



# Hands-on: debugging (Totalview, addr2line, GDB)

[a.marani@ Cineca.it](mailto:a.marani@ Cineca.it)



[www.cineca.it](http://www.cineca.it)



# Debugging on FERMI...

Debugging on FERMI is **no** easy task!

Error messages are often vague, and core files may be rather incomprehensible...

However, there are some useful tools that can help on the task!

Before that, let's see some general advice for the setting of a debug session

# Compiling for a debug session

**3 flags are required for compiling a program that can be analyzed by debugging tools:**

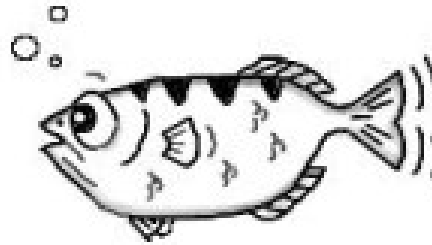
- g** : integrates debugging symbols on your code, making them “human readable” when analyzed from debuggers
- O0** : avoids any optimization on your code, making it execute the instructions in the exact order they’re implemented
- qfullpath** : Causes the full name of all source files to be added to the debug informations



## Other useful flags

- qcheck** Helps detecting some array-bound violations, aborting with SIGTRAP at runtime
- qflttrap** Helps detecting some floating-point exceptions, aborting with SIGTRAP at runtime
- qhalt=<sev>** Stops compilation if encountering an error of the specified lever of severity
- qformat** Warns of possible problems with I/O format specification (C/C++) (printf,scanf...)
- qkeepparm** ensures that function parameters are stored on the stack even if the application is optimized.

**GDB**



**GDB**  
The GNU Project  
Debugger

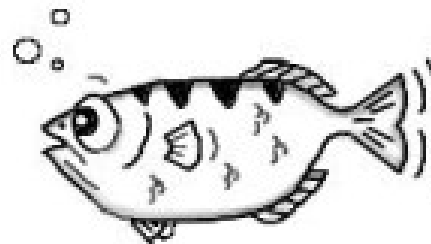
**addr2line**

**Totalview**





On FERMI, GDB is available both for front-end and back-end applications



# GDB

The GNU Project  
Debugger

Front-end: *`gdb <exe>`*

Back-end: *`/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gdb <exe>`*

It is possible to make a post-mortem analysis of the **binary** core files generated by the job

*`/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gdb <exe>  
<corefile>`*

To generate binary core file, add the following envs to runjob:

*`--envs BG_COREDUMPONEXIT=1`*

*`--envs BG_COREDUMPBINARY=*`*

‘\*’ means “all the processes”. It is possible to indicate which ranks generate its core by specifying its number



# GDB – remote access

The Blue Gene/Q system includes support for using GDB real-time with applications running on compute nodes.

IBM provides a simple debug server called gdbserver. Each running instance of GDB is associated with one process or rank (also called GDB client).

Each instance of a GDB client can connect to and debug one process. To debug multiple processes at the same time, run multiple GDB tools at the same time. A maximum of four GDB tools can be run on one job.



...so, how to do that?

# Using GDB on running applications

1) First of all, submit your job as usual;

```
llsubmit <jobscript>
```

2) Then, get your job ID;

```
llq -u $USER
```

3) Use it for getting the BG Job ID;

```
llq -l <jobID> | grep "Job Id"
```

4) Start the gdb-server tool;

```
start_tool --tool /bgsys/drivers/ppcfloor/ramdisk/distrofs/cios/sbin/gdbtool  
--args "rank=<rank #> --listen_port=10000" --id <BG Job ID>
```

5) Get the IP address for your process;

```
dump_proctable --id <BG Job ID> --rank <rank #> --host sn01-io
```





# Using GDB on running applications

6) Launch GDB! (back-end version);

```
/bgsys/drivers/ppcfloor/gnu-linux/bin/powerpc64-bgq-linux-gdb ./myexe
```

7) Connect remotely to your job process;

```
(gdb) target remote <IP address>:10000
```

**8) Start debugging!!!**

(Although you aren't completely free...for example, command 'run' does not work)

If nothing is specified, an unsuccessful job generates a text core file for the processes that caused the crash...

...however, those core files are all but easily readable!

```
++PARALLEL TOOLS CONSORTIUM LIGHTWEIGHT COREFILE FORMAT version 1.0
++LCB 1.0
Program : array_err.x
Job ID : 61743
Personality:
  ABCDET coordinates : 0,0,0,0,1,1
  Rank : 5
  Ranks per node : 4
  DDR Size (MB) : 16384
++ID Rank: 5, TGID: 113, Core: 4, HWTID:0 TID: 113 State: RUN
***FAULT Encountered unhandled signal 0x00000005 (5) (SIGTRAP)
Generated by interrupt.....0x00000005 (Program Exception IP=0x0000000001000438 ESR=0x0000000002000000: Trap )
While executing instruction at.....0x0000000001000438
Dereferencing memory at.....0x0000000000000000
Fault occurred at timebase.....0x00017349a2a0943d
Tools attached (list of tool ids).....None
Currently running on hardware thread....Y
General Purpose Registers:
  r0=0000000000000001 r01=0000001fbfffb9e0 r02=00000000015af7c0 r03=0000000000000005 r04=0000001fbfffbfa60 r05=0000000001548f60 r06=0000000001548f60 r07=000
00000000001
  r08=ffffffffffff0000 r09=000000000000005f r10=8080808080808080 r11=0000000000000000 r12=00000000010003f8 r13=0000001ec0507700 r14=0000000000000000 r15=000
00000000000
  r16=0000000000000000 r17=0000000000000000 r18=0000000000000000 r19=0000000000000000 r20=0000000000000000 r21=0000000000000000 r22=0000000000000000 r23=000
00000000000
  r24=0000000000000000 r25=0000000000000000 r26=00000000013cb4c0 r27=0000001fbfffc100 r28=0000000001565b90 r29=0000000000000000 r30=00000000015a1e68 r31=000
000015a1e80
Special Purpose Registers:
  lr=0000000001000408 cr=0000000084000222 xer=0000000020000000 ctr=000000000000000d
  msr=000000008002f000 dearr=0000000000000000 esr=0000000002000000 fpscr=0000000000008000
  sprg0=0000000000000000 sprg1=0000000000000000 sprg2=0000000000000000 sprg3=0000000000000000 sprg4=0000000000000000
  sprg5=0000000000000000 sprg6=000000000000044800 sprg7=0000000000000000 sprg8=0000000000000000
  srr0=0000000000000438 srr1=0000000000002f00 srr2=0000000000000000 srr3=0000000000000000 mcsrr0=0000000000000000 mcsrr1=0000000000000000
  dbcrr0=0000000000000000 dbcrr1=0000000000000000 dbcrr2=0000000000000000 dbcrr3=0000000000000000 dbar=0000000000000000
Floating Point Registers:
  f0=5500002000000000 1000008800200001 0000000000000000 0000000000000000 f01=0000000000000000 0000000000000000 0000000000000000 0000000000000000
  f02=0000000000000000 0000000000000000 0000000000000000 0000000000000000 f03=0000000000000000 0000000100000000 0000000000000000 0000000000000000
  f04=0000000000000000 0000000000000001 0000000000000001 0000000000000003 f05=0000000000000000 0000000000000000 0000000000000000 0000000000000000
  f06=0000000000000000 0000000000000000 0000000000000000 0000000000000000 f07=0000000000000000 0000000000000000 0000000000000000 0000000000000000
  f08=0000000000000000 0000000000000000 0000000000000000 0000000000000000 f09=0000000000000020 0000000000000020 0000000000000020 0000000000000020
  f10=4059000000000000 4059000000000000 4059000000000000 4059000000000000 f11=4059000000000000 4059000000000000 4059000000000000 4059000000000000
  f12=3fe0000000000000 3fe0000000000000 3fe0000000000000 3fe0000000000000 f13=0000000000000001 0000000000000000 0000000000000000 0000000000000000
  f14=0000000000000000 0000000000000000 0000000000000000 0000000000000000 f15=0000000000000000 0000000000000000 0000000000000000 0000000000000000
  f16=0000000000000000 0000000000000000 0000000000000000 0000000000000000 f17=0000000000000000 0000000000000000 0000000000000000 0000000000000000
  f18=0000000000000000 0000000000000000 0000000000000000 0000000000000000 f19=0000000000000000 0000000000000000 0000000000000000 0000000000000000
  f20=0000000000000000 0000000000000000 0000000000000000 0000000000000000 f21=0000000000000000 0000000000000000 0000000000000000 0000000000000000
  f22=0000000000000000 0000000000000000 0000000000000000 0000000000000000 f23=0000000000000000 0000000000000000 0000000000000000 0000000000000000
  f24=0000000000000000 0000000000000000 0000000000000000 0000000000000000 f25=0000000000000000 0000000000000000 0000000000000000 0000000000000000
  f26=0000000000000000 0000000000000000 0000000000000000 0000000000000000 f27=0000000000000000 0000000000000000 0000000000000000 0000000000000000
```

**addr2line** is an utility that permits to get from this file informations about where the job crashed



# Core files

Blue Gene core files are lightweight text files

Hexadecimal addresses in section STACK describe function call chain until program exception. It's the section delimited by tags: +++STACK / ---STACK

+++STACK

| Frame Address     | Saved Link Reg   |
|-------------------|------------------|
| 0000001ffffff5ac0 | 000000000000001c |
| 0000001ffffff5bc0 | 00000000018b2678 |
| 0000001ffffff5c60 | 00000000015046d0 |
| 0000001ffffff5d00 | 00000000015738a8 |
| 0000001ffffff5e00 | 00000000015734ec |
| 0000001ffffff5f00 | 000000000151a4d4 |
| 0000001ffffff6000 | 00000000015001c8 |

---STACK

In particular, “Saved Link Reg” column is the one we need!



# using `addr2line`

From the core file output, save only the addresses in the Saved Link Reg column:

```
0000000000000001c
000000000018b2678
000000000015046d0
000000000015738a8
000000000015734ec
0000000000151a4d4
000000000015001c8
```

Replace the first eight 0s with 0x:

```
000000000018b2678 => 0x018b2678
```

On page 88 of IBM Redbook “Blue gene/Q application development” you can find a Perl script (`bgqtranslate.pl`) that can do the replacement for you

Lauch `addr2line`:

```
addr2line -e ./myexe 0x018b2678
```

```
addr2line -e ./myexe < addresses.txt
```



TotalView is a GUI-based source code defect analysis tool that gives you control over processes and thread execution and visibility into program state and variables.



It allows you to debug one or many processes and/or threads with complete control over program execution.

Running a Totalview execution in back-end can be a bit tricky, as it requires connection from FERMI to your local machine via ssh tunneling to VNC server.

# Using Totalview: preliminaries

In order to use Totalview, first you need to have downloaded and installed VNCviewer on your local machine.

(<http://www.realvnc.com/download/viewer/>)



Windows users will also find useful Cygwin, a Linux-like environment for Windows. During installation, be sure to select “openSSH” from the list of available packages.

(<http://cygwin.com/setup.exe>)



Once all the required softwares are installed, we are ready to start preparing our Totalview session!





# Using Totalview: preparation

- 1) On FERMI, load tightvnc module;  
*module load tightvnc*
- 2) Execute the script vncserver\_wrapper;  
*vncserver\_wrapper*
- 3) Instructions will appear. Copy/paste to your local machine (Cygwin shell if Windows) this line from those instructions:  
*ssh -L 59xx:localhost:59xx -L 58xx:localhost:58xx -N <username>@login<no>.fermi.cineca.it*  
where *xx* is your VNC display number, and *<no>* is the number of the front-end node you're logged into (01,02,07 or 08)
- 4) Open VNCViewer. On Linux, use another local shell and type:  
*vncviewer localhost:xx*  
On Windows, double click on VNCviewer icon and write *localhost:xx* when asked for the server. Type your VNC password (or choose it, if it's your first visit)

## Using Totalview: job script setting

- 5) Inside your job script, you have to load the proper module and export the DISPLAY environment variable:

```
module load profile/advanced totalview
```

```
export DISPLAY=fen<no>:xx
```

where *xx* and *<no>* are as the above slide (you'll find the correct DISPLAY name to export in vncserver\_wrapper instructions)

- 6) Totalview execution line (inside your LoadLeveler script) will be

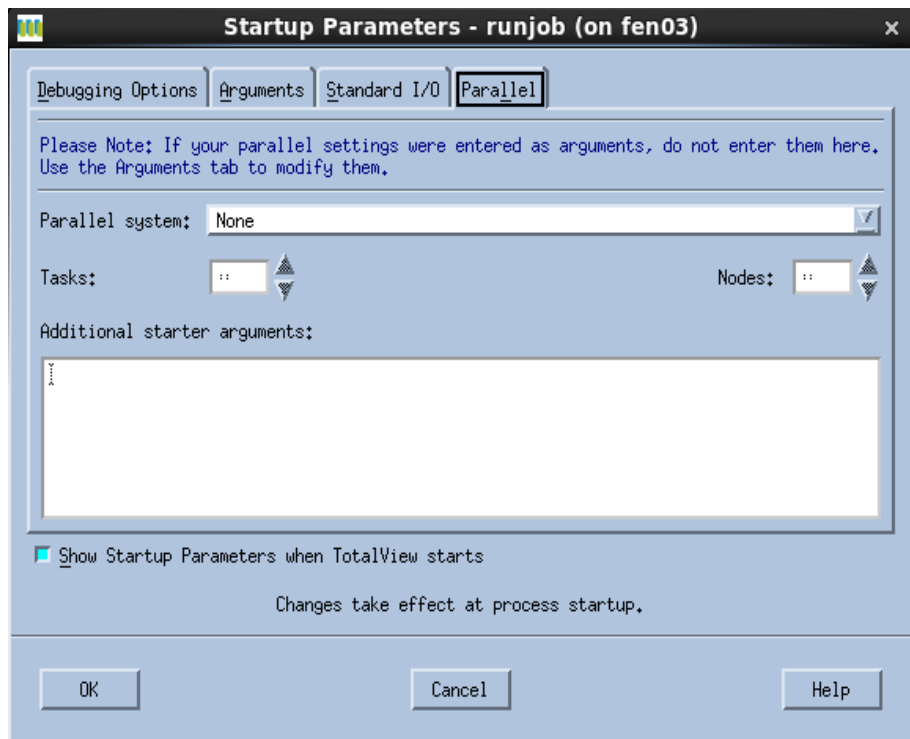
as follows:

```
totalview runjob -a <runjob arguments: --np, --exe, --args...>
```

- 7) Launch the job. When it will start running, you will find a Totalview window opened on your VNCviewer display!  
Closing Totalview will also kill the job.



# Using Totalview: start debugging



Select “BlueGene” as a parallel system, and a number of tasks and nodes according to the arguments you gave to runjob during submission phase.

Click “Go” (the green arrow) on the next screen and your application will start running.

**WARNING:** due to license issues, you are NOT allowed to run Totalview sessions with more than 64 tasks simultaneously!!!



# Out from Totalview

When you've finished using Totalview, please follow this procedure in order to close the session safely:

1) Close VNCviewer on your local machine;

2) Kill the VNCserver on FERMI:

*vncserver kill :x*

*x* is the usual VNC display number, without the initial 0 (if present);

3) On your first local shell, close the ssh tunneling connection with CTRL+C.

# Totalview Remote Display Client

An easier (and maybe safer) way to use Totalview is Totalview **RDC** (Remote Display Client), a simple tool that helps with submitting a job already setted with the proper characteristics (and with no VNC involved)

**TotalView Remote Display Client**

File Help

**TotalView TECHNOLOGIES**

Session Profiles:

- VSC
- VSC bash
- VSC csh

1. Enter the Remote Host to run your debug session:

Remote Host:  User Name:  Commands:

2. As needed, enter hosts in access order to reach the Remote Host:

|   | Host                 | Access By | Access Value         | Commands             |
|---|----------------------|-----------|----------------------|----------------------|
| 1 | <input type="text"/> | User Name | <input type="text"/> | <input type="text"/> |
| 2 | <input type="text"/> | User Name | <input type="text"/> | <input type="text"/> |

3. Enter settings for the debug session on the Remote Host :

TotalView | MemoryScope

Path to TotalView on Remote Host:

Arguments for TotalView:

Your Executable (path & name):

Arguments for Your Executable:

Submit Job to Batch Queueing System:

4. Enter batch submission settings for the Remote Host :

PBS Submit Command:

TotalView PBS Script to Run:

Additional PBS Options:

No session running

RDC procedure isn't fully operative yet, since we encountered some firewall issues that lead to different behaviours depending on the single workstation settings

Our System Administrators are looking into it.  
Connecting with RDC will be soon a possibility!!

