You are welcome to contact our IC hotline:
ic-hotline@lists.uni-karlsruhe.de
In case you want to call us please dial: +49 721 608-8011

This user guide is available in PDF under
http://www.scc.kit.edu/scc/docs/IC/ug/ugic.pdf
Any comments and hints concerning this introduction are welcome. Please send corresponding e-mails to Hartmut.Haefner@kit.edu.
<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 0.9</td>
<td>May 02, 2008</td>
<td>First preliminary version of the User Guide describing the 'InstitutsCluster'</td>
</tr>
<tr>
<td>Version 0.91</td>
<td>June 26, 2008</td>
<td>Minor changes to the User Guide in chapters Libraries and additional chapter CAE Application Codes</td>
</tr>
<tr>
<td>Version 0.92</td>
<td>September 10, 2008</td>
<td>Additional CAE software in chapter CAE Application Codes and changes in Table 1 of the User Guide</td>
</tr>
<tr>
<td>Version 0.93</td>
<td>November 24, 2009</td>
<td>Additional CAE software in chapter CAE Application Codes</td>
</tr>
<tr>
<td>Version 0.94</td>
<td>March 10, 2010</td>
<td>Minor changes in chapter Numerical Libraries</td>
</tr>
<tr>
<td>Version 0.95</td>
<td>May 4, 2010</td>
<td>Additional CAE software in chapter CAE Application Codes (LS-Dyna)</td>
</tr>
<tr>
<td>Version 0.96</td>
<td>September 15, 2010</td>
<td>New MK Library requires other libraries to link</td>
</tr>
</tbody>
</table>
Contents

1 Introduction 6

2 Configuration of InstitutsCluster 6
   2.1 Architecture of InstitutsCluster .............................................. 6
   2.2 Components of InstitutsCluster ............................................... 7

3 Access to InstitutsCluster 9
   3.1 Login .................................................................................... 9
   3.2 Login on a Login Node ............................................................ 9

4 File Systems 9
   4.1 $HOME .................................................................................. 9
   4.2 $WORK .................................................................................. 10
   4.3 Improving Performance on $HOME and $WORK ...................... 10
      4.3.1 Improving Throughput Performance ................................... 11
      4.3.2 Improving Metadata Performance ..................................... 12
   4.4 $TMP ................................................................. 12
   4.5 Moving Files between Local Workstations and InstitutsCluster .. 12
   4.6 Backup and Archiving ............................................................ 13

5 Modules 13
   5.1 The most Important of Supplied Modulefiles ........................... 14
   5.2 Viewing available Modulefiles ............................................... 14
   5.3 Viewing loaded Modulefiles ................................................... 14
   5.4 Loading and Unloading a Modulefile ....................................... 14
   5.5 Creating a Modulefile ............................................................ 15
   5.6 Further important Module Commands ..................................... 16

6 Compilers 16
   6.1 Compiler Options .................................................................... 16
      6.1.1 General Options ............................................................... 16
      6.1.2 Important specific Options of Intel Compilers .................... 17
      6.1.3 Important specific Options of GNU Compilers ................. 17
      6.1.4 Important specific Options of the NAG Fortran Compiler .... 18
   6.2 Fortran Compilers ................................................................... 18
   6.3 C and C++ Compilers ........................................................... 19
7 Parallel Programming

7.1 Parallelization for Distributed Memory

7.1.1 Compiling and Linking MPI Programs
7.1.2 Communication Modes
7.1.3 Execution of Parallel Programs
7.1.4 mpiexec Options

7.2 Programming for Shared Memory Systems

7.3 Distributed and Shared Memory Parallelism

8 Debuggers

8.1 Parallel Debugger ddt

9 Performance Analysis Tools

9.1 Timing of Programs and Subprograms

9.1.1 Timing of Serial or Multithreaded Programs
9.1.2 Timing MPI-parallelized Programs
9.1.3 Timing of Program Sections

9.2 Profiling

10 Numerical Libraries

10.1 Intel Math Kernel Library (MKL)

11 CAE Application Codes

11.1 ABAQUS
11.2 LS-DYNA
11.3 MD Nastran
11.4 PERMAS
11.5 Fluent
11.6 ANSYS CFX
11.7 Star-CD
11.8 STAR-CCM+
11.9 COMSOL Multiphysics
11.10 Matlab
11.11 Pre- and Postprocessors, Visualisation Tools

12 Batchjobs

12.1 The job_submit Command
12.2 Environment Variables for Batch Jobs
12.3 job_submit Examples

12.3.1 Serial Programs
12.3.2 Parallel MPI Programs
1 Introduction

The Computing Center of the University of Karlsruhe hosts a parallel computer called 'InstitutsCluster' for different departments of University Karlsruhe. This supercomputer can only be used by employees of those departments which have taken a share in the costs of Institutscluster. The details how to get an account are available in section 3.

The InstitutsCluster can fulfill the services of a parallel high performance compute server as well as the services of a traditional serial and throughput oriented compute server. This user guide is mainly written for those customers who want to use InstitutsCluster as parallel high performance computer system. But except for a few sections, it is also of interest to those users, who want to run serial programs only.

In order to limit the size of this guide, only the most important information about the use of InstitutsCluster has been collected here. This document is accompanied by many links to resources on the web, especially on the web server of SCC Karlsruhe where more detailed information is available: http://www.rz.uni-karlsruhe.de/ssck/ic

2 Configuration of InstitutsCluster

![InstitutsCluster diagram]

Figure 1: InstitutsCluster at SCC of University of Karlsruhe

2.1 Architecture of InstitutsCluster

The InstitutsCluster is a distributed memory parallel computer where each node has eight Intel Xeon processors, local memory, disks and network adapters. All nodes are connected by a fast InfiniBand 4X DDR interconnect. In addition the file system Lustre, that is connected by coupling the InfiniBand of the file server with the InfiniBand switch of the compute cluster, is added to InstitutsCluster to provide a fast and scalable parallel file system.

The basic operating system on each node is Suse Linux Enterprise 10. On top of this operating system a set of open source software components like e.g. SLURM has been installed. Some of these components are of special interest to end users and are briefly discussed in this document. Others which are of greater importance to system administrators will not be covered by this document.
Nodes of InstitutsCluster may have different roles. According to the services supplied by the nodes, they are separated into disjoint groups. From an end users point of view the different groups of nodes are login nodes, compute nodes, file server nodes and administrative server nodes.

- **Login Nodes**
  The login nodes are the only nodes that are directly accessible by end users. These nodes are used for interactive login, file management, program development and interactive pre- and postprocessing. Several nodes are dedicated to this service but they are all accessible via one address and the Linux Virtual Server (LVS) will distribute the login sessions to the different login nodes.

- **Compute Node**
  The majority of nodes are compute nodes which are managed by a batch system. Users will submit their jobs to the batch system JMS using SLURM as basic system and a job is executed depending on its priority, when the required resources become available.

- **File Server Nodes**
  The hardware of the parallel file system Lustre incorporates some file server nodes; the file system Lustre is connected by coupling the InfiniBand of the file server with the independent InfiniBand switch of the compute cluster. In addition to shared file space there is also local storage on the disks of each node (for details see chapter 4).

- **Administrative Server Nodes**
  Some other nodes are delivering additional services like resource management, external network connection, administration etc. These nodes can be accessed directly by system administrators only.

### 2.2 Components of InstitutsCluster

The Institutscluster consists of

- 200 8-way Intel Xeon X5355 nodes. Each of these nodes contains two Quad-core Intel Xeon processors which run at a clock speed of 2.667 GHz and have 2x4 MB of level 2 cache each. Each node has 16 GB of main memory, 4 local disks (250 GB each) and an adapter to connect to the InfiniBand 4X DDR interconnect.

- 2 of 6 8-way Intel Xeon X5355 nodes can be used by the standard user of InstitutsCluster. Each node runs at a speed of 2.667 GHz and has 2x4 MB of level 2 cache. Each node has 32 GB of main memory, one InfiniBand adapter and 4 local disks (250 GB each).

- 5 8-way Intel Xeon E5345 service nodes. Each node runs at a speed of 2.3 GHz and has 2x4 MB of level 2 cache. Each node has 8 GB of main memory, one InfiniBand adapter and 4 local disks (250 GB each).

- 8 8-way Intel Xeon E5320 file server nodes. They are part of the scalable, parallel file system Lustre that is tied to InstitutsCluster via a separate InfiniBand network. The global shared storage of the file system has a capacity of 380 TiB and is subdivided into a part used for home directories and a larger part for non permanent files. The directories in the larger part of the file system are often called work directories. The details are described in chapter 4.

An important component of InstitutsCluster is the InfiniBand 4X DDR interconnect. All nodes are attached to this interconnect which is characterized by its very low latency of below 2 microseconds and a point to point bandwidth between two nodes of more than 1300 MB/s. Especially the very short latency makes the parallel system ideal for communication intensive applications and applications doing a lot of collective MPI communications.

With these types of nodes InstitutsCluster can meet the requirements of a broad range of applications:
• applications that are parallelized by the message passing paradigm and use high numbers of processors will run on a subset of the 200 eight-way nodes and exchange messages over the InfiniBand interconnect,

• applications that are parallelized using shared memory either by OpenMP or explicit multi-threading with Pthreads can run within the eight-way nodes.
3 Access to InstitutsCluster

To login on InstitutsCluster an account is necessary. Employees of the departments of University Karlsruhe and KIT respectively that have a share in the costs of InstitutsCluster have to fill a form, that can be downloaded from the website http://www.rz.uni-karlsruhe.de/download/downloads, and send it (per fax or mail) to BIT8000.

3.1 Login

The InstitutsCluster is a distributed memory parallel computer with two dedicated login nodes. These two login nodes and also the other login nodes are equipped with 8 cores, 32 GB main memory, local disks and network adapters. The operating system Linux (Suse Linux Enterprise Server 10, kernel version 2.6.22) runs on all nodes (and all other nodes), so that working on a single node of InstitutsCluster is comparable with working on a workstation.

3.2 Login on a Login Node

2 login nodes are available for the general public on InstitutsCluster. The selection of the login node is done automatically. If you are connecting another time to a login node, the sessions might run on a different login node of InstitutsCluster. Only the secure shell ssh is allowed to login. Other commands like telnet or rlogin are not allowed for security reasons.

A connection to InstitutsCluster can be established by the command

ssh user-id@ic1.rz.uni-karlsruhe.de

When logging in you must type your password; when logging in the first time the password should be changed.

If you are using OpenSSH (usually installed on Linux based systems) and you want to use a GUI-based application on InstitutsCluster like e.g. the debugger DDT, you should use the command

ssh -X user-id@ic1.rz.uni-karlsruhe.de

with the option -X.

4 File Systems

On InstitutsCluster the parallel file system Lustre is used for globally visible user data. Lustre is mainly developed by Cluster File Systems Inc (CFS). CFS has recently been acquired by Sun Microsystems.

Initial directories on the file system Lustre are created for each user, and environment variables $HOME and $WORK point to these directories. Within a batch job there is a further directory $TMP. Some of the characteristics of the file systems are shown in Table 1.

The physical location of the file systems is shown in Fig 2.

4.1 $HOME

The home directories of InstitutsCluster users are located in the parallel file system Lustre. You have access to your home directory from all nodes of InstitutsCluster. A regular backup of these directories to tape archive is automatically done.
Table 1: File Systems and Environment Variables

<table>
<thead>
<tr>
<th>Property</th>
<th>$TMP</th>
<th>$HOME</th>
<th>$WORK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visibility</td>
<td>local</td>
<td>global</td>
<td>global</td>
</tr>
<tr>
<td>Lifetime</td>
<td>batch job</td>
<td>permanent</td>
<td>&gt; 7 days</td>
</tr>
<tr>
<td>Disk space</td>
<td>375 GB</td>
<td>76 TB</td>
<td>305 TB</td>
</tr>
<tr>
<td>Quotas</td>
<td>no</td>
<td>if required</td>
<td>if required</td>
</tr>
<tr>
<td>Backup</td>
<td>no</td>
<td>yes (default)</td>
<td>no</td>
</tr>
<tr>
<td>Read perf./node</td>
<td>100 MB/s</td>
<td>600 MB/s</td>
<td>600 MB/s</td>
</tr>
<tr>
<td>Write perf./node</td>
<td>200 MB/s</td>
<td>700 MB/s</td>
<td>800 MB/s</td>
</tr>
<tr>
<td>Total read perf.</td>
<td>n*100 MB/s</td>
<td>1700 MB/s</td>
<td>6200 MB/s</td>
</tr>
<tr>
<td>Total write perf.</td>
<td>n*200 MB/s</td>
<td>1500 MB/s</td>
<td>5500 MB/s</td>
</tr>
</tbody>
</table>

global: all nodes of InstitutsCluster access the same file system;
local: each node of InstitutsCluster has its own file system;
permanent: files are stored permanently;
batch job: files are removed at end of the batch job.

The $HOME directories are used to hold those files that are permanently used like source codes, configuration files, executable programs etc. But remember that the performance of the $WORK file system is usually much better than the performance of the $HOME file system.

For each user group (i.e. one institute) a fixed amount of disk space for home directories is reserved. The disk space is not controlled by so-called quotas; but quotas can be introduced if it will be necessary. You can check the currently used disk space for your user group with the command show_quota (not yet available). The disk space limit of each user group (institute) can be queried by the command kontingent_get (not yet available).

4.2 $WORK

On InstitutsCluster there is additional file space that can be accessed using the environment variable $WORK.

The work directories are used for files that have to be available for a certain amount of time, e.g. a few days. These are typically restart files or output data that have to be postprocessed.

All users can create large temporary files. But in order to be fair to your colleagues who also want to use this file system, large files which are no longer needed should be removed. The Computing Center automatically removes old files in this file system which are older than 28 days. However, the guaranteed lifetime for files in $WORK is only 1 week.

The file system used for $WORK directories is also the parallel file system Lustre. This file system is especially designed for parallel access and for a high throughput to large files. The $WORK file system shows high data transfer rates of more than 5 GB/s write and read performance when the data are accessed in parallel. When you are designing your application you should consider that the performance of parallel file systems is generally better if data is transferred in large blocks and stored in few large files.

4.3 Improving Performance on $HOME and $WORK

There are special commands which might help to improve throughput and metadata performance on $HOME and $WORK.
4.3.1 Improving Throughput Performance

Depending on your application some adaptations might be necessary if you want to reach the full bandwidth of the file systems $HOME and $WORK. Parallel file systems typically stripe files over storage subsystems, i.e. large files are separated into stripes and distributed to different storage subsystems. The storage subsystems of InstitutsCluster are Infortrend RAID systems (Transtec PROVIGO 610). These RAID systems are attached to each server pair and the file system uses 6 software RAID6 volumes which vertically use all of these RAID systems. Each volume can reach up to 130 MB/s for writes and up to 150 MB/s for reads. The file system $WORK uses 48, the file system $HOME uses 12 volumes. By default, files of the file system $WORK are striped across 2 volumes and files of $HOME are striped across 1 volume.

However, you can change the stripe count of a directory and of newly created files. New files and directories inherit the stripe count from the parent directory. E.g. if you want to enhance throughput on a single file which is created in the directory $WORK/my_output_dir you can use the command

```
$ lfs setstripe $WORK/my_output_dir 0 -1 8
```

to change the last parameter of the stripe count to 8. If the single file is accessed from one task it is not beneficial to further increase the stripe count because the local bus and the interconnect will become the bottleneck. If many tasks and nodes use the same output file you can further increase the throughput by using all available storage subsystems with the following command:

```
$ lfs setstripe $WORK/my_output_dir 0 -1 -1
```

Note that the stripe count parameter -1 indicates that all available storage subsystems should be used. If all tasks write to the same file you should make sure that overlapping file parts are seldom used and that it is most beneficial if a single task uses file sizes which are multiples of 4 MB (4 MB is the default stripe size).

If you change the stripe count of a directory the stripe count of existing files inside this directory is not changed. It is also possible to only change the stripe count of empty files. If you want to change the stripe count of existing non-empty files, change the stripe count of the parent directory, copy the files to new files, remove the old files and move the new files back to the old name.

Also note that changes on the striping parameters (e.g. stripe count) are not saved in the backup, i.e. if directories have to be recreated this information is lost and the default stripe count will be used.
Therefore, you should annotate for which directories you made changes to the striping parameters so that you can repeat these changes if required.

### 4.3.2 Improving Metadata Performance

Metadata performance on parallel file systems is usually not as good as with local file systems. In addition, it is usually not scalable, i.e. a limited resource. Therefore, you should omit metadata operations whenever possible. For example, it is much better to have few large files than lots of small files.

There is a possible optimization which improves the performance of the `ls` command. On modern Linux systems, the GNU `ls` command often uses colorization by default to visually highlight the file type; this is especially true if the command is run within a terminal session. This is because the default shell profile initializations usually contain an alias directive similar to the following for the `ls` command:

```
alias ls='ls -color=ttty'
```

However, running the `ls` command in this way for files on a Lustre file system requires a `stat()` call to be used to determine the file type. This can result in a performance overhead, because the `stat()` call always needs to determine the size of a file, and that in turn means that the client node must query the object size of all the backing objects that make up a file. As a result of the default colorization setting, running a simple `ls` command on a Lustre file system often takes as much time as running the `ls` command with the `-l` option (the same is true if the `-F`, `-p`, or the `-classify` option, or any other option that requires information from a `stat()` call, is used.). To avoid this performance overhead when using `ls` commands, add an alias directive similar to the following to your shell startup script:

```
alias ls='ls -color=none'
```

### 4.4 $TMP

While all tasks of a parallel application access the same `${HOME}` and `${WORK}` directory, the `${TMP}` directory is local to each node on InstitutsCluster. Different tasks of a parallel application use different directories when they do not utilize one node.

This directory should be used for temporary files being accessed by single tasks. On all compute nodes the underlying hardware are four 246 GB disks per node.

Each time a batch job is started, a subdirectory is created on each node assigned to the job. `${TMP}` is newly set; the name of the subdirectory contains the Job-id and the starting time so that the subdirectory name is unique for each job. This unique name is then assigned to the environment variable `${TMP}` within the job. At the end of the job the subdirectory is removed.

### 4.5 Moving Files between Local Workstations and InstitutsCluster

You should transfer files between InstitutsCluster and your workstation by using the command `scp`. You can transfer files in both directions. In special cases the passive `ftp` command (only from InstitutsCluster to your workstation) can be used.

`scp` has a similar syntax like `rcp`, i.e. files on a remote computer system are identified by prefixing the file name with the computer name and user-id. File name and computer name are separated by a colon, while user-id and computer name are separated by the sign `@`.

A small example is to copy the file `mydata` from user `xy01` on the computer `ws.institute.uni-karlsruhe.de` into your `${HOME}` directory on InstitutsCluster. To accomplish this you may enter the following command on InstitutsCluster:

```
scp xy01@ws.institute.uni-karlsruhe.de:mydata $HOME/mydata
```

You will find further information on the `scp` command in the corresponding man page.
4.6 Backup and Archiving

There are regular backups of all data of the home directories. With the following commands you can access the saved data:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsm_q_backup</td>
<td>shows one, multiple or all files stored in the backup device</td>
</tr>
<tr>
<td>tsm_restore</td>
<td>restores saved files</td>
</tr>
</tbody>
</table>

Table 2: Commands for Backup

The option -h shows how to use both commands.

Files of the directories $HOME and $WORK can be archived. With the following commands you can use the archive:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsm_archiv</td>
<td>archives files</td>
</tr>
<tr>
<td>tsm_d_archiv</td>
<td>deletes files from the archive</td>
</tr>
<tr>
<td>tsm_q_archiv</td>
<td>shows files in the archive</td>
</tr>
<tr>
<td>tsm_retrieve</td>
<td>retrieves archived files</td>
</tr>
</tbody>
</table>

Table 3: Commands for Archiving

The option -h shows how to use the commands.

More detailed information you can find on the following website: http://www.rz.uni-karlsruhe.de/rz/sw/tsm/xc

5 Modules

The InstitutsCluster supports the use of Modules software to make it easier to configure and modify the user environment. Modules software enables dynamic modification of your environment by the use of modulefiles. A modulefile contains information to configure the shell for an application. Typically, a modulefile contains instructions that alter or set shell environment variables, such as PATH and MANPATH, to enable access to various installed software.

One of the key features of using modules is to allow multiple versions of the same software to be used in your environment in a controlled manner. For example, two different versions of the Intel C compiler can be installed on the system at the same time - the version used is based upon which Intel C compiler modulefile is loaded.

The software stack of InstitutsCluster provides a number of modulefiles. You can also create your own modulefiles. Modulefiles may be shared by many users on a system, and users may have their own collection of modulefiles to supplement or replace the shared modulefiles.

A modulefile does not provide configuration of your environment until it is explicitly loaded. That is, the specific modulefile for a software product or application must be loaded in your environment (with the module load command) before the configuration information in the modulefile is effective.

The modulefiles that are automatically loaded for you when you log in to the system can be displayed by the command module list. You only have to load further modulefiles, if you want to use additional software packages or to change the version of an already loaded software.

By default the modulefiles
dot adds the current directory to your environment variable PATH,
gcc loads GNU C/C++ and Fortran90/95 compiler in a stable version,
openmpi loads OpenMPI in a stable version,

and - if necessary - further modulefiles will be loaded when logging in.

5.1 The most Important of Supplied Modulefiles

<table>
<thead>
<tr>
<th>Modulefile</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dot</td>
<td>adds the current directory to your environment variable PATH</td>
</tr>
<tr>
<td>gcc</td>
<td>loads GNU C/C++ and Fortran90/95 compiler in a stable version</td>
</tr>
<tr>
<td>intel</td>
<td>loads Intel C/C++ and Fortran90/95 compiler in a stable version</td>
</tr>
<tr>
<td>openmpi</td>
<td>loads OpenMPI in latest version</td>
</tr>
<tr>
<td>ddt</td>
<td>loads graphical debugger in latest version</td>
</tr>
<tr>
<td>mkl</td>
<td>loads Intel MKL for Intel compiler in a stable version</td>
</tr>
<tr>
<td>nag-compilers</td>
<td>loads NAG Fortran compiler in a stable version (not yet available)</td>
</tr>
<tr>
<td>itchk</td>
<td>loads Intel Intel Thread Checker and Profiler in a stable version (not yet available)</td>
</tr>
<tr>
<td>itac</td>
<td>loads Intel trace collector and trace analyzer in latest version (not yet available)</td>
</tr>
</tbody>
</table>

Table 4: Important Supplied Modulefiles

All the above mentioned software packages can be used by all users.

5.2 Viewing available Modulefiles

Available modulefiles are modulefiles that available for you to load. A modulefile must be loaded before it provides changes to your environment, as described in the introduction to this section. You can view the modulefiles that are available on the system by issuing the `module available` command:

```
module available
```

5.3 Viewing loaded Modulefiles

A loaded modulefile is a modulefile that has been explicitly loaded in your environment by the module load command. To view the modulefiles that are currently loaded in your environment, issue the module list command:

```
module list
```

5.4 Loading and Unloading a Modulefile

You can load a modulefile in to your environment to enable easier access to software that you want to use by executing the `module load` command. You can load a modulefile for the current session, or you can set up your environment to load the modulefile whenever you log in to the system.

You can load a modulefile for your current login session as needed. To do this, issue the `module load` command as shown in the following example, which illustrates the DDT debugger modulefile being loaded:

```
module load
```
module load ddt or module add ddt

Loading a modulefile in this manner affects your environment for the current session only.

If you frequently use one or more modulefiles that are not loaded when you log in to the system, you can set up your environment to automatically load those modulefiles for you. A method for doing this is to modify your shell startup script to include instructions to load the modulefile automatically.

For example, if you want to automatically load the DDT debugger modulefile when you log in, edit your shell startup script to include the following instructions. This example assumes that you use bash as your login shell. Edit the $HOME/.bashrc file as follows:

```bash
# if the 'module' command is defined, $MODULESHOME
# will be set
if [ -n "$MODULESHOME" ]; then
  module load ddt
fi
```

From now on, whenever you log in, the DDT debugger modulefile is automatically loaded in your environment.

In certain cases, you may find it necessary to unload a particular modulefile before you can load another modulefile in to your environment to avoid modulefile conflicts.

You can unload a modulefile by using the module unload or module rm command, as shown in the following example:

```bash
module unload ddt or module rm ddt
```

Unloading a modulefile that is loaded by default makes it inactive for the current session only - it will be reloaded the next time you log in.

### 5.5 Creating a Modulefile

If you download or install a software package into a private directory, you can create your own (private) modulefile for products that you install by using the following general steps:

1. create a private modulefiles directory,
2. copy an existing modulefile (as a template) or copy the corresponding default modulefile out of a subdirectory - if available - of the path /opt/modules/modulefiles into the private modulefiles directory,
3. edit and modify the modulefile accordingly,
4. register the private directory with the module use command.

A user installing a random product or package should look at the manpages for modulefiles, examine the existing modulefiles, and create a new modulefile for the product being installed using existing modulefiles as a template. To view modules manpages, type:

```bash
man module or man modulefile
```
5.6 Further important Module Commands

The command module help [modulefile...] prints the usage of each sub-command.
The commands module display modulefile [modulefile...]
or module show modulefile [modulefile...] display information about a modulefile. The above
mentioned commands will list the full path of the modulefile and all (or most) of the environment
changes the modulefile will make if loaded. It will not display any environment changes found within
conditional statements.

The command module whatis [modulefile [modulefile...]] displays the modulefile information
set up by the module-whatis commands inside the specified modulefiles. If no modulefiles are specified
all whatis information lines will be shown.

The command module use [-a|-append] directory [directory...] prepends directory [directory...] to the MODULEPATH environment variable. The –append flag will append the directory to
MODULEPATH.

6 Compilers

On InstitutCluster exist different compilers for Fortran (supporting the language standards of Fortran
77, Fortran 90, Fortran 95 and partially Fortran 2003), C and C++. There are two Fortran compiler
families and two C/C++ compiler families. The Fortran compilers consist of the Intel compiler family,
the GNU Fortran77 compiler and the GNU Fortran95 compiler. The C/C++ compilers consist of the
Intel compiler family and the GNU C/C++ compilers in 2 versions. We recommend the latest versions
of the C/C++ and Fortran compilers of the Intel compiler family.

6.1 Compiler Options

6.1.1 General Options

As on other Unix or Linux systems the compilers support the most common options:

-c compile only, do not link the object codes to create an executable program.

-I path specify a directory, which is used to search for module files and include files, and add it to
the include path.

-g include information for a symbolic debugger in the object code.

-O [ level ] create optimized source code. The optimization levels are 0, 1, 2, and 3. The option
-0 is identical to -02. Increasing the optimization level will result in longer compile time, but
will increase the performance of the code. In most cases at least optimization level -O2 should
be selected. The -O2 option of the Intel compilers enables optimizations for speed, including
global code scheduling, software pipelining, predication, and speculation. The NAG Fortran
compilers additionally supports the compiler option -O4. The Intel C/C++ compiler and the
GNU compilers additionally support the compiler option -Os optimizing the code for size.

-p or -pg create code for profiling with the gprof utility. -p is not supported by the NAG Fortran
compiler and by the GNU compilers.

-L path tell the linker to search for libraries in path before searching the standard directories

-1library use the specified library to satisfy unresolved external references

-o name specify the name of the resulting executable program.
6.1.2 Important specific Options of Intel Compilers

All Intel C/C++ compilers can be called by the command `icc`. If you are using pure C++ code, you also can use the C++ compiler `icpc`. All Intel Fortran compilers can be called by the command `ifort`. Intel specific compiler options which are often needed are:

- `-Ob[012]` controls inline expansion with suboptions from 0 (disables inlining) up to 2 (inlines any function, at the compiler’s discretion); suboption 1 is the default.

- `-fast` maximizes speed across the entire program; this option sets options `-O3`, `-ipo` and `-static`. The default is `-nofast`. The option `-static` prevents linking with shared libraries and `-ipo` enables multifile interprocedural (IP) optimizations (between files).

- `-vec_report[012345]` specifies the amount of vectorizer diagnostic information to report; valid suboptions are 0 (produces no diagnostic information) up to 5 (indicates non-vectorized loops and prohibiting data dependency information).

- `-mp` maintains floating-point precision (disables some optimizations). The `-mp` option restricts optimization to maintain declared precision and to ensure that floating-point arithmetic conforms more closely to the ANSI and IEEE standards. For most programs, specifying this option adversely affects performance.

- `-parallel` tells the auto-parallelizer to generate multithreaded code for loops that can be safely executed in parallel; to use this option, you must also specify `-O2` or `-O3`.

- `-ivdep_parallel` tells the compiler that there is no loop-carried memory dependency in any loop following an IVDEP directive.

- `-par_report[0123]` controls the auto-parallelizer’s level of diagnostic messages; valid suboptions are 0 (produces no diagnostic information) up to 3 (indicates diagnostics indicating loops successfully and unsuccessfully auto-parallelized and additional information about any proven or assumed dependencies inhibiting auto-parallelization).

- `-openmp` enables the parallelizer to generate multithreaded code based on OpenMP directives.

- `-openmp_report[012]` controls the OpenMP parallelizer’s level of diagnostic messages; valid suboptions are 0 (produces no diagnostic information) up to 2 (displays diagnostics indicating loops, regions, and sections successfully parallelized and diagnostics indicating successful handling of MASTER constructs, SINGLE constructs, CRITICAL constructs, ORDERED constructs, ATOMIC directives, etc; suboption 1 is the default.

- `-save` is only an Intel Fortran compiler option; it places variables, except those declared as AUTOMATIC, in static memory.

- `-traceback` is only an Intel Fortran compiler option; it tells the compiler to generate extra information in the object file to allow the display of source file traceback information at run time when a severe error occurs.

6.1.3 Important specific Options of GNU Compilers

The GNU C/C++ compiler can be called by the command `gcc`. The GNU Fortran95/2003 compiler can be called by the command `gfortran`; the Fortran compilers supports all options supported by the C/C++ compiler. GNU specific compiler options which are often needed are:

- `-funroll-loops[012]` unrolls loops whose number of iterations can be determined at compile time or upon entry to the loop.
-fprefetch-loop-arrays[0|1|2|3|4|5] generates instructions to prefetch memory to improve the performance of loops that access large arrays.

-static prevents linking with shared libraries.

6.1.4 Important specific Options of the NAG Fortran Compiler

The NAG Fortran compiler can be called by the command f95; the Fortran compiler supports the Fortran 77, Fortran 90 and Fortran 95 standard. NAG specific Fortran compiler options which are often needed are:

-unsafe performs possibly unsafe optimizations that may depend on the numerical stability of the program.

-dusty allows the compilation and execution of "legacy" software by downgrading the category of common errors found in such software from "Error" to "Warning".

-ieee=full|nonstd|stop the option -ieee=full enables all IEEE arithmetic facilities including non-stop arithmetic. The option -ieee=nonstd disables IEEE gradual underflow, producing zero instead of a denormalised number; the resulting program may run faster; non-stop arithmetic is also disabled, terminating execution on floating overflow, divide by zero or invalid operand. The option -ieee=stop enables all IEEE arithmetic facilities except for non-stop arithmetic. The default mode is -ieee=stop.

-C compiles code with all possible runtime checks.

-mtrace traces memory allocation and deallocation.

-gline compiles code to produce a traceback when a runtime error message is generated.

-gc enables automatic garbage collection of the executable program; this option is incompatible with -thread_safe and -mtrace.

-thread_safe compiles code for safe execution in a multi-threaded environment; this must be specified when compiling and also during the link step; it is incompatible with -gc and -mtrace.

-static prevents linking with shared libraries.

6.2 Fortran Compilers

The standard Fortran compiler on InstitutsCluster is Intel's Fortran compiler. You can use different versions of this compiler. All versions of the Intel Fortran compiler support Fortran 77, Fortran 90 and Fortran 95 plus some features of the Fortran 2003 standard and other extensions. A detailed description is available in the different Intel Fortran User Guides and the different Intel Fortran Language References. All documents are available at the HP XC web site: http://www.rz.uni-karlsruhe.de/ssck/ic-manuals

The Intel Fortran compiler may be invoked with several suffixes indicating the format of the source code, expected language standard and some other default options:

Free format source codes should always be stored in files with file name extension .f90, .i90 or .F90 while files containing fixed format source code should have the file name extension .f, .ftn, .for, .i, .F, .FTN, .FOR, .fpp, .FPP.

To compile FORTRAN90/95 source code stored in file my_prog.f90 the appropriate command is

ifort -c -O3 my_prog.f90
### Table 5: Fortran suffix names

<table>
<thead>
<tr>
<th>command</th>
<th>file name suffix</th>
<th>default compiler option for source format</th>
<th>language level</th>
</tr>
</thead>
<tbody>
<tr>
<td>ifort</td>
<td>.f, .ftn, .for, .i</td>
<td>-fixed -72 -nostand</td>
<td></td>
</tr>
<tr>
<td>ifort</td>
<td>.F, .FTN, .FOR, .fpp, .FPP</td>
<td>-fixed -72 -fpp -nostand</td>
<td></td>
</tr>
<tr>
<td>ifort</td>
<td>.f90, .i90</td>
<td>-free -nostand</td>
<td></td>
</tr>
<tr>
<td>ifort</td>
<td>.F90</td>
<td>-free -fpp -nostand</td>
<td></td>
</tr>
</tbody>
</table>

To compile an MPI program the basic compiler name must be substituted by the string `mpif90`. The parallel program `my_MPI_program.f90` therefore should be compiled with the command

```
mpif90 -c -O3 my_MPI_program.f90
```

To compile multithreaded applications (i.e. OpenMP programs) the compiler option `-openmp` is added to the compiler name, i.e. the OpenMP program `my_OpenMP_program.f90` has to be compiled with the command

```
ifort -c -O3 -openmp my_OpenMP_program.f90
```

When a FORTRAN90/95 program uses both parallelization paradigms (MPI and multi threading) then the compiler name must be substituted by the string `mpif90` and the compiler option `-openmp` must be used.

The NAG Fortran compiler also supports the different Fortran standards. The suffix of a filename determines the action `f95` - by this command the NAG Fortran compiler is called - performs upon it.

### Table 6: Fortran suffix names for the NAG Fortran compiler

<table>
<thead>
<tr>
<th>command</th>
<th>file name suffix</th>
<th>source format</th>
<th>language level</th>
</tr>
</thead>
<tbody>
<tr>
<td>f95</td>
<td>.f, .ftn, .for</td>
<td>-fixed</td>
<td>No compiler option;</td>
</tr>
<tr>
<td>f95</td>
<td>.F</td>
<td>-fixed -fpp</td>
<td>Fortran95 standard;</td>
</tr>
<tr>
<td>f95</td>
<td>.f90, .f95</td>
<td>-free</td>
<td>-strict95 for strict Fortran95 code</td>
</tr>
<tr>
<td>f95</td>
<td>.F90, .F95</td>
<td>-free -fpp</td>
<td>-dusty for &quot;legacy&quot; code</td>
</tr>
</tbody>
</table>

The above mentioned commands can be used with the NAG Fortran compiler, if the name of the Intel Fortran compiler `ifort` is substituted by the name of the NAG Fortran compiler `f95`. The command `mpif77` and `mpif90` respectively can be used with both Intel Fortran compiler and NAG Fortran compiler.

If you want to check the Fortran95 conformity of your program, the usage of the NAG Fortran compiler is the best choice because of the strict checks of the compliance with the standard.

The GNU Fortran compiler supports the Fortran95 standard and some Fortran 2003 features. Again the above mentioned commands can be used with the GNU Fortran compiler, if the name of the Intel Fortran compiler `ifort` is substituted by the name of the GNU Fortran compiler `gfortran`.

### 6.3 C and C++ Compilers

The C and C++ compilers on InstitutsCluster are:

- latest Intel C/C++ compiler of version 10.1, this is the default C/C++ compiler;
- GNU project C/C++ compiler.
The C and C++ compilers on InstitutsCluster are invoked with commands $\texttt{icc}$ or $\texttt{gcc}$. Details may be found in the appropriate man pages or in the compiler manuals 
(http://www.rz.uni-karlsruhe.de/ssck/ic-manuals).

To compile MPI programs the compiler scripts $\texttt{mpicc}$ and $\texttt{mpiCC}$ should be used for C and C++ programs.

To compile a C++ program $\texttt{my\_MPI\_program.C}$ that calls MPI functions the appropriate command is therefore

$\texttt{mpiCC -c -O3 my\_MPI\_program.C}$

To compile an OpenMP program $\texttt{my\_OpenMP\_program.C}$ written in C++ the following command should be used:

$\texttt{mpiCC -c -openmp -O3 my\_OpenMP\_program.C}$

7 Parallel Programming

Different programming concepts for writing parallel programs are used in high performance computing and are therefore supported on InstitutsCluster. This includes concepts for programming for distributed memory systems as well as for shared memory systems.

A program parallelized for distributed memory systems consists of several tasks where each task has its own address space and the tasks exchange data explicitly or implicitly via messages. This type of parallelization is the most portable parallelization technique but may require a high programming effort. It is used on workstation clusters as well as on parallel systems like InstitutsCluster.

In contrast to this, parallelization for shared memory systems is sometimes much easier but restricts the execution of the resulting program to a computer system which consists of several processors which share one global main memory. This type of parallelization can be used within a single node of InstitutsCluster.

A lot of resources on these parallelization environments are available on the web. A starting address could be: http://www.rz.uni-karlsruhe.de/ssck/parallel

7.1 Parallelization for Distributed Memory

For distributed memory systems most often explicit message passing is used, i.e. the programmer has to introduce calls to a communication library to transfer data from one task to another one. As a de facto standard for this type of parallel programming the Message Passing Interface (MPI) has been established during the past years. On InstitutsCluster MPI is part of the parallel environment.

7.1.1 Compiling and Linking MPI Programs

There are special compiler scripts to compile and link MPI programs. All these scripts start with the prefix $\texttt{mpi}$:

$\texttt{mpicc}$ compile and link C programs,

$\texttt{mpiCC}$ compile and link C++ programs,

$\texttt{mpif77}$ or $\texttt{mpif90}$ compile and link Fortran programs. Both variants work together with both Intel, GNU and NAG Fortran compilers which means that it complies with the Fortran 90/95 language specification.
With these compiler scripts no additional MPI specific options for header files, libraries etc. are needed, but all the standard options of the serial compilers are still available.

Further details on MPI may be found at http://www.rz.uni-karlsruhe.de/rd/mpi

7.1.2 Communication Modes

Communication between the tasks of a parallel application can be done in two different ways:

- data exchange using shared memory within a node,
- communication between nodes using the InfiniBand Switch.

On Institutcluster in general more than one task of a parallel application is executed on one node. In the operating mode - exclusive use of nodes - allocated nodes are used exclusively by up to eight MPI-processes or OpenMP-threads of a single batch job. In the operating mode - exclusive use of cores - allocated nodes can be used by different batch jobs (up to eight tasks with a summarized memory request of 16 GB). MPI-processes running on the same node use automatically the shared memory for the communication, i.e. they transfer messages by copying the data within the shared memory of the node. This results in a communication speed of about 1600 MB/s for simple send/receive operations.

Tasks on different nodes communicate over the InfiniBand Switch. Data communication speed can reach about 1300 MB/s.

So we can summarize:

- within a node always communication via shared memory is used,
- tasks running on different nodes communicate over the InfiniBand Switch.

7.1.3 Execution of Parallel Programs

Parallel programs can be started interactively or under control of the batch system.

Interactive parallel programs are launched with the command `mpirun`. They can only be executed on the node you are logged in, i.e. launching the command `mpirun` interactively means that you cannot use another node than this one you are logged in. Especially the following restrictions hold:

- maximum 8 MPI-processes,
- maximum 2 GB virtual memory per MPI-process and
- maximum 10 minutes CPU time per task are allowed.

Batch jobs are launched with the command `job_submit` and allow to start jobs in the development pool with a few nodes or in the production pool with many nodes. To start a parallel application as batch job the shellscript that is usually required by the command `job_submit` must contain the command `mpirun` with the application as input file.

The syntax to start a parallel application is `mpirun [ mpirun_options ] program`

or

`mpirun [ mpirun_options ] -f appfile`
as well for interactive calls as calls within shellscripts to execute batch jobs. The `mpirun_options` are the same for interactive calls and calls within shellscripts to execute batch jobs.

Important for the understanding: the option `-n #` is required calling `mpirun` interactively, but is ignored calling `mpirun` in batch jobs (the number of processors used in batch jobs is controlled by an option of the command `job_submit`). There is no option to specify the number of nodes you want to use, because calling `mpirun` interactively means to always use only one node and calling `mpirun` in a batch job means that the number of nodes is controlled automatically by the batch system.

**Example:**

```bash
#!/bin/bash
#
# This is an example for interactive parallel program execution.
#
# The program my_mpi_program will be run with 4 tasks.
#
mpirun -n 4 my_mpi_program
# A second version launching the same executable on the same number
# of processors is:
#
export MPRUN_OPTIONS="-n 4"
mpirun my_mpi_program
```

### 7.1.4 mpirun Options

**Calling**

`mpirun -?` or `mpirun -h`

prints the usage of the command `mpirun`.

**Calling**

`mpirun -H`

prints the usage of the command `mpirun` with a brief explanation of the options.

<table>
<thead>
<tr>
<th><code>mpirun Option</code></th>
<th>Brief Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-n #</code> or <code>-np #</code></td>
<td>MPI job is run on <code>#</code> processors (option is ignored in batch mode)</td>
</tr>
<tr>
<td><code>-byslot</code></td>
<td>maps the processes by slot in a round-robin scheme (default)</td>
</tr>
<tr>
<td><code>-bynodel</code></td>
<td>maps the processes by node in a round-robin scheme</td>
</tr>
<tr>
<td><code>-V, -version</code></td>
<td>prints version number</td>
</tr>
<tr>
<td><code>-v, -verbose</code></td>
<td>turns on verbose mode</td>
</tr>
<tr>
<td><code>-d</code></td>
<td>turns on debug mode</td>
</tr>
<tr>
<td><code>-f appfile</code></td>
<td>allows to run different executables on different processors; the names of the executables must be stored in <code>appfile</code>; this option must always be the last option!</td>
</tr>
</tbody>
</table>

The last parameter of the command `mpirun` must be either the option `-f appfile` or an executable program and shell script respectively.

Using an executable program and shellscript respectively as last parameter `mpirun` executes the programs in Single Program Multiple Data (SPMD) mode, i.e. the same program is executed by all tasks of the parallel application. Sometimes parallel programs are designed in such a way that different programs are executed by the tasks of a parallel application. This is called Multiple Program Multiple Data (MPMD) mode which is also supported by `mpirun`. To use this mode the option `-f appfile` must be chosen as last parameter.
The format of the application file *appfile* is very simple. In row \( i \) \((i = 1, \text{number of processors})\) the name of the executable that should run on processor \( i-1 \) must be specified. Running a master-slave model on 4 processors means that you have to create the following *appfile*:

```
master
slave
slave
slave
```

The master will run - as usual - on processor 0 and the slaves will run on the processors 1 up to 3.

### 7.2 Programming for Shared Memory Systems

While MPI is a tool for distributed memory systems, OpenMP is targeted to shared memory systems, i.e. one node with several CPUs. OpenMP is an extension to Fortran and C and seems to become a de facto standard for parallel programming of shared memory systems. On Institutcluster the OpenMP specification is supported by the Intel Fortran and C compilers.

To compile programs for shared memory parallelism the Intel compiler option `-openmp` must be selected. The following options can be specified to control the behaviour of thread-parallelized programs:

- `-openmp_report[0|1|2|3]` – controls the level of diagnostic messages regarding OpenMP;
- `-openmp_stubs` – enables the compiler to generate sequential code; the OpenMP directives are ignored and a stub OpenMP library is linked;
- `-par_report[0|1|2|3]` – controls the auto-parallelizer’s level of diagnostic messages;
- `-par_threshold[n]` – sets a threshold for the auto-parallelization of loops based on the probability of profitable execution of the loop in parallel; \([n]\) is an integer from 0 to 100; the default value is 75;
- `-parallel` – enables the auto-parallelizer to generate multithreaded code for loops that can be safely executed in parallel;
- `-threads` – specifies that multithreaded libraries should be linked; this option sets the `-reentrancy` threaded option; the default is `-nothreads`.


### 7.3 Distributed and Shared Memory Parallelism

For certain applications it might be convenient to combine the parallelization techniques for distributed memory and shared memory, i.e. parallelization within a node using shared memory parallelization with OpenMP and parallelization between nodes using message passing with MPI. In these cases the compilation must be started using one of the compiler scripts starting with prefix `mpi` and using the compiler option `-openmp`.

### 8 Debuggers

On Institutcluster the GUI based distributed debugging tool (ddt) may be used to debug serial as well as parallel applications. For serial applications also the GNU gdb or Intel idb debugger may be used. The Intel idb comes with the compiler and information on this tool is available together with the compiler documentation.
In order to debug your program it must be compiled and linked using the \texttt{-g} compiler option. This will force the compiler to add additional information to the object code which is used by the debugger at runtime.

### 8.1 Parallel Debugger ddt

ddt consists of a graphical frontend and a backend serial debugger which controls the application program. One instance of the serial debugger controls one MPI process. Via the frontend the user interacts with the debugger to select the program that will be debugged, to specify different options and to monitor the execution of the program. Debugging commands may be sent to one, all or a subset of the MPI processes.

Before the parallel debugger ddt can be used, it is necessary to load the corresponding module file:

```
module add ddt
```

Now ddt may be started with the command

```
ddt program
```

where \texttt{program} is the name of your program that you want to debug.

![DDT startup window](image)

**Figure 3: DDT startup window**

Fig. 3 shows ddt’s startup window. Before actually starting the debugging session you should check the contents of several fields in this window:
1. The top line shows the executable file that will be run under control of the debugger. In the following lines you may input some options that are passed to your program or to the MPI environment.

2. If your program reads data from stdin you can specify an input file in the startup window.

3. Before starting an MPI program you should check that 'openmpi' is the MPI implementation that has been selected. If this is not the case, you have to change this. Otherwise ddt may not be able to run your program.

   In order to debug serial programs, the selected MPI implementation should be 'none'

   You may also change the underlying serial debugger using the 'change' button. By default ddt uses its own serial debugger, but it may also use the Intel idb debugger.

4. Select the number of MPI processes that will be started by ddt. If you are using ddt within a batch job, you must make sure that this number is identical to the number of MPI tasks (-p option) that you selected with the job_submit command. When you debug a serial program, select 1.

5. After you have checked all inputs in the ddt startup window, you can start the debugging session by pressing the 'run' button.

The ddt window now shows the source code of the program that is being debugged and breakpoints can be set by just pointing to the corresponding line and pressing the right mouse button. So you may step through your program, display the values of variables and arrays and look at the message queues.

Figure 4: DDT window
9 Performance Analysis Tools

After installation and successful test of a program it is essential to analyze its performance. When performance bottlenecks have been detected they should be resolved by restructuring parts of the program, setting of specific compiler or runtime options or by replacing own code by optimized code from numerical libraries (cf. section 10). To get support and help to optimize your program you should contact the technical support staff at University of Karlsruhe Computing Center (cf. section 13).

For most serial as well as parallel programs the processing power of the CPU is the limiting resource when running the program. This means that the performance analysis should concentrate on CPU usage. For parallel programs additionally the communication overhead has to be analyzed. On InstitutsCluster the communication time is included in the CPU time. So a ratio between CPU time and real time that is close to one doesn’t tell anything about communication efficiency.

The most important questions in performance analysis are:

- Is the CPU used effectively, i.e. is there nearly no idle time?
- Is the communication organized in the right way so that no task is waiting for data to be sent from or to another task?
- Is the processor used most efficiently, i.e. what MFlop/s rate is achieved?

The performance analysis of a parallel program may therefore consist of the following steps:

1. Check the ratio of user CPU time, system CPU time and real (wall clock) time.
2. Analyze the communication behaviour of the program.
3. Find those parts of the program which consume the highest amount of CPU time.
4. Do a detailed analysis how the functional units of the processors are utilized.

For these steps different tools are supplied and will be described in the next sections. When the most time consuming parts of the program and possible bottlenecks have been identified, then the next step is to restructure these parts of the program to improve the performance.

9.1 Timing of Programs and Subprograms

The simplest way to do a first timing analysis of a program is to use the `time` command to analyze a serial or multithreaded program.

9.1.1 Timing of Serial or Multithreaded Programs

To do a very first analysis of CPU usage of a serial or multithreaded application, just write the command time in front of the program name. i.e. in order to run the program `my_serial_program` under control of `time` enter the command

```
time my_serial_program [ options ]
```

After termination of the program you will get some additional lines of output which may look like:

```
real 2m9.051s
user 1m49.312s
sys 0m0.106s
```
In this example we see that the program used 1 min 49.312 sec CPU time in user mode and 0.106 sec in system mode. The system CPU time is the time which is consumed by operating system functions working for the application program. Most of this time is caused by input and output operations. The system CPU time should i.g. not exceed a few percent of the user CPU time. In those cases where the program has exclusive access to the resources of a node (e.g. in batch jobs running in the production class) the realtime should not be much higher than the sum of user and system CPU time. Otherwise the program seems to be waiting for completion of I/O operations.

In case of multithreaded programs the CPU time could be much higher than the real wall clock time. The CPU time is the sum of the times required by all threads of the program. The following example shows the measurement of a program running with two threads on a two way node:

```
real  3m24.255s
user  6m47.652s
sys   0m0.261s
```

As we see in this example the user time is nearly twice the real time. So the two threads of the program use the two CPUs without high overhead for I/O operations.

### 9.1.2 Timing MPI-parallelized Programs

In case of MPI parallelized programs the `time` command can only deliver information on real time, but not on CPU time used by the parallel application.

### 9.1.3 Timing of Program Sections

A more detailed timing is the measurement of CPU time and real time for certain sections of the program. This may be accomplished by inclusion of some timing calls into the program. Fortran programmers could use the Fortran 95 subprograms `CPU_TIME` and `DATE_AND_TIME`. C and C++ programmers should use the appropriate system calls like `times` or `getrusage`.

In a parallel MPI program the MPI function `MPI_Wtime` may also be used to measure the wall clock time.

A simple Fortran example may look like:

```fortran
SUBROUTINE timer (real_time, cp_time)
  !
  ! timer computes :
  !
  ! real_time: the real time in seconds since midnight
  !
  ! cp_time : the CPU time consumed by the program since program start
  !
  REAL(KIND=4) :: real_time, cp_time
  INTEGER, DIMENSION(8) :: values
  CALL CPU_TIME (cp_time)
  CALL DATE_AND_TIME (VALUES = values)
  real_time = ((values(5) * 60.) + values(6)) * 60. + values(7) + &
              values(8)/1000.
END SUBROUTINE timer
```

27
PROGRAM timer_example

REAL(KIND=4) :: real_time0, cp_time0
REAL(KIND=4) :: real_time, cp_time

CALL timer (real_time0, cp_time0)

! The real time and CPU time used by subroutine compute
! will be measured.

CALL compute

CALL timer (real_time, cp_time)

real_time = real_time - real_time0
cp_time = cp_time - cp_time0

PRINT *, 'real time needed for compute : ', real_time, ' sec.'
PRINT *, 'CPU time needed for compute : ', cp_time , ' sec.'

END PROGRAM timer_example

9.2 Profiling

Profiling is used to identify those parts of a program that consume the highest amount of CPU time. In many cases more than 90% of the CPU time is used in less than 5% of the source code of the program. These most time consuming parts of the program should be optimized carefully. In some cases is is possible to replace own code by a call to some optimized functions or subprograms from highly tuned libraries (cf. section 10).

The profiling tool \texttt{gprof} is available on many Unix or Linux systems. The information you may get from this tools is:

- a flat profile with information on CPU usage by all subroutines and functions of the program and a
- a call graph profile which gives information not only on each function and subprogram, but also on its callees (number of calls, CPU time used by callee etc.).

To use the \texttt{gprof} utility the following steps are required:

1. Compile and link the program with \texttt{-pg} option.

2. Run the program as usual. When the program terminates a file \texttt{gmon.out} will be created. In case of a parallel program several output files \texttt{gmon.out.\(i\)} are written, where \(i\) is the task id.
3. To create the profiles, run

  gprof program gmon.out*

where program is the name of your executable program. To create a profile for only one or a
certain subset of tasks of a parallel application, you should replace the string gmon.out* by a
list of file names.

10  Numerical Libraries

Up to now only the Intel Math Kernel Library (MKL) has been installed on InstitutsCluster. Tuned
implementations of well established open source libraries are part of MKL.

10.1 Intel Math Kernel Library (MKL)

The Intel Math Kernel Library includes functions from following areas:

- Basic Linear Algebra Subprograms (BLAS);
- Sparse BLAS (basic vector operations on sparse vectors);
- LAPACK routines for solving systems of linear equations, least-squares problems, eigenvalue
  and singular value problems;
- direct sparse solver routines (PARDISO) and iterative solvers with user-defined routines like
  MVM;
- Vector Math Library (VML) provides highly optimized vector implementations of computa-
  tionally intensive core mathematical functions like ADD, MUL, SQR, SIN, COS, TAN, POW, EXP,
  etc; these functions could result in a significant speed up of the program, when many calls to
  basic mathematical functions consume a significant portion of the overall CPU time;
- Vector Statistical Library (VSL) today includes Convolution/Correlation and an extensive col-
  lection of Random Number Generators;
- Fast Fourier Transforms (FFT)s and
- LINPACK - Intel offers two LINPACK benchmark packages, one for SMP machines and one fpr
distributed memory computer systems.

Before linking a program with the Intel MK Libraries, a module must be loaded to set some environ-
ment variables:

  module add mkl

Now the program may be linked using the MK Libraries: In the following example we assume that
a Fortran program myprog.f90 and a C program myprog.c will be compiled and linked against the
Intel MK Libraries for the usage of BLAS-routines. The appropriate options at link time are:

- BLAS

  ifort -o myprog myprog.f90 -L$MKLPATH -lmkl_intel_lp64 -lmkl_intel_thread
  -lmkl_core -liomp5
  icc -o myprog myprog.c -L$MKLPATH -lmkl_intel_lp64 -lmkl_intel_thread
  -lmkl_core -liomp5
• FFT, VML, VSL etc.

```bash
ifort -o myprog myprog.f90 -L$MKLPATH -lmkl_intel_lp64 -lmkl_intel_thread
   -lmkl_core -liomp5 -lm
icc -o myprog myprog.c -L$MKLPATH -lmkl_intel_lp64 -lmkl_intel_thread
   -lmkl_core -liomp5 -lm
```

For usage of all other libraries of MKL the Intel Math Kernel Library Link Line Advisor should be used. It can be found on the website [http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor/](http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor/) As OS you must select - Linux; as processor architecture you must select - Intel(R) 64; in the box for dynamic or static linking you should select - dynamic; if you are choosing a cluster library (like e.g. BLACS) of MKL, you must choose OpenMPI as MPI library.

11 CAE Application Codes

The InstitutCluster is especially suited for solving physical problems. Up to now only one CAE program is available.

• structural mechanics: ABAQUS, LS-Dyna, MD Nastran, Permas;
• fluid dynamics: Fluent, ANSYS CFX, Star-CD, Star-CCM+.

All these codes are parallelized and should be started under control of the batch system requiring both the correct launching of the program and submitting it as a batch job. To facilitate this for the user, a command is provided which handles both tasks.

Other applications are

• COMSOL Multiphysics;
• Matlab;
• Pre/Postprocessing, Visualization: EnSight, Gambit, HyperWorks, ICEM CFD.

11.1 ABAQUS

ABAQUS is a widely used program, based on the Finite Element technique, to solve problems covering the following features:

• linear and nonlinear stress/displacement problems,
• heat transfer and mass diffusion,
• acoustics,
• coupled problems (thermo-mechanical, thermo-electrical and more),
• all these problems may be static or dynamic (with implicit and explicit time integration),
• a large variety of material models are available,
• submodeling and substructuring,
• mesh adaption,
• design optimization,
• and much more.

A complete overview can be found in http://www.abaqus.com/products/products_standard.html.

The ABAQUS documentation can be accessed interactively by the command abaqus doc

To start ABAQUS in batch modes the following command should be used:


Parameters are:

- j jobname
- t CPU-time in minutes
- m main memory in MByte
- c job-class (p or d; default is p)
- T real time in minutes (optional)
- p number of parallel tasks (default is 1)
- i inputfile without .inp (optional)
- o old jobname on *RESTART and *POST OUTPUT (optional)
- f new or append
- u user-subroutine

-D selection of the Direct Solver in ABAQus/Standard (if p > 1): y or n (default is n)
- s string with further options

11.2 LS-DYNA

LS-DYNA is a general-purpose, implicit and explicit finite element program that is employed to analyze the nonlinear static and dynamic response of three-dimensional inelastic structures. Its fully automated contact analysis capabilities and error-checking features have enabled users worldwide to solve successfully many complex crash and forming problems.

In addition to LS-DYNA the tool LS-PREPOST for pre- and postprocessing is available. There are also well established interfaces to HyperWorks. More information about the program can be found in http://www.dynamore.de, where also manuals and tutorials are available.

The main applications are:

• Large Deformation Dynamics and Contact Simulations
• Crashworthiness Simulation
• Occupant Safety Systems
• Metal Forming
• Metal, Glass, and Plastics Forming
• Multi-physics Coupling
• Failure Analysis

The submitting command is as follows:

```bash
lsdynajob -j ID -t TIME -m MEMORY [-c CLASS] [-p PROCS] [-T TIME] [-s STRING]
```

Parameters are:

- `j` jobname
- `t` CPU-time in minutes
- `m` main memory in MByte
- `c` job-class (p or d; default is p)
- `p` number of parallel tasks (default is 1)
- `T` real time in minutes
- `s` string with further options

### 11.3 MD Nastran

MD Nastran is also a finite element code to solve structural mechanics problems. A short description can also be found in [http://www.mscsoftware.com/products](http://www.mscsoftware.com/products) As pre- and postprocessors for Nastran models Patran and HyperMesh licenses are available.

The documentation is in PDF and can be found in the directory

/software/all/msc/nastran/md20071/doc/pdf

on InstitutsCluster.

The command is

```bash
nastranjob -j ID -t TIME -m MEMORY [-c CLASS] [-p PROCS] [-e SCRATCHDIR] [-T TIME] [-s STRING]
```

Parameters are:

- `j` jobname
- `t` CPU-time in minutes
- `m` main memory in MByte
- `c` job-class (p or d, default is p)
- `T` real time in minutes (optional)
- `p` number of parallel tasks (default is 1)
- `e` directory to store scratch files ($WORK or $TMP; default is $WORK)
- `s` string with further options

32
The capabilities of the most CFD codes include

- stationary and instationary flow,
- laminary and turbulent flow,
- compressible and incompressible flow,
- multiphase and multiparticle flows,
- chemical reactions and combustion,
- newtonian and non-newtonian fluids,
- free surfaces,
- coupled heat transfer and convection,
- and many more.

11.4 PERMAS

PERMAS is also a general purpose finite element program, which the whole spectrum of functionalities of a widespread analysis code. A detailed description can be found in [http://www.intes.de](http://www.intes.de). Since PERMAS is a pure analysis code, model generation and results visualization must be performed by external programs. PERMAS offers a lot of interfaces to well-established pre- and postprocessors, such as MSC.Patran and HyperWorks. Currently the license is limited to one process with up to 8 parallel threads.

The documentation is online and can be accessed by input of the command `permasdoc`.

The PDF version is available in the directory `/software/all/intes/documentation/onldoc_v11.120/permas`.

PERMAS is well parallelized in thread based mode. Therefore it cannot be processed on multiple nodes and the number of processors is limited to the number of cores of a single node.

PERMAS is invoked as a batch job by the command

```bash
permasjob -j ID -t time -m MEMORY -c CLASS [-T TIME] [-p PROCS] [-e SCRATCH] [-s STRING]
```

Parameters are:

- `-j projectname`
- `-t CPU-time in minutes`
- `-m main memory in MByte`
- `-c job-class (p or d; default is p)`
- `-T real time in minutes (optional)`
- `-p number of parallel tasks (p ≤ 8, default is 1)`
- `-e specify the environment variable for scratch files ($WORK or $TMP; default is $TMP)`
- `-s string with further options (optional)`
11.5 Fluent

At the moment the solver of Fluent 6.3.26 is installed. The preprocessor and mesh generator for Fluent, Gambit, is available for Windows and several Linux distributions. For a graphical representation with Fluent, the Fluent code itself can be used or an installation of any visualisation code like e.g. EnSight are suited. The documentation is provided interactively in the Fluent GUI after pushing the Help button. General information is presented under [http://www.fluent.com](http://www.fluent.com).

The current license allows up to 12 parallel processes per Fluent job. The batch command is:

```
fluentjob -j ID -v VERSION -t CPU-time -m MEMORY [-c CLASS] [-T TIME] [-p PROCS]
```

Parameters are:

- `-j jobname`
- `-t CPU-time in minutes`
- `-m main memory in MByte`
- `-c job-class (p or d; default is p)`
- `-T real time in minutes (optional)`
- `-p number of parallel tasks (default is 1)`
- `-v 2d|3d|2ddp|3ddp for different FLUENT versions`

More information can be found at [http://www.rz.uni-karlsruhe.de/produkte/1404.php](http://www.rz.uni-karlsruhe.de/produkte/1404.php).

11.6 ANSYS CFX

ANSYS CFX consists of 3 modules:

- CFX-Pre to import the mesh and formulate the model,
- CFX-Solver to configure and start the solver,
- CFX-Post to postprocess the results.

The mesh can be generated by codes like ANSYS ICEM CFD or ANSYS Workbench, which must be installed on local sites. CFX-Pre and CFX-Post can be used interactively on local installations or on the login nodes of InstitutsCluster. The solver should be operated in batch mode:

```
cfx5job -j IDENT -t TIME -m MEMORY [-c QUEUE] [-R NAME] [-p PROCS]
[-s STRING] [-T TIME]
```

Parameters are:

- `-j jobname` without .def
- `-t CPU-time in minutes`
- `-m main memory in MByte`
- `-c job-class (p or d; default is p)`
- `-T real time in minutes (optional)`
- `-p number of parallel tasks (default is 1)`
-R name of the result file for restart

-s string with further options

The maximum licensed number of parallel processes is 50. The documentation is available by the online help system or as PDFs in the directory /software/all/ansys_inc/v110/CFX/help/pdf. More information can be found under http://www.ansys.com/products/cfx.asp.

11.7 Star-CD

The Star-CD suite contains the meshing and modeling modules pro-STAR and pro-am (the automatic mesher). The solver can be started from the GUI of these modules or as a batch job:

starcdjob -j CASE -t TIME -m MEMORY [-c QUEUE] [-p PROCS] [-s STRING] [-T TIME]

Parameters are:

-j case-name
-t CPU-time in minutes
-m main memory in MByte
-c job-class (p or d; default is p)
-T real time in minutes (optional)
-p number of parallel tasks (default is 1)
-s string with further options

The parallelisation is licensed for up to 124 processors. The documentation is online available as PDF. The product’s web site is http://www.cd-adapco.com

11.8 STAR-CCM+

STAR-CCM+ is parallel development to CD-adapco’s STAR-CD CFD code with similar functionality but a complete different user interface and workflow. An overview can be found on the web site http://www.cd-adapco.com. In interactive mode STAR-CCM+ can be started by the command

starccm+

Be sure to provide enough memory by opening a Xterm on an exclusive node via a job_submit command. A STAR-CCM+ model may be prepared, the solution process should be performed as a batch job:

ccm+job -j IDENT -t TIME -m MEMORY [-p PROCS] [-c QUEUE] [-T TIME]

Parameters are:

-j name of a simulation file filename.sim
-t CPU-time in minutes
-m main memory in MBytes
-c job-class (p or d; default is p)
The complete documentation is online and available as PDF.

11.9 COMSOL Multiphysics

COMSOL Multiphysics is an application for almost all engineering regions based on the Finite Element Method. It is able to couple all physical areas like structural mechanics, fluids, heat etc.

Basically, COMSOL Multiphysics is interactively oriented and an access to the program goes over a GUI. Nevertheless it is possible to run COMSOL in batch mode and thus under the JMS environment using the `job_submit` command.

- Create the model as usual via the GUI and save it as `filename.mph`.

A COMSOL job for batch processing is a string of the form

```
comsol [options] batch -input filename.mph [-output filename_out.mph]
```

The file `filename_out.mph` in the `-output` option contains the model and the result. If it is omitted, the input file is overwritten. There are several options possible, the most important are listed below. A complete description can be found in the "Installations and Operations Guide":

-32 running the model as a 32-bit model;
-64 running the model as a 64-bit model;
-blas selecting a specific Blas library; the default is auto, other alternative can be found in the "Installations and Operations Guide";
-np number of parallel tasks number of processors; running the model in SMP mode on one node;
-tmp scratch file system

Example:
```
job_submit -c p -p 1/4 -t 10 -m 2000 "comsol -64 -np 4 batch -input filename.mph"
```

The memory must be chosen large enough. Check it out by trial and error. The documentation is online, the COMSOL web site is [http://www.comsol.de/](http://www.comsol.de/).

11.10 Matlab

Matlab is an extremely versatile program for problems covering the areas mathematics, engineering, biology, financial, statistics and a lot more. Matlab can be used interactively, but for large numerical problems it may be advisable to run it in batch mode. For this some feature should be deactivated and a m-File must be provided.

```
matlab -nodesktop -nojvm -nosplash < filename.m
```

runs a Matlab job without opening the usual desktop and the Java VM. The welcome screen is suppressed. Any graphics from any commands in the m-File is also suppressed. This command should be run under `job_submit`, especially if large memory and cpu times are needed and if the job should run multithreaded.

Further optimization of Matlab can be achieved.
• by enabling the toolbox path cache: (File >> Preferences... >> General), check the boxes in the "Toolbox path caching" area and press the button;

• on computers or nodes with multiple processors Matlab will determine the maximum number of cores and will distribute threads on these by default. This should be considered by the parameter 
   -p 1/n in the job_submit command. The number of threads can be specified explicitly by a command 
   maxNumCompThreads(n) in the M-File; if multithreading should be prevented, the following option should be set as a Matlab startup option: -singleCompThread

The documentation is online available, the web site can be found in http://www.mathworks.com/

11.11 Pre- and Postprocessors, Visualisation Tools

There are several tools for modeling, meshing and postprocessing. These are

• EnSight
• Gambit
• HyperWorks
• ANSYS ICEM CFD

A detailed description is available on http://www.rz.uni-karlsruhe.de/produkte/produkte.php and links given there. There is no specific handling of these programs on InstitusCluster.

12 Batchjobs

As described in section 2, the majority of the nodes of InstitutsCluster is managed by the batch system. Batch jobs are submitted using the command job_submit. The main purpose of the job_submit command is to specify the resources that are needed to run the job. job_submit will then queue the job into the input queue. The jobs are organized into different job classes like development or production. For each job class there are specific limits for the available resources (number of nodes, number of CPUs, maximum CPU time, maximum memory etc.). These limits may change from time to time. The current settings are listed with the command job_info. The command job_queue [-l] shows your queued jobs in standard or in long format.

<table>
<thead>
<tr>
<th>Important Batch commands</th>
<th>Brief Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_submit</td>
<td>submits an job and queues it in an input queue.</td>
</tr>
<tr>
<td>job_cancel</td>
<td>cancels an job from the input queue or a running job.</td>
</tr>
<tr>
<td>job_info</td>
<td>shows the different input queues and their specific limits for the available resources.</td>
</tr>
<tr>
<td>job_queue</td>
<td>shows your queued or running jobs in standard or long format.</td>
</tr>
<tr>
<td>jobacct</td>
<td>shows several resource data of your (running, cancelled or completed) jobs.</td>
</tr>
</tbody>
</table>

When the resources requested by a certain job become available and when no other job with higher priority is waiting for these resources, then the batch system will start this job.
12.1 The job_submit Command

The syntax of the job_submit command is available with

```
job_submit -H
```

The most important options are:

```
job_submit -t time -m mem -c class[+] -p i[/j] [-T time] [-M mem] [-J "jobname"]
[-l af|aF|Af|AF] [-A account] [-N[s][b][c|C|e|E]] [-i file] [-o file]
[-e file|+] [-x[+|-]] job
```

- `time`: maximum CPU time (minutes) on each CPU that is allocated to the job. The job will be terminated when one task exceeds its CPU time limit.

- `-T time`: maximum elapsed time (minutes). The job will be terminated, when this time is exceeded. For many applications the elapsed time will not be much higher than the CPU time. Exceptions are I/O intensive applications which need a much higher elapsed time than CPU time. If this option is omitted, the default value is a function of the requested CPU time ($T = 1.01 \times t + 1$; $t$ is time from `-t`).

- `mem`: maximum memory requirement per task in Mega Bytes. The 8-way nodes of InstitutsCluster are equipped with 16 GB of main memory.

- `-M mem`: maximum virtual memory requirement per task in Mega Bytes. This option allows users to use the memory management of the operating system and can strongly downgrade the system performance, if it is not properly used. So this option is only available for special users.

- `-J "jobname"`: the job gets the name `jobname`. `jobname` is an arbitrary string of maximum 16 chars.

- `-l af|aF|Af|AF`: the sign a or alternatively A means that account information is switched on or off; the sign f or alternatively F means that displaying of floating point exceptions is switched on or off. Default is `-l af`.

- `-A account`: additional accounting information (only for special customers). `account` is a text string.

- `-N[s][b][c|C|e|E][:mailaddress]`: this option allows the automatic sending of mails on the basis of events:
  - `s`: submitting the job triggers the sending of a mail to `mailaddress`.
  - `b`: starting the job triggers the sending of a mail to `mailaddress`.
  - `c`: complete end of the job triggers the sending of a mail to `mailaddress`. Begin and end of STDOUT and STDERR will be sent by mail.
  - `C`: complete end of the job triggers the sending of a mail to `mailaddress`. Complete STDOUT and STDERR will be sent by mail.
  - `e`: Erroneous end of the job triggers the sending of a mail to `mailaddress`. Begin and end of STDOUT and STDERR will be sent by mail.
  - `E`: Erroneous end of the job triggers the sending of a mail to `mailaddress`. Complete STDOUT and STDERR will be sent by mail.

If the mailaddress is omitted the mailaddress bound to the userid will be chosen.

- `-p i[/j]`: number of tasks ($i$) and threads per task ($j$)
  - default: $j = 1$

  This option defines how many processors are required to run the job.

- If the job is single threaded, i.e. it is a serial or a pure MPI program without any usage of OpenMP or other multithreading techniques, then one CPU per task is needed. The format of this option is `-p i` where $i$ is the number of tasks.
• When the program is an OpenMP parallelized program which does not contain any MPI calls, then the number of tasks is 1 and the number of threads must not exceed 8. The format of -p option in this case is \(-p 1/j\) where \(j\) is the number of threads.

• When both parallelization techniques (e.g. MPI and OpenMP) are used, then \(i\) is the number of MPI tasks and \(j\) is the number of threads per MPI task. The command job_info shows the valid combinations of \(i, j\) and mem.

\(-c\) class[\(+\)]: this option defines the job class. The sign \(+\) means higher priority (only available for special customers). Two job classes are available on InstitutsCluster:

\(d\): jobs in this class will start immediately, but do not have exclusive access to any resources of InstitutsCluster. All cores of this class are operated in mode 'shared', i.e. multiple tasks can be executed simultaneously on a single core. Performance measurements are not reasonable in this class.

The class \(d\) is typically used for program development and test.

\(p\): jobs in class production are distinguished by the exclusive access of nodes or cores. Thus two different operating modes can be chosen. In the first operating mode - exclusive use of nodes - always whole nodes, i.e. all 8 cores of one node, are accessed. This can lead to unused cores. In the second operating mode - exclusive use of cores - only several cores of one node are used exclusively. This implicitly means that unused cores of the node can be used by another program and thus the memory is not used exclusively.

\(-i\) stdin_file: when the option \(-i\) stdin_file has been selected the job will be executed as if job `< stdin_file` would have been specified (default: /dev/null).

\(-o\) stdout_file: the standard output of the job is written to the file named in this option. When the \(-o\) option is omitted the default output file is Job_$JID.out where $JID is a unique identification number of a job. It is created when job_submit is launched.

\(-e\) stderr_file: the error messages of the job are written to the selected file. If this option is omitted all error messages are written to a file Job_$JID.err. When a job does not generate any error messages, then the standard error file is deleted at job termination. Choosing \(-e +\) means to concatenate standard error with standard output, i.e. to write standard error into the standard output file.

\(-x[\+|\-]\): this option should be used carefully. If \(-x+\) is chosen, exclusive access of nodes will be used. This means that no further executables will run on the requested nodes and that all nodes will completely charged onto your account. If \(-x-\) is chosen, exclusive use of nodes will be switched off. This means that further executables can run on the requested nodes (but not on the requested cores) and that only the processors which are used by your executable will be charged onto your account. If this option is omitted the batch system decides if exclusive use of nodes or exclusive use of cores will be chosen to run the job.

\(job\): this is a parallel program call or a shell script to be executed on InstitutsCluster. When the invocation of the job requires additional arguments, the parallel program call or the script with arguments may be enclosed in quotes or double quotes, but they can also be omitted.

Important remark: please read this paragraph before starting jobs in the production pool! First, the production pool contains nodes with 8 cores. Second, it always holds the equation: number_of_requested_processors = number_of_requested_tasks * number_of_requested_treads \((p = i * j)\). If you are asking for more than 8 cores the batch system must allocate more than one node. In this case exclusive use of nodes is chosen automatically \((-x+)\). If you want to switch off exclusive use of cores you must choose the option \(-x-\). If you are asking for 8 or less cores and for 16 GB or less of main memory, then the batch system decides that your job will run within one node. If you are using \(p < 8\) cores, then be aware that \(8 - p\) cores are idling (and completely accounted)! In this case “shared” use of cores is chosen automatically \((-x-\)). If you want to switch on exclusive use of the whole node you must choose the option \(-x+\).
12.2 Environment Variables for Batch Jobs

Parameters can also be set by environment variables. The syntax is `export JMS_parameter=value`. Examples are: `export JMS_t=10; export JM_o=stdout_file; export JMS_job="mpirun a.out"`. Parameters set in the command line overwrite parameters set by the environment. The command `job_submit` replaces chosen parameters by the appropriate environment variables and exports them to the user job.

Now some useful environment variables will be explained. You can get the complete list of environment variables by calling the shell command `set` in a batch job.

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Brief Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS_t</td>
<td>contains the value of the CPU time limit which has been defined with option -t. This value can be used to compute the amount of CPU time within a program that is still available for the computation.</td>
</tr>
<tr>
<td>JMS_T</td>
<td>contains the maximum elapsed time as specified with the option -T.</td>
</tr>
<tr>
<td>JMS_m</td>
<td>contains the memory requirement as specified with the -m option.</td>
</tr>
<tr>
<td>JMS_Nnodes</td>
<td>contains the allocated number of nodes.</td>
</tr>
<tr>
<td>JMS_p</td>
<td>contains the requested number of processors.</td>
</tr>
<tr>
<td>JMS_tasks</td>
<td>contains the number of MPI tasks specified with the first value i in the -p option.</td>
</tr>
<tr>
<td>JMS_threads</td>
<td>contains the number of threads per task (process) as specified with the second value j in the -p option. After setting JMS_tasks the following assignment is done: <code>OMP_NUM_THREADS=$JMS_tasks</code></td>
</tr>
<tr>
<td>JMS_c</td>
<td>defines the job class as specified with option -c.</td>
</tr>
<tr>
<td>JMS_start_time</td>
<td>gives the starting time of the job if it is in state running.</td>
</tr>
<tr>
<td>JMS_submit_time</td>
<td>gives the time at which the job has been submitted.</td>
</tr>
<tr>
<td>JMS_submit_node</td>
<td>contains the name of the node the job has been started on.</td>
</tr>
<tr>
<td>JMS_node0</td>
<td>contains the name of the first node.</td>
</tr>
<tr>
<td>JMS_nodes</td>
<td>lists all used node names. If e.g. 8 processors per node are used the used nodes are listed eight times.</td>
</tr>
<tr>
<td>JMS_stdin</td>
<td>contains the name of the input file of the job.</td>
</tr>
<tr>
<td>JMS_stdout</td>
<td>contains the name of the output file of the job.</td>
</tr>
<tr>
<td>JMS_stderr</td>
<td>contains the name of the standard error file of the job.</td>
</tr>
<tr>
<td>JMS_pwd</td>
<td>contains the name of the output directory of the job.</td>
</tr>
<tr>
<td>JMS_user</td>
<td>contains the userid of the user who has submitted the job.</td>
</tr>
<tr>
<td>JMS_group</td>
<td>contains the groupid of the user who has submitted the job.</td>
</tr>
<tr>
<td>TMP and TEMP</td>
<td>contain the working directory for temporary files of the job.</td>
</tr>
</tbody>
</table>

12.3 job_submit Examples

12.3.1 Serial Programs

1. To submit a serial job that runs the script `job.sh` and that requires 5000 MB of main memory, 3 hours of CPU time and 4 hours of wall clock time the command

```
job_submit -t 180 -T 240 -m 5000 -p 1 -c p job.sh
```

may be used. The high wall clock time (`-T 240`, i.e. 4 hours) is necessary when the program does a lot of I/O. In most other cases the wall clock time will only be slightly larger than the CPU time (`-t` option). Further jobs can and probably will run on the node your job runs on.

2. Now we want to resubmit the same job, but a certain argument, e.g. `-n 100` has to be passed to the script, i.e. the command `job.sh -n 100`, has to be executed within the batch job. The appropriate `job_submit` command is now:
12.3.2 Parallel MPI Programs

For your understanding you must know: **parallel programs (no shell scripts) must be launched by calling mpirun parallel program; shell scripts only run on the first processor!**

1. We want to run 4 tasks of the program `my_par_program` within a batch job in the job class development. Each task has a CPU time limit of 10 minutes and the memory requirement per task is 3000 MB. The wall clock time limit is set to 1 hour. This may be necessary when the nodes for the development class are heavily loaded and many other processes are using these nodes at the same time. The appropriate `job_submit` command is

   ```
   job_submit -t 10 -T 60 -m 3000 -p 4 -c d "mpirun my_par_program"
   ```

   or

   ```
   job_submit -t 10 -T 60 -m 3000 -p 4 -c d mpirun my_par_program
   ```

2. The same program will now be run in the production class on 16 processors. The maximum CPU time is 4 hours, the memory requirement per task is 3500 MB and exclusive use of nodes is switched off.

   ```
   job_submit -t 240 -m 3500 -p 16 -c p -x- mpirun my_par_program
   ```

   As one node only contains 16 GB and the memory requirement per 4 tasks is 14 GB, only 4 tasks can be run per node. Thus the job runs on 4 nodes. On the other 4 cores of each of the 4 nodes further jobs with the overall memory requirement of 2 GB per node can be run.

3. A third job sample includes the following functions:

   - create a subdirectory `Job_Output` within `$WORK`,
   - select `$WORK/Job_Output` as current working directory,
   - run 64 tasks of the program `my_par_program` (CPU time limit: 3 hours, memory requirement: 2000 MB). The program `my_par_program` is stored in `$HOME/bin`.

In order to accomplish this, a shell script is needed. Let `job.ksh` be the name of this script. Its content is:

```ksh
#!/bin/ksh
#
cd $WORK
if [ ! -d "Job_Output" ]
  then
  mkdir Job_Output
fi

cd Job_Output

mpirun $HOME/bin/my_par_program
```

To submit this job, use the commands

```
chmod u+rx job.ksh
defaults -c p -p 64 -t 180 -m 2000 job.ksh
```
The if statement enables the user to run this job several times. It will not abort while trying to create a directory Job_Output that already exists.

When the script job.ksh is executed, the master node of this job will run the shell commands like `cd` or `mkdir` and any other serial commands or programs. Only parallel programs launched by the command `mpirun` will be executed on all these processors that are requested by the -p option of the command `job_submit`. The job will run on 8 nodes (8 tasks per node) with exclusive use of nodes. Adding the option -x- will take no effect because there are no idle cores on the requested nodes.

4. In order to run several parallel and serial programs within one batch job, again a ksh script is needed which contains the commands to start the programs.

Let us assume that we want to run the two parallel programs `my_first_parallel_prog` and `my_second_parallel_prog`. Both programs are stored in the directory $HOME/project/bin. Before starting the second program we want to copy a file `results_1` from the current working directory, which is $WORK, into the $HOME directory. The job script job_2.ksh may now look like this:

```bash
#!/bin/ksh

cd $WORK

mpirun $HOME/project/bin/my_first_parallel_program

if [ "$?" = "0" ]
then
  # program terminated successfully, copy data and start next program
  cp results_1 $HOME/results_1
  if [ "$?" = "0" ]
  then
    # file result_1 has been copied successfully, next program may be started
    mpirun $HOME/project/bin/my_second_parallel_program
  else
    echo 'File results_1 could not be copied into $HOME directory'
    exit 1
  fi
else
  echo 'Program my_first_parallel_program terminated abnormally'
  exit 2
fi
```

To run this script on 32 cores (CPU time limit: 4 hours, memory limit: 5000 MB) use the `job_submit` command:

```bash
job_submit -c p -p 32 -t 240 -m 5000 job_2.ksh
```

Within this job first the cd command is executed on the master node. Then the program `my_first_parallel_prog` is executed with 32 MPI tasks on 11 nodes (3 - once only 2 - cores per node). The default mode - exclusive use of nodes - is chosen. So 5 (once only 6) cores per node are idling. When all tasks have been terminated, the master node copies the file `results_1` into the $HOME directory before the parallel execution of `my_second_parallel_prog` is initiated.
12.3.3 Multithreaded Programs

For programs based on OpenMP the OpenMP specification defines an environment variable
OMP_NUM_THREADS to select the number of threads. For details on these variable see the documentation
of Fortran and C compiler at
http://www.rz.uni-karlsruhe.de/ssck/ic-manuals

The variable OMP_NUM_THREADS is automatically initialized by job_submit to the value of j as specified
by the option -p i/j.

The following examples illustrate the usage of job_submit command with multithreaded applications:

1. run the program my_openmp_prog with 2 threads, a CPU time limit of 4 hours per thread and
   a memory requirement of 2 GB:
   
   job_submit -c p -p 1/2 -t 240 -T 300 -m 2000 my_openmp_prog

   Because this program has not been parallelized with MPI, it consists of one single process that
   is split into several (in this case) 2 threads. This is described by the option -p 1/2.

   The operating system computes the CPU time on a per process basis, i.e. the CPU times of
   all threads of a process are added. To reflect this, the job_submit command multiplies the
   requested CPU time by the number of threads. In this case we have a limit of 8 hours. If there
   is a good load balance between the threads, each thread may consume approximately 4 hours of
   CPU time. Since the job will run on one node and the option -x is omitted, the default mode -
   exclusive use of cores - will be chosen.

   If there is a poor load balance among threads the available time for this job is limited by the
   wall clock time, i.e. five hours.

2. Now we want to run the same program with 6 threads and 3 hours of CPU time per thread
   and again 2 GB of main memory. In addition we want to select dynamic scheduling, i.e. the
   amount of work in parallelized loops is dynamically assigned to 6 threads. In a shell script
   the environment variable OMP_SCHEDULE is set to dynamic to inform the runtime system about
   dynamic scheduling.

   The script openmp_job.ksh looks like:

   #!/bin/ksh
   #
   export OMP_SCHEDULE="dynamic"
   #
   my_openmp_prog

   It is submitted to the batch system with the command

   job_submit -c p -p 1/6 -t 180 -T 300 -m 2000 openmp_job.ksh

12.3.4 Programs using MPI and OpenMP

When running programs using distributed memory parallelism (e.g. MPI) as well as shared memory
parallelism (e.g. OpenMP) both arguments (i and j) of the -p option of the job_submit command
must be specified.

1. To run the program my_parallel_program on 32 8-way nodes with eight threads per node and
   16 GB of main memory per MPI task, the job_submit command may look like:

   job_submit -c p -t 240 -T 300 -m 16000 -p 32/8 mpirun my_parallel_prog

2. To run a program with 2 MPI tasks and 4 threads on one 8-way node, the job_submit command
   will be:

   job_submit -c p -t 240 -T 300 -m 16000 -p 2/4 mpirun my_parallel_prog
12.4 Commands for Job Management

There exist several commands to list, cancel or query jobs. To identify an individual job, a unique job-id which is determined by job_submit is associated with each job.

The job_submit command returns a message

job_submit: Job job_id has been submitted.

The job_id is the unique identification of this specific job and will be used in all job management commands to identify the job.

The job management commands to list, cancel or query jobs are job_queue, job_cancel and job_info.

job_queue [-l] The output of job_queue lists all jobs, its job identification and its specific requirements. If you are using the option -l, further informations will be printed.

A sample output of job_queue is:

<table>
<thead>
<tr>
<th>job-id</th>
<th>c</th>
<th>P</th>
<th>n/i/j</th>
<th>t</th>
<th>T</th>
<th>m</th>
<th>queued</th>
<th>s</th>
<th>start</th>
<th>end(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>474</td>
<td>p</td>
<td>t</td>
<td>1/8/1</td>
<td>10</td>
<td>11</td>
<td>2000</td>
<td>22/15:57</td>
<td>r</td>
<td>22/15:57</td>
<td>22/16:07</td>
</tr>
</tbody>
</table>

- The first column shows the complete job identification. To select a job within the job management commands it is sufficient to specify the job-id (it is a numerical value).
- The second column displays the job class (column c). The job in this example belongs to class production (p). Jobs in class development will show a d in this column.
- The next column displays the partition (column P) the job runs in. The job in this example runs in the partition "thin nodes" (t). There are no further partitions.
- The next n/i/j shows how many nodes, MPI-processes and threads per node are requested for this job.
- The next two columns show the requested time in minutes.
- The column m shows the requested memory.
- The column queued and start and end show the times when the job is queued and when it has been started and when it will end at the latest time in the format day/wall clock time.
- The column s gives the status of the job which is r for running, w for waiting or L for looping in job chains (see next section).

job_cancel deletes a waiting job from the input queue or aborts a running job.

To delete the job with job-id 565 from the input queue, just enter the command

job_cancel 565

job_info lists the current settings for the job classes, i.e. the limits for CPU time, number of nodes, amount of memory, number of jobs per user etc.

These values may change from time to time reflecting the varying requirements and available resources.

12.5 Command for Showing System Data

The command job_acct shows system data of your running, cancelled or completed jobs. To identify an individual job, a unique job-id which is determined by job_submit is associated with each job.

The job_submit command returns a message
**job_submit:** Job **job-id** has been submitted.

The *job-id* is the unique identification of this specific job and must be used in the command **job_acct** to identify the job. The syntax of the command is

**job_acct** *job-id*.

The command shows data like number of cpus, sum of cpu time over all processors, elapsed cpu time, maximum physical memory by any process, maximum virtual memory by any process, maximum number of minor page faults for any process, total number of voluntary context switches for all processes, the status of the job and an error variable.

By or for any process means that the maximum over all processors is calculated and for all processes means that the sum over all processes is calculated.

If hardware address translation fails, the CPU switches into a special execution context (virtual memory context) to ensure that a physical memory page is allocated for the virtual page, and the page is then refreshed from disk if necessary. The page table entry is also copied into the Translation Lookaside Buffer (TLB). Such hardware address translation failures are called page faults. If a page has to be read from disk, it is called a hard or major page fault; on InstitutsCluster paging is not allowed (all data are kept in memory), thus the major page fault rate is always zero. A soft or minor page fault occurs if a page is attached the first time, thus the number of such faults should be about the maximum physical memory divided by 4 KB.

Voluntary context switches represent the number of times a process releases its CPU time-slice voluntarily before it’s time-slice allocation is expired. This usually occurs when the process needs an external resource, like making an I/O call for more data.

A important remark on the displayed number of cores:
for parallel programs the displayed number of cores usually means the number of accounted cores, i.e. 8 cores are accounted both for serial programs and for all kinds of parallel programs from 2 up to 8 cores, if the option `-x+` is chosen explicitly or per default! If the option `-x-` is chosen explicitly or per default, the number of displayed cores differs from the number of accounted cores, because the number of displayed cores will always be 8 and the number of accounted cores will be the number of requested cores.

### 12.6 Job Chains

The CPU time requirements of many applications exceed the limits of the job classes. In those situations it is recommended to solve the problem by a job chain. A job chain is a sequence of jobs where each job automatically starts its successor. To implement a job chain, the program must be prepared for restarting and the job script must contain some additional statements for file management and for starting of the next member in the chain.

A program that enables a restart functionality must write the intermediate results, that are needed to resume the computation, to an output file before a time limit is reached. This results in the following structure of the program:

```plaintext
if (first_run ) then
    initiate computation
else
    read restart_file
end if

main_loop: do
    compute next_step
    if (time_limit reached) then
```
The job script that implements a job chain contains a `job_submit` command to resubmit the same script again or to submit a different job script. The script must also save and rename the restart files. Care must be taken, that the chain can easily be restarted when it has been broken accidentally.

The following rules are especially important for job chains:

- The `job_submit` command to submit the next member of the job chain should always be activated at the end of a job. `job_submit` will check if the time interval from start of the job until submission of the next job exceeds a certain threshold value (30 seconds). If not, the submitted job will not run automatically. This feature will prevent job chains from looping, i.e. by mistake every few seconds a new job may be submitted without doing any reasonable work. Looping job chains are indicated by an 'L' in the status column of the `job_queue` output.

### 12.6.1 A Job Chain Example

Within a batch job we want to run the parallel program `my_par_prog`. The program writes its intermediate results to a file named `restart`. This file is the input file for the next step in the job chain, i.e. the next invocation of `my_par_prog` will read this file. Each single job in the job chain will use 4 hours of CPU time. The job chain will terminate, when 20 jobs have been executed. The job script is `job_chain_1.bash`.

```bash
#!/bin/bash
#
# Sample job scripts job_shain_1.bash
# implements a simple job chain
#
# The chain will terminate after at most 20 runs.
#
MAX_JOBS=20
#
if [ "$JOB_COUNTER" = "" ]
then
  echo "============================================================================"
  echo "= Variable JOB_COUNTER not initialized ="
  echo "= job chain aborted! ="
  echo "============================================================================"
  exit 1
fi
#
# Run the program my_par_prog and save the return code in RETURN
#
mpirun my_par_prog
RETURN=$?
#
# Check the return code of my_par_prog
#"
if [ "$RETURN" = "0" ]
then
#
# program terminated successfully
#
# - save restart file,
#
cp restart Restart_Files/restart_$JOB_COUNTER
RETURN_CP=$?
#
# Check return code of cp command
#
if [ $RETURN_CP != "0" ]
then
    echo "======================================================="
    echo "= Copy command failed, restart file not saved! ="
    echo "= job chain aborted! ="
    echo "================================================================="
    exit 2
fi
#
# Restart files may also be copied into tape archive using tsm_archiv command
#
# archive Restart_Files/restart_$JOB_COUNTER
# RETURN_ARCHIVE=$?
#
# check return code of archive command
#
# if [ "$RETURN_ARCHIVE" != "0" ]
# then
#    echo "================================================================="
#    echo "= Archive command failed, restart file not archived! ="
#    echo "= job chain aborted! ="
#    echo "================================================================="
#    exit 3
# fi
#
# Old restart files may be deleted here if they are no longer needed.
#
# if [ $JOB_COUNTER -gt 1 ] ; then
#    rm Restart_Files/restart_'expr $JOB_COUNTER - 1'
# fi
#
# - increment JOB_COUNTER
#
#   JOB_COUNTER='expr $JOB_COUNTER + 1'
#
# - submit next job
#
# if [ $JOB_COUNTER -lt $MAX_JOBS ]
then
    job_submit
# The new job will be submitted with the same parameter as the last job
# using JMS_ environment variables!!!
else
    47
To initiate this job chain the following command must be entered:

```bash
export JOB_COUNTER=0
job_submit -c p -m 1000 -t 240 -p 64 job_chain_1.bash
```

In the above job script the file `restart` is copied to the directory `Restart_Files` and the value of `JOB_COUNTER`, i.e. the actual number of this job within the job chain, is appended to the file name. It must be checked carefully if this command has been completed successfully. All results computed so far are stored in restart files. These files should be saved from time to time.

When this job chain has been interrupted, it can easily be restarted manually. The latest restart file has to be copied from the directory `Restart_Files` to the working directory and must be renamed `restart`. Then the environment variable `JOB_COUNTER` must be set to the correct values, i.e. the number of jobs that have been completed successfully. Now the job chain can be started again with the `job_submit` command.

Instead of running a job a fixed number of times within a job chain, another method may be to terminate the chain when the program signals a successful completion of the whole computation by a nonzero return code.

### 12.6.2 Get remaining CPU Time

One problem with a job chain is to determine the amount of time that is still available for computation. The Fortran subroutine `time_left` will compute this value assuming that all the CPU time is spent in one program. An application program may call this subprogram and write the restart file and terminate execution, when the remaining time falls below a certain limit.

```fortran
SUBROUTINE time_left (time_remaining)
!
! time_left computes the difference between the environment variable
! $JMS_t and the CPU time consumed from start of the program.

IMPLICIT NONE

include 'mpif.h'

REAL time_remaining
```

48
REAL cputime, max_cpu_time
CHARACTER*10 string_max_time
INTEGER ierror, max_time

! Get CPU time consumed by each task and compute the maximum value
CALL cpu_time (cputime)

CALL MPI_Allreduce (cputime, max_cpu_time, 1, MPI_REAL, 
&                  MPI_MAX, MPI_COMM_WORLD, ierror)

! getenv delivers the value of environment variable JMS_t
CALL getenv ('JMS_t', string_max_time)

! Convert this value into integer format
READ (string_max_time, *) max_time

! Compute the remaining CPU time

  time_remaining = REAL(max_time)*60. - max_cpu_time

END

13  Technical Contacts at SCC of University Karlsruhe

• IC Hotline
  Email: ic-hotline@lists.uni-karlsruhe.de
  Phone: +49 721 608-8011