Valgrind Helgrind/DRD and Intel Thread Checker

Uladzimir Karneyenka
Computer Science Department
San Jose State University
San Jose, CA 95192
408-924-1000
Uladzimir.Karneyenka@sjsu.edu

1. ABSTRACT

The purpose of this project is to demonstrate and compare the usage of the thread error analyzers provided with Valgrind suit and Intel Inspector XE 2013 on Linux platform. Demonstration part would consist of installation procedures and requirements for the tools and usage. Comparison is done on several high-level criterias - cost, usability, performance, etc.

2. INTRODUCTION

This paper demonstrates and compares two thread error analyzers - Intel Inspector XE 2013 and Valgrind. The Intel's product is an enterprise grade software with extensive capability and customizability. Valgrind, on the other hand is an open-source software focused strictly to extract as much relevant information about the executable as possible to facilitate the debugging and optimization. These two products share quite a bit of similarity, and if an experienced developer decides to switch from one to another, he would be able to do so, quickly and harmlessly. Although, as it will be shown, it is probably best not to pick sides, but use both tools in conjunction with each other.

3. INSTALLATION

3.1. Intel Inspector XE 2013 installation

Installation on Linux Mint 15 of this tool is quite seamless - just have to download the installation tarball from the Intel website (https://software.intel.com/en-us/intel-inspector-xe), unpack it, and then, run the installation script which can be found in the directory which was created when the download tarball was unpacked. There are two installation scripts available - shell-based and GUI-based -, I used the former with the default setup which puts the installation into /opt/intel directory.

Intel Inspector XE 2013 is not a freeware, but it has a trial period of 30 days for a developer to figure out whether this tool is the right choice. Nevertheless, even to install a trial version you have to provide a valid serial key. Intel asks some information, such as a valid email address, before one can download the tarball, and that email is where Intel sends the trial serial key, which you will have to provide during the installation process.

To run Intel Inspector XE 2013, the user have to source its environment script - **inspxe-vars.sh** -, which could be found under <*installation_dir*>**/inspector_xe_2013**. Personally, I just sourcing it in the **.bashrc** so I would not have to worry about forgetting doing so everytime I open a new terminal. At this point, the tool should work without any further adjustments. But, if you are running a recent distribution of Linux Ubuntu, or LInux Mint, then, you might have to modify your system by setting a value stored in **/proc/sys/kernel/yama/ptrace_scope** to 1. If you encounter some other issues, I recommend to skim through Release Notes document which should be stored in <*installation_dir*>**/inspector_xe_2013/documentation/en/** by the installer.

3.2. Valgrind installation

Valgrind toolset installation is a bit more involved if compared to Intel Inspector. Also, by itself, Valgrind is shell based, so all the interactions are performed through the terminal. Luckily, there are available GUI implementations by third parties, and for this demo I am using Valkyrie, an "in-house" GUI maintained by the Valgrind developers.

To install Valgrind, I downloaded a tarball with a source code from http://valgrind.org. The installation procedure for most open source software is usually similar to each other, and Valgrind follows the same pattern. Unpack the tarball and note the directory where unpacked files are. In terminal, navigate to the directory with the unpacked files, and run/execute **configure** script. If the configuration step passed, in the same terminal, run/execute **make**, and on

success, **make install**. For my system, I had all the dependencies met (such as, having g++ installed) so, there weren't any surprises. Complications started popping up when I tried to install Valkyrie.

Valkyrie GUI is developed on Qt4 framework, and as such, in order to build the source code, you have to have Qt4 installed on your system, with all the libraries and binaries visible system-wide. In turn, Qt4 installation/build has some dependencies which must be met. Fortunately, Qt4 has a very good documentation, and it specifically lists the required library dependencies. For example, the dependencies for Linux based platforms, Qt4 dependencies are listed at (http://qt-project.org/doc/qt-4.8/requirements-x11.html), and coincidentally, I had some libraries missing, and had to install them manually. For instance, I had Xext missing on my system, and per notes listed on that page, I installed libxext-dev development package using **apt-get** tool. After installing the dependencies for Qt4 and installing Qt4 framework itself, I also build and installed Valkyrie GUI. Unfortunately, the version I downloaded had a slight setback - in the source code, a system header file was not listed which cause a compile time error complaining about having unresolved symbols. After researching on Google search and modifying the source code by listing a missing header file, the build and installation went through and I was able to run Valkyrie on top of Valgrind.

4. EXAMPLE FROM INTEL INSPECTOR

4.1 Analyze using Intel Inspector XE 2013

As any other enterprise level software Intel Inspector XE 2013 comes with some tutorials and examples. I used one of the examples specifically designed for the multithreading debugging to learn how to use and navigate through the interface in order to localize possible problems. Example by itself, while having a bit convoluted source code, used a very simple example of a data race condition, where a single variable was accessed and modified by two threads.

In order to analyse a possible problematic multithreaded code using Intel Inspector, the developer has to follow several simple steps (from Intel Documentation):

1. Set up the project
    a. Create a project
    b. Configure and select executable to analyse.
   Collect results
. Configure the analyser
a. Run the executable
   Investigate

4.1.1. Setting up a project

For the best result, Intel recommends that the executable in question has to be build using the following set of rules:

- Enable debug information
- Disable compiler optimizations
- Use dynamic runtime library
- Disable compiler flag responsible for runtime error checks

These rules are recommended in order to avoid false positives and to have source code information during analysis.

The procedure to set up a project for the executable is straightforward. Launch the Intel Inspector from the terminal by running **inspxe-gui** command which should bring up the GUI for the Intel Inspector. Choose **File > New > Project…**, which brings up a dialog with two fields - **Project Name**, and **Location**. **Project Name** can be anything meaningful to the developer. **Location** field can be left untouched, which would let Intel Inspector to save all the project related information on its own, also, this field is not for the location where your source code or executable, it is for the project's housekeeping. After you done filling these fields, hit **Create Project**, which should bring you to the **Project Properties** dialog box. **Project Properties** dialog has several tabs, but the one of interest is **Target**. Under this tab, three field are of outmost importance - **Application**, **Application parameters**, and **Working directory**. While, they are pretty much self-explanatory, I would like to point out that **Working directory** should point out to the directory your application is expecting to be run from, otherwise, it might not pick up some dependences - resources or libraries - that were referenced from inside the executable by something other than absolute address.

4.1.2. Collecting results and Investigate

Intel Inspector is not just a tool for multithreading debugging, it's a toolset consisting of several debugging tools that allow threading error analysis, memory error analysis, and customized analysis. For this exercise, we are interested in a threading error analysis tool, and to selected it, choose **File > New > Analysis**, which should open a new tab in the main window (Figure 1.). There, the developer can customize the complexity and the behavior of the analyser. Intel Inspector allows not only analyse, detect and show the problematic areas in the code, but it also allows to run the debugger right on the spot where and when the problem occurs.

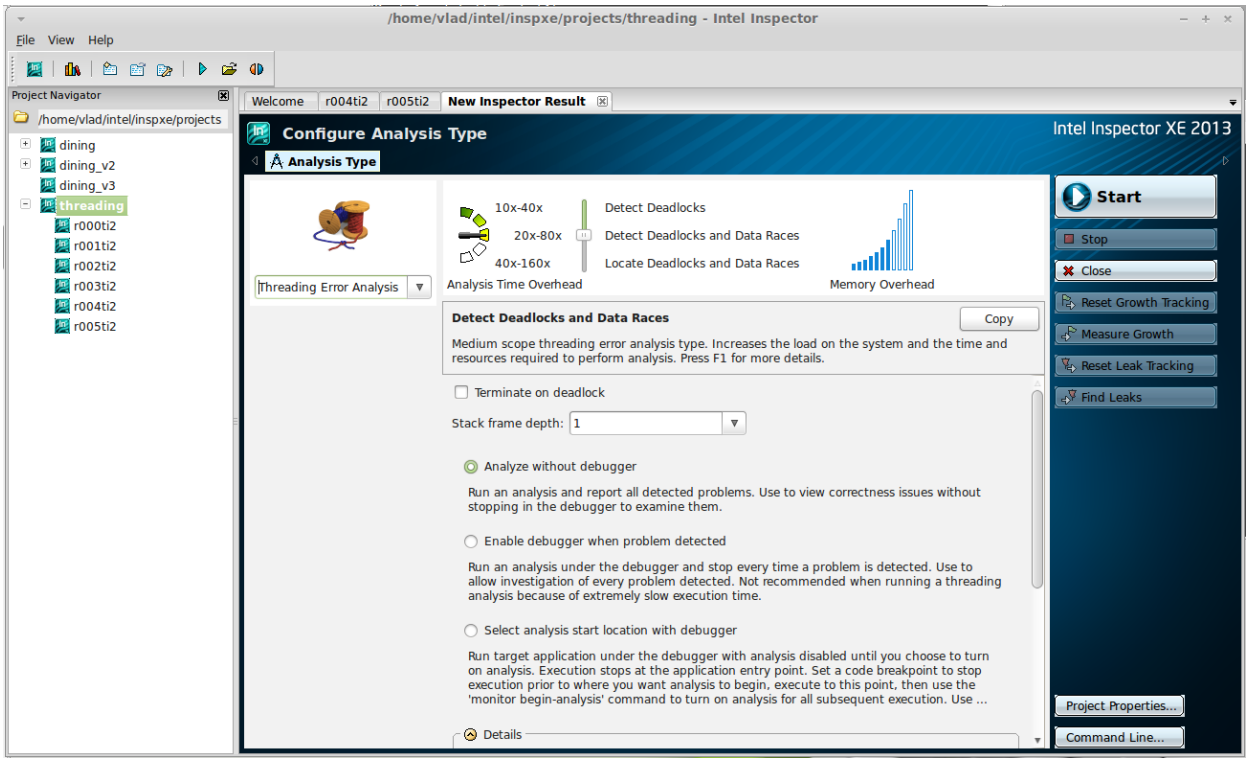

Figure 1 Analysis configuration window

After the configuration for the desired analysis, the developer can start the program execution, whence Intel Inspector would collect and process the runtime information. Upon completing the execution of the program in question, Intel Inspector will automatically navigate to **Summary** tab(Figure 2.), which is effectively is a hub for further investigation by the developer.

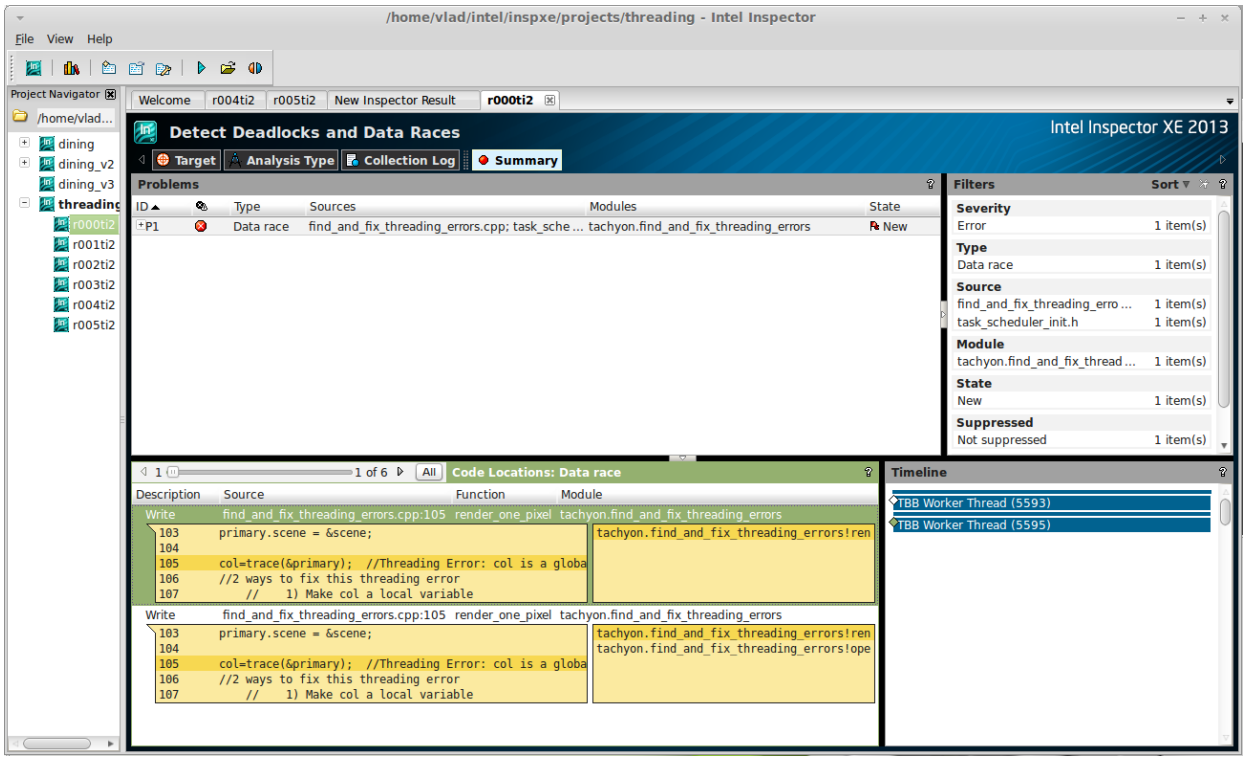

Figure 2 Analysis summary window

For this example, it is clearly shown that there is a data race in two threads, which are trying to modify the same variable. It is quite helpful to get the results from the analysis in such a vivid form, but Intel Inspector doesn't stop there - all the objects in the Analysis summary window are instrumented with context menus and actions. The developer can easily navigate to the original source code and work on it in order to remedy the problem. Intel Inspector comes with a greate documentation that explain what type of a data race it found and what are the possible solutions. Furthermore,by the means of Intel Inspector, the developer can select a particular problem from the list found, investigate it further, debug it, adjust/fix it, rebuild, and re-analyse in a continuous cycle till the problem is gone.

4.2 Analyze using Valgrind/Helgrind/Valkyrie

The procedure for thread error analysis is very much the same in Valgrind as in Intel Inspector, but with some tiny intricacies.

Setting up a project

If Valgrind and Valkyrie were installed properly, launch Valkyrie by running **valkyrie** in the terminal. That should bring up the Valkyrie GUI.
To create a project, choose **File > New Project…**. This will bring up a dialog box that will ask for the project directory where, as in Intel Inspector, all the project housekeeping files will be kept. It does not necessarily have to be the location where the executable-under-test or the source code is. After the project's directory is set and project created, the developer has to set up the tool to run the executable-under-test. To do so, choose **Edit > Options**. In the dialog that opens, the developer has to set up Valkyrie options to point to the location of the executable-under-test and the working directory. If everything was set correctly, Valgrind/Valkyrie is ready to analyse the executable.

4.2.1 Collecting results and Investigate

In Valgrind toolset, the counterparts for Intel Inspector Thread Error Analyzer are Helgrind and DRD. Helgrind is a bit more advanced thread analyzer than DRD and produces more accurate results; also, Valkyrie implements the interface for Helgrind, so, for this exercise, only Helgrind will be used to analyze the executable.

Valkyrie can work with two tools from Valgrind toolset - Memcheck and Helgrind. So, the developer has to set Helgrind to be the current tool for the analysis, which accomplish simply by choosing **Tools > Helgrind**. Valkyrie does not list any particular options for Helgrind, so there is not much customization available.

To start Helgrind, choose **Process > Run**. The output of the Helgrind's analysis is dumped into a XML file under

project's directory, and the Valkyrie's window shows the output in a more organized fashion (Figure 3).

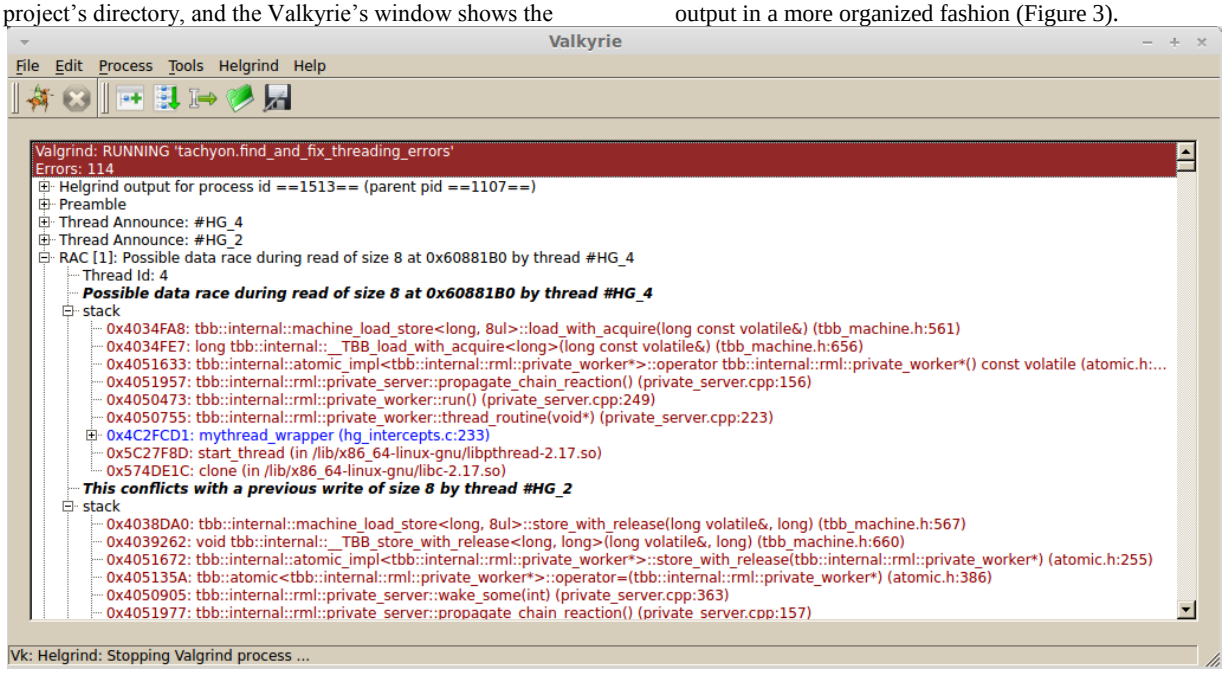

Figure 3 Helgrind results view

Valkyrie parses Helgrind's output XML file and lists errors in the order as they occur during the runtime. The executable that was analyzed in this example is the same that was analyzed in Intel Inspector, and the error count and error location is different. Nevertheless, as Figure 4 shows, Helgrind also detects the same error and location Intel Inspector found in its analysis run.

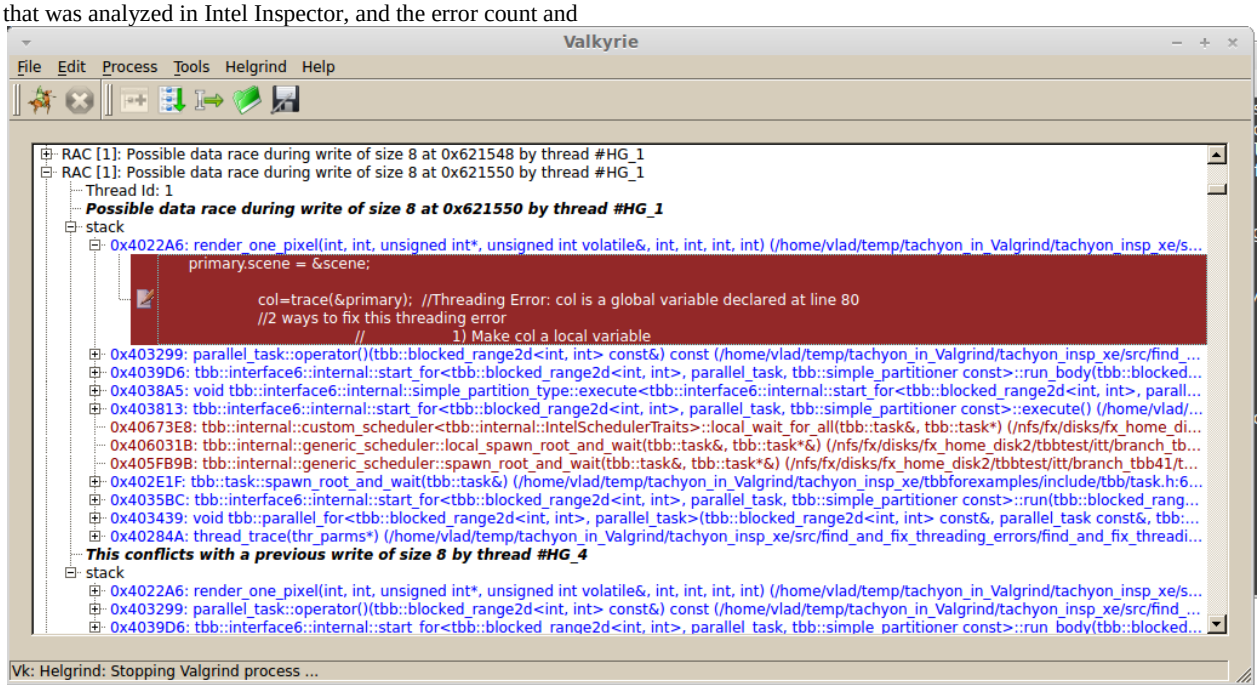

Figure 4 Helgrind result view with expected error found

Valkyrie allows for minor formatting adjustment for the output to improve readability, while maintaining the availability of the information. Also, as Intel Inspector, Valkyrie allows quick jump to the source code, saving the developer time searching for the source file and line manually.

5. TOOL COMPARISON

Table 1. Product comparison

	Valgrind	Intel Inspector XE
Installation	High	Low
Usage complexity	Low	Low
Presentation complexity	High	Low
Instrumentation	Low	High
Performance	Low	High
Cost	Free	$800+ (Free 30 days trial)

Installation on Linux is always a challenge, especially for a development tools. And while the complexity of installing Valgrind/Valkyrie was not of a great surprise, the ease of installation of Intel Inspector XE 2013 was pleasantly harmless and fast. Installing Valgrind/Valkyrie without deeper understanding of the OS setup could be really time-consuming and aggravating.

Usage complexity on both - Valgrind/Valkyrie and Intel Inspector is quite straightforward, no misleading options and configurations. The time required to set up the project and start working on the problems at hand was reasonably short.

Presentation complexity "award" definitely goes to Valgrind/Valkyrie. There is no doubt that in order to sieve through the output produced by Helgrind some substantial knowledge and understanding of the errors dumped is required. It is possible, that Intel Inspector also detected those numerous errors found by Helgrind, but it, probably, did some post-processing in order to get rid of all possible "false positives" and irrelevant. On the other hand, it is possible that Helgrind was created by developers for developers with more strict assumption in order not to miss something that might turn out to be very much relevant.

Instrumentations of the results on Intel Inspector is extremely helpful. Effectively, Intel Inspector plays a role of a developer's hub, where the developer can analyze, debug, and repair his code from within one program, thus, reducing turn-around time. From this point of view, Valgrind/Valkyrie loses a bit, nevertheless, it does not lose where it matters the most - finding the problems in the the code.

Comparing **Performance** between these two products could be unjust to any of them. In order to fairly compare the performance, one has to get into tiny details of what analyzers are reading, checking, skipping, testing, etc. The lack of expertise prevents me to state any opinion on the matter. But, it seems, that running Intel Inspector analyzing an executable is a bit faster than doing so using Valgrind/Valkyrie. The probable cause for Valgrind/Valkyrie to lag being Intel Inspector is the fact that the output from Helgrind is stored in XML format which then read into Valkyrie where it get's parsed to generate GUI representation, which might be very time consuming.

6. CONCLUSION

Using Intel Inspector XE 2013 to work on the problem was very productive when I was working on Dining Professors example code I wrote to try out these analyzers. From the point of view of a beginner, Intel Inspector is extremely helpful and resourceful in dealing with threading errors. But, I assume, as a beginner developer would gain more experience, Valgrind should not be overlooked. The reason for that, is while smart analyzers are really helpful by simplifying things and can speed up the debugging stage, sometimes they can be too smart, hiding some details that might be actual problems.

7. REFERENCES

[1] Intel Inspector XE 2013 Help files
[2] intel.com
[3] valgrind.org