The NCIT and CoLaborator Cluster Resources Guide

Version 2.0

Adrian Lascateu, Cristina Ilie
{adrian.lascateu|cristina.ilie}@cti.pub.ro

Alexandru Herisanu, Sever Apostu, Alexandru Gavrila
{heri|sever|alexg}@hpc.pub.ro

Release date: 31 July 2009

Contents

1 Latest Changes

This is the first draft of a document concerning the use of NCIT and CoLaborator cluster resources. It was developed during the GridInitiative 2008 Summer School.

Several other versions will follow, please consider this a work in progress.

v.1.0	jul 2008	S.A, A.G	Initial release
v.1.1	2 nov 2008	H.A	Added examples, reformatted LaTex code
v.2.0	jul 2009	A.L, C.I.	Added chapters 7, 8 and 9 Updated chapters 3, 5 and 6

2 Introduction

The **NCIT** (National Center for Information Technology) of "Politehnica" University of Bucharest started back in 2001 with the creation of **CoLaborator**, a Research Base with Multiple Users (R.B.M.U) for High Performance Computing (HPC) , that took part of a World Bank project. **CoLaborator** was designed as a path of communication between Universities, at a national level, using the national network infrastructure for education and research.

Back in 2006, **NCIT**'s infrastructure was enlarged with the creation of a second, more powerful, computing site, more commmonly referred to as the **NCIT Cluster**.

Both clusters are used in research and teaching purposes by teachers, PhD students, grad students and students alike. In the near future a single sign-on (SSO) scenario will be implemented, with the same users' credentials across both sites, using the already existing LDAP infrastructure behind the `http://curs.cs.pub.ro` project.

This document was created with the given purpose of serving as an introduction to the parallel computing paradigm and as a guide to using the clusters' specific resources. You will find within the next paragraphs descriptions of the various existing hardware architectures, operating and programming environments, as well as further (external) information as the given subjects.

2.1 The Cluster

Although the two computing sites are different administrative entities and have different locations, the approach we will use throughout this document will be that of a single cluster with various platforms. This approach is justified also by the future 10Gb Ethernet link between the two sites.

Given both the fact that the cluster was created over a rather large period of time (and it keeps changing, since new machines will continue to be added), and that there is a real need for software testing on multiple platforms, the clusters structure is a heterogenous one, in terms of hardware platforms and operating/programming environments.

There are, currently, four different platforms available on the cluster:

- Intel x86 Linux
- Intel x86_64 Linux
- Sun SPARC Solaris
- Sun Intel x86_64 Solaris

Not all of these platforms are currently given a frontend, the only frontends available at the moment being **fep.hpc.pub.ro** and **fep.grid.pub.ro** machines (**F**ront **E**nd **P**rocessors). The machines behind them are all running non-interactive, being available **only** to jobs sent **through** the frontends.

2.2 Software Overview

The tools used in the **NCIT and CoLaborator cluster** for software developing are Sun Studio, OpenMP and OpenMPI. For debugging we use the TotalView Debugger and Sun Studio and for profiling and performance analysis - Sun Studio Performance Tools.

Sun Studio and GNU compilers were used to compile our tools. The installation of all the tools needed was done using the repository available online at `http://storage.grid.pub.ro/GridInitRepo/GridInitRepo.repo`.

2.3 Further Information

The latest version of this document will be kept online at `http://gridinitiative.ncit.pub.ro/clusterguide`.

You may find the latest news on the **CoLaborator** teaching activities and ongoing projects on `http://hpc.pub.ro`.

For any questions or feedback on this document, please feel free to write us at **mail@hpc.pub.ro**.

3 Hardware

This section covers in detail the different hardware architectures available in the cluster.

General Configuration

3.1 Intel Xeon Based Machines

3.1.1 The Intel Xeon Processor

The Intel Xeon Processor refers to many families of Intel's x86 multiprocessing CPUs for dual-processor (DP) and multi-processor (MP) configuration on a single motherboard targeted at non-consumer markets of server and workstation computers, and also at blade servers and embedded systems.The Xeon CPUs generally have more cache than their desktop counterparts in addition to multiprocessing capabilities.

The Intel Xeon Processor with 800MHz System Bus available at **NCIT Cluster** is a 90nm process technology processor, with a 3.00 Ghz clock frequency, a 16KB L1 Cache, 1024KB full speed L2 Cache with 8-way associativity and Error Correcting Code (ECC).

It provides binary compatibility with applications running on previous members of Intel's IA-32 architecture, hyper-threading technology, enhanced branch prediction and enables system support for up to 64 GB pf physical memory.

3.1.2 IBM eServer xSeries 336

The IBM eServer xSeries 336 servers available at **NCIT Cluster** are 1U rack-mountable corporate business servers, each with one Intel Xeon 3.0 GHz processor with Intel Extended Memory 64 Technology and upgrade possibility, Intel E7520 Chipset Type and a Data Bus Speed of 800MHz.

They come with 512MB DDR2 SDRAM ECC main memory working at 400 MHz (upgradable to a maximum of 16GB), one Ultra320 SCSI integrated controller and one UltraATA 100 integrated IDE controller. Network wise, they have two network interfaces, Ethernet 10Base-T/100Base-TX/1000BaseT (RJ-45).

Eight such machines are available at **NCIT Cluster**, one of which is the second frontend, **fep.grid.pub.ro**.

More information on the IBM eServer xSeries 336 can be found on IBM's support site, http://ibm.com/systems.

3.1.3 Fujitsu-SIEMENS PRIMERGY TX200 S3

The Fujitsu-SIEMENS PRIMERGY TX200 S3 servers available at **NCIT Cluster** have each two Intel Dual-Core Xeon 3.0Ghz with Intel Extended Memory 64 Technology and upgrade possibility, Intel 5000V Chipset Type and a Data Bus Speed of 1066MHz. These processors have 4096KB of L2 Cache, ECC.

They come with 1024MB DDR2 SDRAM ECC main memory, upgradable to a maximum of 16GB, 2-way interleaved, working at 400 MHz., one 8-port SAS variant controller, one Fast-IDE controller and a 6-port controller.

They have two network interfaces, Ethernet 10Base-T/100Base-TX/1000BaseT(RJ-45).

More information on the Fujitsu-SIEMENS PRIMERGY TX200 S3 can be found on Fujitsu-SIEMENS's site, http://primergy.fujitsu.com.

3.2 Fujitsu Esprimo Machines

Core machines. There are curently 66 Fujitsu Esprimo, model P5905, available. They each have an Intel Pentium 4 3.0Ghz CPU, with 2048KB L2 cache, 2048MB DDR2 man memory (upgradable to a maximum of 4GB) working at 533Mhz. Storage SATAII (300MB/s) 250 GB. More information can be found here.

3.3 IBM Blade Center H

There are three chassis and each can fit 14 blades in 9U. You can find general information about the model here. Currently there are two types of blades installed: 32 Intel based **HS21** blades and 4 Cell based **QS22** blades. In the near future, two AMD based **LS22** blades (AMD Opteron processor, 6 cores) will be added.

3.3.1 HS 21 blade

There are 32 H21 blades of which 28 are used for hte batch system and 4 are for developement. Each blade has an Intel Xeon quad-core processor at 2Ghz with 2x6MB L2 cache, 1333Mhz FSB and 16GB of memory. Full specifications here.

3.3.2 QS 22 blade

The Cell based QS22 blade features two dual core 3.2 GHz IBM PowerXCell 8i Processors, 512 KB L2 cache per IBM PowerXCell 8i Processor, plus 256 KB of local store memory for each eDP SPE. Memory capacity 8GB. For more features QS 22 features.

3.4 Storage System

The storage system is composed of the following DELL solutions: 2 PowerEgde 2900 and 2 PowerEdge 2950 servers, and 4 PowerVault MD1000 Storage Arrays.

3.5 UltraSPARC II Based Machines

3.5.1 The UltraSPARC IIs Processor

The UltraSPARC IIs Processor has a 64-bit SPARC V9 VIS instruction set, a 360480Mhz clock frequency and has one core. It was developed back in 1999, with 32KB L1 Cache, 10248192KB L2 Cache and no L3 Cache. This version of the processor with 4096MB of main memory can be found on the Sun Enterprise 220R and 420R.

The Sun Enterprise 10000 comes with 8192MB L2 Cache processors.

More information on the UltraSPARC IIs Processors can be found at `http://en.wikipedia.org/wiki/SPARC`

3.5.2 Sun Enterprise 220R

The Sun Enterprise 220R comes with two 450 MHz UltraSPARC IIs processors, 16KB data and 16KB instruction L1 Cache on chip per processor, and 4MB L2 Cache external per processor. It has 512MB of main memory, divided between 4 60-ns memory modules of 128MB each. Network wise, the machines come with four Ethernet/Fast Ethernet (10BASE-T/100BASE-T) twisted-pair standard connectors (RJ-45) or one MII for external transceiver connection.

Three such machines are available at **CoLaborator**, two of them forming a HA cluster in the near future.

More information on the Sun Enterprise 220R machine can be found at http://sunsolve.sun.com.

3.5.3 Sun Enterprise 420R

The Sun Enterprise 420R comes with up to four 450 MHz UltraSPARC IIs processors, 16KB data and 16KB instruction L1 Cache on chip per processor, and 4MB L2 Cache external per processor. It has 1024MB of main memory, divided between 4 60-ns memory modules of 256MB each and a datapath of up to 1.78GB/sec throughput. Network wise, the machines come with four Ethernet/Fast Ethernet (10BASE-T/100BASE-T) twisted-pair standard connectors (RJ-45), self-sensing.

One Sun Enterprise 420R is available at **CoLaborator**, namely one of the frontends, **fep.hpc.pub.ro**.

More information on the Sun Enterprise 420R machine can be found at http://sunsolve.sun.com.

3.5.4 Sun Enterprise 10000

The Sun Enterprise 10000 (codenamed Starfire) is a high-end multiprocessor datacenter server capable of up to 64 UltraSPARC IIs processors. The one available at **CoLaborator** comes with 32 466Mhz UltraSPARC IIs processors, with 32KB L1 Cache per processor and 8192KB L2 external Cache.

It uses a Gigaplane-XB interconnect with 12.8 GB/second throughput, a memory data bus of 576 bits, a data bus of 144 bits per board and a CPU data bus of 144 bits.

The system is equiped with 8 System Boards, each of them containing four processors and four banks of eight memory SIMMs on each system board and two Control Boards which control the system JTAG, clock, fan, power, serial interface, and Ethernet interface functions. Main memory's capacity is of 32GB, divided between the System Boards, with up to four memory expansion options per System Board.

Storage wise, the system supports a maximum of two Sun StorEdge A3500 arrays, a high availability disk subsystem with hardware RAID Storage up to 1 Tbyte per cabinet using 18 Gbyte disks. It also comes equiped with a QUANTUM tape device, using digital linear tape (DLT) with autoloaders and libraries.

The physical machine is logically divided between two domains, called HPCDOM and ORADOM, each of them being allocated CPU and memory resources, with the posibility of dynamicaly changing ratios.

More information on the Sun Enterprise 10000 machine can be found at http://sunsolve.sun.com and on Sun's site, at http://www.sun.com/servers/highend/e10000.

4 Operating systems

The two operating systems running in the **NCIT and CoLaborator Cluster** are Solaris and Linux.

Solaris is the only operating system currently available on the SPARC systems from **CoLaborator**. Linux runs on the x86_64 and x86 platforms from Intel available at the **NCIT Cluster**.

Although both operating systems are UNIX-derived, there are several differences between them. In the following subsections, we'll try to explain some of them.

4.1 Linux

Linux is the UNIX-like operating system. It's name comes from the Linux kernel, originally written in 1991 by Linus Torvalds. The system's utilities and libraries usually come from the GNU operating system, announced in 1983 by Richard Stallman.

The Linux release used at the **NCIT Cluster** is a RHEL (Red Hat Enterprise Linux) clone called Scientific Linux, co-developed by Fermi National Accelerator Laboratory and the European Organization for Nuclear Research (CERN).

The Linux kernel version is displayed by the command:

```
$ uname -r
```

whereas the distribution release is displayed by the command:

```
$ cat /etc/issue
```

4.2 Solaris

Solaris is the UNIX-based operating system introduced by Sun Microsystems since 1992. It comes as a successor to SunOS 4 and, opposed to it, it's not BSD-derived but SVR4-based.

It was first identified as SunOS 5, but Sun decided to name SunOS 5.2 as Solaris 2, with the intention of encompassing not only SunOS, but also the **OpenWindows graphical user interface** and **Open Network Computing (ONC)** functionalities.

The SunOS minor version is included in the Solaris release number, such that, the command:

```
$ uname -r
```

will show the corresponding SunOS release level.

Solaris also supports the Linux platform ABI, allowing it to run native Linux binaries on x86 systems. This feature is called "Solaris Containers for Linux Applications" (SCLA), based on the branded zones functionality introduced in Solaris 10 8/07.

4.3 Addressing Modes

Both Solaris and Linux support 64bit addressing, thus programs can be compiled and linked either in 32 or 64bit mode. This has no influence on the capacity or precision of floating point numbers (4 or 8 byte real numbers), affecting only memory addressing, the usage of 32 or 64bit pointers. Obviously, programs requiring more than 4GB of memory have to use the 64bit addressing mode. It's necessary to specify the addressing mode at compile and at link time, since the default mode is 32bit on Solaris and 64bit on Linux.

5 Environment

5.1 Login

Logging into UNIX-like systems is done through the secure shell (**SSH**). Since usually the SSH daemon is installed by default both on Solaris and Linux systems. You can log into each one of the cluster's frontends from your local UNIX machine, using the ssh command:

```
$ ssh username@fep.hpc.pub.ro
$ ssh username@fep.grid.pub.ro
```

Depending on your local configuration it may be necessary to use the -Y flag to enable the trusted forwarding of graphical programs.

Logging into one of the frontends from your local Windows machine is done with the use of a windows SSH client such as putty.

5.2 File Management

Every user of the cluster has a home directory on a shared filesystem within the cluster. This is **$HOME**=/export/home/*username*.

Transfering files to Solaris or Linux from your local UNIX-like machine is done through the secure copy command **scp**, e.g:

```
$ scp localfile username@fep.hpc.pub.ro:
$ scp -r localdirectory username@fep.grid.pub.ro:
```

The default directory where scp copies the file is the home directory. If you want to specify a different path where to save the file, you should write the path after ":"

Example:

```
$ scp localfile username@fep.hpc.pub.ro:your/relative/path/to/home
$ scp localfile username@fep.hpc.pub.ro:/your/absolute/path
```

Transfering files from Solaris or Linux to your local UNIX-like machine is done through the secure copy command **scp**, e.g:

```
$ scp username@fep.hpc.pub.ro:path/to/file /path/to/destination/on/local/machine
```

WinSCP is a **scp** client for Windows that provides a graphical file manager for copying files to and from the cluster.

NX Client is a solution for secure remote access, desktop virtualization, and hosted desktop deployment. It is used also on Linux and also on Windows.

If you want to copy an archive from a link in your current directory it is easier to use wget. You should use *Copy Link Location* (from your browser) and paste the link as parameter for wget.

Example: wget http://link/for/download

5.3 Module Package

The **Module package** provides for the dynamic modification of the user's environment. Initialization scripts can be loaded and unloaded to alter or set shell environemnt variables such as `$PATH` or `$LD_LIBRARY_PATH`, to choose for example a specific compiler version or use software packages.

The advantage of the modules system is that environment changes can easily be undone by unloading a module. Dependencies and conflicts can be easily controlled. If, say, you need mpi with gcc then you'll just have to load both the gcc compiler and mpi-gcc modules. The module files will make all the necessary changes to the environment variables.

Note: The changes will remain active only for your current session. When you exit, they will revert back to the initial settings.

For working with modules, the **module** command is used. The most important options are explained in the following.

To get help about the module command you can either read the manual page (man module), or type

```
$ module help
```

To get the list of available modules type

```
$ module avail
----------------------- /opt/modules/Modules/3.2.5/modulefiles --------------
dot         module-cvs  module-info modules     null        use.own

------------------------- /opt/modules/modulefiles -------------------------------
apps/hrm               compilers/intel-11.0_081       mpi/openmpi-1.3.2_gcc-4.1.2
apps/matlab            compilers/pgi-7.0.7            mpi/openmpi-1.3.2_gcc-4.4.0
apps/uso09             compilers/sunstudio12.1        mpi/openmpi-1.3.2_intel-11.0_081
batch-system/sge-6.2u3           debuggers/totalview-8.6.2-2
cell/cell-sdk-3.1                grid/gLite-UI-3.1.31-Prod
compilers/gcc-4.1.2              java/jdk1.6.0_13-32bit
compilers/gcc-4.4.0              java/jdk1.6.0_13-64bit
```

An available module can be loaded with

```
$ module load [module name] -> $ module load compilers/gcc-4.1.2
```

A module which has been loaded before but is lo longer needed can be removed by

```
$ module unload [module name]
```

If you want to use another version of a software or e.g. another compiler, we strongly recommend to switch between modules.

```
$ module switch [oldfile] [newfile]
```

This will unload all modules from bottom up to the oldle , unload the oldle , load the newle and then reload all previously unload modules. Due to this procedure the order of the loaded modules is not changed and dependencies will be rechecked. Furthermore some modules adjust their environment variables to match previous loaded modules.

You will get a list of loaded modules with

```
$ module list
```

A short information about the software initialized by a module can be obtained by

```
$ module whatis [file]
e.g.: $ module whatis compilers/gcc-4.1.2
compilers/gcc-4.1.2  : Sets up the GCC 4.1.2 (RedHat 5.3) Environment.
```

You can add a directory with your own module files with

```
$ module use path
```

Note : If you loaded module files in order to compile a program, you probably have to load the same module files before running that program. Otherwise some necessary libraries may not be found at program start time. This is also true if using the batch system!

5.4 Batch System

A batch system controls the distribution of tasks (batch jobs) to the available machines or resources. It ensures that the machines are not overbooked, to provide optimal program execution. If no suitable machines have available resources, the batch job is queued and will be executed as soon as there are resources available. Compute jobs that are expected to run for a large period of time or use a lot of resources should use the batchsystem in order to reduce load on the frontend machines.

You may submit your jobs for execution on one of the available queues. Each of the queues has an associated environment.

To display queues summary:

```
$ qstat -g c [-q queue]
CLUSTER QUEUE                   CQLOAD   USED    RES  AVAIL  TOTAL aoACDS  cdsuE
--------------------------------------------------------------------------------
all.q                             0.00      0      0    192    200      0      8
gridinit.q                        0.02     24      0      0     24      0      0
uso09                             0.00      0      0      1      1      0      0
```

5.4.1 Sun Grid Engine

To submit a job for execution over a cluster, you have two options: either specify the command directly, or provide a script that will be executed. This behavior is controlled by the "-b y—n" parameter as follows: "y" means the command may be a binary or a script and "n" means it will be treated as a script.

Some examples of submitting jobs (both binaries and scripts).

```
$ qsub -q [queue]  -b y [executable] -> $ qsub -q queue_1 -b y /path/my_exec

$ qsub -pe [pe_name] [no_procs] -q [queue]  -b n [script]
e.g: $ qsub -pe pe_1 4 -q queue_1 -b n my_script.sh
```

To watch the evolution of the submited job, use **qstat**. Running it without any arguments shows information about the jobs submited by you alone.

To see the progress of all the jobs use the -f flag. You may also specify which queue jobs you are interested in by using the -q [queue] parameter, e.g:

```
$ qstat [-f] [-q queue]
```

Typing "watch qstat" will automatically run qstat every 2 sec. To exit type "Ctrl-C".
In order to delete a job that was previously submitted invoke the qdel command, e.g:

```
$ qdel [-f] [-u user_list] [job_range_list]
```

where:
-f - forces action for running jobs
-u - users whose jobs will be removed. To delete all the jobs for all users use -u "*".

An example of submitting a job with SGE looks like that:

```
$ cat script.sh
#!/bin/bash
‘pwd‘/script.sh
$ chmod +x script.sh
$ qsub -q queue_1 script.sh   (you may omit -b and it will behave like -b n)
```

To display the sumited jobs of all users(-u "*") or a specified user, use:

```
$ qstat [-q queue] [-u user]
```

To display extended information about some jobs, use:

```
$ qstat -t [-u  user]
```

To print detailed information about one job, use:

```
$ qstat -j job_id
```

6 Programming

This section covers in detail the programming tools available on the cluster, including compilers, debuggers and profiling tools.

6.1 Compilers

6.1.1 General Compiling and Linker hints

To access non-default compilers you have to load the appropriate module le - using module avail to see the availables modules and the module load to load the modul (see *5.3 Module Package*). You canthen access the compilers by there original name, e.g. g++, gcc, gfortran, or by environment-variables $CXX, $CC or $FC. When, however, loading more than one compiler module, you have to be aware that environment variables point to the compiler loaded at last.

For convenient switching between compilers and platforms, we added environment variables for the most important compiler flags. These variables can be used to write a generic makefile which compiles on all our Unix like platforms:

- **FC, CC, CXX** -a variable containing the appropiate compiler name.
- **FLAGS_DEBUG** -enable debug information.
- **FLAGS_FAST** - include the options which usually offer good performance.For many compiler this will be the -fast option.
- **FLAGS_FAST_NO_FPOPT** - like fast, but disallow any floating point optimizations which will have an impact on rounding errors.
- **FLAGS_ARCH32** - build 32 bit executables or libraries.
- **FLAGS_ARCH64** - build 64 bit executables or libraries.
- **FLAGS_AUTOPAR** - enable autoparallelization, if the compiler supports it.
- **FLAGS_OPENMP** - enable OpenMP support, if supported by the compiler.

To produce debugging information in the operating systems native format use -g option at compile time.

In general we recommend to use the same ags for compiling and for linking. Otherwise the program may not run correctly or linking may fail.

The order of the command line options while compiling and linking does matter. If you get unresolved symbols while linking, this may be caused by a wrong order of libraries. If a library xxx uses symbols out of the library yyy, the library yyy has to be right of xxx in the command line, e.g ld ... -lxxx -lyyy.

The search path for header les is extended with the -Idirectory option and the library search path with the -Ldirectory option.

The environment variable `ld_library_path` species the search path where the program loader looks for shared libraries. Some compile time linker, e.g. the Sun linker, also use this variable while linking, while the GNU linker does not.

6.1.2 GNU Compilers

The **GNU C/C++/Fortran** compilers are available by using the binaries gcc, g++, g77 and gfortran or by environment variables `$CXX, $CC or $FC`. If you cannot access them you have to load the appropiate module file as is described in section *5.3 Module Package*.

For further references the manual pages are available.

Some options to compile your program and increase their performance are:

- -m32 or -m64, to produce code with 32-bit or 64-bit addresing - as mentioned above, the default is platform dependant
- -march=opteron, to optimize for the Pentium processor (NCIT Cluster)
- -mcpu=ultrasparc optimize for the Ultrasparc I/II processors (CoLaborator)
- -O2 or -O3, for different levels of optimization
- -malign-double, for Pentium specific optimization
- -ffast-math, for floating point optimizations

GNU Compilers with versions above 4.2 support OpenMP by default. Use the -fopenmp flag to enable the OpenMP support.

6.1.3 Sun Compilers

We use Sun Studio 6 on the Solaris machines (soon to be upgraded) and Sun Studio 12 on the Linux machines. Nevertheless, the use of these two versions of Sun Studio is pretty much the same.

The **Sun Studio** development tools include the Fortran95, C and C++ compilers. The best way to keep your applications free of bugs and at the actual performance level we recommend you to recompile your code with the latest production compiler. In order to check the version that your are currently using use the flag -V.

The commands that invoke the compilers are **cc, f77, f90, f95** and **CC**.

An important aspect about the Fortran 77 compiler is that from Sun Studio 7 is no longer available. Actually, the command **f77** invokes a script that is a wrapper and it is used to pass

the necessary compatibility options, like **-f77**, to the f95 compiler. We recommend adding **-f77 -trap=common** in order to revert to f95 settings for error trapping. At the link step you may want to use the **-xlang=f77** option(when linking to old f77 object binaries). Detailed information about compatibility issues between Fortran 77 and Fortran 95 can be found in http://docs.sun.com/source/816-2457/5_f77.html

For more information about the use of Sun Studio compilers you may use the man pages but you may also use the documentation found at http://developers.sun.com/sunstudio/documentation.

6.1.4 Intel Compilers

Use the module command to load the Intel compilers into your environment. The curent version of Intel Compiler is 11.1. The Intel C/C++ and Fortran77/Fortran90 compilers are invoked by the commands **icc** — **icpc** — **ifort** on Linux. The corresponding manual pages are available for further information. Some options to increase the performance of the produced code include -O3 high optimization -fp-model fast=2 enable oating point optimization -openmp turn on OpenMP -parallel turn on auto-parallelization In order to read or write big-endian binary data in Fortran programs, you can use the compiler option **-convert big_endian**.

6.1.5 PGI Compiler

PGI compilers are a set of commercially available Fortran, C and C++ compilers for High Performance Computing Systems from Portland Group.

PGI Compiler:

- PGF95 - for fortran
- PGCC - for c
- PGC++ - for c++

PGI Recommended Default Flags:

- -fast A generally optimal set of options including global optimization, SIMD vectorization, loop unrolling and cache optimizations.
- -Mipa=fast,inline Aggressive inter-procedural analysis and optimization, including automatic inlining.
- -Msmartalloc Use optimized memory allocation (Linux only).
- –zc_eh Generate low-overhead exception regions.

PGI Tuning Flags

- -Mconcur Enable auto-parallelization; for use with multi-core or multi-processor targets.
- -mp Enable OpenMP; enable user inserted parallel programming directives and pragmas.

- -Mprefetch Control generation of prefetch instructions to improve memory performance in compute-intensive loops.
- -Msafeptr Ignore potential data dependencies between C/C++ pointers.
- -Mfprelaxed Relax floating point precision; trade accuracy for speed.
- -tp x64 Create a PGI Unified Binary which functions correctly on and is optimized for both Intel and AMD processors.
- -Mpfi/-Mpfo Profile Feedback Optimization; requires two compilation passes and an interim execution to generate a profile.

6.2 OpenMPI

OpenMPI RPM's are available compiled both for 32and 64bit machines. It was compiled with both Sun Studio and GNU Compilers and the user may select which one to use depending on the task.

The compilers provided by OpenMPI are mpicc, mpiCC, mpic++, mpicxx, mpif77 and mpif90. Please note that mpiCC, mpic++ and mpicxx all invoke the same C++ compiler with the same options. Another aspect is that all of these commands are only wrappers that actually call opal_wrapper. Using the -show flag does not invoke the compiler, instead it prints the command that would be executed. To find out all the possible flags these commands may receive, use the -flags flag.

To compile your program with mpicc, use:

```
$ mpicc -c pr.c
```

To link your compiled program, use:

```
$ mpicc -o pr pr.o
```

To compile and link all at once, use:

```
$ mpicc -o pr pr.c
```

For the others compilers the commands are the same - you only have to replace the compiler's name with the proper one.

The mpirun command executes a program, like

```
$ mpirun [options] <program> [<args>]
```

The most used option specifies the number of cores to run the job: -n #. It is not necessary to specify the hosts on which the job would execute because this will be managed by Sun Grid Engine.

6.3 OpenMP

On Linux machines. GNU C Compiler now provides integrated support for OpenMP. To compile your programs to use "#pragma omp" directives use the -fopenmp flag in addition to the gcc command.

When using Solaris, in order to get OpenMP support, use Sun Studio compilers with the following flags:
- -xopenmp, to activate OpenMP
- -xloopinfo, for loop parallelization messages

- -xO3
- -xautopar, for auto parallelization

Below is a simple example of a C program using OpenMP. It only creates a number of threads and uses them to print a Ḧello Worldm̈essage.

```
$ cat hello.c
 #include <omp.h>
 #include <stdio.h>

 int main (int argc, char *argv[]) {
   int th_id, nthreads;
   #pragma omp parallel private(th_id)
   {
     th_id = omp_get_thread_num();
     printf("Hello World from thread %d\n", th_id);
     #pragma omp barrier
     if ( th_id == 0 ) {
       nthreads = omp_get_num_threads();
       printf("There are %d threads\n",nthreads);
     }
   }
   return 0;
 }
```

To compile using the Sun Studio Compilers, use:

```
$ cc -xopenmp -xO3 -o hello.c
```

The following environment variables can affect the program at runtime:

- OMP_SCHEDULE - Sets schedule type for parallel for, for, directives/pragmas with schedule type RUNTIME specified.
- OMP_NUM_THREADS - Sets the number of threads to use during execution of a parallel region.
- OMP_DYNAMIC - Enables or disables dynamic adjustment of the number of threads available for execution of parallel regions.
- OMP_NESTED - Enables or disables nested parallelism

For more information about shared memory parallelization see **chapter 7**.

6.4 Debuggers

6.4.1 Sun Studio Integrated Debugger

Sun Studio includes a debugger for serial and multithreaded programs. You will find more information on how to use this environment online. You may start debugging your program from *Run - Debugg executabile*.

There is a series of actions that are available from the *Run* menu but which may be specified from the command line when starting Sun Studio. In order to just start a debuggind session, you can attach to a running program by:

```
$ sunstudio -A pid[:program_name]
```

or from *Run - Attach Debugger*. To analyse a core dump, use:

```
$ sunstudio -C core[:program_name]
```

or *Run - Debug core file*

6.4.2 TotalView

To use TotalView over a cluster there are a few steps to follow. It is very important to have the $HOME directory shared over the network between nodes. We will submit a SGE job which will open a terminal from where we launch TotalView. The program is already compiled in the shared $HOME directory. These are the steps to follow:

```
$ xhost +
```

will permit X11 connections.

Next we need to find out the DISPLAY and the port the X11 connection is forwarded. For example, if DISPLAY is host:14.0 the connection port will be 6014.

```
$ echo $DISPLAY
```

Now the script to be used with SGE needs to be written. For example the script may look like this one:

```
$ cat script.sh
#!/bin/bash
#$ -cwd
#$ -q fs-dual

setenv DISPLAY fep.grid.pub.ro:14.0

/usr/bin/xterm

$ chmod +x script.sh
```

Submitting the job is done with:

```
$ qsub script.sh
```

When the xterm window will appear, we will launch TotalView. If the window does not appear check the job output. This should give you some clue and why your script failed; maybe you misspelled a command or maybe the port for the X11 forwarding is closed from the firewall. You should check these two things first.

When TotalView launches we need to select the program we want to debug. We may also attach to a running program. From the Parallel tab we need to select the compiler used to obtain the binary file and how many task to run on how many nodes.

When this phase is finished a window appears. The control buttons are located in the upper part of the window. The program's code is shown in a sub-window and breakpoints can be toggled and removed by clicking on the left side bar.

7 Parallelization

Parallelization for computers with shared memory (SM) means the automatic distribution of loop iterations over several processors(autoparallelization), the explicit distribution of work over the processors by compiler directives (OpenMP) or function calls to threading libraries, or a combination of those.

Parallelization for computers with distributed memory (DM) is done via the explicit dis- tribution of work and data over the processors and their coordination with the exchange of messages (Message Passing with MPI).

MPI programs run on shared memory computers as well, whereas OpenMP programs usu- ally do not run on computers with distributed memory.

There are solutions that try to achieve the programming ease of shared memory paralleliza- tion on distributed memory clusters. For example Intels Cluster OpenMP oers a relatively easy way to get OpenMP programs running on a cluster.

For large applications the hybrid parallelization approach, a combination of coarse-grained parallelism with MPI and underlying ne-grained parallelism with OpenMP, might be attrac- tive, in order to use as many processors eciently as possible.

Please note, that large computing jobs should not be started interactively, and that when submitting use of batch jobs (see chapter 4.6 on page 25), the GridEngine batch system determines the distribution of the MPI tasks on the machines to a large extent.

7.1 Shared Memory Programming

For shared memory programming, OpenMP is the de facto standard (http://www.openmp.org). The OpenMP API is dened for Fortran, C and C++ and consists of compiler directives (resp. pragmas), runtime routines and environment variables.

In the parallel regions of a program several threads are started. They execute the contained program segment redundantly, until they hit a worksharing construct . Within this construct, the contained work (usually do- or for-loops) is distributed among the threads. Under normal conditions all threads have access to all data (shared data). But pay attention: if data, accessed by several threads, is modied, then the access to this data must be protected with critical regions or OpenMP locks .

Also private data areas can be used, where the individual threads hold their local data. Such private data (in OpenMP terminology) is only visible to the thread owning it. Other threads will not be able to read or write private data.

Note: In many cases, the stack area for the slave threads must be increased by changing a compiler specic environment variable (e.g. Sun Studio: stacksize, Intel:kmp_stacksize), and the stack area for the master thread must be increased with the command ulimit -s xxx (zsh shell, specication in KB) or limit s xxx (C-shell, in KB).

Hint: In a loop, which is to be parallelized, the results must not depend on the order of the loop iterations! Try to run the loop backwards in serial mode. The results should be the same. This is a necessary, but not a sucient condition!

The number of the threads has to be specied by the environment variable omp_num_threads.

Note: The OpenMP standard does not specify the value for omp_num_threads in case it is not set explicitly. If omp_num_threads is not set, then Sun OpenMP for example starts only 1 thread, as opposed to the Intel compiler which starts as many threads as there are processors available.

On a loaded system fewer threads may be employed than specied by this environment variable, because the dynamic mode may be used by default. Use the environment variable omp_dynamic to change this behavior. If you want to use nested OpenMP, the environment variable omp_nested=true has to be set.

The OpenMP compiler options have been sumarized in the following. These compiler flags will be set in the environment variables FLAGS_AUTOPAR and FLAGS_OPENMP (as explained in section 6.1.1)

Compiler	**flags_openmp**	**flags_autopar**
Sun	-xopenmp	-xautopar -xreduction
Intel	-openmp	-parallel
GNU	-fopenmp (4.2 and above)	n.a. (planned for 4.3)
PGI	-mp	-Mconcur -Minline

An example program using OpenMP is /export/home/stud/alascateu/PRIMER/PROFILE/openmp_only.c

7.1.1 Automatic Shared Memory Parallelization of Loops

The Sun Fortran, C and C++-compilers are able to parallelize loops automatically. Success or failure depends on the compilers ability to prove it is safe to parallelize a (nested) loop. This is often application area specic (e.g. nite dierences versus nite elements), language (pointers and function calls may make the analysis dicult) and coding style dependent. The appropriate option is -xautopar which includes -depend -xO3. Although the -xparallel option is also available, we do not recommend to use this. The option combines auto- matic and explicit parallelization, but assumes the older Sun parallel programming model is used instead of OpenMP. In case one would like to combine automatic parallelization and OpenMP, we strongly suggest to use the -xautopar -xopenmp combination. With the option -xreduction, automatic parallelization of reductions is also permitted, e.g. accumulations, dot products etc., whereby the modication of the sequence of the arithmetic operation can cause dierent rounding error accumulations.

Compiling with the option -xloopinfo supplies information about the parallelization. The compiler messages are shown on the screen.

If the number of loop iterations is unknown during compile time, then code is produced, which decides at run-time whether a parallel execution of the loop is more ecient or not (alternate coding).

Also with automatic parallelization, the number of the used threads can be specied by the environment variable omp_num_threads.

7.1.2 GNU Compilers

With version 4.2 the GNU compiler collection supports OpenMP with the option **-fopenmp**. It supports nesting using the standard OpenMP environment variables. In addition to these variables you can set the default thread stack size in kilobytes with the variable **GOMP_STACKSIZE**.

CPU binding can be done with the **GOMP_CPU_AFFINITY** environment variable. The variable should contain a space or comma-separated list of CPU's. This list may contain different kind of entries: either single CPU numbers in any order, a range of CPU's (M-N) or a range with some stride (M-N:S). CPU numbers are zero based. For example, **GOMP_CPU_AFFINITY="0 3 1-2 4-15:2"** will bind the initial thread to CPU 0, the second to CPU 3, the third to CPU 1, the fourth to CPU 2, the fifth to CPU 4, the sixth through tneth to CPU's 6, 8, 10, 12 and 14 respectively

and then start assigning back from the beggining of the list. **CGOMP_CPU_AFFINITY=0** binds all threads to CPU 0.

Automatic Shared Memory Parallelization of Loops Since version 4.3 the GNU compilers are able to parallelize loops automatically with the option **-ftree-parallelize-loops=[threads]**. However the number of threads to use have to be specified at compile time and thus are fix at runtime.

7.1.3 Intel Compilers

By adding the option -openmp the OpenMP directives are interpreted by the Intel compilers. Nested OpenMP is supported too.

The slave threads stack size can be increased with the environment variable kmp_stacksize=megabytes M.

Attention: By default the number of threads is set to the number of processors. It is not recommended to set this variable to larger values than the number of processors available on the current machine. By default, the environment variables omp_dynamic and omp_nested are set to false.

Intel compilers provide an easy way for processor binding. Just set the environment variable kmp_affinity to compact or scatter, e.g.

```
$ export KMP_AFFINITY=scatter
```

Compact binds the threads as near as possible, e.g. two threads on dierent cores of one processor chip. Scatter binds the threads as far away as possible, e.g. two threads, each on one core on dierent processor sockets.

Automatic Shared Memory Parallelization of Loops The Intel Fortran, C and C++ compilers are able to parallelize certain loops automatically. This feature can be turned on with the option **-parallel**. The number of the used threads is specied by the environment variable OMP_NUM_THREADS.

Note: using the option -O2 enables automatic inlining which may help the automatic parallelization, if functions are called within a loop.

7.1.4 PGI Compilers

By adding the option -mp the OpenMP directives, according to the OpenMP version 1 speci- cations, are interpreted by the PGI compilers.

Explicit parallelization can be combined with the automatic parallelization of the compiler. Loops within parallel OpenMP regions are no longer subject to automatic parallelization. Nested parallelization is not supported. The slave threads stack size can be increased with the environment variable mpstkz=megabytes M

By default omp_num_threads is set to 1. It is not recommended to set this variable to a larger value than the number of processors available on the current machine. The environment variables omp_dynamic and omp_nested have no eect!

The PGI compiler oers some support for NUMA architectures like the V40z Opteron systems with the option -mp=numa. Using NUMA can improve performance of some parallel applications by reducing memory latency. Linking -mp=numa also allows you to use the environment variables

mp_bind, mp_blist and mp_spin. When mp_bind is set to yes , parallel processes or threads are bound to a physical processor. This ensures that the operating system will not move your process to a dierent CPU while it is running. Using mp_blist, you can specify exactly which processors to attach your process to. For example, if you have a quad socket dual core system (8 CPUs), you can set the blist so that the processes are interleaved across the 4 sockets (MP_BLIST=2,4,6,0,1,3,5,7) or bound to a particular (MP_BLIST=6,7).

Threads at a barrier in a parallel region check a semaphore to determine if they can proceed. If the semaphore is not free after a certain number of tries, the thread gives up the processor for a while before checking again. The mp_spin variable denes the number of times a thread checks a semaphore before idling. Setting mp_spin to -1 tells the thread never to idle. This can improve performance but can waste CPU cycles that could be used by a dierent process if the thread spends a signicant amount of time in a barrier.

Automatic Shared Memory Parallelization of Loops The PGI Fortran, C and C++ compilers are able to parallelize certain loops automatically. This feature can be turned on with the option -Mconcur. The number of the used threads is also specied by the environment variable OMP_NUM_THREADS.

Note: Using the option -Minline the compiler tries to inline functions. So even loops with function calls may be parallelized.

7.2 Message Passing with MPI

MPI (Message-Passing Interface) is the de-facto standard for parallelization on distributed memory parallel systems. Multiple processes explicitly exchange data and coordinate their work flow. MPI specifies the interface but not the implementaion. Therefore, there are plenty of implementations for PC's as well as for supercomputers. There are freely available implementations and commercial ones, which are particularly tuned for the target platform. MPI has a huge number of calls, although it is possible to write meaningful MPI applications just employing some 10 of these calls.

An example program using MPI is /export/home/stud/alascateu/PRIMER/PROFILE/mpi.c.

7.2.1 OpenMPI

The Open MPI Project (www.openmpi.org) is an open source MPI-2 implementation that is developed and maintained by a consortium of academic, research, and industry partners. Open MPI is therefore able to combine the expertise, technologies, and resources from all across the High Performance Computing community in order to build the best MPI library available. Open MPI offers advantages for system and software vendors, application developers and computer science researchers.

The compiler drivers are **mpicc** for C, **mpif77** and **mpif90** for FORTRAN, **mpicxx** and **mpiCC** for C++.

mpirun is used to start a MPI program. Refere to the manual page for a detailed description of mpirun (`$ man mpirun`).

We have several Open MPI implementations. To use the one suitable for your programs, you must load the appropriate module (remember to also load the corresponding compiler module). For example, if you want to use the **PGI** implementation you should type the following:

```
$ module list
Currently Loaded Modulefiles:
```

```
  1) batch-system/sge-6.2u3            4) switcher/1.0.13
  2) compilers/sunstudio12.1           5) oscar-modules/1.0.5
  3) mpi/openmpi-1.3.2_sunstudio12.1
$ module avail

[...]
-------------------- /opt/modules/modulefiles --------------------
apps/hrm                          debuggers/totalview-8.6.2-2
apps/matlab                       grid/gLite-UI-3.1.31-Prod
apps/uso09                        java/jdk1.6.0_13-32bit
batch-system/sge-6.2u3            java/jdk1.6.0_13-64bit
cell/cell-sdk-3.1                 mpi/openmpi-1.3.2_gcc-4.1.2
compilers/gcc-4.1.2               mpi/openmpi-1.3.2_gcc-4.4.0
compilers/gcc-4.4.0               mpi/openmpi-1.3.2_intel-11.0_081
compilers/intel-11.0_081          mpi/openmpi-1.3.2_pgi-7.0.7
compilers/pgi-7.0.7               mpi/openmpi-1.3.2_sunstudio12.1
compilers/sunstudio12.1           oscar-modules/1.0.5(default)
```

Load the PGI implementation of MPI

```
$ module switch mpi/openmpi-1.3.2_pgi-7.0.7
```

Load the PGI compiler

```
$ module switch compilers/pgi-7.0.7
```

Now if you type **mpicc** you'll see that the wrapper calls **pgcc**.

7.2.2 Intel MPI Implementation

Intel-MPI is a commercial implementation based on mpich2 which is a public domain implementation of the MPI 2 standard provided by the Mathematics and Computer Science Division of the Argonne National Laboratory.

The compiler drivers **mpifc**, **mpiifort**, **mpiicc**, **mpiicpc**, **mpicc** and **mpicxx** and the instruction for starting an MPI application mpiexec will be included in the search path.

There are two dierent versions of compiler driver: **mpiifort**, **mpiicc** and **mpiicpc** are the compiler driver for Intel Compiler. **mpifc**, **mpicc** and **mpicxx** are the compiler driver for GCC (GNU Compiler Collection).

To use the Intel implementation you must load the apporpriate modules just like in the PGI exemple in the OpenMPI section.

Examples:

```
$ mpiifort -c ... *.f90
$ mpiicc -o a.out *.o
$ mpirun -np 4 a.out:
$ ifort -I$MPI_INCLUDE -c prog.f90
$ mpirun -np 4 a.out
```

7.3 Hybrid Parallelization

The combination of MPI and OpenMP and/or autoparallelization is called hybrid paralleliza- tion. Each MPI process may be multi-threaded. In order to use hybrid parallelization the MPI library has to support it. There are 4 stages of possible support:

1. single - multi-threading is not supported.
2. funneled - only the main thread, which initializes MPI, is allowed to make MPI calls.
3. serialized - only one thread may call the MPI library at a time.
4. multiple - multiple threads may call MPI, without restrictions.

You can use the MPI_Init_thread function to query multi-threading support of the MPI implementation.

A quick example of a hybrid program is /export/home/stud/alascateu/PRIMER/PROFILE/hybrid.c. It is a standard Laplace equation program, with MPI support, in witch a simple OpenMP matrix multiply program was inserted. Thus, every process distributed over the cluster will spawn multiple threads that will multiply some random matrices. The matrix dimensions where augmented so the program would run sufficient time to collect experiment data with the Sun Analyzer presented in the **Performance / Runtime Analysis Tools** section.

Tu run the program (C environment in the example) compile it as a MPI program but with OpenMP support:

```
$ mpicc -fopenmp hybrid.c -o hybrid
```

Run with (due to the Laplace layout you need 4 processors):

```
$ mpirun -np 4 hybrid
```

7.3.1 Hybrid Parallelization with Intel-MPI

Unfortunately Intel-MPI is not thread safe by default. Calls to the MPI library should not be made inside of parallel regions if the library is not linked to the program. To provide full MPI support inside parallel regions the program must be linked with the option -mt_mpi. Note: If you specify either the -openmp or the -parallel option of the Intel C Compiler, the thread safe version of the library is used. Note: If you specify one of the following options for the Intel Fortran Compiler, the thread safe version of the library is used:

1. -openmp
2. -parallel
3. -threads
4. -reentrancy
5. -reentrancy threaded

8 Performance / Runtime Analysis Tools

This chapter describes tools that are available to help you assess the performance of your code, identify potential performance problems, and locate the part of the code where most of the execution time is spent. It also covers the installation and run of an Intel MPI benchmark.

8.1 Sun Sampling Collector and Performance Analyzer

The Sun Sampling Collector and the Performance Analyzer are a pair of tools that you can use to collect and analyze performance data for your serial or parallel application. The Collector gathers performance data by sampling at regular time intervals and by tracing function calls. The performance information is gathered in so called experiment files, which can then be displayed with the analyzer GUI or the er_print line command after the program has nished. Since the collector is part of the Sun compiler suite the studio compiler module has to be loaded. However programs to be analyzed do not have to be compiled with the Sun compiler, the GNU or Intel compiler for example work as well.

8.1.1 Collecting experiment data

The first step in profiling with the Sun Analyzer is to obtain experiement data. For this you must compile your code with the **-g** option. After that you can either run **collect** like this

```
$ collect a.out
```

or use the *GUI*.

To use the GUI to collect experiment data, start the analyzer (**X11 forwarding must be enabled** - `$ analyzer`), go to *Collect Experiment* under the *File* menu and select the **Target**, **Working Directory** and add **Arguments** if you need to. Click on **Preview Command** to view the command for collecting experiment data only. Now you can submit the command to a queue.

Some example of scripts used to submit the command (the path to **collect** might be different):

```
$ cat script.sh
#!/bin/bash

qsub -q [queue] -pe [pe] [np] -cwd -b y  \
  "/opt/sun/sunstudio12.1/prod/bin/collect -p high -M CT8.1 -S on -A on -L none \
                   mpirun -np 4 -- /path/to/file/test"

$ cat script_OMP_ONLY.sh
#!/bin/bash

qsub -q [queue]  [pe] [np] -v OMP_NUM_THREADS=8 -cwd -b y  \
  "/opt/sun/sunstudio12.1/prod/bin/collect  -L none \
                -p high -S on -A on /path/to/file/test"

$ cat scriptOMP.sh
```

```
#!/bin/bash

qsub -q [queue] [pe] [np] -v OMP_NUM_THREADS=8 -cwd -b y  \
  "/opt/sun/sunstudio12.1/prod/bin/collect -p high -M CT8.1 -S on -A on -L none \
                    mpirun -np 4 -- /path/to/file/test"
```

The first one uses MPI tracing for testing MPI programs, the second one is intended for Openmp programs (that's why it sets the OMP_NUM_THREADS variable) and the last one is for hybird programs (they use both MPI and Openmp).

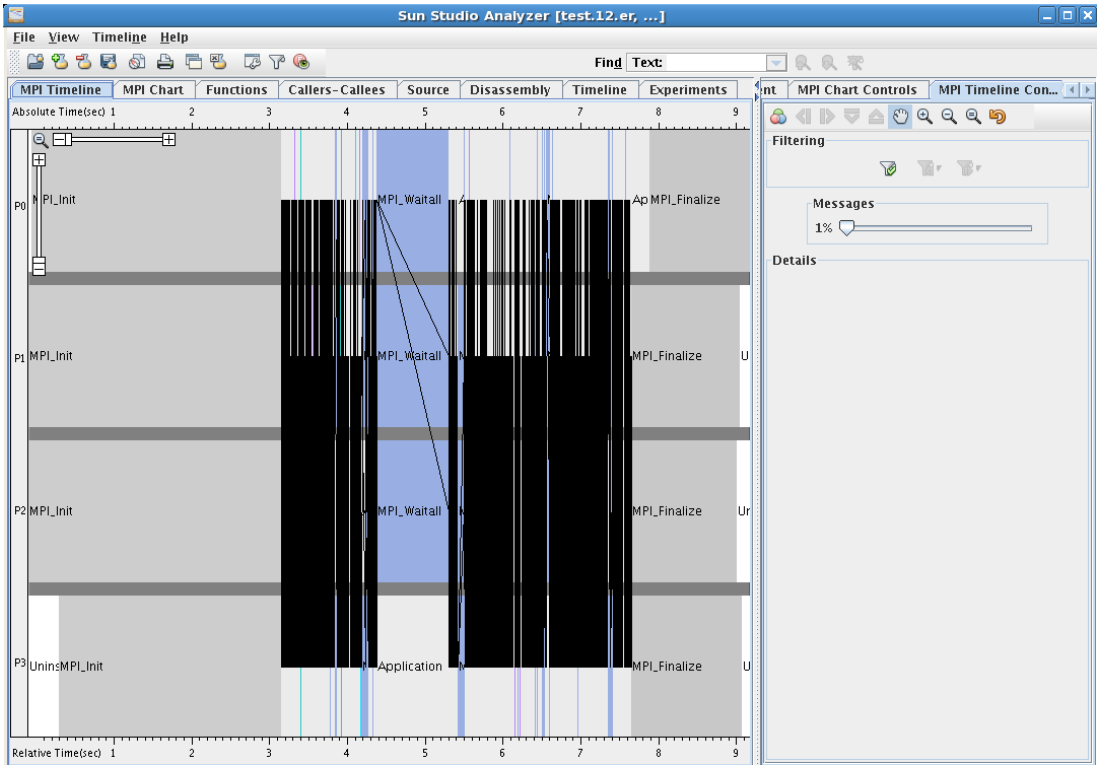
Some of the parameters used are explained in the following. You can find more information in the manual (`$ man collect`).

- -M CT8.1; Specify collection of an MPI experiment. CT8.1 is the MPI version installed.
- -L size; Limit the amount of profiling and tracing data recorded to size megabytes. None means no limit.
- -S interval; Collect periodic samples at the interval specified (in seconds). **on** defaults to 1 second.
- -A option; Control whether or not load objects used by the target process should be archived or copied into the recorded experiment. **on** archive load objects into the experiment.
- -p option; Collect clock-based profiling data. **high** turn on clock-based profiling with the default profiling interval of approximately 1 millisecond.

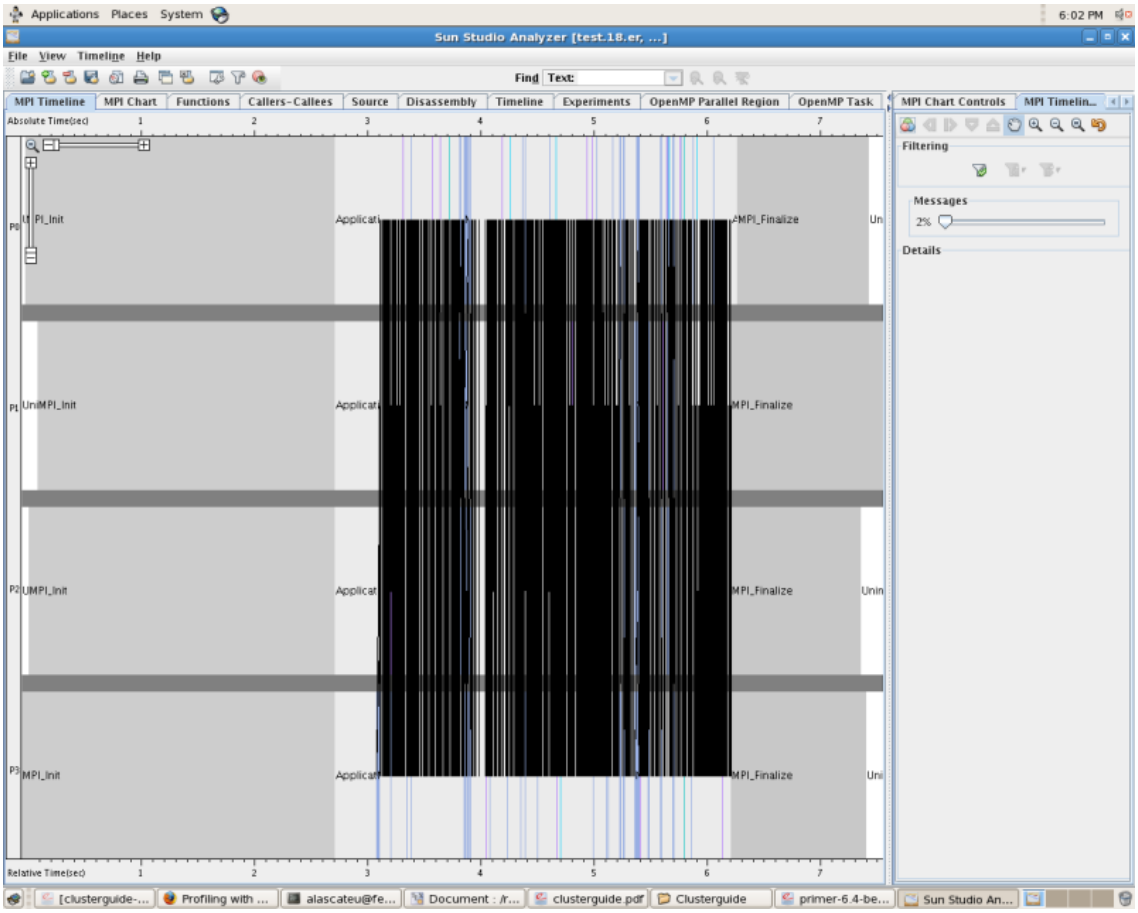
8.1.2 Viewing the experiment results

To view the results, open the analyzer and go to *File - Open Experiment* and select the experiment you want to view. A very good tutorial for analyzing the data can be found here. *Performance Analyzer MPI Tutorial* is a good place to start.

The following screenshots where taken from the analysis of the programs presented in the **Parallelization section** under **Hybrid Parallelization**.



MPI only version



Hybrid (MPI + Openmp) version

8.2 Intel MPI benchmark

The Intel MPI benchmark - **IMB** is a tool for evaluating the performance of a MPI installation. The idea of IMB is to provide a concise set of elementary MPI benchmark kernels. With one executable, all of the supported benchmarks, or a subset specified by the command line, can be run. The rules, such as time measurement (including a repetitive call of the kernels for better clock synchronization), message lengths, selection of communicators to run a particular benchmark (inside the group of all started processes) are program parameters.

8.2.1 Installing and running IMB

The first step is to get the package from here. Unpack the archive. Make sure you have the Intel compiler module loaded and a working OpenMPI installation. Go to the */imb/src* directory. There are three benchmarks available: IMB-MPI1, IMB-IO, IMB-EXT. You can build them separately with:

```
$ make <benchmark name>
```

or all at once with:

```
$ make all
```

Now you can run any of the three benchmarks using:

```
$ mpirun -np <nr_of_procs> IMB-xxx
```

NOTE: there are useful documents in the /imb/doc directory detailing the benchmarks.

8.2.2 Submitting a benchmark to a queue

You can also submit a benchmark to run on a queue. The following two scripts are examples:

```
$ cat submit.sh
#!/bin/bash
qsub -q [queue] -pe [pe] [np] -cwd -b n [script_with_the_run_cmd]

$ cat script.sh
#!/bin/bash
mpirun -np [np] IMB-xxx
```

To submit you just have to run:

```
$ ./submit.sh
```

After running the IMB-MPI1 benchmark on a queue with 24 processes the following result was obtained (only parts are shown):

```
#---------------------------------------------------
#    Intel (R) MPI Benchmark Suite V3.2, MPI-1 part
#---------------------------------------------------
# Date                  : Thu Jul 23 16:37:23 2009
# Machine               : x86_64
# System                : Linux
# Release               : 2.6.18-128.1.1.el5
```

```
# Version               : #1 SMP Tue Feb 10 11:36:29 EST 2009
# MPI Version           : 2.1
# MPI Thread Environment: MPI_THREAD_SINGLE

# Calling sequence was:

# IMB-MPI1

# Minimum message length in bytes:   0
# Maximum message length in bytes:   4194304
#
# MPI_Datatype                   :   MPI_BYTE
# MPI_Datatype for reductions    :   MPI_FLOAT
# MPI_Op                         :   MPI_SUM
#
#

# List of Benchmarks to run:

# PingPong
# PingPing
# Sendrecv
# Exchange
# Allreduce
# Reduce
# Reduce_scatter
# Allgather
# Allgatherv
# Gather
# Gatherv
# Scatter
# Scatterv
# Alltoall
# Alltoallv
# Bcast
# Barrier

[...]

#---------------------------------------------------------------------------
# Benchmarking Exchange
# #processes = 2
# ( 22 additional processes waiting in MPI_Barrier)
#---------------------------------------------------------------------------
      #bytes #repetitions  t_min[usec]  t_max[usec]  t_avg[usec]   Mbytes/sec
      524288           80      1599.30      1599.31      1599.31      1250.54
```

```
     1048576           40      3743.45      3743.48      3743.46      1068.53
     2097152           20      7290.26      7290.30      7290.28      1097.35
     4194304           10     15406.39     15406.70     15406.55      1038.51

[...]

#---------------------------------------------------------------------------
# Benchmarking Exchange
# #processes = 24
#---------------------------------------------------------------------------
      #bytes #repetitions  t_min[usec]  t_max[usec]  t_avg[usec]   Mbytes/sec
           0         1000        75.89        76.31        76.07         0.00
           1         1000        67.73        68.26        68.00         0.06
           2         1000        68.47        69.29        68.90         0.11
           4         1000        69.23        69.88        69.57         0.22
           8         1000        68.20        68.91        68.55         0.44
      262144          160     19272.77     20713.69     20165.05        48.28
      524288           80     63144.46     65858.79     63997.79        30.37
     1048576           40     83868.32     89965.37     87337.56        44.46
     2097152           20     91448.50    106147.55     99928.08        75.37
     4194304           10    121632.81    192385.91    161055.82        83.17

[...]

#---------------------------------------------------------------
# Benchmarking Alltoallv
# #processes = 8
# ( 16 additional processes waiting in MPI_Barrier)
#---------------------------------------------------------------
      #bytes #repetitions  t_min[usec]  t_max[usec]  t_avg[usec]
           0         1000         0.10         0.10         0.10
           1         1000        18.49        18.50        18.49
           2         1000        18.50        18.52        18.51
           4         1000        18.47        18.48        18.47
           8         1000        18.40        18.40        18.40
          16         1000        18.42        18.43        18.43
          32         1000        18.89        18.90        18.89
          68         1000       601.29       601.36       601.33
       65536          640      1284.44      1284.71      1284.57
      131072          320      3936.76      3937.16      3937.01
      262144          160     10745.08     10746.09     10745.83
      524288           80     22101.26     22103.33     22102.58
     1048576           40     44044.33     44056.68     44052.76
     2097152           20     88028.00     88041.70     88037.15
     4194304           10    175437.78    175766.59    175671.63
```

```
[...]

#---------------------------------------------------------------
# Benchmarking Alltoallv
# #processes = 24
#---------------------------------------------------------------
      #bytes #repetitions  t_min[usec]  t_max[usec]  t_avg[usec]
           0         1000         0.18         0.22         0.18
           1         1000       891.94       892.74       892.58
           2         1000       891.63       892.46       892.28
           4         1000       879.25       880.09       879.94
           8         1000       898.30       899.29       899.05
      262144           15    923459.34    950393.47    938204.26
      524288           10   1176375.79   1248858.31   1207359.81
     1048576            6   1787152.85   1906829.99   1858522.38
     2097152            4   3093715.25   3312132.72   3203840.16
     4194304            2   5398282.53   5869063.97   5702468.73
```

As you can see, if you specify 24 processes then the benchmark will also run the 2,4,8,16 tests. You can fix the minimum number of processes to use with:

```
$ mpirun [...] <benchmark> -npmin <minimum_number_of_procs>
```

NOTE: Other useful control commands can be found in the Users Guide (/doc directory) under section 5.

9 Application Software and Program Libraries

9.1 Automatically Tuned Linear Algebra Software (ATLAS)

The BLAS (Basic Linear Algebra Subprograms) are routines that provide standard building blocks for performing basic vector and matrix operations. The Level 1 BLAS perform scalar, vector and vector-vector operations, the Level 2 BLAS perform matrix-vector operations, and the Level 3 BLAS perform matrix-matrix operations. Because the BLAS are efficient, portable, and widely available, they are commonly used in the development of high quality linear algebra software, LAPACK for example.

The ATLAS (Automatically Tuned Linear Algebra Software) project is an ongoing research effort that provides C and Fortran77 interfaces to a portably efficient BLAS implementation, as well as a few routines from /htmladdnormallinkLAPACKhttp://math-atlas.sourceforge.net/

9.1.1 Using ATLAS

To initialize the environment use:

- module load blas/atlas-9.11_sunstudio12.1 (compiled with gcc)
- module load blas/atlas-9.11_sunstudio12.1 (compiled with sun)

To use level1-3 functions available in atlas see the prototypes functions in cblas.h and use them in your examples. For compiling you should specifiy the necessary libraries files.

Example:

- gcc -lcblas -latlas example.c
- cc -lcblas -latlas example.c

It is recommended the version compiled with gcc.It is almost never a good idea to change the C compiler used to compile ATLAS's generated double precision (real and complex) and C compiler used to compile ATLAS's generated single precision (real and complex), and it is only very rarely a good idea to change the C compiler used to compile all other double precision routines and C compiler used to compile all other single precision routines . For ATLAS 3.8.0, all architectural defaults are set using gcc 4.2 only (the one exception is MIPS/IRIX, where SGI's compiler is used). In most cases, switching these compilers will get you worse performance and accuracy, even when you are absolutely sure it is a better compiler and flag combination!

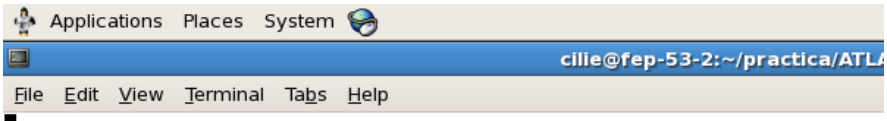
9.1.2 Performance

```
./xsl3blastst


---------------------------------- GEMM -----------------------------------
TST# A B    M    N    K ALPHA  LDA  LDB  BETA  LDC  TIME MFLOP SpUp  TEST
==== = = ==== ==== ==== ===== ==== ==== ===== ==== ===== ===== ==== =====
   0 N N  100  100  100   1.0 1000 1000   1.0 1000  0.01 250.0 1.00 -----
   0 N N  100  100  100   1.0 1000 1000   1.0 1000  0.00   0.0 0.00 PASS
   1 N N  200  200  200   1.0 1000 1000   1.0 1000  0.02 800.0 1.00 -----
   1 N N  200  200  200   1.0 1000 1000   1.0 1000  0.01 1333.2 1.67 PASS
   2 N N  300  300  300   1.0 1000 1000   1.0 1000  0.07 749.9 1.00 -----
   2 N N  300  300  300   1.0 1000 1000   1.0 1000  0.00   0.0 0.00 PASS
   3 N N  400  400  400   1.0 1000 1000   1.0 1000  0.16 800.0 1.00 -----
   3 N N  400  400  400   1.0 1000 1000   1.0 1000  0.02 6399.7 8.00 PASS
   4 N N  500  500  500   1.0 1000 1000   1.0 1000  0.33 762.1 1.00 -----
   4 N N  500  500  500   1.0 1000 1000   1.0 1000  0.03 8928.3 11.71 PASS
   5 N N  600  600  600   1.0 1000 1000   1.0 1000  0.65 666.6 1.00 -----
   5 N N  600  600  600   1.0 1000 1000   1.0 1000  0.06 7199.5 10.80 PASS
   6 N N  700  700  700   1.0 1000 1000   1.0 1000  1.03 664.7 1.00 -----
   6 N N  700  700  700   1.0 1000 1000   1.0 1000  0.08 9025.7 13.58 PASS
   7 N N  800  800  800   1.0 1000 1000   1.0 1000  1.72 596.7 1.00 -----
   7 N N  800  800  800   1.0 1000 1000   1.0 1000  0.14 7110.7 11.92 PASS
   8 N N  900  900  900   1.0 1000 1000   1.0 1000  2.18 667.5 1.00 -----
   8 N N  900  900  900   1.0 1000 1000   1.0 1000  0.19 7754.8 11.62 PASS
   9 N N 1000 1000 1000   1.0 1000 1000   1.0 1000  2.90 688.7 1.00 -----
   9 N N 1000 1000 1000   1.0 1000 1000   1.0 1000  0.26 7691.8 11.17 PASS

10 tests run, 10 passed

~
~
```




```
./xdl3blastst


---------------------------------- GEMM -----------------------------------
TST# A B    M    N    K ALPHA  LDA  LDB  BETA  LDC  TIME MFLOP SpUp  TEST
==== = = ==== ==== ==== ===== ==== ==== ===== ==== ===== ===== ==== =====
   0 N N  100  100  100   1.0 1000 1000   1.0 1000  0.01 166.7 1.00 -----
   0 N N  100  100  100   1.0 1000 1000   1.0 1000  0.00   0.0 0.00 PASS
   1 N N  200  200  200   1.0 1000 1000   1.0 1000  0.04 400.0 1.00 -----
   1 N N  200  200  200   1.0 1000 1000   1.0 1000  0.01 2000.0 5.00 PASS
   2 N N  300  300  300   1.0 1000 1000   1.0 1000  0.09 586.9 1.00 -----
   2 N N  300  300  300   1.0 1000 1000   1.0 1000  0.01 6750.0 11.50 PASS
   3 N N  400  400  400   1.0 1000 1000   1.0 1000  0.18 711.1 1.00 -----
   3 N N  400  400  400   1.0 1000 1000   1.0 1000  0.03 4571.1 6.43 PASS
   4 N N  500  500  500   1.0 1000 1000   1.0 1000  0.39 637.7 1.00 -----
   4 N N  500  500  500   1.0 1000 1000   1.0 1000  0.07 3676.3 5.76 PASS
   5 N N  600  600  600   1.0 1000 1000   1.0 1000  0.68 635.3 1.00 -----
   5 N N  600  600  600   1.0 1000 1000   1.0 1000  0.10 4319.7 6.80 PASS
   6 N N  700  700  700   1.0 1000 1000   1.0 1000  1.14 601.7 1.00 -----
   6 N N  700  700  700   1.0 1000 1000   1.0 1000  0.22 3118.0 5.18 PASS
   7 N N  800  800  800   1.0 1000 1000   1.0 1000  2.06 497.1 1.00 -----
   7 N N  800  800  800   1.0 1000 1000   1.0 1000  0.29 3555.3 7.15 PASS
   8 N N  900  900  900   1.0 1000 1000   1.0 1000  2.26 646.2 1.00 -----
   8 N N  900  900  900   1.0 1000 1000   1.0 1000  0.32 4613.6 7.14 PASS
   9 N N 1000 1000 1000   1.0 1000 1000   1.0 1000  2.97 672.9 1.00 -----
   9 N N 1000 1000 1000   1.0 1000 1000   1.0 1000  0.47 4273.2 6.35 PASS

10 tests run, 10 passed
```

```
./xsl3blastst


---------------------------------- GEMM -----------------------------------
TST# A B    M    N    K ALPHA  LDA  LDB  BETA  LDC  TIME MFLOP SpUp  TEST
==== = = ==== ==== ==== ===== ==== ==== ===== ==== ===== ===== ==== =====
   0 N N  100  100  100   1.0 1000 1000   1.0 1000  0.01 250.0 1.00 -----
   0 N N  100  100  100   1.0 1000 1000   1.0 1000  0.00   0.0 0.00 PASS
   1 N N  200  200  200   1.0 1000 1000   1.0 1000  0.02 800.0 1.00 -----
   1 N N  200  200  200   1.0 1000 1000   1.0 1000  0.01 1333.2 1.67 PASS
   2 N N  300  300  300   1.0 1000 1000   1.0 1000  0.07 749.9 1.00 -----
   2 N N  300  300  300   1.0 1000 1000   1.0 1000  0.00   0.0 0.00 PASS
   3 N N  400  400  400   1.0 1000 1000   1.0 1000  0.16 800.0 1.00 -----
   3 N N  400  400  400   1.0 1000 1000   1.0 1000  0.02 6399.7 8.00 PASS
   4 N N  500  500  500   1.0 1000 1000   1.0 1000  0.33 762.1 1.00 -----
   4 N N  500  500  500   1.0 1000 1000   1.0 1000  0.03 8928.3 11.71 PASS
   5 N N  600  600  600   1.0 1000 1000   1.0 1000  0.65 666.6 1.00 -----
   5 N N  600  600  600   1.0 1000 1000   1.0 1000  0.06 7199.5 10.80 PASS
   6 N N  700  700  700   1.0 1000 1000   1.0 1000  1.03 664.7 1.00 -----
   6 N N  700  700  700   1.0 1000 1000   1.0 1000  0.08 9025.7 13.58 PASS
   7 N N  800  800  800   1.0 1000 1000   1.0 1000  1.72 596.7 1.00 -----
   7 N N  800  800  800   1.0 1000 1000   1.0 1000  0.14 7110.7 11.92 PASS
   8 N N  900  900  900   1.0 1000 1000   1.0 1000  2.18 667.5 1.00 -----
   8 N N  900  900  900   1.0 1000 1000   1.0 1000  0.19 7754.8 11.62 PASS
   9 N N 1000 1000 1000   1.0 1000 1000   1.0 1000  2.90 688.7 1.00 -----
   9 N N 1000 1000 1000   1.0 1000 1000   1.0 1000  0.26 7691.8 11.17 PASS

10 tests run, 10 passed

~
~
```

```
./xzl3blastst


---------------------------------- GEMM -----------------------------------
TST# A B    M    N    K     ALPHA  LDA  LDB      BETA  LDC TIME MFLOP SpUp  TEST
==== = = ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ===== ==== =====
   0 N N  100  100  100  1.0  0.0 1000 1000  1.0  0.0 1000  0.0 666.6 1.00 -----
   0 N N  100  100  100  1.0  0.0 1000 1000  1.0  0.0 1000  0.0   0.0 0.00 PASS
   1 N N  200  200  200  1.0  0.0 1000 1000  1.0  0.0 1000  0.1 640.0 1.00 -----
   1 N N  200  200  200  1.0  0.0 1000 1000  1.0  0.0 1000  0.0 3199.8 5.00 PASS
   2 N N  300  300  300  1.0  0.0 1000 1000  1.0  0.0 1000  0.2 1124.9 1.00 -----
   2 N N  300  300  300  1.0  0.0 1000 1000  1.0  0.0 1000  0.1 2454.4 2.18 PASS
   3 N N  400  400  400  1.0  0.0 1000 1000  1.0  0.0 1000  0.5 1066.6 1.00 -----
   3 N N  400  400  400  1.0  0.0 1000 1000  1.0  0.0 1000  0.1 4266.4 4.00 PASS
   4 N N  500  500  500  1.0  0.0 1000 1000  1.0  0.0 1000  1.1 909.0 1.00 -----
   4 N N  500  500  500  1.0  0.0 1000 1000  1.0  0.0 1000  0.2 4166.4 4.58 PASS
   5 N N  600  600  600  1.0  0.0 1000 1000  1.0  0.0 1000  1.7 1002.3 1.00 -----
   5 N N  600  600  600  1.0  0.0 1000 1000  1.0  0.0 1000  0.4 3999.8 3.99 PASS
   6 N N  700  700  700  1.0  0.0 1000 1000  1.0  0.0 1000  2.4 1135.7 1.00 -----
   6 N N  700  700  700  1.0  0.0 1000 1000  1.0  0.0 1000  0.7 3897.5 3.43 PASS
   7 N N  800  800  800  1.0  0.0 1000 1000  1.0  0.0 1000  4.2 978.9 1.00 -----
   7 N N  800  800  800  1.0  0.0 1000 1000  1.0  0.0 1000  1.0 4031.2 4.12 PASS
   8 N N  900  900  900  1.0  0.0 1000 1000  1.0  0.0 1000  5.7 1016.7 1.00 -----
   8 N N  900  900  900  1.0  0.0 1000 1000  1.0  0.0 1000  1.3 4365.0 4.29 PASS
   9 N N 1000 1000 1000  1.0  0.0 1000 1000  1.0  0.0 1000  6.7 1186.2 1.00 -----
   9 N N 1000 1000 1000  1.0  0.0 1000 1000  1.0  0.0 1000  1.9 4319.4 3.64 PASS

10 tests run, 10 passed
```

9.2 MKL - Intel Math Kernel Library

Intel Math Kernel Library (Intel MKL) is a library of highly optimized, extensively threaded math routines for science, engineering, and financial applications that require maximum performance. Core math functions include BLAS, LAPACK, ScaLAPACK, Sparse Solvers, Fast Fourier Transforms, Vector Math, and more. Offering performance optimizations for current and next-generation Intel processors, it includes improved integration with Microsoft Visual Studio, Eclipse, and XCode. Intel MKL allows for full integration of the Intel Compatibility OpenMP run-time library for greater Windows/Linux cross-platform compatibility.

9.2.1 Using MKL

To initialize the environment use:

- module load blas/mkl-10.2

  To compile with gcc an example that uses mkl functions you should specify necessary libraries. Example:

- gcc -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5 -lpthread -lm example.c

9.2.2 Performance

```
./xcl3blastst


----------------------------------- GEMM -------------------------------------
TST# A B    M    N    K     ALPHA  LDA  LDB      BETA  LDC TIME MFLOP SpUp  TEST
==== = = ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ===== ==== =====
   0 N N  100  100  100  1.0  0.0 1000 1000  1.0  0.0 1000  0.0 195.2 1.00 -----
   0 N N  100  100  100  1.0  0.0 1000 1000  1.0  0.0 1000  0.0 190.5 0.98 PASS
   1 N N  200  200  200  1.0  0.0 1000 1000  1.0  0.0 1000  0.0 64000.0 1.00 ----
   1 N N  200  200  200  1.0  0.0 1000 1000  1.0  0.0 1000  0.1 1016.0 0.02 PASS
   2 N N  300  300  300  1.0  0.0 1000 1000  1.0  0.0 1000  0.0 9001.1 1.00 -----
   2 N N  300  300  300  1.0  0.0 1000 1000  1.0  0.0 1000  0.2 1285.9 0.14 PASS
   3 N N  400  400  400  1.0  0.0 1000 1000  1.0  0.0 1000  0.1 9482.9 1.00 -----
   3 N N  400  400  400  1.0  0.0 1000 1000  1.0  0.0 1000  0.5 1123.0 0.12 PASS
   4 N N  500  500  500  1.0  0.0 1000 1000  1.0  0.0 1000  0.1 11766.5 1.00 ----
   4 N N  500  500  500  1.0  0.0 1000 1000  1.0  0.0 1000  0.8 1297.2 0.11 PASS
   5 N N  600  600  600  1.0  0.0 1000 1000  1.0  0.0 1000  0.1 12001.8 1.00 ----
   5 N N  600  600  600  1.0  0.0 1000 1000  1.0  0.0 1000  1.3 1367.3 0.11 PASS
   6 N N  700  700  700  1.0  0.0 1000 1000  1.0  0.0 1000  0.2 12647.1 1.00 ----
   6 N N  700  700  700  1.0  0.0 1000 1000  1.0  0.0 1000  1.5 1886.2 0.15 PASS
   7 N N  800  800  800  1.0  0.0 1000 1000  1.0  0.0 1000  0.3 13300.7 1.00 ----
   7 N N  800  800  800  1.0  0.0 1000 1000  1.0  0.0 1000  1.3 3089.5 0.23 PASS
   8 N N  900  900  900  1.0  0.0 1000 1000  1.0  0.0 1000  0.4 13347.6 1.00 ----
   8 N N  900  900  900  1.0  0.0 1000 1000  1.0  0.0 1000  1.4 4064.7 0.30 PASS
   9 N N 1000 1000 1000  1.0  0.0 1000 1000  1.0  0.0 1000  0.6 13159.9 1.00 ----
   9 N N 1000 1000 1000  1.0  0.0 1000 1000  1.0  0.0 1000  1.7 4625.0 0.35 PASS

10 tests run, 10 passed

~
~
```

```
./xsl3blastst


---------------------------------- GEMM -----------------------------------
TST# A B    M    N    K ALPHA  LDA  LDB  BETA  LDC  TIME MFLOP SpUp  TEST
==== = = ==== ==== ==== ===== ==== ==== ===== ==== ===== ===== ==== =====
   0 N N  100  100  100   1.0 1000 1000   1.0 1000  0.06  33.9 1.00 -----
   0 N N  100  100  100   1.0 1000 1000   1.0 1000  0.01 153.9 4.54 PASS
   1 N N  200  200  200   1.0 1000 1000   1.0 1000  0.00   0.0 1.00 -----
   1 N N  200  200  200   1.0 1000 1000   1.0 1000  0.02 1000.2 0.00 PASS
   2 N N  300  300  300   1.0 1000 1000   1.0 1000  0.01 9001.5 1.00 -----
   2 N N  300  300  300   1.0 1000 1000   1.0 1000  0.05 1149.1 0.13 PASS
   3 N N  400  400  400   1.0 1000 1000   1.0 1000  0.01 16002.0 1.00 -----
   3 N N  400  400  400   1.0 1000 1000   1.0 1000  0.11 1153.3 0.07 PASS
   4 N N  500  500  500   1.0 1000 1000   1.0 1000  0.03 8065.6 1.00 -----
   4 N N  500  500  500   1.0 1000 1000   1.0 1000  0.21 1207.9 0.15 PASS
   5 N N  600  600  600   1.0 1000 1000   1.0 1000  0.03 13502.1 1.00 -----
   5 N N  600  600  600   1.0 1000 1000   1.0 1000  0.33 1317.3 0.10 PASS
   6 N N  700  700  700   1.0 1000 1000   1.0 1000  0.06 11829.4 1.00 -----
   6 N N  700  700  700   1.0 1000 1000   1.0 1000  0.52 1309.4 0.11 PASS
   7 N N  800  800  800   1.0 1000 1000   1.0 1000  0.08 12801.9 1.00 -----
   7 N N  800  800  800   1.0 1000 1000   1.0 1000  0.81 1269.1 0.10 PASS
   8 N N  900  900  900   1.0 1000 1000   1.0 1000  0.11 12791.4 1.00 -----
   8 N N  900  900  900   1.0 1000 1000   1.0 1000  1.06 1380.9 0.11 PASS
   9 N N 1000 1000 1000   1.0 1000 1000   1.0 1000  0.14 13891.0 1.00 -----
   9 N N 1000 1000 1000   1.0 1000 1000   1.0 1000  0.69 2915.9 0.21 PASS

10 tests run, 10 passed

~


./xdl3blastst


---------------------------------- GEMM -----------------------------------
TST# A B    M    N    K ALPHA  LDA  LDB  BETA  LDC  TIME MFLOP SpUp  TEST
==== = = ==== ==== ==== ===== ==== ==== ===== ==== ===== ===== ==== =====
   0 N N  100  100  100   1.0 1000 1000   1.0 1000  0.04  44.5 1.00 -----
   0 N N  100  100  100   1.0 1000 1000   1.0 1000  0.03  76.9 1.73 PASS
   1 N N  200  200  200   1.0 1000 1000   1.0 1000  0.00   0.0 1.00 -----
   1 N N  200  200  200   1.0 1000 1000   1.0 1000  0.03 500.1 0.00 PASS
   2 N N  300  300  300   1.0 1000 1000   1.0 1000  0.02 3375.6 1.00 -----
   2 N N  300  300  300   1.0 1000 1000   1.0 1000  0.08 675.1 0.20 PASS
   3 N N  400  400  400   1.0 1000 1000   1.0 1000  0.02 5334.0 1.00 -----
   3 N N  400  400  400   1.0 1000 1000   1.0 1000  0.19 670.3 0.13 PASS
   4 N N  500  500  500   1.0 1000 1000   1.0 1000  0.05 4902.7 1.00 -----
   4 N N  500  500  500   1.0 1000 1000   1.0 1000  0.19 1330.0 0.27 PASS
   5 N N  600  600  600   1.0 1000 1000   1.0 1000  0.07 6085.4 1.00 -----
   5 N N  600  600  600   1.0 1000 1000   1.0 1000  0.59 731.1 0.12 PASS
   6 N N  700  700  700   1.0 1000 1000   1.0 1000  0.11 6181.1 1.00 -----
   6 N N  700  700  700   1.0 1000 1000   1.0 1000  0.95 723.7 0.12 PASS
   7 N N  800  800  800   1.0 1000 1000   1.0 1000  0.17 6132.7 1.00 -----
   7 N N  800  800  800   1.0 1000 1000   1.0 1000  1.04 980.1 0.16 PASS
   8 N N  900  900  900   1.0 1000 1000   1.0 1000  0.22 6509.9 1.00 -----
   8 N N  900  900  900   1.0 1000 1000   1.0 1000  0.41 3591.7 0.55 PASS
   9 N N 1000 1000 1000   1.0 1000 1000   1.0 1000  0.30 6558.4 1.00 -----
   9 N N 1000 1000 1000   1.0 1000 1000   1.0 1000  0.54 3697.4 0.56 PASS

10 tests run, 10 passed
```

```
./xzl3blastst


---------------------------------- GEMM -----------------------------------
TST# A B    M    N    K     ALPHA  LDA  LDB      BETA  LDC TIME MFLOP SpUp  TEST
==== = = ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ===== ==== =====
   0 N N  100  100  100  1.0  0.0 1000 1000  1.0  0.0 1000  0.0 258.1 1.00 -----
   0 N N  100  100  100  1.0  0.0 1000 1000  1.0  0.0 1000  0.0 186.1 0.72 PASS
   1 N N  200  200  200  1.0  0.0 1000 1000  1.0  0.0 1000  0.0 7112.7 1.00 -----
   1 N N  200  200  200  1.0  0.0 1000 1000  1.0  0.0 1000  0.1 581.9 0.08 PASS
   2 N N  300  300  300  1.0  0.0 1000 1000  1.0  0.0 1000  0.0 5400.8 1.00 -----
   2 N N  300  300  300  1.0  0.0 1000 1000  1.0  0.0 1000  0.3 660.7 0.12 PASS
   3 N N  400  400  400  1.0  0.0 1000 1000  1.0  0.0 1000  0.1 6096.2 1.00 -----
   3 N N  400  400  400  1.0  0.0 1000 1000  1.0  0.0 1000  0.1 3737.8 0.61 PASS
   4 N N  500  500  500  1.0  0.0 1000 1000  1.0  0.0 1000  0.2 6580.0 1.00 -----
   4 N N  500  500  500  1.0  0.0 1000 1000  1.0  0.0 1000  0.4 2793.7 0.42 PASS
   5 N N  600  600  600  1.0  0.0 1000 1000  1.0  0.0 1000  0.3 6546.4 1.00 -----
   5 N N  600  600  600  1.0  0.0 1000 1000  1.0  0.0 1000  1.1 1597.3 0.24 PASS
   6 N N  700  700  700  1.0  0.0 1000 1000  1.0  0.0 1000  0.4 6534.3 1.00 -----
   6 N N  700  700  700  1.0  0.0 1000 1000  1.0  0.0 1000  0.7 3838.3 0.59 PASS
   7 N N  800  800  800  1.0  0.0 1000 1000  1.0  0.0 1000  0.6 7112.2 1.00 -----
   7 N N  800  800  800  1.0  0.0 1000 1000  1.0  0.0 1000  1.5 2643.0 0.37 PASS
   8 N N  900  900  900  1.0  0.0 1000 1000  1.0  0.0 1000  0.8 7113.3 1.00 -----
   8 N N  900  900  900  1.0  0.0 1000 1000  1.0  0.0 1000  1.7 3405.1 0.48 PASS
   9 N N 1000 1000 1000  1.0  0.0 1000 1000  1.0  0.0 1000  1.1 7080.7 1.00 -----
   9 N N 1000 1000 1000  1.0  0.0 1000 1000  1.0  0.0 1000  2.2 3610.7 0.51 PASS

10 tests run, 10 passed

~
```

9.3 ATLAS vs MKL - level 1,2,3 functions

The BLAS functions were tested for all 3 levels and the results are shown only for level 3. To summarize the performance tests, it would be that ATLAS loses for Level 1 BLAS, tends to be beat MKL for Level 2 BLAS, and varies between quite a bit slower and quite a bit faster than MKL for Level 3 BLAS, depending on problem size and data type.

ATLAS's present Level 1 gets its optimization mainly from the compiler. This gives MKL two huge advantages: MKL can use the SSE prefetch instructions to speed up pretty much all Level 1 ops. The second advantage is in how ABS() is done. ABS() *should* be a 1-cycle operation, since you can just mask off the sign bit. However, you cannot standardly do bit operation on floats in ANSI C, so ATLAS has to use an if-type construct instead. This spells absolute doom for the performance of NRM2, ASUM and AMAX.

For the Level 2 and 3, ATLAS has it's usual advantage of leveraging basic kernels to the maximum. This means that all Level 3 ops follow the performance of GEMM, and Level 2 ops follow GER or GEMV. MKL has the usual disadvantage of optimizing all these routines seperately, leading to widely varying performance.

References

[1] The RWTH HPC Primer, http://www.rz.rwth-aachen.de/go/id/pil/lang/en

[2] Wikipedia page on SPARC processors, http://www.rz.rwth-aachen.de/go/id/pil/lang/en

[3] Sunsolve page on the Sun Enterprise 220R, http://sunsolve.sun.com/handbook_pub/validateUser.do?target=Systems/E220R/E220R

[4] Sunsolve page on the Sun Enterprise 420R, http://sunsolve.sun.com/handbook_pub/svalidateUser.do?target=Systems/E420R/E420R