



# **SGI® Altix™ Linux and Compiling Environment**

Reiner Vogelsang

SGI GmbH

[reiner@sgi.com](mailto:reiner@sgi.com)

June 22, 2005



# Module Objectives

**After completing the module, you will be able to**

- **Login into an Altix**
- **Help yourself within the Linux environment**
- **Recognize the Altix compiler flow**
- **Compile programs with standard options**
- **Create and use static and dynamic shared libraries**
- **Use some object file analyzers**

# Login procedure

- **Authentication of a user**

- Check of a requested access to a system against rights and permissions of a user account.
- Branch to the so-called home directory of a user.

- **`rlogin -l <user> <host>`**

- Starts a remote terminal session on the target host.
- Insecure! `rlogin`, `rsh` `rcp` are usually deactivated at larger computer sites.

- **`telnet -l <user> <host> [port]`**

- Communicates via the TELNET protocol with a remote host. Insecure connection as well. Use that login method in controlled environments only.

- **`ssh -X -l <user> <host>`**

- Secure login procedure and transfer of user data via RSA/DSA encryption. Even encrypted channel for X11 display forwarding is automatically established. Strongly recommended!

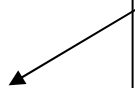
# Shells

- The command `ps` returns information about processes running under Linux. Only processes belonging to a user are shown per default:

```
reiner@dcm24 3> ps
```

PID	TTY	TIME	CMD
4744	pts/1	00:00:00	csH
4777	pts/1	00:00:00	ps

Your shell having the process id 4744 connecting you to terminal pts/1



- The following shells (“command interpreters”) are available:
- `sh` - Bourne shell
- `ksh` - Korn shell
- `tcsh, csh` - C-shell
- `bash` - Bourne-Again shell, the Linux shell

# Helpful Commands: man

- “man” stands for manual and is the Unix “help” command
- Manual pages (“man pages”) are written in troff, the traditional Unix text formatting system.
- The default location of the man pages is /usr/man or /usr/share/man
- If you know the command but you have forgotten a certain option type `man <command>` like “man man”:

## NAME

`man` - format and display the on-line manual pages

`manpath` - determine user's search path for man pages

## SYNOPSIS

```
man [-acdfFhkKtwW] [--path] [-m system] [-p string]
[-C config_file]
```

```
[-M pathlist] [-P pager] [-S section_list] [section]
```

```
name ...
```

# Helpful Commands: man

- If you know the action you would like to perform but don't know the command which serves your needs search the index of the man pages with a keyword:

```
man -k < key word > like man -k manual
```

```
reiner@dcm24 17> man -k manual
```

```
FSG [lsb_release](1) - manual page for FSG lsb_release v1.4
```

```
man (1) - format and display the on-line manual pages
```

```
man [manpath] (1) - format and display the on-line manual pages
```

```
man2html (1) - format a manual page in html
```

- 
- 
-

# Helpful Commands: env and export

- **env** lists the settings of your environment:

```
DISPLAY=reiner.sgi.com:0
```

```
TERM=xterm
```

```
REMOTEHOST=dcm13
```

```
HOME=/ptmp/reiner
```

```
PATH=/sw/com/histx_1.2a/bin:/sw/sdev/intel-
```

```
cc/8.0.069/bin:/sw/sdev/intel-
```

```
fc/8.0.050/bin:/ptmp/reiner/bin:/usr/kerberos/bin:/usr/local/bin:/bi
```

```
n:/usr/bin:/usr/X11R6/bin:/ptmp/reiner/scratch_fc/reiner/PRISM/pytho
```

```
n/bin:/usr/gnu/bin:/usr/freeware/bin:/usr/local/bin:/ptmp/reiner/scr
```

```
atch_fc/reiner/PRISM/local/bin
```

```
LD_LIBRARY_PATH=/sw/com/histx_1.2a/lib:/sw/sdev/mkl/7.0.007/mkl70/lib
```

```
/64:.....
```

- Check important variables like **PATH**, **LD\_LIBRARY\_PATH**, **DISPLAY**
- Global variables are set in the Bash by  
`export <var name>=<value>` or  
`export <var name>=${<var name>}:value`

# Helpful Commands: ulimit

- Provides control over the resources available to the shell and to processes started by it.
  - `ulimit -a` reports all the limits of your resources
  - Look for the `stacksize` which is ridiculously small under Redhat. Codes with huge stack requirements may abort with a core! Increase `stacksize` to global limits by `ulimit -s unlimited`
  - Set the size of core dumps to zero: `ulimit -c 0`  
This prevents unintended file system hogs!



# Helpful Commands: topology, hwinfo

- `topology` and `hwinfo` replace `hinv` (Redhat, ProPack 3).
- Those commands partially reflect the information of the special directory tree `/proc`, `/sys` or `/var/lib/hardware/`:

```
reiner@dcm27 103> /usr/sbin/hwinfo -disk
```

```
21: SCSI c00.0: 10600 Disk
```

```
.....
```

```
SysFS ID: /block/sdaa
```

```
SysFS BusID: 12:0:0:0
```

```
SysFS Device Link: /devices/pci0000:14/0000:14:01.1/host12/12:0:0:0
```

```
....
```

```
Model: "SGI ST373453FC"
```

```
Driver: "qla2300", "sd"
```

```
Device File: /dev/sdaa (/dev/sg26)
```

```
Device Files: /dev/sdaa, /dev/disk/by-path/pci-0000:14:01.1-scsi-0:0:0:0
```

```
Attached to: #17 (Fibre Channel)
```

# Helpful Commands: topology, hwinfo

```
reiner@dcm27 104> topology
```

```
Serial number: N0001045
```

```
Partition number: 0
```

```
2 C-Bricks
```

```
4 Routers
```

```
1 TIO-Brick
```

```
16 CPUs
```

```
61.35 Gb Memory Total
```

```
...
```

Node	ID	asic	NASID	Memory
------	----	------	-------	--------

-----

0	001c05#2	SHub_1.2	0	8063296 kB
---	----------	----------	---	------------

1	001c05#1	SHub_1.2	2	8077312 kB
---	----------	----------	---	------------

```
...
```

CPU	ID	Family	Rev	Speed	data	inst	L2	L3
-----	----	--------	-----	-------	------	------	----	----

-----

0	001c05#2a	Itanium 2	1	1600	16K	16K	256K	6144K
---	-----------	-----------	---	------	-----	-----	------	-------

1	001c05#2c	Itanium 2	1	1600	16K	16K	256K	6144K
---	-----------	-----------	---	------	-----	-----	------	-------

# Helpful Commands: topology, hwinfo(cont.)

- `/proc/cpuinfo` contains essential CPU information

```
processor      : 15
vendor        : GenuineIntel
arch          : IA-64
family        : Itanium 2
model         : 1
revision      : 5
archrev       : 0
features      : branchlong
cpu number    : 0
cpu regs      : 4
cpu MHz       : 1500.000000
itc MHz       : 1500.000000
BogoMIPS      : 16.74
```

# Helpful Commands: topology, hwinfo (cont.)

- `/proc/pal/cpu<0-nnn>` contains more advanced information:

```
cat /proc/pal/cpu0/cache_info
```

•

•

Data/Instruction Cache level 3:

```
Size           : 6291456 bytes
Attributes     : Unified WriteBack
Associativity  : 24
Line size      : 128 bytes
Stride         : 128 bytes
Store latency  : 7 cycle(s)
Load latency   : 14 cycle(s)
Store hints    : [Reserved]
Load hints     : [Non-temporal, level 1]
Alias boundary : 4096 byte(s)
Tag LSB        : 18
Tag MSB        : 49
```

# Helpful Commands: topology, hwinfo (cont.)

- `/sys/devices/system/node/*/meminfo` the memory statistics per node

`-nmumactl -hardware` lists installed and free memory per node

```
node 0 size: 7874 MB
```

```
node 0 free: 7183 MB
```

```
node 1 size: 7888 MB
```

```
node 1 free: 7429 MB
```

```
node 2 size: 7888 MB
```

```
node 2 free: 7540 MB
```

```
node 3 size: 7887 MB
```

```
...
```

- Look out for nodes with few free memory. Application will allocate from remote nodes.
- Use `bcfree -afs`, a tool for freeing pages in the buffer or slab caches. More subtle is the usage of `posix_fadvise`.

# Helpful Commands: gtopology

- `gtopology` prints optionally a listing of hardware component and their relationship describing the network topology of your Altix system.
- `gtopology` displays a graphical interpretation of the network topology.
  - Helpful to understand performance issues due to missing links or nodes.

• `reiner@dcm24 2> gtopology`

```
Machine dcm24.munich.sgi.com : 16 Processors / 8 nodes/ 2  
routers
```

```
Interconnect: 1 ;Level= 1 ;n-obj= 4
```

```
Interconnect: 1 ;Level= 2 ;n-obj= 1
```

```
Interconnect: 2 ;Level= 1 ;n-obj= 4
```

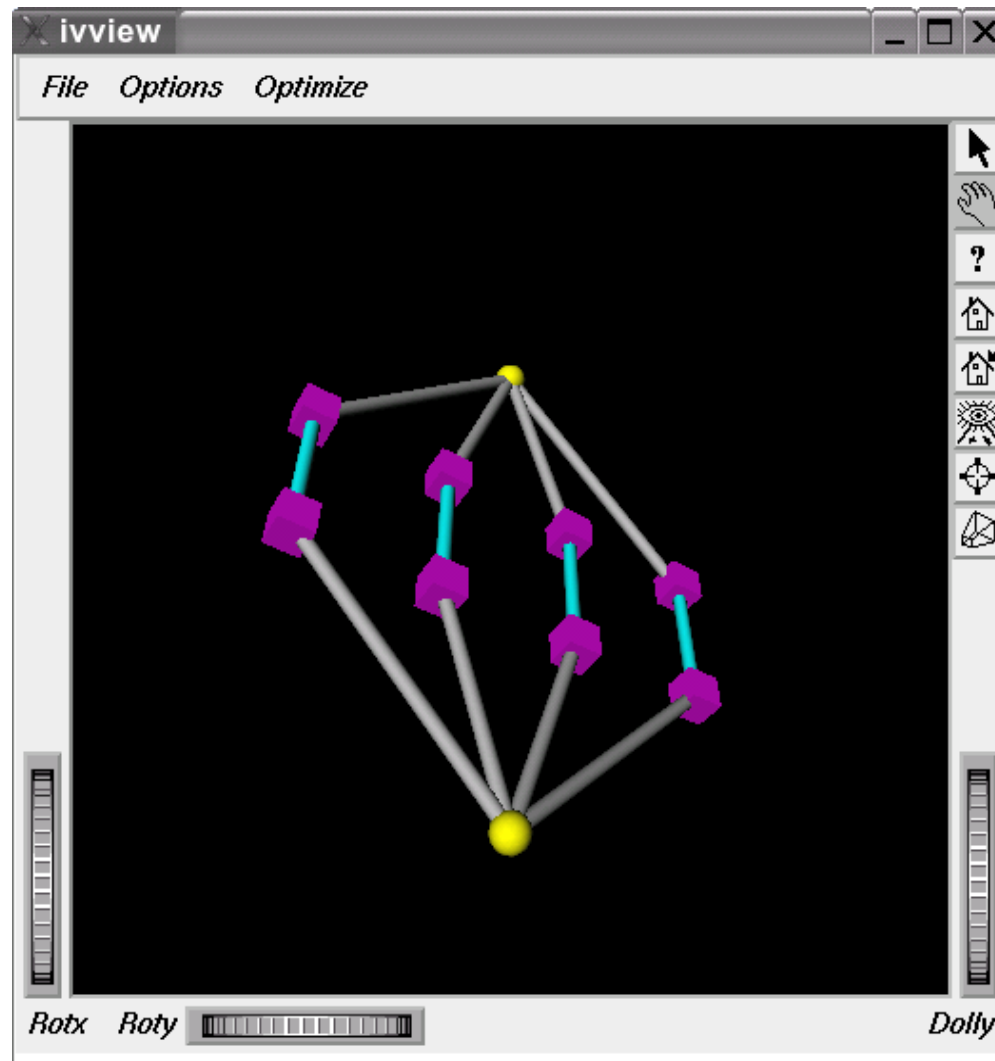
```
Interconnect: 2 ;Level= 2 ;n-obj= 1
```

```
Found 10 topology objects :
```

```
Found 24 Links between the 10 objects
```

```
Max_hops= 3 between node0: 001c05/0 and most_remote:  
001c18/1 :
```

# Helpful Commands: topology and gtopology



# Helpful Commands: top

- `top` provides an ongoing look at processor activity in real time. It displays a listing of the most CPU-intensive tasks on the system, and can provide an interactive interface for manipulating processes. It can sort the tasks by CPU usage, memory usage and runtime can be better configured than the standard `top` from the `procps` suite. Most features can either be selected by an interactive command or by specifying the feature in the personal or system-wide configuration file.

- Example of a `.toprc` file:

```
RCfile for "top with windows"           # shameless braggin'  
Id:a, Mode_altscr=0, Mode_rixps=1, Delay_time=3.000, Curwin=0  
Def      fieldscur=AEHIOQTWKNMbcdfgJplrsuvyzX  
         winflags=34105, sortindx=12, maxtasks=0  
         summclr=1, msgscclr=1, headclr=3, taskclr=1
```

- Example of an alias to get a snapshot of your own processes:

```
revenue.engr.sgi.com:reiner 7> alias topme  
top -b -n 1 | sort -n | grep reiner
```



# Helpful Commands: top (cont.)

- Output of top with the sample .toprc:

```
  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  P COMMAND
 8332 root        0  -20   9808 6496 4736 R    1   0.0   49:34.08  0  lim
17559 sgi         25   0 30.8g  15m 4352 R  200   0.0   0:47.44  53  mxm4.mpi.x
17560 sgi         25   0 30.8g  15m 4352 R  200   0.0   0:47.44  55  mxm4.mpi.x
17558 sgi         25   0 30.8g  15m 4352 R  200   0.0   0:47.43  50  mxm4.mpi.x
17557 sgi         25   0 30.8g  15m 4352 R  200   0.0   0:47.40  49  mxm4.mpi.x
17408 sgi         17   0  4112 2624 1584 R    1   0.0   0:02.05  1  top
```

# C/C++ and Fortran Compilers

- Intel compilers
- 7.1 Compilers shipped March 2003
- 8.0 Compilers released December 2003
  - Switch to a DEC compiler frontend, efc -> ifort, ecc -> icc
- 8.1 Compilers released June 2004
- 9.0 Compilers released February 2005
- Fortran supports OpenMP 2.0
- C/C++ compatible with gcc and C99 standard (subset)
- GNU Fortran and C
- Enable easy migration from 32-bit platforms to Altix
- Included in the standard Linux distribution
- ORC (Open Research Compiler) Fortran and C
  - Available at *<http://ipf-orc.sourceforge.net/>*

# Compiling a Program

- Compile line:

```
icc [ option(s) ] filename.{c|C|cc|cpp|cxx|i}  
ifort [ option(s) ] filename.{f|for|ftn|f90|fpp}  
gcc|g++ [ option(s) ] filename.{c|C|cc|cxx|m|i|ii}  
g77 [ option(s) ] filename.{f|for|F|fpp}
```

- Filename requires the appropriate extension:

```
% icc main.c  
% ifort main.f  
% g77 main.f[or]  
% g++ main.C
```

# Common Compiler Options

- `-o <file_name>` Renames the output file
- `-g` Turns debug mode on, does NOT change opt. level.
- `-r8` Converts all intrinsic REAL to DOUBLE PRECISION, default reals are 4 byte entities as well as the integers
- `-c` Compile only
- `-O[0 | 1 | 2 | 3]` Optimization levels, O3 turns prefetching.
- `-parallel` Auto-parallelizer
- `-openmp` Turns on OpenMP directives
- `-openmp_profile` Profile of OpenMP directives

# Common Compiler Options

- `-mp`

**Use `-mp` to limit floating-point optimizations and maintain declared precision**

- `-mp1`

**Less performance impact**

- `-IPF-fltacc`

**Try to maintain floating point accuracy**

# Floating-point Underflow

- Many processors do not handle denormalized arithmetic (for gradual underflow) in hardware.

Whether environments support gradual underflow is very implementation dependent, and may lead to differences in numerical results.

- The Intel compiler provides the  
–ftz  
option to force flushing denormalized numbers to zero.

# Endianess

- The Intel IA64 as well as the rest of the Intel processor family is working with byte-wise little-endian address representation.

– The number 1025 bit-wise represented and grouped in 4 bytes:

00000000 00000000 00000100 00000001

^MSB

^LSB

–	Big Endian	Little Endian
00	00000000	00000001
01	00000000	00000100
02	00000100	00000000
03	00000001	00000000

– In rare cases even bytes can be little-endian.

# Endianess (cont.)

- **Big endian systems are**
    - **SGI MIPS/Irix (Origin 3000, 2000,...)**
    - **HP PA Risc**
    - **Sun Sparc**
    - **IBM Power RISC**
    - **NEC vector systems, Cray vector systems**
  - **To read/write big endian binary data you HAVE to set (Intel 9.x, 8.x and 7.x compilers):**
    - F\_UFMTENDIAN=big (applies to all units)**
    - F\_UFMTENDIAN=big:10,20 (applies to unit 10 and 20 only)**
- or compile with**
- **convert big (Intel 9.x and 8.x compilers)**

**(See Intel® Fortran Compiler for Linux\* Systems User's Guide, Volume I: Building Applications)**



# Modules

- `module` is a user interface that provides utilities for the dynamic modification of a user's environment, i.e., users do not have to modify their `PATH` and other environment variables by hand to access the compilers, loader, libraries, and utilities.

If enabled, modules can be used on the SGI Altix Series to customize the compiling environment.

To access the software on the SGI Altix Series, do the following (typically `MODULESHOME` will be `/opt/modules/x.y.z`, where `x.y.z` is the modules package version):

# Modules

- C shell initialization (in *.cshrc*):

```
source ${MODULESHOME}/init/csh
module load intel-compilers-latest mpt-1.9-1rel
module load scsl-1.4.1-1
```

- Bourne shell initialization (in *.profile*):

```
. ${MODULESHOME}/init/sh
module load intel-compilers-latest mpt-1.9-1rel
module load scsl-1.4.1-1
```

- Korn shell initialization (in *.profile*):

```
. ${MODULESHOME}/init/ksh
module load intel-compilers-latest mpt-1.9-1rel
module load scsl-1.4.1-1
```

# Modules

- To view which modules are available on your system (any shell):

```
% module avail
```

```
----- /sw/com/modulefiles -----  
SCCS                ivision.R  
admin              ivision.lnk  
...  
capd               mpt-1.9-1  
...  
epic.5.1           scsl-1.4.1rel  
...  
intel-compilers-latest transcript.4.0  
...
```

# Modules

- To list which modules are in your environment (any shell):

```
% module list
```

Currently Loaded Modulefiles:

```
1) intel-compilers-latest      3) scsl-1.4.1rel
2) mpt-1.9-1
```

- See `man module` for more options

# Libraries

- Libraries are files that contain one or more object (.o) files
- Libraries are used to
  - Protect a company's investment in software development by allowing to ship only object code to customers and developers
  - Simplify local software development by ``hiding'' compilation detail
- In UNIX, libraries are sometimes called *archives*

# Static Versus Dynamic Libraries

- **Static library**
  - Calls to library components are satisfied at link time by copying text from the library into the executable.
- **Dynamic library**
  - As the program starts, all needed libraries are linked into the program.
- When loaded into memory, the library can be accessed by multiple programs.
- Dynamic libraries are formed by creating a Dynamically Shared Object (DSO) file.

# Handling of Static Libraries

- Create a library with three object files:

```
% ar -q libutil.a object1.o object2.o object3.o
```

- List the contents of the archive:

```
% ar -t libutil.a
```

```
object1.o
```

```
object2.o
```

```
object3.o
```

- Add a file to the archive:

```
% ar -q libutil.a object4.o
```

- Replace an object with a newer version:

```
% ar -r libutil.a object4.o
```

- Delete an object from the archive:

```
% ar -d libutil.a object4.o
```

# Using Static Libraries

- To use a static library, include the library on the compile line:

```
% gcc -o myprog myprog.c func1.o libutil.a
```

- If the library is named *lib<name>.a* and it is not in a standard library directory, use the `-L<dir>` and `-l<name>` options:

```
% gcc -o myprog myprog.c func1.o -L./libs -lutil
```

- In the above example, if both a dynamic and static libraries exist in the same directory, the dynamic library is chosen first
- To use the static version of standard libraries, use the full path name of the library or the `-static` option:

```
% gcc myprog.c /usr/lib/libm.a
```

or

```
% gcc myprog.c -static -lm
```



# Creating Dynamic Libraries

- To create a dynamic library with a series of object files:

```
% ld -shared object1.o object2.o -o libops.so
```

- To create a DSO from an existing static library:

```
% ld -shared -whole-archive libutil.a -o libutil.so
```

# Using Dynamic Libraries

- To use a dynamic library, include the library on the compiler line:

```
% gcc -o myprog myprog.c func1.o libops.so
```

```
% gcc -o myprog myprog.c func1.o -L./libs -lops
```

```
% gcc myprog.c -lm
```

- When using `-l<string>` and, within a directory, both `lib<string>.a` and `lib<string>.so` exist, the DSO library is used

- If your dynamic library is not in the standard directories, the run-time linker `ld.so` cannot find it unless you

- Use the `-rpath <directory>` option during linking:

```
% gcc -o myprog myprog.c -Wl,-rpath -Wl,./libs -L  
./libs -lops
```

or

- Set the `LD_LIBRARY_PATH` environment variable before running the executable:

```
% setenv LD_LIBRARY_PATH ./libs
```

```
% myprog
```

# Libraries Included with the Intel Compilers

- **libguide.a, libguide.so**
  - for support of OpenMP-based program
- **libsvml.a**
  - short vector math library
- **libirc.a**
  - Intel support for PGO (profile-guided optimization) and CPU dispatch
- **libimf.a, libimf.so**
  - Intel math library
- **libcprts.a, libcprts.so**
  - Dinkumware C++ library
- **libunwind.a, libunwind.so**
  - Unwinder library
- **libcxa.a, libcxa.so**
  - Intel runtime support for C++ features

# More Intel Libraries

- **Mathematical Kernel Library (MKL)**
  - Link against `-lmkl`
- **Intel's scientific and engineering floating point math library**
- **Initially only basic linear algebra subroutines (BLAS) and fast Fourier transformations (FFT)**
- **Address:**
  - Solvers such as linear algebra package (LAPACK) and BLAS
  - Eigenvector/eigenvalue solvers (BLAS, LAPACK)
  - Some quantum chemistry needs (dgemm)
  - PDEs, signal processing, seismic, solid-state physics (FFTs)
  - General scientific, financial - vector transcendental functions, vector markup language (VML)
- **Don't use MKL on small counts!**
- **Documentation usually available in `<MKL install directory>/`**

# More Intel Libraries

- Intel Integrated Performance Primitivesbasic/common peration
- Link against -lipp
  - Low structure
  - Atomic (does one thing)
  - No I/O
  - Low overhead
  - No local storage
- Contains functions for multimedia, matrix processing for visualization..., cryptography and string processing.....
- Documentation: <http://www.intel.com/software/products/ipp/>

# SGI ProPack Libraries

- **Message Passing Toolkit**

- **libmpi**

- **Implements the Message Passing Interface**
    - **Fully compliant with MPI-1 standard**
    - **Plus a couple of MPI-2 features**
      - MPI-IO based on ROMIO
      - MPI-2 on-sided communication
      - MPI-2 process creation and communicator handling
      - NO dynamic process creation
    - **Thread-safe and OpenMP interoperability implemented**
    - **Documentation available by man pages and <http://www-unix.mcs.anl.gov/mpi/>**

# SGI ProPack Libraries

- **Message Passing Toolkit (cont.)**

- **libshmem**

- **Implements Cray's one-sided communication library (get and put).**
    - **Sub us latencies achievable.**
    - **You have to handle remote addresses yourself.**
    - **However, works across partitions!**
    - **Documentation: man intro\_shmem**

# SGI ProPack Libraries

- **SGI Scientific Library**

- **libscs**

- **Contains optimized version of LAPACK and BLAS.**
    - **Provide FFTs and sparse solvers.**
    - **OpenMP parallelized and thread safe!**
    - **Has certain performance advantages over MKL in application environments.**
    - **Documentation: man**



# SGI ProPack Libraries

- **Flexible File I/O layer**
  - **libffio, libeag\_ffio**
    - **Implements a subset of Cray's FFIO layer**
      - **Especially tailored for I/O caching and user customized buffer handling**
    - **You may need that for VERY I/O demanding applications like NASTRAN.**
    - **Defines an API similar to fopen, fread, fwrite. Calls fopen, fread... instead.**
    - **Documentation: man intro\_ffio**

# Getting Information about Object Files and Libraries

- `file` Lists the general properties of file
- `size` Lists the size of each section of the object file
- `readelf` Lists the content of an ELF object
- `ldd` Lists shared library dependencies
- `nm` Lists the symbol table information
- `objdump` Dissambles object and executable
- `objcopy` Let you manipulate symbols of a binary
- `strip` Removes the symbol table and relocation bits from executable
- `c++filt` Demangles names from C++

# Listing File Properties and File Size

- Use `file(1)` for information about object files and executables

```
% file main
```

```
main: ELF 64-bit LSB executable, IA-64, version 1,  
      dynamically linked (uses shared libs), not  
stripped
```

# Estimating Memory Requirements of a Program

- `size(1)` reports the size of a program
- Reported size is the minimum space required

```
$ size main
```

```
   text      data      bss      dec      hex filename
   3254       788       80     4122     101a main
```

```
$ size -A main
```

```
main :
```

```
section          size          addr
.interp          24  4611686018427388360
.note.ABI-tag    32  4611686018427388384
.hash            144  4611686018427388416
.dynsym          408  4611686018427388560
.dynstr          244  4611686018427388968
.
.
.
.debug_abbrev    252          0
.debug_line      0          0
Total            9384
```

# Getting Information About ELF Files

- Use `readelf(1)` to inspect sections of an ELF (Executable and Linking Format) file:

```
readelf [options] filename1 [filename2...]
```

- You can print the ELF header, section headers, DSO library list, library information, and so on, by specifying different options (see man page).

- List dynamic shared library list using `readelf` or `ldd`:

```
% readelf -d main
```

Dynamic segment at offset 0xf40 contains 24 entries:

Tag	Type	Name/Value
0x0000000000000001	(NEEDED)	Shared library: [libuti
1.so]		
0x0000000000000001	(NEEDED)	Shared library: [libc.s
o.6.1]		
0x000000000000000f	(RPATH)	Library rpath: [.]
0x000000000000000c	(INIT)	0x400000000000006a0
. . .		
0x0000000000000000	(NULL)	0x0

# Getting Information About ELF Files

```
•% ldd main
    libutil.so => ./libutil.so (0x20000000000048000)
    libc.so.6.1 => /lib/libc.so.6.1 (0x20000000000204000)

    /lib/ld-linux-ia64.so.2 => /lib/ld-linux-
ia64.so.2 (0x20000000000000000)
```

# Listing Global Symbol Table Information

- Use `nm(1)` to list global symbol table information for object files and archives

```
% nm example.o
```

```
U func5
```

```
00000000000000000000 G global_initialized
```

```
00000000000000000050 C global_uninitialized_array
```

```
00000000000000000000 T main
```

```
U sqrt
```

# Using nm(1) To Find Unresolved Symbols

- When compiling a program, you may get an error from the linker that a symbol is unresolved (it cannot find where the symbol is defined):

```
% cc myprog.c -lmy_lib
ld:
Unresolved:
Missing_Symbol
```

- If you do not know where Missing\_Symbol is defined, you can search available object files and libraries for the symbol.
- For example, use a combination of `nm` and `grep` to search for this symbol in local object files, libraries, and DSOs:

```
% foreach i (*.o *.a *.so)
? nm $i | grep Missing_Symbol | grep ' T '
? echo $i
? echo
? end
```



# Disassembling Object Files

- To disassemble an object file, use `objdump(1)`:

```
objdump -d filename1 [filename2...]
```

- See how the optimizer is rearranging your source code
- Hand-tune in assembly
- Use the `-S` option to mix source, if possible, with the assembly code
- You can rename symbols on the fly using `objcopy`:  

```
objcopy --redefine-sym sgemm=dgemm <code_with_clashes>
```

# Stripping Executables of Symbol Table Information

- Use `strip(1)` to remove all symbol table information, thereby decreasing the size of your executables:

```
strip [options] filename1 [filename2...]
```

```
% icc -g -o main main.o libutil.a -lm
```

```
% ls -l main
```

```
-rwxr-
```

```
x-- 1 gerardo sdiv 259839 Apr 15 10:45 main*
```

```
% strip main
```

```
% ls -l main
```

```
-rwxr-
```

```
x-- 1 gerardo sdiv 211912 Apr 15 10:45 main*
```

- Stripped executables cannot be debugged symbolically, and

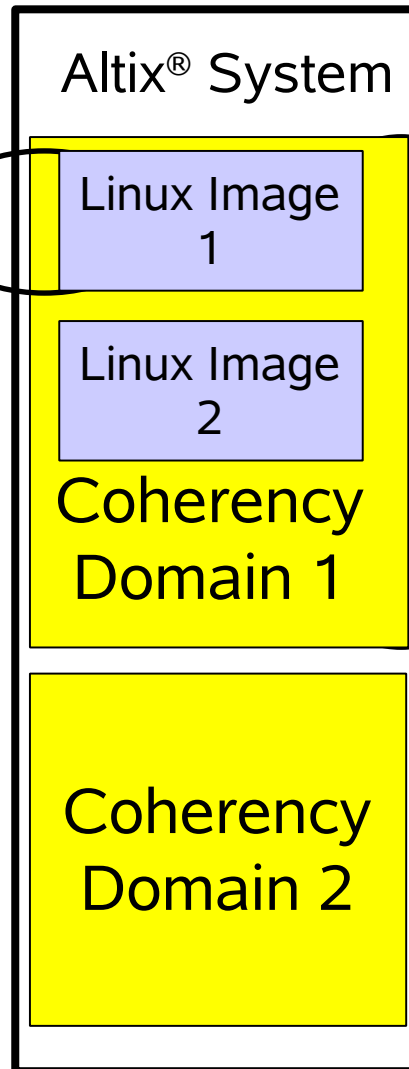
`nm(1)` gives an error.

- Stripping also provides a measure of intellectual property protection when distributing binary code.

# Parallel Programming Models on Shub 1.2-based Altix 3000 Systems

**Intra-Host  
( $\leq 512$  P)**

- OpenMP
- Pthreads
- MPI
- SHMEM™
- Global memory segments



**Intra-Coherency Domain (512P)**

- MPI
- SHMEM
- Global memory Segments

**Spanning Entire System**

- MPI

# Programming Models

**Altix and NUMALink are efficient for:**

- OpenMP**
- MPI**
- CAF (Co-Array Fortran)**
- UPC**
- Global shared memory segment programming**
- OpenMP combined with MPI**

# Compiling and Running an OpenMP Program

- Parallelism expressed by directives

```
c$omp parallel do private(i,j)
  do i = 1 , IXDIM
    do j = 1 , IYDIM
      a(i,j) = a(i,j ) + b (i,k) * c (kk,j)
    enddo
  enddo
c$omp end parallel do
```

- Parallel code uses Pthreads

```
• export OMP_NUM_THREADS=4
export KMP_MONITOR_STACKSIZE=200k
export KMP_STACKSIZE=2M
ifort -openmp -o myapplication myapplication.f
./myapplication
```

# Compiling and Running a MPI Program

- **Parallelism expressed by explicit calls of MPI library which performs data exchange via message passing**

```
call mpi_send ( c(1,k),IXDIM,mpi_real,left,  
+             iccol , mpi_comm_world, ierr )
```

- **Usually SGI MPI uses shared memory for data exchange**

- `ifort -o myapplication myapplication.c -lmpi`

OR

```
icc -o myapplication myapplication.c -lmpi
```

```
export MPI_DSM_VERBOSE=1
```

```
mpirun -stats -v -np 4 myapplication < input
```

# Shell Scripts: Why?

- **Let you create new commands.**
- **Gathering of individual commands of a repeated workflow.**
- **Easy way to remember and to use environment settings for a job.**

# Shell Scripts: Example for Running an OpenMP Job

```
#!/bin/sh -x
export OMP_NUM_THREADS=4
if [ "$1" != "" ];then
export OMP_NUM_THREADS=$1
fi
export F_UFMTENDIAN=big
export KMP_MONITOR_STACKSIZE=200k
export KMP_STACKSIZE=512000000
#
NO_OF_CPUS=`cat /proc/cpuinfo | fgrep processor | wc -l`
PEL=`expr ${NO_OF_CPUS} \- 1`
PE0=`expr ${PEL} \- ${OMP_NUM_THREADS} \+ 1`
BASEDIR=`pwd`
WORKDIR=/tmp/reiner/work_`date '+%m%d%y%H%M%S'`.${OMP_NUM_THREADS}
if [ ! -d ${WORKDIR} ]; then
    mkdir -p ${WORKDIR}
fi
cd ${WORKDIR}
cp ${BASEDIR}/insph_ns_bh_v1 .
cp ${BASEDIR}/ns14.00001 .
cp ${BASEDIR}/Shen_ASCII.dat .
cp ${BASEDIR}/sph_nsbh2 .
(/usr/bin/time dplace -x2 -c${PE0}-${PEL} ./sph_nsbh2 )>stdout \
2>&1
```



# Shell Scripts: Differences between Linux and SYSV/BSD UNIX

- In some cases the Linux ksh behaves much more like bash:

```
cat ${expid}.date | read year month day jobnum
```

- A subshell is spawned for read
- After termination of the subshell variables are left uninitialized!

```
if [ -f * ]; then ....
```

- \* does not expand under Linux! Looks for a file '\*'.

# Recommendations: make

- **Use make and Makefile to manage your software project**
  - **Let you describe dependencies between sources, objects and libraries.**
  - **Only refresh of items which are out of sync. Not a plain redo of the whole compilation.**
  - **Easy synchronization of with source code versioning systems like SCCS or CVS.**

# Example: Makefile

```
#Declarations
EXE=mxm4.mpi.x
OBS=mxm4.mpi.o
LIB=mylib.a
SYSLIBS=-Vaxlib -lmpi
FC=ifort
LD=ifort
FFLAGS=-openmp
LDFLAGS=-openmp
AR=ar
ARFLAGS=rv
#Implicit rules
.SUFFIXES:
.SUFFIXES: .a .o .f
.o.a:
    $(AR) $(ARFLAGS) $@ $<
    rm -f $*.o
.f.o:
    $(FC) $(FFLAGS) -c $<
```

```
#Explicit rules
all:$(EXE)
$(LIB):$(LIB)(setup.o) \
        $(LIB)(verify.o)

$(EXE):$(OBS) $(LIB)
    $(LD) $(LDFLAGS) -o $@ $(OBS)\
        $(LIB) $(SYSLIBS)
```

# Recommendations: Track Compiler Versions

- Sometimes a protocol how a binary was generated gets lost. How can I detect the version of the compiler afterwards?
- `objdump -j .comment -s <executable>` extracts the comments section which contains the command line options of your compile step:

```
VER=`ifort -V 2>&1 | awk ' /Build/ {print $7}'`  
ifort -Difort_version=$VER foo.c -o foo.exe  
objdump -j .comment -s foo.exe
```

```
0310 3032332f 696e636c 75646520 2d6f7065 023/include -ope  
0320 6e6d7020 2d446966 6f72745f 76657273 nmp -Difort_vers  
0330 696f6e3d 6c5f6663 5f70635f 382e312e ion=l_fc_pc_8.1.  
0340 30323320 2d632049 6e74656c 28522920 023 -c Intel(R)  
0350 466f7274 72616e20 436f6d70 696c6572 Fortran Compiler
```

# Lab

- **Go to the directory `SGI_programming_environment/labs/[fsrc_mp, csrc_mp]`.**
- **Inspect the Makefile and have a look how the library is created.**
- **Generate the hybrid application (MPI + OpenMP).**
- **Choose a small number of OpenMP threads.**
- **Start the hybrid code as a MPI job (4 MPI tasks).**
- **Play with tools like `top` and the object analyzers.**
  - On which CPUs is my job running?
- **Create a DSO from the static library.**

**sggi<sup>®</sup>**