HP-SEE

Hands-on session: Overview and usage of PARADOX software stack

www.hp-see.eu

Vladimir Slavnic
Institute of Physics Belgrade, Serbia
slavnic@ipb.ac.rs

The HP-SEE initiative is co-funded by the European Commission under the FP7 Research Infrastructures contract no. 261499

Overview

- Overview of PARADOX software stack
  - Compilers
  - Performance tools: debuggers and profilers
  - Libraries
  - Application software
- Using compilers
  - Useful flags
- Using numerical libraries
  - Intel MKL libraries (linking guide)
- Debugging simple codes using GDB
- Finding memory leaks with Valgrind
- Performing simple profiling using gprof

Compilers

- PARADOX cluster supports development of software in C/C++ and Fortran programming languages through different toolchains:
  - GNU Compiler Collection
  - Intel Compilers
  - Portland Group Compilers
  - IBM XL Compilers (available on tPARADOX)

GNU Compilers Collection

- GNU Compiler Collection or GCC is standard compiler on most modern Unix-like computer operating systems
- Includes front ends for compiling:
  - C
  - C++
  - Objective-C
  - Fortran
  - Java
  - Ada
  - Go
- Most important for HPC are the C/C++ and Fortran languages as most applications designed for such platforms are written in one or more of these language specifications

GNU Compilers Collection

- Ported to a wide variety of processor architectures
- Widely deployed as a tool in commercial, proprietary and closed source software development environments
- Performs well on a variety of native and cross targets
- Available for most embedded platforms

- http://gcc.gnu.org/

Intel Compilers

- Intel's suite of compilers with front ends for C, C++, and Fortran:
  - C, C++ - icc, icpc
  - FORTRAN – ifort
- Compilers are distributed through different package suites:
  - Intel Parallel Studio, Intel Parallel Studio XE, the Intel C++ Composer package, the Intel C++ Composer XE package, the Intel Composer XE package and the Intel Cluster Studio
- Compilers available for GNU/Linux, Mac OS and Microsoft Windows systems

Intel Compilers

- Intel tunes its compilers to optimize for its hardware platforms therefore producing highly optimized code for Intel processors
- Different optimization features and multithreading capabilities
  - Automatic vectorizer that can generate SSE, SSE2, SSE3, SSSE3, SSE4 and AVX SIMD instructions,
  - Supports both OpenMP 3.1 and automatic parallelization for symmetric multiprocessing
- http://software.intel.com/en-us/articles/intel-compilers/

PGI Compilers

- The Portland Group, Inc or PGI
- Set of commercially available Fortran, C and C++ compilers for high-performance computing system
- Incorporate:
  - global optimization
  - vectorization
  - software pipelining
  - shared-memory parallelization
  - capabilities targeting both Intel and AMD processors

PGI Compilers

HP-SEE
High-Performance Computing Infrastructure
for South East Europe's Research Communities

- PGI supports the following high-level languages:
  - Fortran 77, Fortran 95, Fortran 2003
  - High Performance Fortran (HPF)
  - ANSI C99
  - ANSI/ISO C++
- Recently PGI has been involved in the expansion of the use of GPGPUs for high-performance computing, developing CUDA Fortran with NVIDIA Corporation and PGI Accelerator Fortran and C compilers which use programming directives
- http://www.pgroup.com/

IBM XL Compilers

- Proprietary suite of compilers from IBM developed for its platforms:
  - POWER
  - Cell CPUs
  - Blue Gene systems

- Well established in HPC
- Support for C, C++ and FORTRAN
- Producing highly optimizing code for IBM CPUs
- Access to highly-tuned libraries, optimization utilities and development utilities
- Recommended to use with IBM hardware

Using compilers

- GCC compiler toolchain is, by default, available on the users path:
  - $ gcc
  - $ gfortran
  - $ f77
- All Intel tools are installed at /opt/intel directory
- In order to use them user should source provided scripts, for example:

```
$ source /opt/intel/composerxe/bin/compilervars.sh intel64
```

- Intel tools are not present at PARADOX worker nodes (only at ui.ipb.ac.rs) so linking with static Intel libraries (static compilation) is necessary when executables will run on batch system and they are using Intel libraries (for example, libiomp5.so used with OpenMP executables)

Important flags (1)

- -o exe_file : names the executable exe_file
- -c : generates the correspondent object file. Does not create an executable.
- -g : compiles in a debugging mode
- -Idir_name : specifies the path where include files are located
- -Ldir_name : specifies the path where libraries are located
- -l<lib_name> : asks to link against the lib<libname>
- -pg: profiling with gprof (needed at the compilation)
- -Wall : show all reasonable warnings
- -static-intel : link Intel provided libraries statically (ICC)

Important flags (2)

- Optimizations:
  - -O0, -O1, -O2, -O3: optimization levels (intel default : -O2)
  - Intel specific:
    - -ip, -ipo: inter-procedural optimizations (mono and multi files)
    - -fast: default high optimization level (-O3 -ipo -static)
    - -opt_report: generates a report which describes the optimization in stderr (-O3 required)
    - -x<code> generate specialized code to run exclusively on processors (SSE4.1, SSE4.2, AVX…)

- Compile with OpenMP:
  - gcc: -fopenmp
  - icc: -openmp

Simple optimization example (1)

- Copy source code test.c from /tmp/simple_optimization_example/ directory on ui.ipb.ac.rs to /nfs/username directory:
  - $ cp /tmp/simple_optimization_example/test.c /nfs/demoXXX
- Make a simple submit script for it (or use one from your serial job)

Simple optimization example (2)

- Try to compile it with different flags or compilers (gcc and icc with O0, O2, O3) and measure execution times:

```
$ gcc –o test -Wall -O0 test.c -lm
$ time ./test
real    0m13.388s
user    0m13.370s
sys     0m0.010s

$ gcc –o test -Wall -O1 test.c -lm
$ time ./test
real    0m10.030s
user    0m10.030s
sys     0m0.000s
```

Performance tools

- Debuggers:
  - TotalView Debugger
  - GDB
  - PGI pgdbg
  - Intel Debugger
  - IBM Parallel Debugger (PDB) on tPARADOX

- Profilers:
  - gprof
  - PGI pgprof
  - Intel Vtune
  - Valgrind

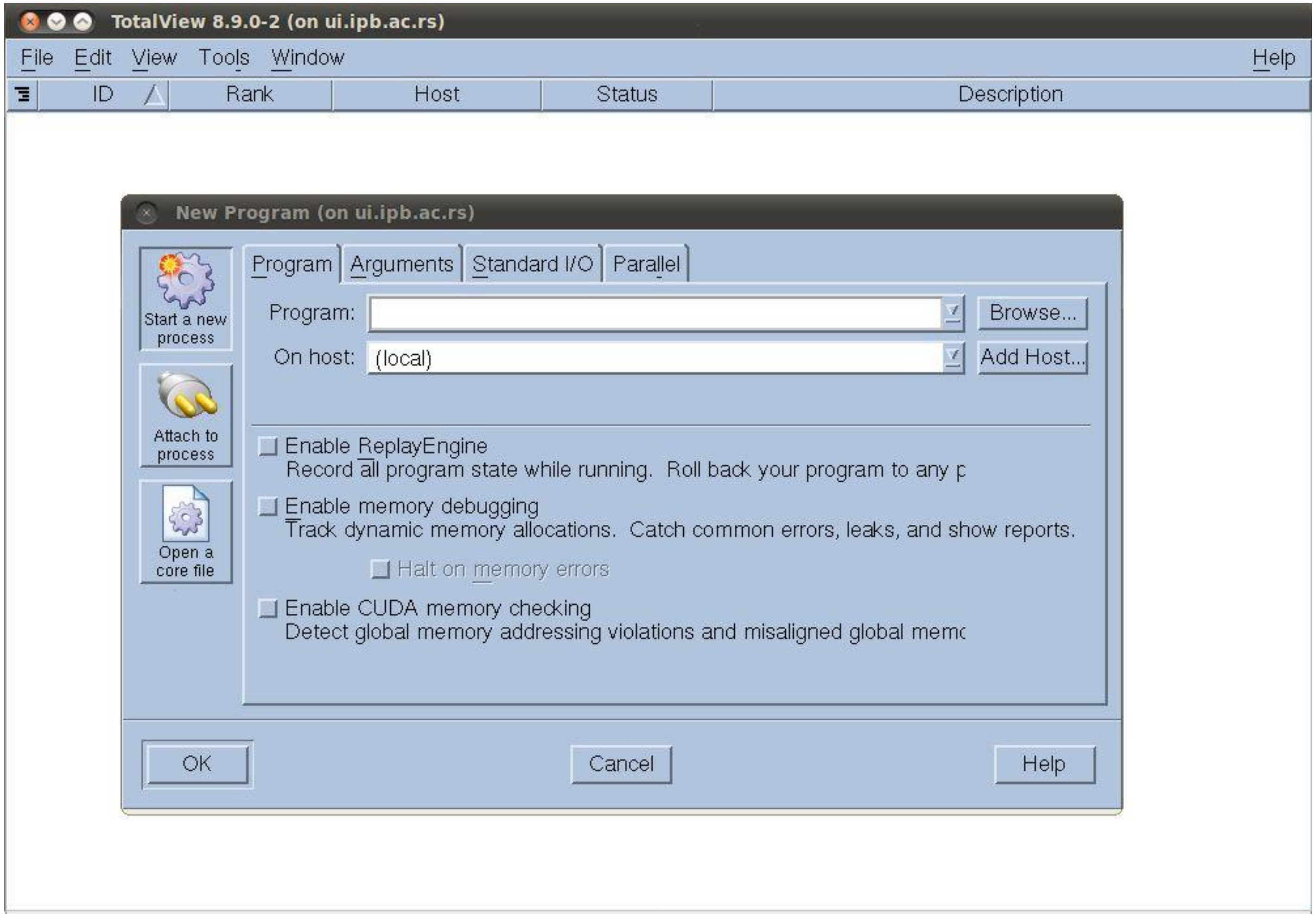
TotalView Debugger (1)

- TotalView is a GUI-based source code defect analysis tool
- It allows you to debug one or many processes and/or threads
- Gives control over program execution, from basic debugging operations like stepping through code to sophisticated techniques that are becoming more commonplace in the high performance computing world such as Graphic Processor Support (debug CUDA)
- Especially designed for use with complex, multi-process and/or multi-threaded applications

TotalView Debugger (2)

- Support for threads, OpenMP, MPI, or GPUs.
- Provides analytical displays of the state of your running program
- TotalView works with C, C++ and Fortran applications written for Linux (including the Blue Gene platforms), UNIX and Mac OS X platforms
- One of the most popular HPC debuggers

TotalView Debugger (3)


TotalView 8.9.0-2 (on ui.ipb.ac.rs)

New Program (on ui.ipb.ac.rs)

Program:

On host: (local)

☐ Enable ReplayEngine
Record all program state while running. Roll back your program to any p

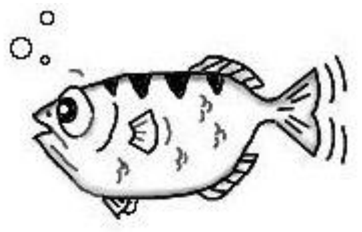
☐ Enable memory debugging
Track dynamic memory allocations. Catch common errors, leaks, and show reports.

☐ Halt on memory errors

☐ Enable CUDA memory checking
Detect global memory addressing violations and misaligned global memc

http://www.roguewave.com/products/totalview-family/totalview.aspx

GDB (1)



- GDB, the GNU Project debugger
- Allows user to see:
  - What is going on `inside' another program while it executes
  - What another program was doing at the moment it crashed

- GDB can do four main kinds of things (plus other things in support of these) to help user catch bugs :
  - Start program, specifying anything that might affect its behavior
  - Make program stop on specified conditions
  - Examine what has happened, when program has stopped
  - Change things in program, so user can experiment with correcting the effects of one bug and go on to learn about another

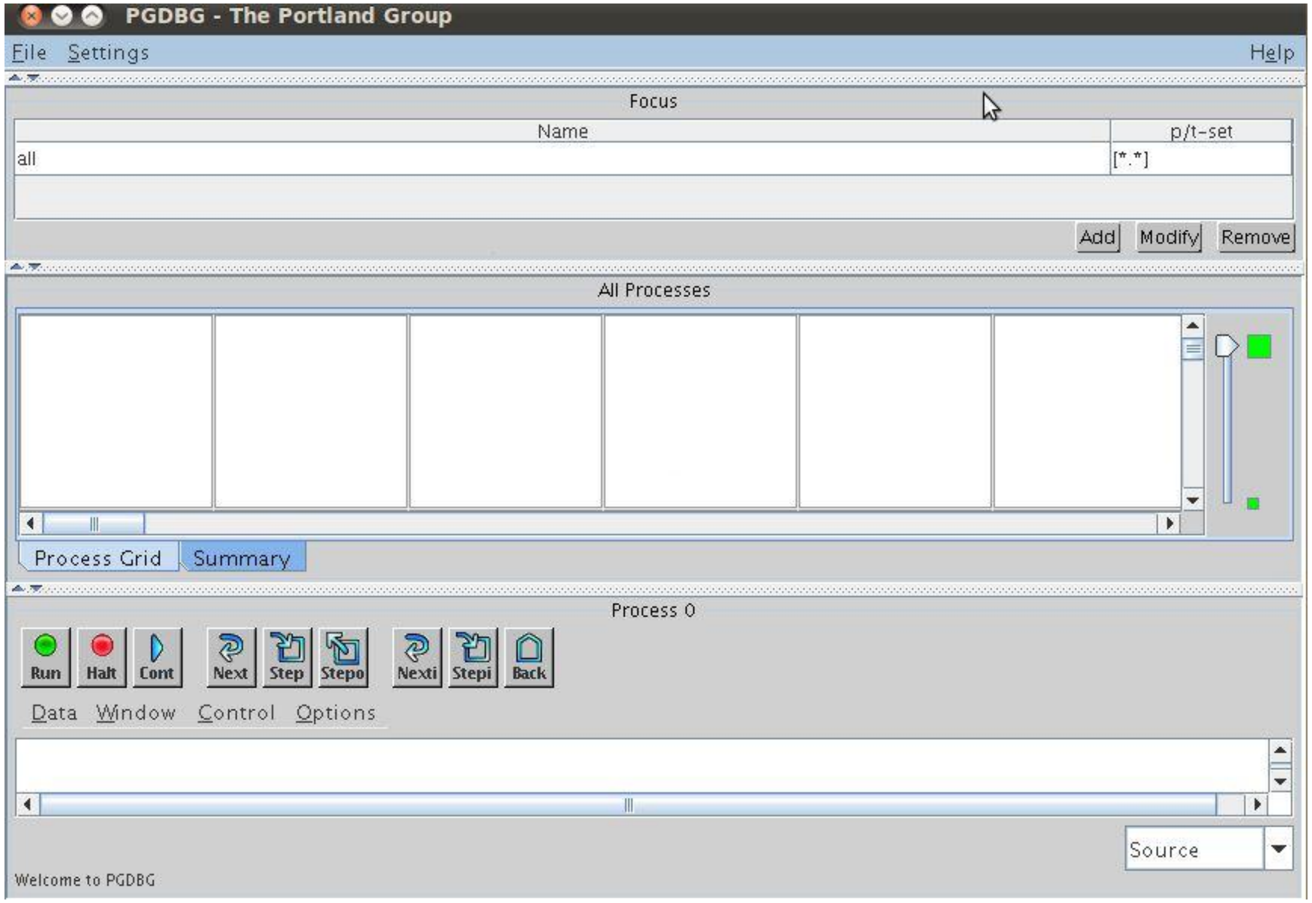
GDB (2)

- The program being debugged can be written in:
  - Ada
  - C
  - C++
  - Objective-C
  - Pascal
  - and many other languages…
- Programs might be executing on the same machine as GDB (native) or on another machine (remote)
- GDB can run on most popular UNIX and Microsoft Windows variants
- http://www.gnu.org/s/gdb/

PBDBG (1)

- PGDBG is a symbolic debugger from PGI for Fortran, C, C++ and assembly language programs
- It provides debugger features, such as execution control using breakpoints, single-stepping, and examination and modification of application variables, memory locations, and registers
- PGDBG supports debugging of certain types of parallel applications:
  - Multi-threaded and OpenMP applications
  - MPI applications
  - Hybrid applications, which use multiple threads or OpenMP as well as multiple MPI processes on Linux clusters

PBDBG (2)



http://www.pgroup.com/products/pgdbg.htm

Intel Debugger (1)

- The Intel Debugger (IDB) provides support for debugging programs written in C, C++, and Fortran 77, 90 and 95
- It provides a choice of command-line and graphical user interface (GUI) on the Linux Eclipse platform
- A part of the Intel® C++ Composer XE 2011 for Linux and Intel® Fortran Composer XE 2011 for Linux
- The Intel® Debugger can debug both single and multithreaded applications
- Provides OpenMP windows with information about current tasks, teams, task waits, barriers, task spawn trees and locks

Intel Debugger (2)

- The Intel Debugger (IDB) features:
  - Attaches to (and detaches from) a running process and debugs the matching program
  - Loads a program into (and unloads a program from) the debugger, automatically creating and deleting processes as necessary
  - Supports multiple-process debugging, associating with one or more programs
  - See processes and examine detailed process state
  - Set breakpoints for a specific process
  - Supports remote debugging of applications on embedded Intel architecture (using a remote agent)
  - Debugs programs with shared libraries
  - Debugs core files
  - …
- http://software.intel.com/en-us/articles/idb-linux/

IBM Parallel Debugger

- Part of IBM Parallel Environment (PE)
- Standard for debugging on POWER systems
- The parallel debugger (pdb) presents user with a single command line interface that supports most dbx/gdb execution control commands
- http://www-03.ibm.com/systems/software/parallel/

Debugging simple codes using GDB

HP-SEE
High-Performance Computing Infrastructure for South East Europe's Research Communities

- Alternatives:
  - Using printf() (adding trace to program)
  - Use debbuger with whom you can:
    - attach to running process
    - change the value of variables at run-time
    - make program stop on specific conditions
    - list source code
    - print variables type
    - inspect a process that has crashed
    - ...
- Choice is easy

GDB basic usage (1)

- Compiling (examples are present at /tmp/gdb):
  - Enable debugging with flag -g :

  ```
  $ gcc –g –o test gdb1.c
  ```
- Source code and executable one to one mapping is made
- Symbol table is created
- Optimization can change things!

- Load executable:

  ```
  $ gdb ./test
  ```
- Symbols are loaded and we can run program (VM)
- We see a command prompt:

  ```
  $(gdb)_
  ```

GDB basic usage (2)

HP-SEE
High-Performance Computing Infrastructure for South East Europe's Research Communities

- Commands:
  - run-Start execution
  - list [arg] -List source code around argument
  - break [arg] -Add a “break point” at arg
  - delete n-Delete break point number n
  - print [arg] -Print the content of arg
  - continue-Continue execution after a break
  - next-Execute next line
  - step-Step into next line (enters functions)
  - backtrace-History of function calls
  - help–Shows help
  - kill-Kill program witout quitting gdb
  - quit-Quit gdb

GDB basic usage (3)

- Type run and program will start:
  - $(gdb) run <arg1, arg2 ...>
- set args – set arguments for next running
- list – list lines of source code (10 lines around argument are displayed):

  list

  list linenum

  list function

  list driver.c:20

GDB basic usage (4)

- Setting breakpoints
  - Set a breakpoint at specific line on current source code file:
    - (gdb) break 8
  - Set a breakpoint at specific function:
    - (gdb) break my_function
  - Set a breakpoint at specific line on some source file :
    - (gdb) break parsing.cc:45

GDB basic usage (5)

- Examining the stack:
  - backtrace - Print backtrace of all stack frames
  - frame - Select and print a stack frame
  - up -Select and print stack frame that called this one
  - down - Select and print stack frame called by this one
  - info locals - Local variables of current stack frame
  - info args - Local arguments of current stack frame

GDB basic usage (6)

- Watchpoints
  - Set on variables (expressions) -variable must be in current scope
  - watch –Set a watchpoint for an expression.
  - rwatch-Set a read watchpoint for an expression.
  - awatch-Set a read/write watchpoint for an expression.
  - Disable–turn off watchpoint

GDB basic usage (9)

- Catchpoints
  - Set on events (C++ exceptions or the loading of a shared library and others)
  - catch EVENT–event can be :
    - throw -The throwing of a C++exception
    - catch -The catching of a C++ exception
    - exec -A call to `exec'
    - fork -A call to `fork'
    - load -A loading of any library
    - load LIBNAME -A loading of specific library
    - unload -Unloading of library
    - thread_start –Starting any threads, just after creation

GDB basic usage (8)

- Inspecting variables
  - ptype–print the data type of a variable
    ```
    (gdb) ptypemyvar
    type = double
    ```
  - print–view the value of a variable
    ```
    (gdb) print i
    $4 = -107
    ```
  - Inspecting an array:
    ```
    (gdb) p myIntArray
    $46 = {0, 1, 2, 3, 4, 5}
    ```
    - (gdb) p myIntArray[3]@7
    - $54 = {3, 4, 5, 10, 1107293224,1079194419, -1947051841}

GDB basic usage (9)

- Inspecting a structure:
  - (gdb) p myStruct
  - $2 = {name = 0x40014978 “Mile mikic‟, EyeColour = 1}
  - (gdb) print myStruct.name
  - $6 = 0x40014978 "Mile Mikic‟
- set - Changing variable value (must be in current context):
  - (gdb) set imidate = 14
- All Fortran variables must be in lowercase!!!

GDB basic usage (10)

Debbuging a running process (gdb2.c):

- attach pid (from gdb) -attach to the running process with pid

```
$ gdb
(gdb) attach 17399
Attaching to process 17399….
```

- $ gdb program pid (outside gdb)

```
Attaching to program: /home/vlada/temp/test2, process 8165
Reading symbols from /lib64/libc.so.6...(no debugging symbols found)...done.
Loaded symbols for /lib64/libc.so.6
Reading symbols from /lib64/ld-linux-x86-64.so.2...(no debugging symbols found)...done.
0x00000030e8c9a510 in __nanosleep_nocancel () from /lib64/libc.so.6
```

- detach – detach from process
- Change variables

gprof

- Used to determine which parts of a program are taking most of the execution time
- gprof is the standard Unix profiler
- It produces basic performance reports including
  - Call counts
  - Timing information
  - Descendent information for called functions

- http://sourceware.org/binutils/docs/gprof/

PGI pgprof

- PGPROF is a powerful interactive postmortem statistical analyzer by PGI
- Supports:
  - MPI process-parallel programs
  - OpenMP thread-parallel programs
  - Programs incorporating PGI Accelerator directives and CUDA Fortran.
- PGPROF can be used to visualize and diagnose the performance of the components of your program
- PGPROF associates execution time with the source code and instructions of your program, allowing you to see where and how execution time is spent
- http://www.pgroup.com/products/pgprof.htm

Intel Vtune (1)

- Intel VTune Amplifier XE
- Commercial application for software performance analysis for 32 and 64-bit x86 based machines
- Threading and performance optimization tool for C/C++ and Fortran developers
- Both GUI and command line interfaces
- Basic features work on both Intel and AMD hardware, advanced hardware-based sampling requires an Intel-manufactured CPU
- Offers various kinds of code profiling:
  - Stack sampling
  - Thread profiling
  - Hardware event sampling

Intel Vtune (2)

- The profiler result consists of details such as time spent in each sub routine which can be drilled down to the instruction level
- The tool can be also used to analyze thread performance
- http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/

Valgrind

- Valgrind is a multipurpose code profiling and memory debugging tool for Linux
- It allows user to run program in Valgrind's own environment that monitors memory usage such as calls to malloc and free (or new and delete in C++)
- If there is usage uninitialized memory, write off the end of an array, or forget to free a pointer, Valgrind can detect it

- http://valgrind.org/

Simple Valgrind examples: Memory leak (1)

- Memory leaks are among the most difficult bugs to detect
- They don't cause any outward problems until you've run out of memory and your call to malloc suddenly fails
- One mistake can be costly if your program runs for long enough and follows that branch of code
- Copy /tmp/valgrind/leak.c to your /nfs/username directory:

```
$ cp /tmp/valgrind/leak.c  /nfs/demoxxx
```

- Compile source code:

```
$ gcc -o test leak.c
```

- Start valgrind:

```
$ valgrind --tool=memcheck ./test
```

Simple Valgrind examples: Memory leak (2)

- Observe output
- Enable leak check

```
$ valgrind --tool=memcheck --leak-check=yes ./test
```

- Observe output
- Compile your program with –g flag:

```
$ gcc -g -o test leak.c
```

- Run with valgrind again:

```
$ valgrind --tool=memcheck --leak-check=yes ./test
```

- Output now shows exact line in source code where the lost memory was allocated

Simple Valgrind examples: Invalid pointer

- Copy /tmp/valgrind/point.c to your /nfs/username directory:

```
$ cp /tmp/valgrind/point.c  /nfs/demoxxx
```

- Compile source code:

```
$ gcc –g -o test2 point.c
```

- Start valgrind:

```
$ valgrind --tool=memcheck --leak-check=yes ./test2
```

- Observe output
- We're using a pointer outside the range - an 'Invalid write'

Simple profiling with gprof (1)

- Profiling using gprof involves following steps:
- Compile your program using the -pg option (you must use the -g option as well if you want line by line profiling):
  - Copy /tmp/gprof/prof.cpp to your /nfs/username directory:

```
$ cp /tmp/gprof/prof.cpp  /nfs/demoxxx
```

  - Compile source code:

```
$ g++ –pg -o test prof.c
```

Simple profiling with gprof (2)

- Run program as normal (through the batch system). This will create a file called gmon.out -- if this file hasn't been created, it's probably because you did not compile with the -pg option.

- Run gprof with program name as argument

```
$ gprof  test gmon.out
```

- It is useful to pipe the output to a textfile that can be viewed in a text editor:

```
$ gprof  test gmon.out > profile.txt
```

Simple profiling with gprof (3)

HP-SEE
High-Performance Computing Infrastructure for South East Europe's Research Communities

- Observe the output

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 60.80      0.78     0.78        1   778.28   778.28  exponential()
 40.27      1.29     0.52        1   515.48   515.48  sinFunc()
```

Libraries

- LAPACK
- BLAS
- FFTW
- SPRNG
- Intel MKL
- NumPy & SciPy
- MPI libraries
- IBM ESSL, IBM PESSL, IBM MASS (tPARADOX)
- ...

LAPACK

HP-SEE
High-Performance Computing Infrastructure for South East Europe's Research Communities

- LAPACK (Linear Algebra PACKage) is a software library for numerical linear algebra
- It provides routines for:
  - solving systems of linear equations and linear least squares
  - eigenvalue problems
  - singular value decomposition
  - includes routines to implement the associated matrix factorizations such as LU, QR, Cholesky and Schur decomposition.
- LAPACK was originally written in FORTRAN 77 and is now written in Fortran 90
- The routines handle both real and complex matrices in both single and double precision
- http://www.netlib.org/lapack/

BLAS

- The BLAS (Basic Linear Algebra Subprograms)
- 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
- 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. LAPACK
- http://netlib.org/blas/

FFTW

- FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data
- FFTW's performance is typically superior to that of other publicly available FFT software,
- Competitive with vendor-tuned codes
- In contrast to vendor-tuned codes FFTW's performance is portable: the same program will perform well on most architectures without modification
- “Fastest Fourier Transform in the West”
- http://www.fftw.org/

SPRNG (1)

- SPRNG (Scalable Parallel Random Number Generators) is one of the best libraries for generating pseudorandom numbers for parallel applications
- It provides user-friendly interfaces for parallel C/C++ and Fortran applications
- Enables the user to easily obtain multiple and sufficiently uncorrelated pseudorandom number streams on different processors, with no inter-process communication.

SPRNG (2)

- The algorithms implemented in SPRNG are:
  - Combined Multiple Recursive Generator
  - The 48 bit Linear Congruential Generator
  - The 64 bit Linear Congruential Generator
  - The modified Lagged Fibonacci Generator
  - The multiplicative Lagged Fibbonacci Generator
  - The Prime Modulus Linear Congruential Generator
- http://sprng.cs.fsu.edu/

Intel MKL (1)

- Intel Math Kernel Library (Intel MKL) is a library of optimized, extensively threaded math routines for solving large science, engineering, and financial computational problems
- The library supports C/C++ and Fortran language APIs
- The most of Intel MKL is threaded which enables more efficient usage of today’s multi-core processors and easier parallelization of developers application or implementation of hybrid programming model

Intel MKL (2)

- Threading is performed through OpenMP, and the following mathematical domains are supported:
  - Sparse Linear Algebra – Sparse BLAS, Sparse Format Converters, PARDISO Direct Sparse Solver, Iterative Sparse Solvers and Pre-conditioners
  - Fast Fourier Transforms
  - Optimized LINPACK benchmark
  - Vector Math Library
  - Statistics Functions – Vector Random Number Generators, Summary Statistics Library
  - Cluster Support – ScaLAPACK, Cluster FFT
- Support page: http://software.intel.com/en-us/articles/intel-mkl/#support
- http://software.intel.com/en-us/articles/intel-mkl/

NumPy & SciPy (1)

- NumPy is the fundamental package needed for scientific computing with Python
- It contains among other things:
  - Powerful N-dimensional array object
  - Sophisticated (broadcasting) functions
  - Tools for integrating C/C++ and Fortran code
  - Useful linear algebra, Fourier transform, and random number capabilities
- SciPy (pronounced "Sigh Pie") is open-source software for mathematics, science, and engineering
- The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation

NumPy & SciPy (2)

- The SciPy library is built to work with NumPy arrays, and provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization
- Together, they run on all popular operating systems, are quick to install, and are free of charge
- NumPy and SciPy are easy to use, but powerful enough to be depended upon by some of the world's leading scientists and engineers
- http://www.scipy.org/
- http://numpy.scipy.org/

MPI Libraries (1)

HP-SEE
High-Performance Computing Infrastructure
for South East Europe's Research Communities

- Supported implementations:

  - MPICH
  - MPICH2
  - OPENMPI
  - MVAPICH & MVAPICH2 (on tPARADOX Cluster)

MPI Libraries (2)

- MPICH
  - MPICH is an open-source, portable implementation of the full MPI-1 standard specification for a wide variety of parallel computing environments,
  - Besides complete implementation of version 1.2 of the MPI Standard, MPICH also implements significant parts of MPI-2, particularly in the area of parallel I/O
  - Although discontinued for some time, MPICH implementation (with the last version 1.2.7) is one of the most frequently used MPI library packages. I
  - Widely used on all types of computing resources: small clusters, Grid sites and large-scale HPC facilities
  - Supports C, C++, F77, and F90
  - http://www.mcs.anl.gov/research/projects/mpi/mpich1-old/

MPI Libraries (3)

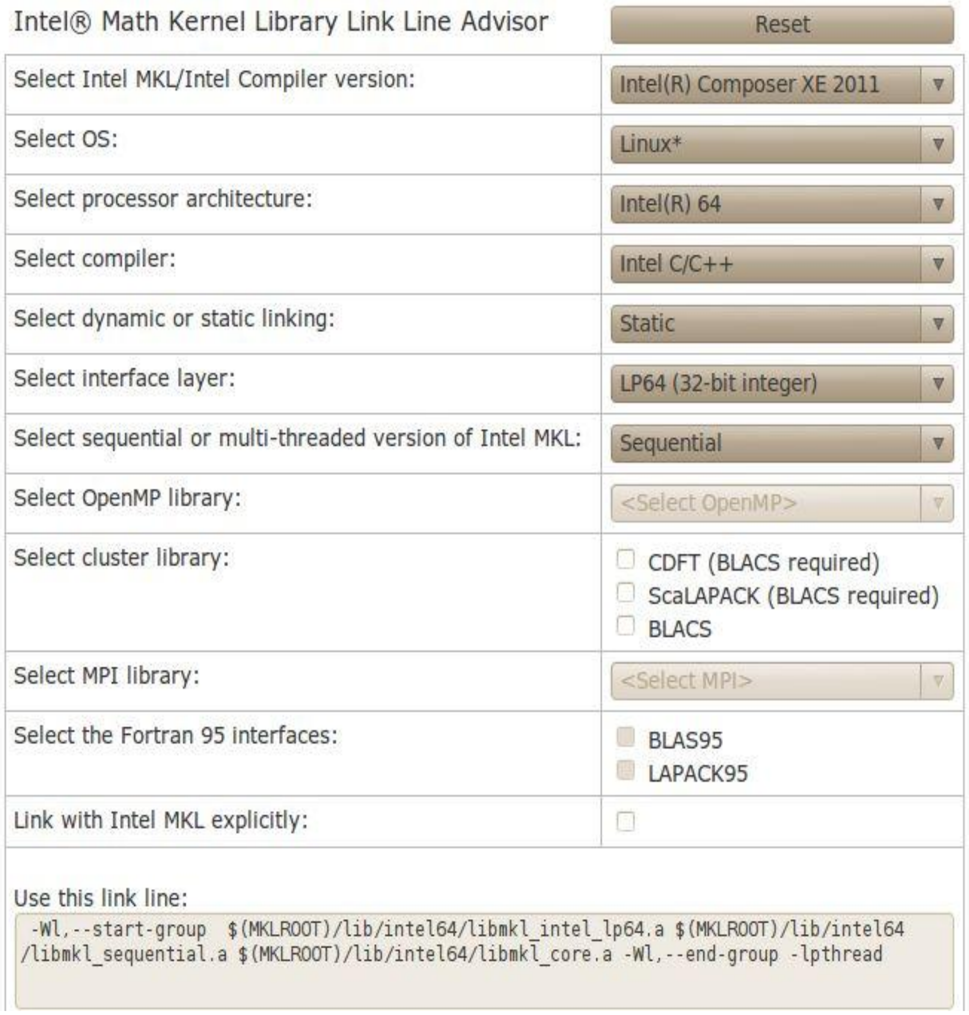
- MPICH-2:
  - MPICH-2 is a high-performance, widely portable implementation of the MPI (Message- Passing Interface) standard, designed to implement all of MPI-1 and MPI-2 specifications (including dynamic process management, one-sided operations, parallel I/O, and other extensions)
  - MPICH-2 implementation replaces MPICH-1 and it is recommended to be used instead of MPICH-1
  - It provides several different process managers (process managers are basically external (typically distributed) agents that spawn and manage parallel jobs) such as Hydra, MPD, SMPD etc.
  - http://www.mcs.anl.gov/research/projects/mpich2/

MPI Libraries (4)

- OPENMPI:
  - The Open MPI Project is an open source MPI-2 implementation that is developed and maintained by a consortium of academic, research, and industry partners
  - Features implemented for Open MPI include:
    - Full MPI-2 standards conformance
    - Thread safety and concurrency
    - Dynamic process spawning
    - Network and process fault tolerance
    - Support network heterogeneity
    - Single library supports all networks
  - http://www.open-mpi.org/

Intel MKL linking guide (1)

- You can use Intel MKL link line advisor:
  - http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor/
- Make choices based on your platform and needs and on-line tool will generate the right link line


Intel® Math Kernel Library Link Line Advisor

Select Intel MKL/Intel Compiler version:	Intel(R) Composer XE 2011
Select OS:	Linux*
Select processor architecture:	Intel(R) 64
Select compiler:	Intel C/C++
Select dynamic or static linking:	Static
Select interface layer:	LP64 (32-bit integer)
Select sequential or multi-threaded version of Intel MKL:	Sequential
Select OpenMP library:	<Select OpenMP>
Select cluster library:	☐ CDFT (BLACS required) ☐ ScaLAPACK (BLACS required) ☐ BLACS
Select MPI library:	<Select MPI>
Select the Fortran 95 interfaces:	☐ BLAS95 ☐ LAPACK95
Link with Intel MKL explicitly:	☐

Use this link line:

```
 -Wl,--start-group  $(MKLROOT)/lib/intel64/libmkl_intel_lp64.a $(MKLROOT)/lib/intel64
/libmkl_sequential.a $(MKLROOT)/lib/intel64/libmkl_core.a -Wl,--end-group -lpthread
```

Intel MKL linking guide (2)

- As for the Intel compiler source the script for the environment initialization (there is a MKL within the compiler suite):
  - `$ source /opt/intel/composerxe/bin/compilervars.sh intel64`
- Or use specific MKL environment script:
  - `$ source /opt/intel/mkl/10.2.3.029/tools/environment/mklvarsem64t.sh`

Intel MKL linking example (1)

- Copy source code test.c from /tmp/mkl/matrix.c directory on ui.ipb.ac.rs to /nfs/username directory:

```
$ cp /tmp/mkl/matrix.c  /nfs/demoXXX
```

- Export MKL environment:

```
$ source /opt/intel/composerxe/bin/compilervars.sh intel64
```

- Compile code with threaded and sequential version of MKL libraries

Intel MKL linking example (2)

- Sequential:

```
icc -o matrix_sequential matrix.c -mkl=sequential -static-intel
```

or

```
$ icc -o matrix_sequential  matrix.c  -Wl,--start-group
${MKLROOT}/lib/intel64/libmkl_intel_lp64.a
${MKLROOT}/mkl/lib/intel64/libmkl_sequential.a
${MKLROOT}/lib/intel64/libmkl_core.a -Wl,--end-group  -lpthread -static-intel
```

- Threaded:

```
$ icc -o matrix_threaded matrix.c -mkl -static-intel
```

or

```
$ icc -o vlada_thread  matrix.c  -Wl,--start-group
${MKLROOT}/lib/intel64/libmkl_intel_lp64.a
${MKLROOT}/lib/intel64/libmkl_intel_thread.a
${MKLROOT}/lib/intel64/libmkl_core.a -Wl,--end-group -openmp -lpthread -static-
intel
```

Intel MKL linking example (3)

- Use serial and OpenMP job submit scripts to submit serial and threaded version of program using reasonable matrix size (3000-5000)
- Use time command with your binary
- Compare outputs

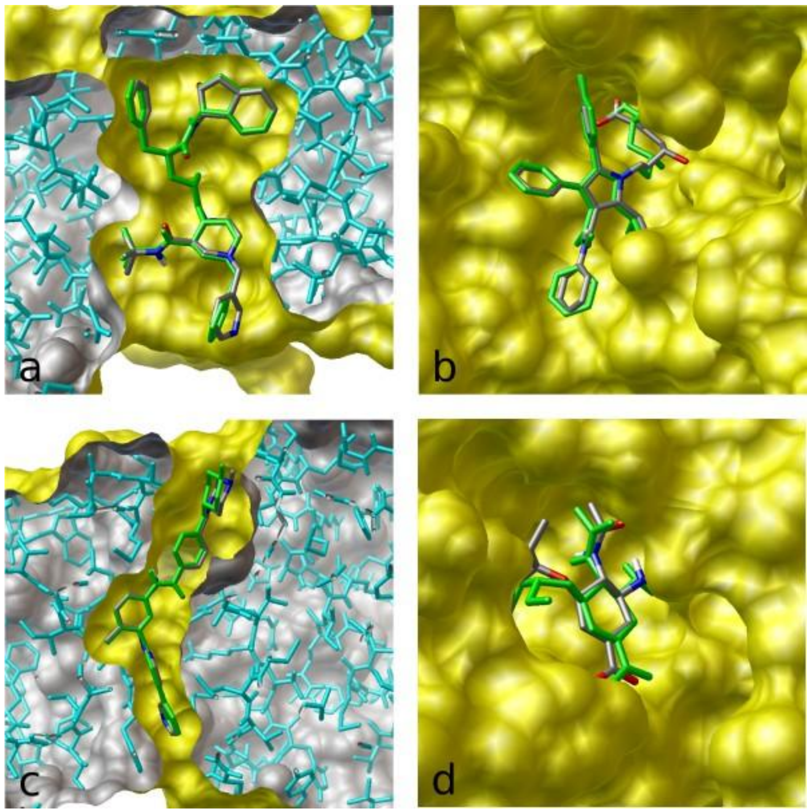
```
$time ./matrix_sequential 5000
Dgemm_multiply(). Elapsed time = 28.56 seconds

real        0m29.535s
user        0m28.794s
Sys         0m0.734s

$ time ./matrix_threaded 5000
Dgemm_multiply(). Elapsed time = 31.3 seconds

real        0m4.974s
user        0m31.483s
Sys         0m1.036s
```

Applications

- AutoDock Vina
- CPMD
- GAUSSIAN
- NAMD
- …

AutoDock Vina

- AutoDock Vina is a open-source program for drug discovery, molecular docking and virtual screening
- Implements multi-core capability
- For its input and output, Vina uses the same PDBQT molecular structure file format used by AutoDock
- http://vina.scripps.edu/



CPMD

HP-SEE
High-Performance Computing Infrastructure for South East Europe's Research Communities

- The CPMD code is a parallelized plane wave/pseudopotential implementation of Density Functional Theory, particularly designed for ab-initio molecular dynamic
- CPMD runs on a large variety of different computer architectures
- Features:
  - Wavefunction optimization: direct minimization and diagonalization
  - Geometry optimization: local optimization and simulated annealing
  - Molecular dynamics: NVE, NVT, NPT ensembles
  - Time-dependent DFT (excitations, molecular dynamics in excited states)
  - …
- http://www.cpmd.org/

GAUSSIAN

HP-SEE
High-Performance Computing Infrastructure
for South East Europe's Research Communities

- Computational chemistry software used by chemists, chemical engineers, biochemists, physicists and other scientists worldwide.
- Starting from the fundamental laws of quantum mechanics, Gaussian predicts the energies, molecular structures, vibrational frequencies and molecular properties of molecules and reactions in a wide variety of chemical environments
- Features:
  - Molecular mechanics
  - Semi-empirical calculations
  - SCF methods
  - Møller-Plesset perturbation theory
- http://www.gaussian.com/

NAMD

HP-SEE
High-Performance Computing Infrastructure for South East Europe's Research Communities

- Not (just) Another Molecular Dynamics program
- Parallel molecular dynamics code designed for high-performance simulation of large biomolecular systems
- Based on Charm++ parallel objects
- Noted for its parallel efficiency and often used to simulate large systems (millions of atoms)
- http://www.ks.uiuc.edu/Research/namd/

Support

- PARADOX cluster support:
  - hpc-admin@ipb.ac.rs