

HP XC4000 User Guide

Version 1.5

Universität Karlsruhe (TH), Rechenzentrum

November 25, 2009

You are welcome to contact our XC hotline:

`xc-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

<http://www.rz.uni-karlsruhe.de/rz/docs/HP-XC/ug/ug4k.pdf>.

Any comments and hints concerning this introduction are welcome. Please send corresponding e-mails to haefner@rz.uni-karlsruhe.de.

Revision history

Version	Date
Version 0.9 First preliminary version of the User Guide describing HP XC4000 system	January 11, 2007
Version 1.0 First complete version of the User Guide	March 12, 2007
Version 1.1 Changes because of software updates	December 18, 2007
Version 1.2 Minor Changes in chapter File Systems	April 21, 2008
Version 1.3 Minor Changes in chapter CAE Application Codes	September 11, 2008
Version 1.4 Changes due to upgrade to XC-software 4.0	August 4, 2009
Version 1.5 Additional CAE software in chapter CAE Application Codes	November 25, 2009

Contents

1	Introduction	6
2	Configuration of HP XC4000	6
2.1	Architecture of HP XC4000	6
2.2	HP XC4000 at SSC Karlsruhe	7
3	Allocation of Resources	9
3.1	Initial Online Project Proposal	9
3.2	Follow-on Online Proposal	9
4	Interactive Login	9
4.1	X-Windows Applications	10
5	File Systems	10
5.1	\$HOME	10
5.2	\$WORK	11
5.3	Improving Performance on \$HOME and \$WORK	11
5.3.1	Improving Throughput Performance	11
5.3.2	Improving Metadata Performance	12
5.4	\$TMP	13
5.5	Moving Files between Local Workstations and HP XC	13
5.6	Backup and Archiving	13
6	Modules	14
6.1	The most Important of Supplied Modulefiles	14
6.2	Viewing available Modulefiles	15
6.3	Viewing loaded Modulefiles	15
6.4	Loading and Unloading a Modulefile	15
6.5	Creating a Modulefile	16
6.6	Further important Module Commands	16
7	Compilers	17
7.1	Compiler Options	17
7.1.1	General Options	17
7.1.2	Important specific Options of Intel Compilers	17
7.1.3	Some specific Options of GNU Compilers	18
7.1.4	Important specific Options of the PGI Compiler	19
7.2	Fortran Compilers	19
7.3	C and C++ Compilers	20

8	Parallel Programming	21
8.1	Parallelization for Distributed Memory	21
8.1.1	Compiling and Linking MPI Programs	21
8.1.2	Communication Modes	22
8.1.3	Execution of Parallel Programs	22
8.1.4	mpirun Options	23
8.1.5	HP MPI Environment Variables	25
8.1.6	Input/Output Processing	25
8.2	Programming for Shared Memory Systems	25
8.2.1	Shared Memory Programming with Intel Compiler	26
8.2.2	Shared Memory Programming with PGI Compiler	26
8.3	Distributed and Shared Memory Parallelism	26
9	Debuggers	26
9.1	Parallel Debugger ddt	27
10	Performance Analysis Tools	29
10.1	Timing of Programs and Subprograms	29
10.1.1	Timing of Serial or Multithreaded Programs	29
10.1.2	Timing MPI-parallelized Programs	30
10.1.3	Timing of Program Sections	30
10.2	Analysis of Communication Behaviour	32
10.2.1	Counter Instrumentation	32
10.3	Profiling	35
11	Numerical Libraries	36
11.1	AMD Core Math Library (ACML)	36
11.2	Linear Solver Package (LINSOL)	39
12	CAE Application Codes	39
12.1	ABAQUS	40
12.2	LS-DYNA	41
12.3	MSC.Marc/Mentat	42
12.4	MD Nastran	43
12.5	PERMAS	44
12.6	Fluent	44
12.7	ANSYS CFX	45
12.8	Star-CD	45
12.9	STAR-CCM+	46
12.10	COMSOL Multiphysics	46
12.11	Matlab	47
12.12	PowerFlow	47

13 Batchjobs	48
13.1 The <code>job_submit</code> Command	49
13.2 Environment Variables for Batch Jobs	51
13.3 <code>job_submit</code> Examples	51
13.3.1 Serial Programs	51
13.3.2 Parallel MPI Programs	52
13.3.3 Multithreaded Programs	53
13.3.4 Programs using MPI and OpenMP	54
13.4 Commands for Job Management	54
13.5 Command for Showing System Data	55
13.6 Job Chains	56
13.6.1 A Job Chain Example	57
13.6.2 Get remaining CPU Time	59
14 Technical Contacts at University of Karlsruhe Computing Center	60

1 Introduction

The Computing Center of the University of Karlsruhe operates a parallel computer HP XC4000 (abbreviated: HP XC) as high performance computer of the federal state Baden-Württemberg; this super-computer will be provided for projects from the universities of our federal state requiring computing power which can not be satisfied by local resources. The details how to get an account are available in section 3 or on the website of SSC Karlsruhe (<http://www.rz.uni-karlsruhe.de/ssck/access>).

In order to limit the size of this guide, only the most important information about the use of the HP XC system has been collected here. This document is accompanied by many links to resources on the web, especially on the web server of the Computing Center of the University of Karlsruhe where more detailed information is available: <http://www.rz.uni-karlsruhe.de/ssck/hpxc4000>

2 Configuration of HP XC4000

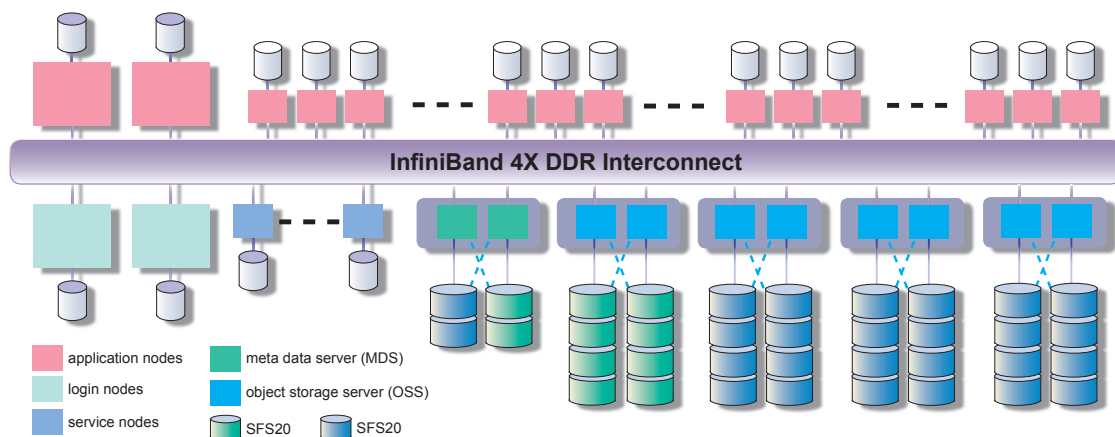


Figure 1: HP XC4000 at Computing Center of University of Karlsruhe

2.1 Architecture of HP XC4000

The HP XC4000 system is a distributed memory parallel computer where each node has two or four AMD Opteron sockets, local memory, disks and network adapters. Each socket comprises 2 cores. In addition special file server nodes are added to the XC cluster to provide a fast and scalable parallel file system. All nodes are connected by a fast InfiniBand 4X interconnect.

The basic operating system on each node is HP XC Linux for HPC, a Linux implementation which is compatible to Redhat AS 5.1. On top of this operating system a set of open source as well as proprietary software components constitute the XC software environment. 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 an XC cluster 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 SLURM and a job is executed depending on its priority, when the required resources become available.
- **File Server Nodes**
Special dedicated nodes are used as servers for the HP StorageWorks Scalable File Share (HP SFS). HP SFS is a parallel and scalable file system product from HP based on the Lustre file system. In addition to shared file space there is also local storage on the disks of each node (for details see chapter 5).
- **Administrative Server Nodes**
Some other nodes are delivering additional services within the XC cluster like resource management, external network connection, administration etc. These nodes can only be accessed directly by system administrators.

2.2 HP XC4000 at SSC Karlsruhe

The HP XC at SSC Karlsruhe consists of

- 2 four-way HP ProLiant DL585 login nodes. Each of these nodes contains four AMD Opteron DC sockets with 2 cores each running at a clock speed of 2.6 GHz and is equipped with 32 GB of main memory.
- 750 two-way ProLiant DL145 G2 nodes. Each of these nodes contains two AMD Opteron sockets running at a clock speed of 2.6 GHz and is equipped with 16 GB of main memory. Each socket has 2 cores with 1 MB of level 2 cache each. Thus each node provides a peak performance of 20.8 GFLOP/s. Two 72 GB local disks and an adapter to the InfiniBand interconnect are additional features of each node.
- 2 four-way HP ProLiant DL585 nodes. Each of the two nodes contains four AMD Opteron DC sockets with 2 cores each running at a clock speed of 2.6 GHz and is equipped with 128 GB of main memory. Thus each node provides a peak performance of 41.6 GFLOP/s. Eight 146 GB local disks and an adapter to the InfiniBand interconnect are additional features of each node.
- 10 two-way HP ProLiant DL380 G4 file server nodes which run the HP Scalable File System (SFS). 8 nodes have the Object Storage Server (OSS) role and the 2 remaining nodes the Meta Data Server (MDS) role. Each of the file server nodes is dual connected to multiple storage systems allowing automatic failover in case of a server failure. The storage systems comprise 36 HP SFS20 enclosures which provide the usable capacity of 56 TB global shared storage within HP XC4000 Cluster. This storage is subdivided into a part used for home directories and a larger part for non permanent or scratch files. The details are described in chapter 5.
- 8 two-way HP ProLiant DL385 and alternatively DL585 service nodes. Each of these nodes contains two AMD Opteron DC sockets with 2 cores each running at a clock speed of 2.6 GHz and is equipped with 8 GB of main memory.

A very important component of HP XC4000 is the InfiniBand 4X interconnect. All nodes are attached to this interconnect which is characterized by its very low latency of just 3 microseconds and a point to point bandwidth between two nodes of about 1500 MB/s for two single tasks. Especially the very

short latency makes the XC system ideal for communication intensive applications and applications doing a lot of collective MPI communications.

With this type of nodes HP XC4000 meets the requirements of applications that are parallelized by the message passing paradigm and use high numbers of processors.

With these different types of nodes HP XC4000 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 four-way DL145 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 on the 8-way nodes. Message passing on more than one 8-way node is not possible.

3 Allocation of Resources

3.1 Initial Online Project Proposal

You don't have an account yet! Please read the following section.

Staff from science or research get an account by writing an online proposal via the website http://www.rz.uni-karlsruhe.de/rd/hpxc_proposal.

Together with the incoming of the online proposal at SCC the members of the project get accounts on the HP XC4000 system. During the review procedure the (unconfirmed) accounts stay valid. At a positive decision of the steering committee the accounts are confirmed and all project members have access to HP XC4000 within the limits of the granted resources of the project.

To evaluate the needed resources within a project it is possible to get pre-accounts for 4 weeks within a planned project. If an online proposal arrives within these 4 weeks the pre-accounts get (unconfirmed) accounts otherwise the pre-accounts will be closed. To get pre-accounts an email containing a brief information about the planned project and its project manager must be sent to ssck-projects@lists.uni-karlsruhe.de.

3.2 Follow-on Online Proposal

Please check the resources of your project by the command `kontingent_get` if your project state is "yellow".

Attention: If the state of your project is "red" your batch jobs will be rejected!

Is your granted CPU-time exhausted you have to write an follow-on online proposal via the website http://www.rz.uni-karlsruhe.de/rd/follow-on_proposal. In the follow-on proposal a detailed project description is not required; therefore you can only use this kind of form if your reviewed project does not change substantially. If your permanent disk space is exhausted, please contact one staff member of the list on the website <http://www.rz.uni-karlsruhe.de/ssck/contact>. If solely your granted timeframe is exhausted you can extend your timeframe by writing an email to ssck-projects@lists.uni-karlsruhe.de.

4 Interactive Login

Instead of `telnet` or `rsh` the secure shell `ssh` is used to login to the HP XC4000 system (abbreviated: HP XC). The secure shell provides higher security when accessing remote computers via the public internet. All data (including passwords) that are transferred between your workstation and the HP XC are encrypted by the secure shell.

To log in to HP XC the following command should be used:

```
ssh user-id@hwwxc2.hww.de oder ssh user-id@xc2.rz.uni-karlsruhe.de
```

where *user-id* is your user identification on HP XC.

On machines administered by the Computing Center of University of Karlsruhe you may run into problems with the previous command. The reason is that on these machines port 24 is used by the secure shell version 2. Your login command then is

```
ssh2 -p 22 user-id@hwwxc2.hww.de oder ssh2 -p 22 user-id@xc2.rz.uni-karlsruhe.de
```

For details on `ssh` read the corresponding man page or consult the appropriate web sites at University of Karlsruhe: <http://www.rz.uni-karlsruhe.de/dienste/ssh>.

4.1 X-Windows Applications

Because the connection between your local workstation and the HP XC is established via `ssh`, usually no further actions are needed to enable X-Windows applications like GUI-based debuggers, graphical pre- and postprocessors etc. If you are using OpenSSH (Linux uses OpenSSH!) you have to switch on the option `-X`. The command to log in to HP XC would be e.g.:

```
ssh -X user-id@xc2.rz.uni-karlsruhe.de
```

In order not to break the security rules of `ssh`:

- the environment variable `$DISPLAY` should never be changed. It is transparently handled by `ssh`.
- The command `xhost +` should never be used on your local workstation. This may give all other users of HP XC access to your local workstation.

5 File Systems

On HP XC the parallel file system HP SFS is supported. It is based on the Lustre file system. Lustre has been mainly developed by Cluster File Systems Inc. This company has 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 3.

Property	<code>\$TMP</code>	<code>\$HOME</code>	<code>\$WORK</code>
Visibility	local	global	global
Lifetime	batch job	project	> 7 days
Disk space	72 GB	16 TB	48 TB
Quotas	no	if required	if required
Backup	no	yes (default)	no
Read perf./node	60 MB/s	320 MB/s	320 MB/s
Write perf./node	60 MB/s	400 MB/s	400 MB/s
Total read perf.	n*60 MB/s	1100 MB/s	3300 MB/s
Total write perf.	n*60 MB/s	800 MB/s	2300 MB/s
global : all nodes of HP XC access the same file system; local : each HP XC node has its own file system; project : files are stored permanently till the end of the project; batch job: files are removed at end of the batch job.			

Table 1: File Systems and Environment Variables

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

5.1 `$HOME`

The home directories of HP XC users are located in the parallel file system Lustre. You have access to your home directory from all nodes of HP XC. A regular backup of these directories to tape archive is automatically done. The `$HOME` directories are used to hold those files that are permanently used

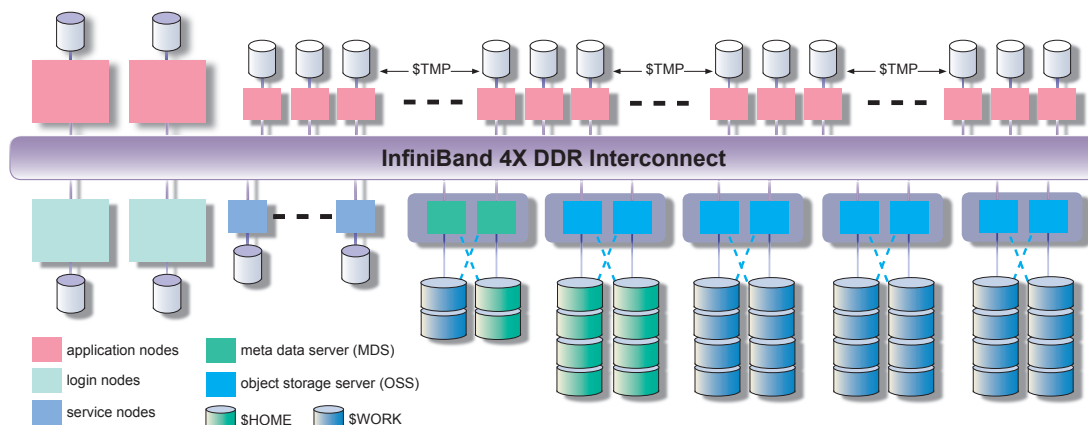


Figure 2: File systems on HP XC

like source codes, configuration files, executable programs etc. But the performance of the `$WORK` file system is usually two times better than the performance of the `$HOME` file system.

For each user group (i.e. one project or one institute) a fixed amount of disk space for home directories is reserved. If required, the disk space will be enforced by so-called quotas. You can check the currently used disk space and quota limits of your user group by the command `show_quota`. This command also displays what you should do in case quotas are exceeded.

5.2 `$WORK`

On HP XC 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.

It is planned to activate quotas on a per group basis with the next SFS software version. However, they will be set to a very high value so that they will only apply if a user group consumes more than 30 percent of the file system. 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 file 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 2 GB/s write performance and more than 3 GB/s 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.

5.3 Improving Performance on `$HOME` and `$WORK`

There are special commands which might help to improve throughput and metadata performance on `$HOME` and `$WORK`.

5.3.1 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 HP XC4000 are called SFS20 and they can reach about 100 MB/s for writes and about 150 MB/s for reads. The file system `$WORK` uses 24, the file system `$HOME` uses 8 SFS20. By default, files of the file system `$WORK` are striped across 4 storage subsystems and files of `$HOME` are striped across 1 storage subsystem¹.

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 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. Also it is only possible to 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.

5.3.2 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 are some possible optimizations which improve the performance of the `ls` and of the `rm` commands.

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=tty'`

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.). If you want that your `ls` commands are incurring this performance overhead, add an alias directive similar to the following to your shell startup script: `alias ls='ls -color=none'`

If the `rm` command is issued to delete lots of files, e.g. `rm -rf`, the command can sometimes take a long time to complete the operation. The primary reason for this is that each file is unlinked in

¹The default stripe count of `$HOME` has been changed from 4 to 1 during the maintenance in November 2007.

a random order and this causes excessive disk-seeking at the server. Since files with adjacent inode numbers are typically adjacent on disk the speed of unlink operations can be increased by pre-sorting the directory entries by inode number. The HP SFS software includes a library and a script that you can use to pre-sort the directory entries. You can do this in either of two ways:

- Edit your script to prefix existing `rm` commands with an LD preload library, as in the following example: `LD_PRELOAD=/usr/opt/hpls/lib/fast_readdir.so /bin/rm myfiles`
- Change your script to replace invocations of the `rm` command with the wrapper script supplied with the HP SFS software, as shown in the following example: `/bin/sfs_rm myfiles`

5.4 \$TMP

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

This directory should be used for temporary files being accessed by single tasks. On 2-way nodes the underlying hardware is a single 72 GB disk per node. On the 8-way nodes the underlying hardware is a fast disk array with about 730 GB capacity.

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.

5.5 Moving Files between Local Workstations and HP XC

You should transfer files between HP XC and your workstation by using the command `scp`. You can transfer files in both directions. In special cases the passive `ftp` command (only from HP XC 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 HP XC. To accomplish this you may enter the following command on HP XC:

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

You will find further information on the `scp` command in the corresponding man page.

5.6 Backup and Archiving

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

Command	Description
<code>tsm_q_backup</code>	shows one, multiple or all files stored in the backup device
<code>tsm_restore</code>	restores saved files

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:

Command	Description
<code>tsm_archiv</code>	archives files
<code>tsm_d_archiv</code>	deletes files from the archive
<code>tsm_q_archive</code>	shows files in the archive
<code>tsm_retrieve</code>	retrieves archived files

Table 3: Commands for Archiving

The option `-h` shows how to use the commands.

More detailed informations you can find on the following website:

<http://www.rz.uni-karlsruhe.de/rz/sw/tsm/xc>

6 Modules

The HP XC4000 Cluster 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 XC software 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`,

`intel` loads Intel Fortran90/95 and C/C++ compiler in the latest stable version,

`hp-mpi` loads HP MPI in a stable version,

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

6.1 The most Important of Supplied Modulefiles

All the above mentioned software packages can be used by all users - both from academic and from industrial customers.

Modulefile	Description
dot	adds the current directory to your environment variable PATH
intel	loads Intel Fortran90/95 and C/C++ compiler in latest version
pgi	loads PGI Fortran90/95 and C/C++ compiler in latest version
gcc	loads GNU compiler suite containing C/C++ compiler in version 4.1.x and Fortran95 compiler gfortran
hp-mpi	loads HP MPI in (nearly) latest version
fftw	loads Fast(est) Fourier Transform library in the West
ddt	loads graphical debugger in latest version

Table 4: Supplied Modulefiles

6.2 Viewing available Modulefiles

Available modulefiles are modulefiles that have been provided with HP XC system software and are 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 avail[able]` command:

```
module avail[able]
```

6.3 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
```

6.4 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 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:

```
# 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:

```
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.

6.5 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:

```
man module or man modulefile
```

6.6 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-whatism` 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`.

7 Compilers

On HP XC exist different compilers for Fortran (supporting the language standards of Fortran 77, Fortran 90 and Fortran 95), C and C++. There are three compilers supporting both Fortran and C/C++. The Fortran compilers consist of the latest Intel compiler, the GNU Fortran77 compiler `g77`, the GNU Fortran90 compiler `gfortran` and the PGI Fortran95 compiler. The C/C++ compilers consist of the latest Intel compiler, the GNU C/C++ compilers in 2 versions and the PGI C/C++ compiler. We recommend the latest Intel C/C++ and Fortran compilers.

7.1 Compiler Options

Subsequently the most important compiler options are explained. If your application code is tested and production jobs shall be performed with your code, you should use the (in the different compiler modules) predefined compiler options for the compilation of your code. They are made available via the environment variables `$CFLAGS` for C code and `$FFLAGS` for Fortran code.

7.1.1 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.
- Ipath* 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 -O is identical to -O2. 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 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 PGI compiler, the PathScale compiler and by the GNU compilers.
- L*path* tell the linker to search for libraries in *path* before searching the standard directories
- l*library* use the specified library to satisfy unresolved external references
- o *name* specify the name of the resulting executable program.

7.1.2 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[0|1|2] 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).

- ip enables additional interprocedural optimizations for single file compilation.
- ipo enables interprocedural optimizations between files.
- vec_report[0|1|2|3|4|5] 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[0|1|2|3] 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[0|1|2] 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.
- xW detects non-compatible processors.
- axP optimizes code for processors with streaming SIMD extensions (here: Opteron).

7.1.3 Some specific Options of GNU Compilers

The GNU C/C++ compiler in version 3 can be called by the command `gcc`. The GNU C/C++ compiler in version 4 can be called by the command `gcc4`. The GNU Fortran77 compiler can be called by the command `g77`; the GNU Fortran95/2003 compiler can be called by the command `gfortran`; the Fortran compilers support all options supported by the C/C++ compiler. GNU specific compiler options which are often needed are:

- funroll-loops[0|1|2] 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.

`-ffast-math` sets `-fno-math-errno`, `-funsafe-math-optimizations`, `-fno-trapping-math`, `-ffinite-math-only`, `-fno-rounding-math`, `-fno-signaling-nans` and `-fcx-limited-range`. This option should never be turned on by any `-O` option since it can result in incorrect output for programs which depend on an exact implementation of IEEE or ISO rules and specifications respectively for math functions.

`-static` prevents linking with shared libraries.

7.1.4 Important specific Options of the PGI Compiler

PGI specific compiler options which are often needed are:

`-fast` generally optimal set of flags for the target.

`-fastsse` generally optimal set of flags for targets that include SSE/SSE2 capability.

`-mp[=align, [no]numa]` interpret and process user-inserted shared-memory parallel programming directives.

`-Mcache_align` where possible, align data objects of size greater than or equal to 16 bytes on cache-line boundaries.

`-Mconcur` enable auto-concurrentization of loops. Multiple processors or cores will be used to execute parallelizable loops.

`-M[no]flushz` do/dont set SSE flush-to-zero mode.

`-M[no]ipa` invokes inter-procedural analysis and optimization.

7.2 Fortran Compilers

The standard Fortran compiler on HP XC 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 upcoming 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/xc4k-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:

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

Table 5: Fortran suffix names for Intel Fortran compiler

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`.

The following tables shows the allowed suffixes and the compiler actions for the PGI Fortran compiler:

To compile FORTRAN90/95 source code stored in file `my_prog.f90` the appropriate command e.g. for the Intel and the PGI Fortran compiler respectively is

command	file name suffix	explanation
pgf95	.f	Fortran source file,
pgf95	.F, .FOR	Fortran source file that can contain macros and preprocessor directives
pgf95	.f90, .f95	Fortran 90/95 source file that is in free format,
pgf95	.F95	Fortran 90/95 source file that can contain macros and preprocessor directives,
pgf95	.hpf	High Performance Fortran source file.

Table 6: Fortran suffix names for PGI Fortran compiler

```
ifort -c -O3 my_prog.f90
or
pgf95 -c -O3 my_prog.f90
```

To compile an MPI program the basic compiler name must be substituted by the string `mpif90` for each chosen compiler (except the GNU compiler family). The parallel program `my_MPI_program.f90` therefore should be compiled with the command

```
mpif90 -c -O3 my_MPI_prog.f90 -openmp
```

To compile multithreaded applications (i.e. OpenMP programs) the compiler option `-openmp` for the Intel compiler and the option `-mp[...]` for the PGI compiler 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_prog.f90
or
pgf95 -c -O3 -mp[=align] my_OpenMP_prog.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` for Intel compiler or `-mp` for PGI compiler must be used.

The above mentioned commands can also be used with the GNU Fortran compiler, if e.g. the name of the Intel Fortran compiler `ifort` is substituted by the name of the GNU Fortran compiler `gfortran`. The command `mpif77` and `mpif90` respectively can be used with all Fortran compilers, i.e. with the Intel, the PGI and the GNU compiler for Fortran.

7.3 C and C++ Compilers

The C and C++ compilers on HP XC are:

- Intel C/C++ compiler in different versions, one of these versions is the default C/C++ compiler;
- GNU project C/C++ compiler 4.1 and
- PGI C/C++ compiler.

The C compilers from Intel, GNU and PGI on HP XC are invoked with the commands `icc`, `gcc` and `pgcc`. The C++ compilers from Intel, GNU and PGI on HP XC are invoked with the commands `icpc`, `g++` and `pgCC`. Details may be found in the appropriate man pages or in the compiler manuals (<http://www.rz.uni-karlsruhe.de/ssck/xc4k-manuals>).

To compile MPI programs the compiler scripts `mpicc` and `mpiCC` should be used for C and C++ programs.

To compile a C++ program `my_MPI_program.C` that calls MPI functions the appropriate command is therefore

```
mpiCC -c -O3 my_MPI_program.C
```

The first command compiles an OpenMP program `my_OpenMP_program.C` written in C++ with the Intel compiler and the second command compiles the same program with the PGI compiler:

```
mpiCC -c -openmp -O3 my_OpenMP_program.C
```

or

```
mpiCC -c -mp[=align] -O3 my_OpenMP_program.C
```

8 Parallel Programming

Different programming concepts for writing parallel programs are used in high performance computing and are therefore supported on HP XC4000. 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 HP XC.

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 HP XC4000.

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>.

8.1 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 HP XC MPI is part of the parallel environment.

8.1.1 Compiling and Linking MPI Programs

There are special compiler scripts to compile and link MPI programs. All these scripts start with the prefix `mpi`:

```
mpicc compile and link C programs,
```

```
mpicc.mpich compile and link C programs in MPICH compatibility mode,
```

```
mpiCC compile and link C++ programs,
```

```
mpiCC.mpich compile and link C++ programs in MPICH compatibility mode,
```

```
mpif77 or mpif90 compile and link Fortran77 and Fortran90 programs. Intel, PGI and GNU Fortran compilers work together with both variants. If MPICH compatibility is required, call mpif77.mpich or mpif90.mpich.
```

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/ssck/mpl>

8.1.2 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 HP XC4000 in general more than one task of a parallel application is executed on one node. The 2x2-way Opteron nodes are allocated exclusively to one batch job with four tasks of the batch job running on one node, if you don't request more than 4 GB main memory per processor. Tasks 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 2000 MB/s for simple send/receive operations.

Tasks on different nodes communicate over the InfiniBand Switch. Data communication speed can reach about 1500 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.

8.1.3 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 4 tasks,
- maximum 2 GB virtual memory per task 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 shellscrip 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:

```
# This is an example for interactive parallel program execution.
# The program my_mpi_program will be run with 4 tasks.
#
# Enter the following command on the command line
mpirun -T -n 4 my_mpi_program
#
# A second version launching the same executable on the same number
# of processors is:
export MPIRUN_OPTIONS="-T -n 4"
mpirun my_mpi_program
```

8.1.4 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.

mpirun Option	Brief Explanation
<code>-n #</code> or <code>-np #</code> <code>-m block cyclic</code>	MPI job is run on # processors (option is ignored in batch mode) distributes tasks among nodes blockwise or cyclically to the processors; block: assigns tasks in order to each CPU on one node before assigning any to the next node (default); cyclic: assigns tasks "round robin" across all allocated nodes
<code>-cpu_bind</code> [=block cyclic ...]	binds a rank to an ldom (locality domain is defined here as one node) to prevent a process from moving to a different ldom after startup; options: block: blockwise dist on each ldom according to (packed) rank id cyclic: cyclic dist on each ldom according to (packed) rank id (default) ...: many other options are allowed; description by <code>man mpirun</code> <code>-cpu_bind</code> means "no cpu binding"
<code>-mpich</code>	runs the application in MPICH compatibility mode
<code>-version</code>	prints the major and the minor version numbers
<code>-v</code>	turns on verbose mode
<code>-d</code>	turns on debug mode
<code>-p</code>	turns on pretend mode
<code>-ck</code>	turns on extended pretend mode
<code>-j</code>	prints the HP MPI job ID
<code>-T</code>	prints user and system time for each MPI rank
<code>-1sided</code>	enables one-sided communication
<code>-i options</code>	enables runtime instrumentation profiling for all processes (refer to HP MPI User Guide for an explanation of <code>-i options</code>)
<code>-stdio=options</code>	specifies standard IO options (refer to HP MPI User Guide for an explanation of <code>-stdio=options</code>); in batch mode all options except for <code>-stdio=p</code> are ignored
<code>-nopreload</code>	omits preloading of the standard MPI library; has to be used if the application is linked with a different MPI library like mtmpi (multi-threaded) or pmpi (profiling)
<code>-f appfile</code>	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!

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 ($i = 1, number_of_processors$) 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.

8.1.5 HP MPI Environment Variables

The following environment variables can be used to control HP MPI.

HP MPI Environment Variable	Brief Explanation
MPLDLIB_FLAGS	controls runtime options when using the diagnostics library
MPLFLAGS	modifies the general behaviour of HP MPI
MPLGLOBMEMSIZE	specifies the amount of shared memory allocated for all processes in MPI applications
MPLINSTR	enables counter instrumentation for profiling HP MPI applications
MPLMT_FLAGS	controls runtime options when using the thread-compliant version of HP MPI
MPLSHMEMCNTL	controls the subdivision of each process's shared memory
MPLWORKDIR	sets a new working directory
MPIRUN_OPTIONS	sets <code>mpirun</code> options
MPLPAGE_ALIGN_MEM	causes the HP MPI library to page align and page pad memory
MPLMAX_WINDOW	specifies the maximum number of windows a rank can have at the same time

Please refer to HP MPI User Guide for a more detailed explanation of the environment variables (<http://www.rz.uni-karlsruhe.de/ssck/xc4k-manuals>).

8.1.6 Input/Output Processing

By default standard input is read by all tasks, i.e. the option `stdio=i` is set per default.

By default `mpirun` collects the output written by all tasks and merges this to one stdout file. In order to distinguish the output of individual tasks the option `-stdio=p` may be selected:

Example:

```
mpirun -stdio=p -n 4 mpi_hello

2: hello world
3: hello world
0: hello world
1: hello world
```

In the above example the output is unordered, but the output of the processors can be identified by its task id.

8.2 Programming for Shared Memory Systems

While MPI is a tool for distributed memory systems, OpenMP is targeted to shared memory systems, i.e. one HP XC 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 HP XC4000 the OpenMP specification is supported by Intel and PGI Fortran and C compilers.

8.2.1 Shared Memory Programming with Intel Compiler

To compile programs for shared memory parallelism 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`. This option is only available in Fortran.

8.2.2 Shared Memory Programming with PGI Compiler

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

- `-Mconcur[=option [,option,...]]` – Instructs the compiler to enable auto-concurrentization of loops. If `-Mconcur` is specified, multiple processors will be used to execute loops that the compiler determines to be parallelizable. There are many suboptions (see the PGI User Guide under (<http://www.rz.uni-karlsruhe.de/ssck/xc4k-manuals>)).
- `-Mnoopenmp` – when used in combination with the `-mp` option, causes the compiler to ignore OpenMP parallelization directives or pragmas.

8.3 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 an HP XC 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`.

9 Debuggers

On HP XC4000 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 `-g` compiler option. This will force the compiler to add additional information to the object code which is used by the debugger at runtime.

9.1 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 *program* is the name of your program that you want to debug.

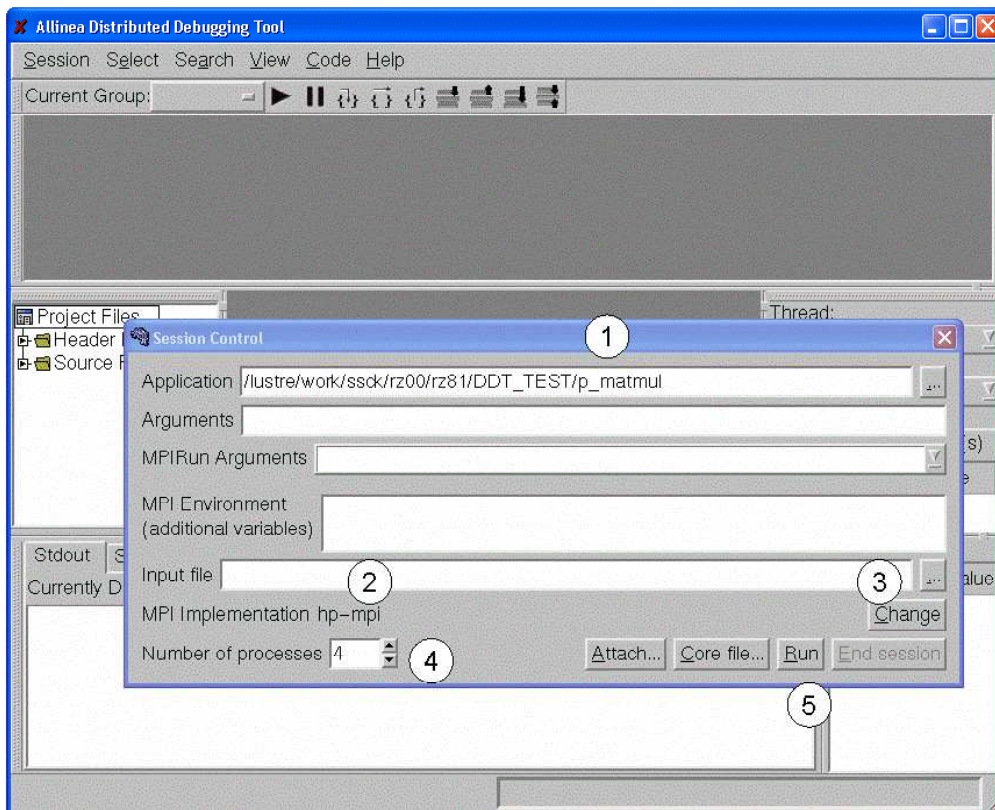


Figure 3: DDT startup window

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.

- Before starting an MPI program you should check that 'hp-mpi' 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.

- 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.
- 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.

More details on ddt including the Users' Guide are available on the web:
<http://www.rz.uni-karlsruhe.de/ssck/xc4k-manuals>.

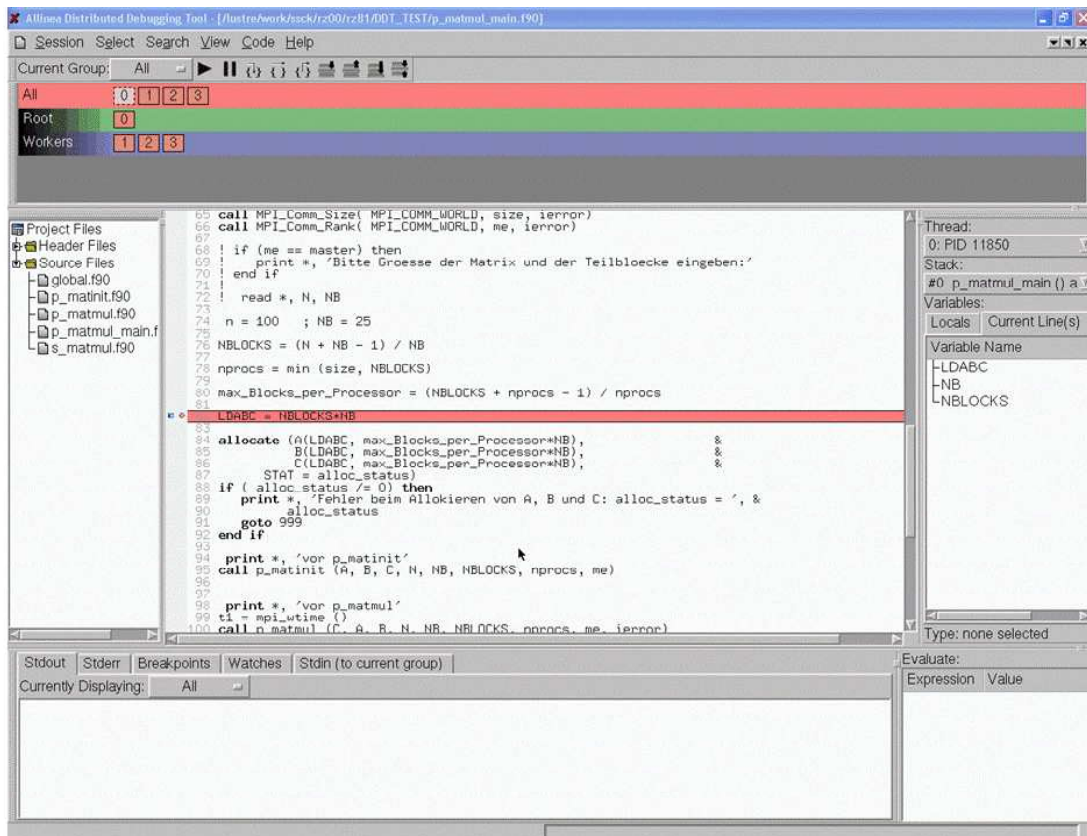


Figure 4: DDT window

10 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 11). To get support and help to optimize your program you should contact the technical support staff at University of Karlsruhe Computing Center (cf. section 14).

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 HP XC4000 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 MFlops 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.

10.1 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 or to use the `-T` option of the `mpirun` command in case of an MPI parallelized program.

10.1.1 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.

10.1.2 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.

This information can easily be made available by selecting the `mpirun` option `-T`. In this case one additional line of output will be generated for each MPI process including the user and system CPU time consumed by this process. The output for an application with 12 MPI processes may look like:

MPI Rank	User (seconds)	System (seconds)
0	1478.10	2.80
1	1477.38	3.56
2	1477.71	3.46
3	1478.03	2.73
4	1477.56	2.74
5	1477.71	2.64
6	1478.01	2.56
7	1477.51	3.38
8	1477.55	2.62
9	1477.57	2.65
10	1477.73	2.67
11	1477.63	2.67

Total:	17732.49	34.48

In this example the system time is just 0.2% of the user user CPU time. But you should be aware that all the communication is done in user mode, i.e. the communication time is included in the user CPU time. This includes the time where the program is waiting for the arrival of a message.

10.1.3 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:

```
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

!-----

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
```

10.2 Analysis of Communication Behaviour

To reach reasonable performance for parallel applications it is essential to have a good load balancing between all tasks (i.e. all tasks should do nearly the same amount of work). Additionally the communication operations should be organized in such a way that there is no need for any task to wait for a long time in order to communicate with other tasks.

As the communication time is part of the user CPU time, the program must be instrumented in order to see which portion of the CPU time is spent for communication and which portion is spent for computation.

10.2.1 Counter Instrumentation

To get a first impression what amount of time is spent in computing and in communication the counter instrumentation supported by the HP MPI library may be used.

In order to activate this, just select the `mpirun` option `-i name`, or the environment variable `MPI_INSTR`:

```
mpirun -i name ...
```

or

```
export MPI_INSTR=name    resp.    setenv MPI_INSTR name
mpirun ...
```

When your program reaches the `MPI_Finalize` function, a file named `name.instr` will be created. This file includes a lot of statistics on time spent in MPI functions, communication hot spots and message length.

The following example will demonstrate the usage and the results obtained with this `mpirun` option:

```
mpirun -T -i test_1 a.out
```

The program is executed as part of a batch job. The results of counter instrumentation are stored in the file `test_1.instr` and consists of three major sections:

- Application Summary,
- Routine Summary by Rank and
- Message Summary by Rank Pair.

Application Summary The application summary shows the overall distribution of compute time and communication time. In this example (c.f. Fig. 5 more than 20% of the total CPU time is spent in MPI routines. This is much more than you should see usually. The MPI time per MPI process (rank) varies between approx. 5% and more than 33%. This indicates that there seems to be a problem with load balancing. Especially MPI process 0 (about 461 seconds against 312 up to 433 seconds on the other processes) seems to have much more work than the other MPI processes.

Routine Summary by Rank Figure 6 shows a summary of the MPI functions called by one MPI process of the application. Counter instrumentation creates one such table for each MPI process. The MPI time is dominated by calls to `MPI_Allreduce` and `MPI_Comm_free`. Nearly 80 seconds are consumed in `MPI_Allreduce` and one out of the 9206 calls took about 5 seconds and one call to `MPI_Comm_free` took about 8 seconds. Both MPI functions (`MPI_Allreduce` as well as `MPI_Comm_free`) are blocking collective MPI operations requiring all processes of an MPI communicator to participate in the communication. The large time consumed by these processes clearly indicates that there is a problem with load balancing in this application.

Version: HP MPI [not a product] B6060BA - Linux IA64 (gcc)
 Date: Thu Dec 9 17:03:32 2004

Processes: 8

User time: 79.61%
 MPI time : 20.39% [Overhead:20.39% Blocking:0.00%]

 ----- Instrumentation Data -----

Application Summary by Rank (second):

Rank	Proc CPU Time	User Portion	System Portion
0	484.506836	478.850586(98.83%)	5.656250(1.17%)
1	484.655273	484.486328(99.97%)	0.168945(0.03%)
2	484.627930	484.437500(99.96%)	0.190430(0.04%)
3	484.718750	484.550781(99.97%)	0.167969(0.03%)
4	484.715820	484.510742(99.96%)	0.205078(0.04%)
5	484.597656	484.424805(99.96%)	0.172852(0.04%)
6	484.572266	484.407227(99.97%)	0.165039(0.03%)
7	484.871094	484.684570(99.96%)	0.186523(0.04%)

Rank	Proc Wall Time	User	MPI
0	486.247237	461.618766(94.93%)	24.628471(5.07%)
1	486.247233	382.488008(78.66%)	103.759225(21.34%)
2	486.252954	387.285337(79.65%)	98.967617(20.35%)
3	486.252647	356.813045(73.38%)	129.439602(26.62%)
4	486.246027	433.273408(89.11%)	52.972619(10.89%)
5	486.246026	401.069109(82.48%)	85.176917(17.52%)
6	486.246793	352.797809(72.56%)	133.448984(27.44%)
7	486.246330	321.625072(66.14%)	164.621258(33.86%)

Rank	Proc MPI Time	Overhead	Blocking
0	24.628471	24.628471(100.00%)	0.000000(0.00%)
1	103.759225	103.759225(100.00%)	0.000000(0.00%)
2	98.967617	98.967617(100.00%)	0.000000(0.00%)
3	129.439602	129.439602(100.00%)	0.000000(0.00%)
4	52.972619	52.972619(100.00%)	0.000000(0.00%)
5	85.176917	85.176917(100.00%)	0.000000(0.00%)
6	133.448984	133.448984(100.00%)	0.000000(0.00%)
7	164.621258	164.621258(100.00%)	0.000000(0.00%)

Figure 5: Application summary by rank

Rank	Routine	Statistic	Calls	Overhead(ms)	Blocking(ms)
3	MPI_Allgather		206	5.893707	0.000000
		min		0.000000	0.000000
		max		0.264168	0.000000
		avg		0.028610	0.000000
	MPI_Allreduce		9206	79391.850948	0.000000
		min		0.000954	0.000000
		max		4932.632923	0.000000
		avg		8.623925	0.000000
	MPI_Barrier		1	0.190020	0.000000
	MPI_Bcast		377	9.254694	0.000000
		min		0.007153	0.000000
		max		3.824949	0.000000
		avg		0.024548	0.000000
	MPI_Comm_dup		75	3.186941	0.000000
		min		0.016928	0.000000
		max		0.164986	0.000000
		avg		0.042493	0.000000
	MPI_Comm_free		75	47660.134077	0.000000
		min		0.006914	0.000000
		max		8097.234011	0.000000
		avg		635.468454	0.000000
	MPI_Finalize		1	0.049114	0.000000
	MPI_Init		1	1098.448992	0.000000
	MPI_Irecv		14001	407.836676	0.000000
		min		0.010967	0.000000
		max		0.252008	0.000000
		avg		0.029129	0.000000

Figure 6: Routine summary by rank

SRank	DRank	Messages	(minsize,maxsize)/[bin]	Totalbytes
0				
	0	508	(4, 5892)	60912
		498	[0..64]	1992
		10	[4097..16384]	58920
	1	13616	(4, 36760)	16609644
		6061	[0..64]	87996
		3072	[65..256]	477312
		4363	[1025..4096]	13472384
		34	[4097..16384]	261648
		86	[16385..65536]	2310304
	2	13616	(4, 36760)	12043852
		6061	[0..64]	87996
		3072	[65..256]	477312
		4363	[1025..4096]	8895000
		34	[4097..16384]	273240
		86	[16385..65536]	2310304
	. . .			

Figure 7: Message summary by rank pair

Message Summary by Rank Pair: Figure 7 gives an overview of the communication between source rank 0 and destination ranks 0, 1 and 2. The output of counter instrumentation will contain these data for all rank pairs. From rank 0 about 16 GB have been transferred to rank 1 in 13616 messages. There were many short messages (6061) transferring less than 90000 bytes, i.e. less than 15 bytes in average. Combining many of these small messages to a few large messages can sometimes decrease the communication time significantly.

10.3 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 it is possible to replace own code by a call to some optimized functions or subprograms from highly tuned libraries (cf. section 11).

The profiling tool `gprof` is available on many Unix or Linux systems. The information you may get from this tool 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 `gprof` utility the following steps are required:

1. Compile and link the program with `-pg` option.
2. Run the program as usual. When the program terminates a file `gmon.out` will be created. In case of a parallel program several output files `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.

11 Numerical Libraries

Some numerical subprogram libraries have been installed on the HP XC4000 system. Some of these are highly tuned for AMD Opteron processors.

Tuned implementations of well established open source libraries are part of the AMD Core Math Library (ACML).

Basic Linear Algebra Subprograms (BLAS): BLAS is included in AMD ACML.

Linear Algebra Package (LAPACK): the LAPACK library contains many functions for solution of linear systems and eigenvalue problems for dense and banded matrices. It reaches high performance by using the Basic Linear Algebra Subprograms (BLAS) as computational kernels. On HP XC4000 LAPACK is included in AMD ACML.

Scalable LAPACK: on HP XC4000 ScaLAPACK (version 1.8.0) is available for the GNU compiler suite in version 4.1.x.

11.1 AMD Core Math Library (ACML)

The AMD Core Math Library includes functions from following areas:

- Basic Linear Algebra Subprograms (BLAS),
- Sparse BLAS (basic vector operations on sparse vectors),
- a full suite of Linear Algebra (LAPACK) routines. As well as taking advantage of the highly-tuned BLAS kernels, a key set of LAPACK routines has been further optimized to achieve considerably higher performance than standard LAPACK implementations,
- a comprehensive suite of Fast Fourier Transforms (FFTs) in both single-, double-, single-complex and double-complex data types,
- fast scalar, vector, and array math transcendental library routines optimized for high performance on AMD Opteron processors and
- Random Number Generators in both single- and double-precision.

The AMD ACML supports all installed compilers with different linker options for FORTRAN and C/C++ code, but you cannot always use the latest ACML version in cooperation with a certain compiler.

In the following examples we assume that a Fortran program `myprog.f90` will be compiled and linked against the AMD Core Math Library. The appropriate options for the different Fortran compilers at link time are:

- Intel Fortran Compiler (use the second command if you want to use the OpenMP version of ACML)

```
ifort $FFLAGS -o myprog myprog.f90 -L$ACMLPATH -lacml
ifort -openmp $FFLAGS -o myprog myprog.f90 -L$ACMLPATH_MP -lacml_mp
```

- GNU Fortran Compiler

```
gfortran $FFLAGS -m64 -o myprog myprog.f90 -L$ACMLPATH -lacml
(gfortran -fopenmp $FFLAGS -m64 -o myprog myprog.f90 -L$ACMLPATH_MP -lacml_mp
is not possible up to now, because the GNU compiler suite in version 4.1.2 is
installed and OpenMP is initially supported by version 4.2!)
```

- PGI Fortran Compiler

```
pgf90 $FFLAGS -Mcache_align -o myprog myprog.f90 -L$ACMLPATH -lacml
pgf90 $FFLAGS -mp -Mcache_align -o myprog myprog.f90 -L$ACMLPATH_MP -lacml_mp
```

In the following examples we assume that a C program `myprog.c` will be compiled and linked against the AMD Core Math Library. The appropriate options for the different C/C++ compilers at link time are:

- Intel C/C++ Compiler (use the second command if you want to use the OpenMP version of ACML)

```
icc $CFLAGS -c -I$ACMLINCLUDE myprog.c
ifort -nofor-main myprog.o -L$ACMLPATH -lacml
(It is not described if there is an OpenMP version for the Intel C/C++ compiler!)
```

- GNU C/C++ Compiler

```
gcc -m64 $CFLAGS -I$ACMLINCLUDE -o myprog myprog.c -L$ACMLPATH -lacml \
-lgfortran
```

- PGI C/C++ Compiler

```
pgcc -c -Mcache_align $CFLAGS -I$ACMLINCLUDE myprog.c
pgcc -tp=k8-64 -Mcache_align myprog.o -L$ACMLPATH \
-lacml -lpgftnrtl -lm
```

Use the following command if you want to use the OpenMP version of ACML!

```
pgcc -c -mp -Mcache_align $CFLAGS -I$ACMLINCLUDE_MP myprog.c
pgcc -tp=k8-64 -mp -Mcache_align myprog.o \
-L$ACMLPATH_MP -lacml_mp -lpgftnrtl -lm
```

An example follows how to use BLAS routines that are written in Fortran within C/C++ routines. There is an C interface to all ACML routines. The parameter list of the calls can be found in the include file `acml.h` of the compiler you are using. The path to the include file is stored in the environment variable `ACMLINCLUDE`.

```

/* testblas.c - test program for the BLAS dgemv function included in ACML

#include <stdio.h>
#include <acml.h>

double m[] = {
    3, 1, 3,
    1, 5, 9,
    2, 6, 5
};

double x[] = {
    -1, -1, 1
};

double y[] = {
    0, 0, 0
};

int main ()
{
    int i, incx, j, n;
    double alpha, beta;
    char c = 'n';

    n = 3; incx = 1;
    alpha = 1.0;
    beta = 0.0;

    for (i=0; i<3; ++i) {
        for (j=0; j<3; ++j) printf ("%5.1f", m[i*3+j]);
        putchar ('\n');
    }

    dgemv(c, n, n, alpha, m, n, x, incx, beta, y, incx);

    for (i=0; i<3; ++i) printf ("%5.1f\n", y[i]);

    return 0;
}

```

The ScaLAPACK library is only available for the GNU compiler suite in version 4.1.x. If you wish to run a Fortran90/95 program, you have to choose the module `gcc`. When linking your application you have to enter the environment variable `$SCALAFLIBS` and `$SCALACLIBS` respectively.

In the following examples the usage for a simple Fortran program and C program is shown:

```

mpif90 -o myprog myprog.o $SCALAFLIBS
mpicc -o myprog myprog.o $SCALACLIBS

```

Linking Fortran programs the environment variable `$SCALAFLIBS` must be chosen, linking C-programs the environment variable `$SCALACLIBS`.

11.2 Linear Solver Package (LINSOL)

LINSOL is a program package to solve large sparse linear systems. It has been developed at University of Karlsruhe Computing Center. For the simulation of many numerical problems the solution of large and sparse linear systems is required. For these problems the linear solver package LINSOL has been designed. Iterative techniques based on generalized conjugate gradient methods and beyond are implemented. Different polyalgorithms that select appropriate solvers from the whole variety of methods and direct solvers - the (Incomplete) Gauss algorithm for unsymmetric matrices and the (Incomplete) Cholesky algorithm for symmetric matrices - are available. Linsol is fully parallelized using MPI.

Before the usage of LINSOL the following module should be loaded to set some environment variables:

```
module add linsol
```

Then you can either use the LINSOL library or you can use the stand-alone interface that works on matrices in the Harwell-Boeing or LINSOL format.

In the directory `/software/all/linsol/examples` you find a file `exam01.f` that calls the LINSOL library by the Fortran90 interface `LSOLP`. It is compiled and linked here exemplarily for arbitrary Fortran90/95 programs that call the LINSOL library:

- Calling LINSOL serially:

```
ifort -O3 -o exam01 exam01.f -L$LSOL_LIB -llinsol -lnocomm
```

- Calling LINSOL on more than one processor:

```
mpif90 -O3 -o exam01 exam01.f -L$LSOL_LIB -llinsol -lMPI
```

You can see the correct output of the executable `exam01` in the file `exam01.output`.

The stand-alone interface is used by the following command:

```
linnox parameter-file
```

Exemplarily you can solve the matrix `gre512.rua` stored in Harwell-Boeing format and the matrix `oilgen.lsol` stored in LINSOL format. Corresponding to the matrices there are parameter-files `gre512_param` and `oilgen_param`. So the solution of the linear equation can be started by the command:

```
linnox gre512_param or linnox oilgen_param
```

You will find detailed informations on LINSOL on the website

<http://www.rz.uni-karlsruhe.de/produkte/linsol>.

12 CAE Application Codes

The HP XC4000 system is especially suited for solving physical problems. The installed programs may be grouped in application areas

- structural mechanics: ABAQUS, LS-Dyna, MSC.Marc/Mentat, MD Nastran, Permas;
- fluid dynamics: Fluent, ANSYS CFX, Star-CD, Star-CCM+, PowerFlow.

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, for each program a command is provided which handles both tasks.

Other applications are

- COMSOL Multiphysics;
- Matlab.

12.1 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 adaptation,
- design optimization,
- and much more.

A complete overview can be found in http://www.simulia.com/products/product_index.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:

```
abqjob -j ID -t TIME -m MEMORY [-c CLASS] [-T TIME] [-p PROCS] [-i FILE]  
[-o OLD-JOB] [-f FILE] [-u USERSUB] [-D DIRECTSOLVER] [-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)

-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

12.2 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
- 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:

```
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

12.3 MSC.Marc/Mentat

MSC.Marc is a general purpose Finite Element code especially for nonlinear analysis of structural mechanics problems, heat transfer, electrical fields and fluids. Multiphysics problems like heat-structure-coupling and fluid-structure coupling can also be treated.

MSC.Mentat is the the environment for the development of Marc models and the postprocessing of the results.

Some features of MSC.Marc are:

- 1D, 2D, 3D and axisymmetric elements (totally more the 100 elements are available)
- more than 40 material models
- boundary conditions, both transient and temperature dependent, contact
- linear and nonlinear
- fracture mechanics
- dynamics
- heat transfer
- heat-structure coupling
- fluid-structure coupling
- substructure
- Joule heating
- and much more

A complete overview can be found in <http://www.mssoftware.com/products>. The documentation can be found in `/software/ssck/marc/mentat2008r1/doc`. The command for running MSC.Marc in batch mode is:

```
marcjob -j ID -t TIME -m MEMORY [-c CLASS] [-p PROCS] [-i INPUT] [-e SCRATCHDIR]  
[-T TIME] [-u USRSUB] [-r RESTART] [-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)
- i *structure of the input files* (only if p>1): s single input file, m multiple input file
- e *directory to store scratch files* (`$WORK` or `$TMP`; default is `$TMP`)
- T *real time in minutes*
- u *user subroutine*
- r *name of a preceding job*, if this is a restart job
- s *string* with further options

12.4 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>. As pre- and postprocessors for Nastran models MSC.Patran and HyperMesh licenses are available.

The documentation is in PDF and can be found in the directory `/software/ssck/msc/nastran/md20071/doc/pdf_nastran` on HP XC4000.

The current license allows running a Nastran job in batch mode on up to 8 processors. The command is

```
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

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.

12.5 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>. 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 processes.

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

The PDF version is available in `/software/ssck/intes/documentation/onldoc_v12.100`.

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

```
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* ($p \leq 8$, default is 1)
- e *specify the environment variable for scratch files* (\$WORK or \$TMP; default is \$TMP)
- s *string* with further options (optional)

12.6 Fluent

At the moment the solver of Fluent V12.0 ist 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>.

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

12.7 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,
- CFD-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 HP XC4000. 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. More information can be found under <http://www.ansys.com/products/cfx.asp>.

12.8 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>

12.9 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)
-T *real time in minutes* (optional)
-p *number of parallel tasks* (default is 1)

The complete documentation is online and available as PDF.

12.10 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 alternatives 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/>.

12.11 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 features 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/>.

12.12 PowerFlow

PowerFlow is a CFD code based on the Lattice-Boltzmann method. It is suited for exterior and interior flows of Newtonian, compressible fluids. More species or chemical reactions are not covered.

The system is composed of three modules:

- PowerCASE for the creation of the model; several CAD formats and Ansys, Nastran and Patran formats can be imported. Simple geometries can be created directly;
- PowerFLOW is the solver, which uses the three steps "Discretize", "Decompose", "Simulate" to solve the CFD problem;
- PowerVIZ is the postprocessor for further evaluation and visualization.

With our current license, PowerFLOW can be parallelized on 16 processors.

The documentation is PDF and is installed under /software/ssck/powerflow/4.1c/doc. Further information can be found on Exa's homepage <http://www.exa.com>.

PowerCASE and PowerVIZ are invoked by the commands "powercase" and "powerviz". The PowerFLOW command for starting the solver under the batch system is:

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

Parameters are:

- j *projektname*
- 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)
- s *string* with further options (optional)
- T *real time in minutes* (optional)

13 Batchjobs

As described in section 2 the majority of HP XC nodes 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.

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

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.

13.1 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] [-J "jobname"]  
[-l af|aF|Af|AF] [-A account] [-N[s][b][c|C|e|E]] [-i file] [-o file]  
[-e file|+] [-d t|f] job
```

-t 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 ($T = 1.01 * t + 1$; t is time from `-t`).

-m mem: maximum memory requirement per task in Mega Bytes. The 4-way nodes of the HP XC are equipped with 16 GB of main memory and the 8-way nodes with 128 GB per node, i.e. 16 GB per CPU.

-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: Erraneous 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: Erraneous 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 HP XC:
- `d`: jobs in this class will start immediately, but do not have exclusive access to any resources of HP XC. 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 to the requested resources. Always whole nodes are accessed. This can lead to unused processors (in particular on 8-way nodes).
- `-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.
- `-d t|f`: this option should be used carefully. If `-d t` is chosen, thin nodes (i.e. nodes with 4 processors) will be used. If `-d f` is chosen, fat nodes (i.e. nodes with 8 processors) will be used. If this option is omitted the batch system decides if thin nodes or if a singular fat node will be allocated to run the job. The batch system can also automatically migrate the job from thin nodes to a singular fat node and vice versa, if this is possible under the given conditions in terms of required processors and main memory.
- `job`: this is a parallel program call or a shell script to be executed on HP XC. 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 thin nodes with 4 processors and fat nodes with 8 processors. 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 processors the batch system will always allocate thin nodes (you mustn't use the option `-d f`!). If you are asking for more than 16 GB of main memory the batch system will always allocate fat nodes (you mustn't use the system `-d t`!). If you are asking for 8 or less processors and for 16 GB or less of main memory, then the batch system decides if your job will run on thin or fat nodes (in case of omitting the option `-d`). **It is assumed that the throughput time of your job will be lower if you don't use the option `-d`. If you are using the option `-d f` (i.e. you are asking explicitly for a singular fat node) and $p < 8$ processors, then be aware that $8 - p$ processors are idling and completely accounted!**

13.2 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.

Environment Variable	Brief Explanation
JMS_t	contains the value of the CPU time limit which has been defined with option <code>-t</code> . This value can be used to compute the amount of CPU time within a program that is still available for the computation.
JMS_T	contains the maximum elapsed time as specified with the option <code>-T</code> .
JMS_m	contains the memory requirement as specified with the <code>-m</code> option.
JMS_Nnodes	contains the allocated number of nodes.
JMS_p	contains the requested number of processors.
JMS_tasks	contains the number of MPI tasks specified with the first value <i>i</i> in the <code>-p</code> option.
JMS_threads	contains the number of threads per task (process) as specified with the second value <i>j</i> in the <code>-p</code> option. After setting <code>JMS_tasks</code> the following assignment is done: <code>OMP_NUM_THREADS=\$JMS_tasks</code> defines the job class as specified with option <code>-c</code> .
JMS_c	
JMS_start_time	gives the starting time of the job if it is in state running.
JMS_submit_time	gives the time at which the job has been submitted.
JMS_submit_node	contains the name of the node the job has been started on.
JMS_node0	contains the name of the first node.
JMS_nodes	lists all used node names. If e.g. 2 processors per node are used the used nodes are listed two times.
JMS_stdin	contains the name of the input file of the job.
JMS_stdout	contains the name of the output file of the job.
JMS_stderr	contains the name of the standard error file of the job.
JMS_pwd	contains the name of the output directory of the job.
JMS_user	contains the userid of the user who has submitted the job.
JMS_group	contains the groupid of the user who has submitted the job.
TMP and TEMP	contain the working directory for temporary files of the job.

13.3 job_submit Examples

13.3.1 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).

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:

```
job_submit -t 180 -T 240 -m 5000 -p 1 -c p "job.ksh -n 100"
or
job_submit -t 180 -T 240 -m 5000 -p 1 -c p job.ksh -n 100
```

13.3.2 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 32 processors. The maximum CPU time is 4 hours and the memory requirement is 6000 MB.

```
job_submit -t 240 -m 6000 -p 32 -c p mpirun my_par_program
```

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: 4000 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:

```
#!/bin/bash
#
cd $WORK
[[ ! -d Job_Output ]] && mkdir Job_Output
cd Job_Output

mpirun $HOME/bin/my_par_program
```

To submit this job, use the commands

```
chmod u+rx job.ksh
job_submit -c p -p 64 -t 180 -m 4000 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`.

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:

```
#!/bin/bash

cd $WORK

mpirun $HOME/project/bin/my_first_parallel_program

if [[ $? -eq 0 ]]; then
#   program terminated successfully, copy data and start next program

    cp results_1 ${HOME}/results_1

    if [[ $? -eq 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 nodes (CPU time limit: 4 hours, memory limit: 5000 MB) use the `job_submit` command:

```
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. 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.

13.3.3 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/xc4k-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 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.

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/bash
#
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
```

13.3.4 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 4-way (thin) nodes with four threads per node and 10 GB of main memory per MPI task, the `job_submit` command may look like:

```
job_submit -c p -t 240 -T 300 -m 10000 -p 32/4 mpirun my_parallel_prog
```

2. To run a program with 2 MPI tasks and 4 threads on a single 8-way node (you cannot run parallel jobs on multiple 8-way (fat) nodes!), the `job_submit` command will be:

```
job_submit -c p -t 240 -T 300 -m 10000 -p 2/4 mpirun my_parallel_prog
```

13.4 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:

job-id	c	P	n/i/j	t	T	m	queued	s	start	end(t)
32955	p	f	1/8/1	10	11	1000	25/11:00	r	25/11:00	25/11:10

- 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 third column displays the partition (column `P`) the job runs in. The job in this example runs in the partition "fat nodes" (`f`). Further partitions are "thin nodes" (`t`) and "development" (`d`).
- 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.

13.5 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 HP XC4000 system 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 cpus:

in most cases the displayed number of cpus means the number of accounted cpus, i.e. on thin nodes the next higher even number of cpus that is divisible by 4 is accounted if the MPI program runs on an number of cpus, that is not divisible by 4, and on fat nodes 8 processors are accounted both for serial programs and for all kinds of parallel programs from 2 up to 8 processors! Only if you didn't request a fat node, but your application runs on a fat node, the number of displayed cpus differs from the number of accounted cpus, because the number of displayed cpus will always be 8 and the number of accounted cpus will be the next higher even number divisible by 4 of requested cpus.

13.6 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:

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

main_loop: do

    compute next_step

    if (time_limit reached) then

        write new_restart_file
        terminate program

    end if

end do main_loop
```

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.

13.6.1 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`.

```
#!/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 [[ ${RETURN} -eq 0 ]]; then
#
#   program terminated successfully
#   - save restart file,
#
    cp restart Restart_Files/restart_${JOB_COUNTER}
    RETURN_CP=$?
#
# Check return code of cp command

if [[ ${RETURN_CP} -ne 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}` -ne 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
    echo "=====
    echo "=                job chain completed successfully                ="
    echo "=====
fi
exit $?
else
#
# my_par_prog terminated with nonzero return code
#
echo "=====
echo "= my_par_prog terminated with return code `${RETURN}` ="
echo "=                job chain terminated                ="
echo "=====
fi

exit `${RETURN}`

```

To initiate this job chain the following command must be entered:

```
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.

13.6.2 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.

```
      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

      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
```

14 Technical Contacts at University of Karlsruhe Computing Center

- **XC Hotline**
xc-hotline@lists.uni-karlsruhe.de
+49 721 608-8011