent set of boundaries (so that you solve a problem that results in a unique solution)

Activities involved are:

1. identify the position of the boundaries in your problem
2. recognize what kind of boundary you are dealing with (inlets, outlets, walls, symmetry, cyclic...)
3. retrieve the information available from your physical problem for that boundary.

FRACS

SCAI

Neumann and Dirichlet boundary conditions

- Dirichlet boundary condition: value of velocity at a boundary $u(x) = u_{boundary}$
- Neumann boundary condition: gradient normal to the boundary of a velocity at the boundary. $(\nabla u) \cdot n = constant$

FRACS

SCAI

Pressure BC convention

- Pressure boundary conditions require static gauge pressure inputs:

$$P_{static} = P_{abs} - P_{operating}$$

- An operating pressure input is necessary to define the pressure (0 will work well).

FRACS

<p>SCAI</p> <h3>Pressure outlet boundary</h3> <ul style="list-style-type: none"> • Pressure outlet BC can be used in presence of velocity BC at the inlet • Usually in a zero-stress condition is applied over multiple exits (if not known specifically from a-priori knowledge of the problem) • The geometry is driving the pressure distribution and the flow repartition • The static pressure is assumed to be constant over the outlet <p>FRACS</p>	<p>SCAI</p> <h3>Velocity inlets</h3> <ul style="list-style-type: none"> • A flat profile is selected by default (if not known specifically from a-priori knowledge of the problem) • Is very often used as BC to set a known flow-rate (working condition) • Thanks to experimental data is possible to obtain spatial and temporal distribution... <p>FRACS</p>
<p>SCAI</p> <h3>Wall boundaries</h3> <ul style="list-style-type: none"> • Used to bound the limit between fluid and solid regions in our problem • Settings at the wall: <ul style="list-style-type: none"> - Tangential fluid velocity equal to wall velocity (usually zero) - Normal velocity component is set to be zero <p>FRACS</p>	<p>SCAI</p> <h3>Symmetry</h3> <ul style="list-style-type: none"> • When your problem present a symmetry plane in the geometry and in the flow field this kind of boundary can be used to reduce computational cost of the CFD model • General characteristics at the symmetry plane <ul style="list-style-type: none"> - normal velocity equal to zero - normal gradients of all variables equal to zero <p>FRACS</p>
<p>SCAI</p> <h3>Material properties</h3> <ul style="list-style-type: none"> • The physical property of the fluid must be given. • For Newtonian incompressible fluid we have to provide only density and viscosity • Newtonian? • Incompressible? <p>FRACS</p>	<p>SCAI</p> <h3>Density</h3> <ul style="list-style-type: none"> • Density is a fluid property and is defined as: $\rho = \text{Mass/Volume}$ • Usually we consider for water the density value at $T=300\text{ K}$ and $P=105\text{ Pa}$: $\rho = 1000\text{ [Kg/m}^3\text{]}$ <p>FRACS</p>
<p>SCAI</p> <h3>Viscosity</h3> <ul style="list-style-type: none"> • For viscous flow if the relationship between viscous forces (tangential component) and velocity gradient is linear then the fluid is called Newtonian and the slope of the line is a measure of a fluid property called viscosity (dynamic): $\tau = \mu \cdot dv/dx$ • Also a cinematic viscosity can be defined by: $\nu = \mu/\rho$ <p>FRACS</p>	<p>SCAI</p> <h3>Reynolds number</h3> <p>The Reynolds number Re is defined as:</p> $Re = \rho \cdot U \cdot L / \mu$ <p>Here:</p> <ul style="list-style-type: none"> L is a characteristic length (say D in tubes) U is the mean velocity over the section (Q/Area) density and viscosity are: ρ, μ <p>If $Re \gg 1$ the flow is dominated by inertia If $Re \ll 1$ the flow is dominated by viscous effects.</p> $Re = \frac{\text{convective inertia force}}{\text{viscous friction force}}$ <p>FRACS</p>

<p>SCAI</p> <h3>Flow classifications</h3> <p>Laminar vs. turbulent flow</p> <ul style="list-style-type: none"> - Laminar flow: fluid particles move in smooth, layered fashion (no substantial mixing of fluid occurs). - Turbulent flow: fluid particles move in a chaotic, "tangled" fashion (significant mixing of fluid occurs). <p>Steady vs. unsteady flow</p> <ul style="list-style-type: none"> - Steady flow: flow properties at any given point in space are constant in time, e.g. $\rho = \rho(x, y, z)$. - Unsteady flow: flow properties at any given point in space change with time, e.g. $\rho = \rho(x, y, z, t)$. <p>FRACS</p>	<p>SCAI</p> <h3>Incompressible vs. compressible flow</h3> <ul style="list-style-type: none"> - Incompressible flow: volume of a given fluid particle does not change. <ul style="list-style-type: none"> • Implies that density is constant everywhere. • Essentially valid for all liquid flows. - Compressible flow: volume of a given fluid particle can change with position. <ul style="list-style-type: none"> • Implies that density will vary throughout the flow field. <p>FRACS</p>
<p>SCAI</p> <h3>Single phase vs. multiphase flow & homogeneous vs. heterogeneous flow</h3> <ul style="list-style-type: none"> • Single phase flow: fluid flows without phase change (either liquid or gas). • Multiphase flow: multiple phases are present in the flow field (e.g. liquid-gas, liquid-solid, gas-solid). • Homogeneous flow: only one fluid material exists in the flow field. • Heterogeneous flow: multiple fluid/solid materials are present in the flow field (multi-species flows). <p>FRACS</p>	<p>SCAI</p> <h3>Working hypothesis in CFD for general fluids</h3> <ul style="list-style-type: none"> • Continuum hypothesis • Homogenous • Incompressible (density ρ is constant) • Isotropic (same behaviour in all directions) • Newtonian (viscosity (μ and λ) are constant and do not depend on the shear rate) <p>In physical models it is very hard to guarantee all these hypotheses (repeatability and calibration).</p> <p>FRACS</p>
<p>SCAI</p> <h3>Equations of conservation</h3> <p>Two general conservation equations:</p> <ol style="list-style-type: none"> 1. Mass (divergence free) 2. Momentum: Newton's second law: the change of momentum equals the sum of forces on a fluid particle  <p>Control-volume: mesh cell</p> <p>FRACS</p>	<p>SCAI</p> <h3>N-S eqn numerical issues</h3> <p>The set of strength over time (local acceleration) = Transport by convection + Pressure forces + Diffusion (viscous forces) + Source terms</p> <ol style="list-style-type: none"> 1. Non linear 2. Coupled (also in the continuity eqn) 3. Role of pressure (no equation of state for Newtonian, incompressible fluids) 4. Second order derivatives <p>FRACS</p>
<p>SCAI</p> <h3>Iterative methods</h3> <p>All the issues above-mentioned require numerical methods and techniques to build accurate and stable tools to integrate such equations.</p>  <p>Iterative methods</p> <p>FRACS</p>	<p>SCAI</p> <h3>Guessed values and relaxation</h3> <ul style="list-style-type: none"> • The iterative method to move from one iteration to the other uses guessed values. • New values are found using the old value and a guessed one according to $\phi_r^{n+1} = \phi_r^{old} + U(\phi_r^{new, predicted} - \phi_r^{old})$ <p>Here U is the relaxation factor:</p> <ul style="list-style-type: none"> • U = 1 is under-relaxation • U = 1 corresponds to no relaxation. One uses the predicted value of the variable. • U = 1 is over-relaxation. <p>Interpolation schemes are related to the way that I use to build my used value.</p> <p>FRACS</p>

SCAI Conservativeness

The conservation of the fluid property ϕ must be ensured for each cell and globally by the algorithm

Global
Local

Δx
 Δy

FRONT

SCAI Boundedness

Iterative methods start from a guessed value and iterate until convergence criterion is satisfied all over the computational domain.

In order to converge math says that:

1. Diagonal dominant matrix (from the sys. of eqn.)
2. Coefficients with the same sign (positive)

Physically this means:

1. if you don't have source terms the values are bounded by the boundary ones (if the pb is linear)
2. if a property increases its value in one cell then the same property must increase also in all the cells nearby.

Overshoot and undershoot present for certain algorithms is related to this property

FRONT

SCAI Transportiveness

Directionality of the influence of the flow direction must be 'readable' by discretization scheme since it affects the balance between convection and diffusion.

Pure convection phenomena
Pure diffusion phenomena

Flow direction

So called 'false-diffusion' (i.e. numerically induced) is related to this property

FRONT

SCAI Pressure - velocity coupling

- For incompressible N-S eqn there is no explicit equation for P.
- P is involved in the momentum equations.
- V must satisfy also continuity equation.

The so-called 'pressure-velocity' coupling is an algorithms used to obtain a valid relationship for the pressure starting from the momentum and the continuity equation.

The oldest and most popular algorithm is the SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) by Patankar and Spalding 1972.

FRONT

SCAI What is convergence

Convergence: where do I stop my iterations?

- A flow field solution is considered 'converged' when the changes of the properties in the cells from one iteration to another are below a certain fixed value.
- General laws are missing, we have some good rules to understand when we are converged and we can stop to iterate over our solution.

FRONT

SCAI Residuals

- Residual: $R_i = \rho_i \phi_i - \sum \rho_i \phi_i - M_i$
- Usually scaled and normalized

#1 Residual control during computation

FRONT

SCAI Other convergence criteria

#2 Monitor the other quantity on boundaries

#3 Monitor changes on quantities you are interest on

FRONT

SCAI Post-processing & data visualization

Once the CFD model is 'at convergence' the data concerning the flow field in the discretized domain are available in order to be processed for quantitative analysis and for visualization.

Scientific visualization over CFD data is one of the most interesting characteristics of this tool because it allows designers and engineers to get a better understand of the physical phenomenon and so to improve the insight on it.

Once that P and U fields at each point in the domain are available it is possible to perform further processing and extract no matter wich derived quantity at any point with an accuracy that is not reachable by common experimental measurements techniques

FRONT

SCAI Source of errors and uncertainty

- Error: deficiency in a CFD model
Possible sources of error are:
 - Numerical errors (discretization, round-off, convergence)
 - Coding errors (bugs)
 - User errors
- Uncertainty: deficiency in a CFD model caused by a lack of knowledge.
Possible source of uncertainty are:
 - Input data inaccuracies (geometry, BC, material properties)
 - Physical model (simplified hypothesis for the fluid behavior)

SCAI Verification and Validation (V&V)

Verification: "solving the equation right" (Roache '86)
This process quantify the errors.

Validation: "solving the right equations" (Roache '86)
This process quantify the uncertainty.

SCAI How should I use it

CFD is a engineering design tool and like any tool can be used with success only were it suited to solve a certain problem and by people with sufficient knowledge on how to use the tool.

Right tools

- CORRECT SCREW DRIVER TYPES

PHILLIPS
FLAT
TOX
TOX

Wrong tool

SCAI HPC Computer Aided Engineering @ CINECA

Raffaella Portzini Ph.D.
CINECA
SuperComputing Applications
and Innovation Department - SCAI

© 16 and 01/17
Supercomputing, Italy

SCAI Outline

- Open-source CAD and Meshing tools
- Meshing main concepts
- Meshing tools for OpenFOAM

SCAI Overview

SCAI Geometry

- The starting point for all problems is a "geometry" that is in few words the shape of the problem to be analyzed or in other words the computational domain defined by the problem
- Geometry is managed through Computer Aided Design tools (CAD)
- Geometries can be created top-down or bottom-up.
- Top-down refers to an approach where the computational domain is created by performing logical operations on primitive shapes such as cylinders, bricks, and spheres
- Bottom-up refers to an approach where one first creates vertices (points), connects those to form edges (lines), connects the edges to create faces, and combines the faces to create volumes
- Advanced CAD tools have parametric description of the curves used to build the geometry
- CAD allows to define a virtualized description of the geometry components in terms of volumes, surfaces (faces), lines (edges) and points (vertices or nodes)

CAD tolerance and file format standards are still an issue

Mesh

- The computational mesh is the discretized counterpart of the geometry CAD
- The mesh cells are the atomic elements in which the physics of the flow is solved
- Cells are grouped and labeled according to different properties of the numerical boundaries defined for the problem (boundaries or patches in OpenFOAM)
- The mesh has a crucial impact on the CFD modeling since the overall workflow is built on top of the mesh
- Mesh quality parameters evaluation is crucial to move forward in a CFD modeling workflow

Open-source CAD and Meshing tools

Open-source CAD and Meshing tools

Open-source CAD and Meshing tools

- Open source CAE resources CAELINUX: <http://www.caelinux.com/>
- SALOME: <http://www.salome-platform.org/>
- FreeCAD: <http://www.freecadweb.org/>
- Meshing:
 - GMSH: <http://www.gmsh.info/>
 - SALOME: <http://www.salome-platform.org/>

We are not interested in this course on dealing with CAD issues

Mesh

There are several type of mesh that differs for the topology of the elements used to discretize the domain:

1. Single-block, structured grid
2. Multi-block, structured grid
3. Unstructured grid
4. Unstructured Tetrahedral grid
5. Hybrid grid





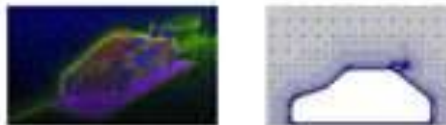
Terminology

Mesh

- Many different cell/element and grid types are available. Choose depends on the problem and the solver capabilities.
- Cell or element types:

Mesh

- Tn mesh: mesh consisting entirely of triangular elements.
- Quad mesh: consists entirely of quadrilateral elements.
- Hex mesh: consists entirely of hexahedral elements.
- Tet mesh: mesh with only tetrahedral elements.
- Hybrid mesh: mesh with one of the following:
 - Triangles and quadrilaterals in 2D
 - Any combination of tetrahedra, prisms, pyramids in 3D
 - Boundary layer mesh: prisms at walls
 - Hexcore: hexahedra in center and other cell types at walls.
- Polyhedral mesh: consists of arbitrary polyhedra.

<p>SCAI</p> <h3>Mesh quality</h3> <ul style="list-style-type: none"> For the same cell count, hexahedral meshes will give more accurate solutions, especially if the grid lines are aligned with the flow. The mesh density should be high enough to capture all relevant flow features. The mesh adjacent to the wall should be fine enough to resolve the boundary layer flow. In boundary layers, quad, hex, and prism/wedge cells are preferred over tri's, tet's, or pyramids. Three main measures of quality (depends on pre-processor): <ul style="list-style-type: none"> Skewness Aspect ratio Non orthogonality <p>FRONT</p>	<p>SCAI</p> <h3>Mesh design</h3> <ul style="list-style-type: none"> Particent flow features should be adequately resolved  <ul style="list-style-type: none"> Cell aspect ratio (width/height) should be near one where flow is multi-dimensional. Quad/hex cells can be stretched where flow is fully-developed and essentially one-dimensional. <p>Flow Direction</p>  <p>FRONT</p>
<p>SCAI</p> <h3>Mesh design</h3> <ul style="list-style-type: none"> Change in cell/element size should be gradual   <ul style="list-style-type: none"> Ideally, the maximum change in grid spacing should be <math>\leq 33\%</math>  $\frac{\Delta x_{i+1}}{\Delta x_i} \leq 1.3$ <p>FRONT</p>	<p>SCAI</p> <h3>Mesh design</h3> <ul style="list-style-type: none"> More cells can give higher accuracy (if well spent). The downside is increased memory and CPU time for mesh creation (complexity) To keep cell count down: <ul style="list-style-type: none"> Use a non-uniform grid to cluster cells only where they are needed Use solution adaption to further refine only selected areas Cell counts of the order: <ul style="list-style-type: none"> 1E5 small/intermediate suitable for desktop pc and for debugging of solver setup 1E6 regular day to day work 1E7 expensive, suitable for HPC centers 1E8 hardly manageable (my 2 cents experience with billion cells mesh http://www.ansys.com/technical-support/forums/forums/thread.jspa?threadID=1310&start=0&resourcetype=attachment_V3-02_Solving_Param_a_030101.pdf) Time mesh generation can be non-negligible <p>FRONT</p>
<p>SCAI</p> <h3>Main sources of errors</h3> <ul style="list-style-type: none"> Mesh too coarse High skewness Large jumps in volume between adjacent cells Large aspect ratios Non orthogonal cells Inappropriate boundary layer mesh <p>FRONT</p>	<p>SCAI</p> <h3>Meshing take home message</h3> <p>Appropriate choice of grid type depends on:</p> <ol style="list-style-type: none"> Geometric complexity (geometry is king) Flow field patterns (BC are queens) Cell and element types supported by solver (need to verify your mesh into the solver) <p>FRONT</p>
<p>SCAI</p> <h3>Meshing tools for openFoam</h3> <ul style="list-style-type: none"> Commercial: Pointwise (Pointwise Inc.) Open-source: snappyHexMesh (OpenCFD Ltd.)  <p>FRONT</p>	<p>SCAI</p> <h3>Pointwise</h3> <p>POINTWISE Reliable 3D Meshing</p> <ul style="list-style-type: none"> Site: http://www.pointwise.com/ General purpose pre-processor for any CFD code (commercial or open-source) Suitable to handle also very complex geometry GUI equipped User-friendly Serial Scriptable (glyph) Rich documentation (tutorials, webinar...) Strongly oriented to automation Sophisticated visual mesh quality check (his own metrics) Support CAD import (most formats) and simple geometry creation <p>FRONT</p>

snappyHexMesh

- + Natural pre-processor of openFOAM
- + No GUI
- + Hard to use with very complex geometry
- + Quite slow learning curve
- + Parallel
- + Scriptable by definition
- + Exhaustive mesh quality information
- + Support all CAD import and simple geometry creation
- + Other geometry manipulation tools are available within the openFOAM toolbox

Pointwise vs snappyHexMesh

Two different philosophies:

- snappyHexMesh:
 - + from boundaries of the domain to object geometry
 - + trying to satisfy quality criteria (user-driven) during the mesh creation
- Pointwise:
 - + from object geometry surfaces up to boundary domains
 - + A posteriori mesh quality check

snappyHexMesh or Pointwise

- + Both tools are well suited in our experience to be used with success in HPC platforms for automated and productive workflow in industrial applications
- + Pro's and con's should be evaluated case by case
- + License business model involved by Pointwise is not a limiting factor as that the cost is small (order of 2k euros) compared to usual CFD solver license cost (order of 10k -100k euros)



Paraview for mesh visualization

In the next tutorial you will learn how to use snappyHexMesh to build meshes suitable to perform CFD analysis in openFOAM.

In order to visualize the resulting mesh generated using snappyHexMesh you can use Paraview.

4 Training Material “HPC Cloud (SURFsara)”

This training was developed by SURFsara and has been held June 11th 2014. The material is available through the shared document server [6].

<h3>SURFsara HPC Cloud Workshop</h3>  <p>www.cloud.sara.nl → Tutorial 2014-06-11</p> <p>UvA HPC and Big Data Course June 2014 Anatoli Danezi, Markus van Dijk cloud-support@surfsara.nl</p> 	<h3>Agenda</h3> <ul style="list-style-type: none"> • Introduction and Overview (current presentation) • Hands on • Lunch • Application design • Hands on cont. • Assignment (approx. 4 hours) <p>SURFsara HPC Cloud workshop June 2014</p>		
<h3>Cloud? What as a Service?</h3> <p>Wikipedia:</p> <ul style="list-style-type: none"> • Cloud Computing is a jargon term without a commonly accepted non-ambiguous scientific or technical definition. • In science Cloud computing is a synonym for distributed computing over a network. <p>... as a Service:</p> <ul style="list-style-type: none"> • SaaS: Software – MS Office 360, gmail • PaaS: Platform – Google App Engine • IaaS: Infrastructure – Amazon EC2, SURFsara HPC Cloud • ...aaS <p>Known cloud-like services:</p> <ul style="list-style-type: none"> • Hosting • Grid computing: massive parallel batch processing <p>SURFsara HPC Cloud workshop June 2014</p>	<h3>Why Cloud?</h3> <p>Benefits</p> <ul style="list-style-type: none"> • No hardware to buy and maintain • No software to buy and maintain (SaaS, PaaS) • No maintenance downtime – live migration of virtual machines • Dynamic scalability – add when needed <p>Drawbacks</p> <ul style="list-style-type: none"> • Control over data – privacy, business secrets, legal obligations (patient data, Patriot Act) • Control over computing – availability, processing power (SLA, overcommitting) • Different environment – virtualization layer, VM management <p>Types:</p> <ul style="list-style-type: none"> • Private / Community / Public • Dedicated / Overcommitted (CPU, network, disk space) <p>SURFsara HPC Cloud workshop June 2014</p>		
<h3>SURFsara Computing</h3> <p>Pre-configured and maintained environments:</p> <ul style="list-style-type: none"> • Cartesius – National Supercomputer: 13984 cores (270TFlops), 41TB RAM, 2.4PB disk • Lisa – National Compute Cluster (VU, UvA, SURF): 6528 cores (46TFlops), 17TB RAM • Grid – International parallel batch processing: 11 sites, 5000 cores, 4PB disk 6PB tape • Hadoop – BigData-parallel processing framework: 700 cores, 1.2PB disk <p>Self-service:</p> <ul style="list-style-type: none"> • HPC Cloud – Cloud computing: 960 HPC cores, 80 Light cores, 8GB RAM/core - Extra: high memory node with 40 cores, 2TB RAM, 6.4TB disk - 500TB shared storage - Nodes connected by fast network <p>Other</p> <ul style="list-style-type: none"> • Beehub – Data storage: WebDAV access • Visualization – render cluster and “Collaboratorium” • Network <p>SURFsara HPC Cloud workshop June 2014</p>	<h3>Who uses the SURFsara HPC Cloud</h3> <table border="0"> <tr> <td> <p>Past:</p> <ul style="list-style-type: none"> • 130 projects completed since January 2011 <p>Current (2014-05-21, a slow day):</p> <ul style="list-style-type: none"> • 120 active projects • 250 login accounts • 160 running VMs using 785 cores • Largest VM: 32 HPC cores, 245GB RAM • Smallest VM: 1/4 core, 2GB RAM • 100TB used for disk images • 70TB used in VirDir (project NASes) <p>Example techniques:</p> <ul style="list-style-type: none"> • Galaxy, RStudio, Matlab • CFD: MPI on virtual cluster, multicore VMs • De novo genome assembly: single machine, multicore VMs, high memory </td> <td> <p>Research fields:</p> <ul style="list-style-type: none"> • Biology • Genetics • Informatics • Chemistry • Ecology • Linguistics • Robotics • Business • Social sciences • Engineering • Humanities </td> </tr> </table> <p>SURFsara HPC Cloud workshop June 2014</p>	<p>Past:</p> <ul style="list-style-type: none"> • 130 projects completed since January 2011 <p>Current (2014-05-21, a slow day):</p> <ul style="list-style-type: none"> • 120 active projects • 250 login accounts • 160 running VMs using 785 cores • Largest VM: 32 HPC cores, 245GB RAM • Smallest VM: 1/4 core, 2GB RAM • 100TB used for disk images • 70TB used in VirDir (project NASes) <p>Example techniques:</p> <ul style="list-style-type: none"> • Galaxy, RStudio, Matlab • CFD: MPI on virtual cluster, multicore VMs • De novo genome assembly: single machine, multicore VMs, high memory 	<p>Research fields:</p> <ul style="list-style-type: none"> • Biology • Genetics • Informatics • Chemistry • Ecology • Linguistics • Robotics • Business • Social sciences • Engineering • Humanities
<p>Past:</p> <ul style="list-style-type: none"> • 130 projects completed since January 2011 <p>Current (2014-05-21, a slow day):</p> <ul style="list-style-type: none"> • 120 active projects • 250 login accounts • 160 running VMs using 785 cores • Largest VM: 32 HPC cores, 245GB RAM • Smallest VM: 1/4 core, 2GB RAM • 100TB used for disk images • 70TB used in VirDir (project NASes) <p>Example techniques:</p> <ul style="list-style-type: none"> • Galaxy, RStudio, Matlab • CFD: MPI on virtual cluster, multicore VMs • De novo genome assembly: single machine, multicore VMs, high memory 	<p>Research fields:</p> <ul style="list-style-type: none"> • Biology • Genetics • Informatics • Chemistry • Ecology • Linguistics • Robotics • Business • Social sciences • Engineering • Humanities 		

Why SURFsara HPC Cloud

SURFsara HPC Cloud is IAAS: Infrastructure As A Service, so you assemble your virtual machine (VM) from the ground up.

General benefits:

- Data and computing in Amsterdam, backups in Almere
- No ties to US and its Homeland Security, Patriot Act
- Others cannot access data in your VM (including SARA personnel)
- Unrestricted Internet access (but fair use), including up/download of data

Technical benefits:

- No overcommitting, you alone use 100% of your core(s)
- Tailor VM to your needs: cores, RAM, disks
- Root access to your VM
- Free choice of OS, packages, versions
- Fast private network for all VMs in your project

SURFsara HPC Cloud workshop June 2014

Why not SURFsara HPC Cloud

SURFsara HPC Cloud is IAAS: Infrastructure As A Service, so you assemble your virtual machine (VM) from the ground up.

Drawbacks:

- No SLA (yet), service during office hours
- You maintain everything in your VM
- You are responsible for all of your VM's behavior
- You must protect yourself against threats from the Internet (DDOS, virus)
- Pay for VM uptime, not just compute time (like gas, light)
- No automatic backups
- Slow disk I/O (designed for computing)
- Your laptop is faster than a 1 core VM
- Interface to construct/start/stop VMs is not user friendly

SURFsara HPC Cloud workshop June 2014

Below decks

VM control: OpenNebula

- Open source
- Adaptable to our needs
- Currently best practice for our situation

Virtualization: KVM, libvirt, CentOS

- Open source
- Low overhead
- Proven track record

Support portal: Redmine

- Open source
- General Cloud wiki
- Per project, wiki and issue tracking

SURFsara HPC Cloud workshop June 2014

Cloud project networks

- Direct Internet access
- One private virtual network per project
- Fast interconnect between VMs
- Squid proxy for non-Internet VMs
- IP and MAC addresses change every launch
- Dynamic DNS

ViDir: shared storage per project

- /i/scratch, #Backup daily to tape
- Mount ViDir in VM: NFS, but with limitations
- Access ViDir via SFTP with private key
- Used for up/download of disk images

Extra

- MySQL server
- NoSQL server

SURFsara HPC Cloud workshop June 2014

Build a Virtual Machine

A "template" is a VM recipe:

- Node type: Large / Medium / Small
- # virtual cores, # real CPUs, RAM
- Private project network
- Internet: needs packet filter
- Your disk images (IDE, virtio), boot from hd / odrom

Create template: "Create VM" wizard or Templates--New

NODE		VCPU / CPU	RAM / CPU	RAM / VCPU
L	HPC	1/1	8GB	8GB
M	light	2/1	8GB	4GB
S	light	4/1	8GB	2GB

SURFsara HPC Cloud workshop June 2014

Set-up assistance

Creating a VM from scratch can be a lot of work.

We provide a set-up wizard: you choose

- CentOS / Ubuntu, Desktop / Server
- # cores and RAM
- Disk size
- Networks: Internet and/or local, webserver reachable from Internet?
- A new VM is created and launched
- Connect to console
- Set up root password and first user
- Connect with SSH

From then on, you must take care of updates etc., the wizard is just a set-up help.

SURFsara HPC Cloud workshop June 2014

Alternatives

Well-known international Clouds:

Note: "US-EU Safe Harbor" worthless

- Amazon Elastic Compute Cloud (EC2)
- Google Compute Engine
- Rackspace

Other:

- Hosting
- Grid Computing
- Hadoop

SURFsara HPC Cloud workshop June 2014

SURFsara HPC Cloud Workshop

HANDS ON

Support portal: <https://www.cloud.sara.nl>
 Search: [Tutorial 2014-06-11](#)

Self-service portal: <https://ui.cloud.sara.nl>
 Make sure you use Google Chrome browser.

UvA HPC and Big Data Course June 2014
 Anatoli Danezi, Markus van Dijk
 cloud-support@surfsara.nl

5 References

- [1] Fortissimo Deliverable D10.5 “The Month 42 Dissemination Report”
- [2] Fortissimo Deliverable D10.6 “The Year 1 Training Report”
- [3] Fortissimo Deliverable D10.7 “The Year 2 Training Report”
- [4] Fortissimo Deliverable D10.8 “The Month 42 Training Report”
- [5] Fortissimo Deliverable D10.9 “Initial Compendium of General Training Material”
- [6] Fortissimo 4PM <https://fortissimo.4pm.si/login.jsf> (access for project partners only)