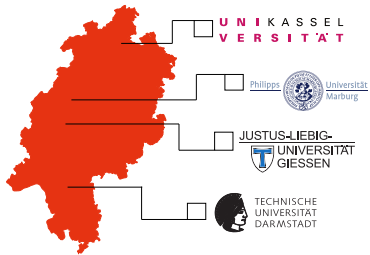
Debugging & Totalview

Hessisches Kompetenzzentrum für Hochleistungsrechnen (HKHLR)

Dr. Christian Iwainsky

V1.0



HKHLR is funded by the Hessian Ministry of Sciences and Arts

HESSEN

Introduction to debugging and Totalview Part I

Topics

- Motivation,
- Debugging and Software Development,
- Environment setup, and Starting Totalview.

"Example 1"

```
1  void foo () {
2      int * myPointer;
3
4      int i;
5
6      for (i=0;i<100;i++) {
7          myPointer[i]=i*i;
8      }
9
10     return myPointer;
11 }
```

"Fix for Example 1"

```
1  void foo () {
2      int * myPointer =
3      (int*) malloc(sizeof(int)*100);
4      int i;
5
6      for (i=0;i<100;i++) {
7          myPointer[i]=i*i;
8      }
9
10     return myPointer;
11 };
```

FIX: Add a properly casted `malloc` to allocate sufficient memory!

"Example 2"

```
1  void doScience () {
2      int * myPointer =
3      (int*) malloc(sizeof(int)*100);
4
5      ...  // Do some work here
6
7
8      return;
9  }
10 int main(){
11     doScience();
12
13     ... // more work here
14 }
```

"Fix for Example 2"

```
1  void doScience () {
2      int * myPointer =
3      (int*) malloc(sizeof(int)*100);
4
5      ...  // Do some work here
6
7      free(myPointer);
8      return;
9  }
10 int main(){
11     doScience();
12
13     ... // more work here
14 }
```

FIX: Add a properly casted `malloc` to allocate sufficient memory!

"Example 3"

```
1  int getPowersOf2_64Bit() {
2      int po2[64];
3
4      po2[0]=1;
5      for (int i=1;i<64;i++)
6          po2[i]=po2[i-1]*2;
7      return po2;
8  }
9  int main(){
10     int powersOf2[64];
11     powersOf2 = getPowersOf2_64Bit();
12
13     ... // more work here
14
15
16     return 0;
17 }
```

"Fix for Example 3"

```
1  int getPowersOf2_64Bit() {
2      int po2 =
3      (int*) malloc(sizeof(int)*64);
4      po2[0]=1;
5      for (int i=1;i<64;i++)
6          po2[i]=po2[i-1]*2;
7      return po2;
8  }
9  int main(){
10     int * powersOf2;
11     powersOf2 = getPowersOf2_64Bit();
12
13     ... // more work here
14
15     free(powersOf2);
16     return 0;
17 }
```

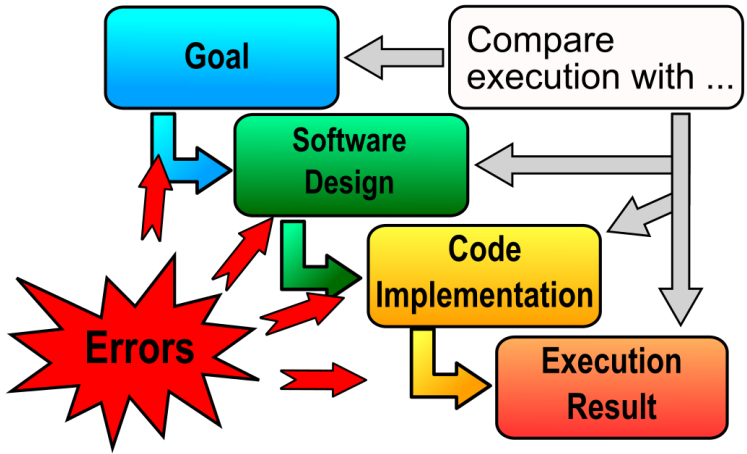
FIX: Add `malloc` to allocate memory, do not return local memory.

Debugging

- **Debugging** is the process of finding and resolving **bugs**
- A **bug** is a defect or problem that prevents the correct operation, results in unintended behaviour, or generates the wrong output
- Debugging strategies may involve
  - Interactive methods, incl. monitoring application, or system
  - Analytical methods, incl. control-flow analysis, log-file analysis, analysis of memory dumps, profiling and tracing,
  - Testing methods: unit testing, integration testing

Src.:wikipedia.org/wiki/Debugging

Debug observations against all levels; check intent vs observation:



TotalView is a comprehensive debugging solution for demanding parallel and multi-core applications: **Interactive debugging**, **Analyzing core-dumps** and **live program inspection**.

- Wide compiler & platform support: C/C++, Fortran, UPC, Assembly and Python.
- Integrated Memory Debugging
- **Reverse Debugging**
- Concurrency & HPC debugging support: **MPI** and **OpenMP**

- Prepare and load minimal Totalview environment:

shell

```
>$ module load totalview
```

- Totalview uses X-Windows to display its UI. A guide for the X-environment is available at HPC-Wiki.info→Linux in HPC→SSH Graphics and File Transfer.
- Two user-interfaces:

The new UI, new features continuously added.

sell

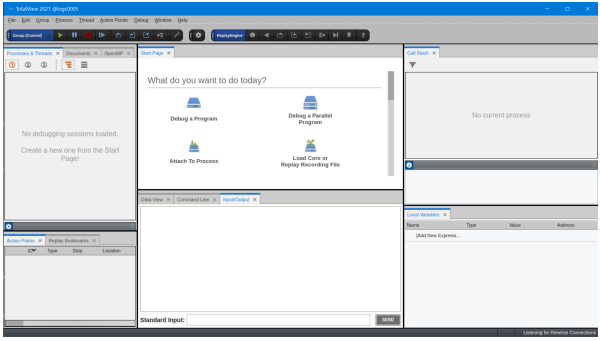
```
>$ totalview -newUI
```

Classic UI, feature complete, to be phased out.

shell

```
>$ totalview -classicUI
```

- New program `Ctrl-N`,
- New parallel program `Ctrl-Shift-P`,
- Attach to running program `Ctrl-T`, and
- Core or Replay file `Ctrl-Shift-L`.



We will discuss Totalview using the program found in the **demo01** folder: It recursively computes factorials. There is a bug regarding the ordering of the programs output.

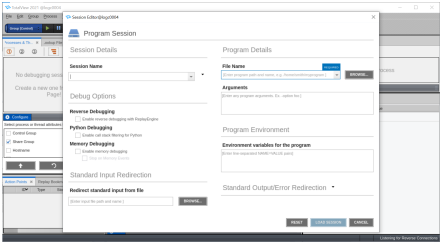
- INPUT.txt — Some example inputs
- Makefile — Maketargets
- demo01.cc — Original sourcefile
- demo01B.cc — Intermediate solution

The makefile has four targets:*demo01.exe*,*demo01A.exe*, *demo01B.exe* and *clean*.
The program accepts input via STDIN or as program arguments.
Please consult the readme.md for more details.

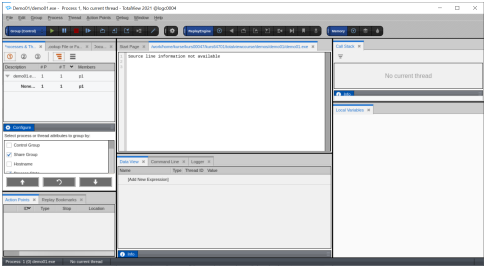
shell

```
>$ ./demo01.exe 1 3 5 7
```

- Session name: enabels to load a previous configuration
- Program: Absolute or relative path to binary / only binary if in $PATH



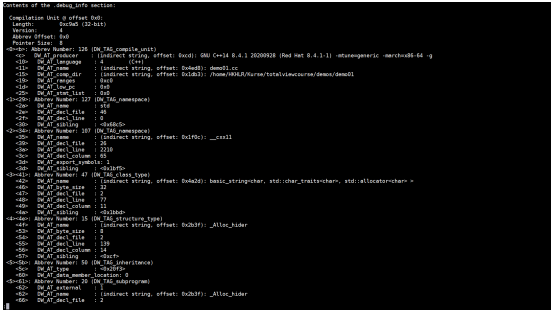
The main window provides a source-code view.
A lack of debug information results in an empty screen or an assembly view[a].



[a] depending on the version of Totalview

Two requirements:

1. Debug markers must be present in binary
   Use compiler option **-g** for every source-file to be debugged
2. Source-code must be present at the original location
   debug information stores the line-numbers and path to the source-file in the executable.



Shell

```
>$ checkDebugInfo.sh <path to binary>
```

- Helper tool by HKHLR.
- Available in "course material → scripts" directory

Example for missing debug infos:

```
[kurs64701@logc0004 demo01]$ checkDebugInfo.sh demo01.exe
No Debuginformation found in demo01.exe
```

Example for available debug infos:

```
[kurs64701@logc0004 demo01]$ checkDebugInfo.sh demo01A.exe
"demo01A.exe" contains debug information.
The following source files were compiled with -g:
        /home/kurse/kurs00047/kurs64701/totalviewcourse/demos/demo01/demo01A.cc
```

We recomment to always use the following compiler flags when debugging:

- -g to get debug informantion and see the source code
- -O0 to *disable* compileroptimization such as code motion, so that one can actually follow along the code when it is executed in the debugger
  - An example why -O0 is important is included as **demo4**
    Without -O0, you can see that step by step execution skips some of the loop header (the line with the for) entirely, because the compiler can optimize it. For example it may fuse two loops into one when the index range is the same.

This segments contents:

- Three examples for bugs and solutions,
- the software development process and bugs,
- required software-environment,
- starting Totalview with the new user-interface,
- the main windows shows source-code and
- debug-information and required compiler flag.