The OpenSHMEM Analyzer

Version 1.0

Oak Ridge National Laboratory
Computer Science Department and Mathematics Division
Computer Science Research Group
Extreme Scale Systems Center

http://www.csm.ornl.gov/essc/

University of Houston
Computer Science Department
High Performance Computing Tools Group

http://web.cs.uh.edu/~hpctools/

July 9, 2014

Contents

List of Figures

Preface

This User's Manual describes the OpenSHMEM Analyzer [2], a tool that provides source code analysis and correctness checks capabilities to the user for OpenSHMEM programs. It provides a range of information about the source program in textual or graphical format. The OpenSHMEM Analyzer relies on web browsers and the graph library GraphViz to render its graphs and use of hyperlinks to navigate to the source code and warning messages.

Audience Description

This manual is a user's guide only. Users are expected to have a basic knowledge of the structure of programs and some experience with C and OpenSHMEM programming. The OpenSHMEM Analyzer is available on most GNU/Linux distributions (e.g. Red Hat and SUSE) with x86/x86-64 processors. It is assumed that users are familiar with the basic commands on these systems.

Organization

This User's Guide is structured into the following sections and appendices:

Section 1: Introduction gives an overview of the OpenSHMEM Analyzer Tool and describes its major goals;

Section 2: Basic Usage of the OpenSHMEM Analyzer describes naming conventions, generated files and how to visualize the graphs and traverse the source code;

Section 3: Features of the OpenSHMEM Analyzer describes the features for local and global analysis of OpenSHMEM programs and how to interpret the information within the graphs;

Appendix A: Download and Install the OpenSHMEM Analyzer describes how to obtain and install the OpenSHMEM Analyzer either as a pre-built executable or in source form;

Appendix B: Ongoing and Future Work is a quick view of on-going work of the OpenSHMEM Analyzer and its future functionality.

Status of Work

The tool is currently in its infancy and there are many enhancements to be made, both with analysis capabilities and with its use and invocation.

1 Introduction

In this chapter we give an overview of the overall features of the OpenSHMEM Analyzer and describe the structure of the system.

1.1 Major Goals of the OpenSHMEM Analyzer

Whenever application software is developed or ported to OpenSHMEM from a new or an existing one, the source code must be carefully analyzed in order to understand many details of the current implementation while at the same time to avoid common errors in OpenSHMEM applications. The OpenSHMEM Analyzer is a tool to support an application developer or code owner who wishes to understand their C application better. It provides a range of information including the structure of a source program in a graphical browsable form. The current input languages for the OpenSHMEM Analyzer are C/C++, OpenSHMEM API 1.0.

OpenSHMEM is a standard for SHMEM library implementations. Many SHMEM libraries exist but they do not conform to a particular standard and have similar but not identical APIs and behavior, which hinders portability. However, significant user efforts are required to parallelize serial codes with OpenSHMEM or further analyze and optimize the performance of OpenSHMEM programs. The OpenSHMEM Analyzer, with its comprehensive intra- and inter-procedural analysis information, can be an indispensable assistant for writing, analyzing and optimizing OpenSHMEM applications. The OpenSHMEM Analyzer has an interactive component that provides a graphical user interface using HTML to display its output and to navigate the source code and error messages within them.

The OpenSHMEM Analyzer is an on-going research project developed collaboratively by Oak Ridge National Laboratories and the University of Houston, with funding from DOD. Its functionality is based on the OpenUH compiler infrastructure, which is maintained by the HPCTools Group at the University of Houston.

The OpenSHMEM Analyzer is intended to be an open source tool available for the OpenSHMEM community.

2 Basic Usage of the OpenSHMEM Analyzer

In this chapter the user will find:

- some hints on preparing programs for the OpenSHMEM Analyzer;
- invoking the OpenSHMEM Analyzer for the first time;
- naming/graphical conventions;
- how to visualize the results or manipulate graphs.

2.1 Preparing Your Program for the OpenSHMEM Analyzer

The first step of preparing a program to be analyzed by the OpenSHMEM Analyzer is to compile it using the tool with a special set of flags. The user is responsible for modifying the application's makefiles, in order to reflect the compiler and flags setting changes. Since the OpenSHMEM Analyzer is based on OpenUH, the following are the main drivers/compilers:

uhcc	C compiler
uhCC	C++ compiler (beta evaluation)

In order to prepare your program, you must add the following flags: -shmem-analyzer -O3 for the compile (-c) and link commands as follows.

Compile commands:

```
$ uhcc -shmem-analyzer -O3 -c myfile.c
$ uhcc -shmem-analyzer -O3 -c myfile2.c
```

Link command:

```
$ uhcc -shmem-analyzer -O3 myfile.o myfile2.o -o myprogram
```

Note: To avoid long compilation times, the less aggressive optimization flag -O2 can be used instead.

In this example, the OpenSHMEM Analyzer will generate a series of files with extensions html, gif, dot, map, and msg.

This flag will, in future, allow additional information to be passed into the Analyzer, so that analyses specific to a language or library can be selected.

2.2 Using the OpenSHMEM Analyzer

There are 2 modes of visualizing the results of the OpenSHMEM Analyzer:

1. when you build your program with the OpenSHMEM Analyzer, it will display the warnings at the command line;

2. the OpenSHMEM Analyzer is also able to display the warnings in a browser together with the program callgraph.

2.2.1 Filename Conventions

The OpenSHMEM Analyzer creates several files with different extensions. The following command:

```
$ uhcc -shmem-analyzer -O2 myfile.c -o myprogram
```

will generate the following files:

myprogram.html	This is the .html file that can be open with a browser. It contains the callgraph and the warning messages from the tools
myprogram.msg	Contains the error messages collected during inter-procedural analysis phase
myfile.c.html	This will contain the HTML version of the source code with syntax highlighted and line number target links
myfile.c.msg	List of error messages generated during intra-procedural analysis phase
myprogram.dot	This file contains the callgraph of the application in the GraphViz format
myprogram.gif	This file contains the .gif image version of the callgraph

2.3 How the OpenSHMEM Analyzer works

The OpenSHMEM Analyzer relies on intra-procedural and inter-procedural analysis to detect potential semantic program errors in the application. The types of warnings reported are displayed during the various compilation phases of the tool. Figure 1 shows how the different phases generate the analyses:

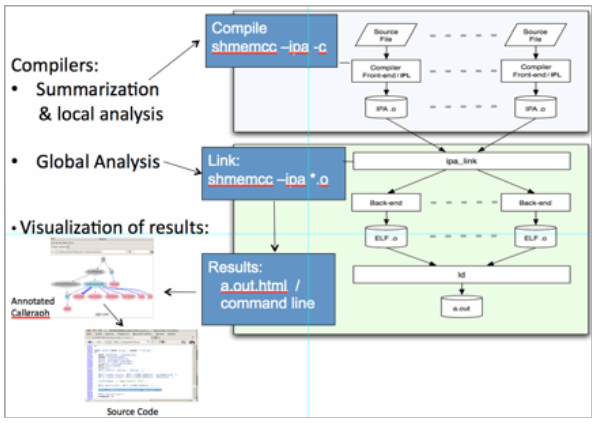

Figure 1: Phases generating the different analyses

The OpenSHMEM Analyzer generates warning messages at different compilation stages of the code. The warning messages are then displayed at the command line or together in the

callgraph. The OpenSHMEM Analyzer does not produce an executable file (in future releases this feature will be enabled).

3 Features of the OpenSHMEM Analyzer

3.1 Overview

The OpenSHMEM Analyzer will generate an HTML file with the callgraph of a program that can be related to the source code via HTML links. In addition, the main HTML file contains a list of warning messages generated by the tool which can be related to the source code via HTML links.

The following command opens the HTML file that contains the callgraph and the warning messages of the program (based on the previous example):

```
$ firefox myprogram.html
```

(Or use any web browser of your choice.)

3.1.1 Callgraph

The call graph gives the structure of the program, where there is a node for each procedure in the program and a directed edge linking a pair of nodes if and only if the procedure corresponding to the source node may invoke the sink node's procedure at run time. If you click on a node, the source text of the corresponding procedure will be displayed in the HTML browser. The edges of the graph represent the different callsites where the procedures are invoked. The user can click on the edges to relate them to the source code. The callgraph nodes are colored in the following format:

Procedures containing OpenSHMEM calls	light blue
Procedures not containing OpenSHMEM calls	silver
Procedures representing OpenSHMEM	pink

For the edges, callsites to OpenSHMEM are colored:

I/O operations (i.e. puts, gets)	blue
Reductions	purple
Broadcasts	red
Atomics	orange
Memory management calls (i.e. dynamic allocation / deallocation of symmetric memory)	yellow
State calls (i.e. num_pes, my_pe)	green
Synchronization calls	red

Figure 2 shows a portion of a callgraph generated for the IS NAS Parallel Benchmark.

In addition to the callgraph, the color legend for the nodes and edges is displayed in Figure 3

When a node or a callsite is clicked, the browser will display the corresponding source code with highlighted syntax formatted in HTML, as in Figure 4.

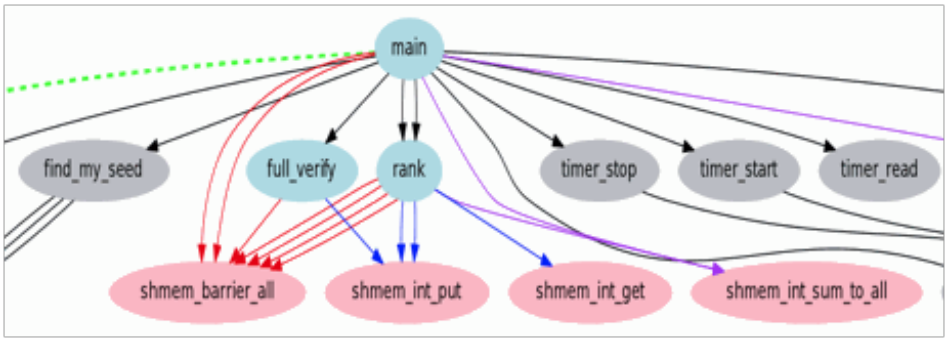

Figure 2: The OpenSHMEM callgraph of the IS NAS parallel benchmark

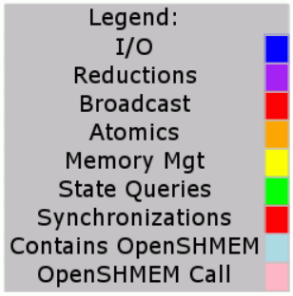

Figure 3: The color legend for the callgraph of an OpenSHMEM program

```
 1 #define N 8
 2 #define M 10
 3 #include <shmem.h>
 4 int targg[N];
 5 long froml[N];
 6 int srcg[N];
 7
 8 int main(void) {
 9       int i, src[N];
10       long lget[N];
11       static int targ_static[N];
12
13       start_pes(0);
14
15       for(i=0; i< N; i++) {
16          src[i] = my_pe() + i;
17          froml[i] = my_pe() + i*i;
18       }
19
20       shmem_int_put(targg, srcg, N+M, 2);
21       shmem_int_put(targ_static, src, N+M, 3);
22       shmem_long_get(lget,froml,N+M,4);
23
24       shmem_barrier_all();  /* sync sender and receiver */
25       return 1;
26
27 }
28
29 void start_pes(int i){}
```

Figure 4: Application source code as displayed in the browser

3.2 Displaying the OpenSHMEM Analyzer Warnings

The OpenSHMEM Analyzer is able to display OpenSHMEM warning messages together with the callgraph. The warning messages are also displayed when the tool is invoked at the command line. When the warning messages are displayed with the callgraph, each message contains an HTML link that relates the message to the source code. Figure 5 is an example of how the OpenSHMEM Analyzer displays its warning messages together with the source code.

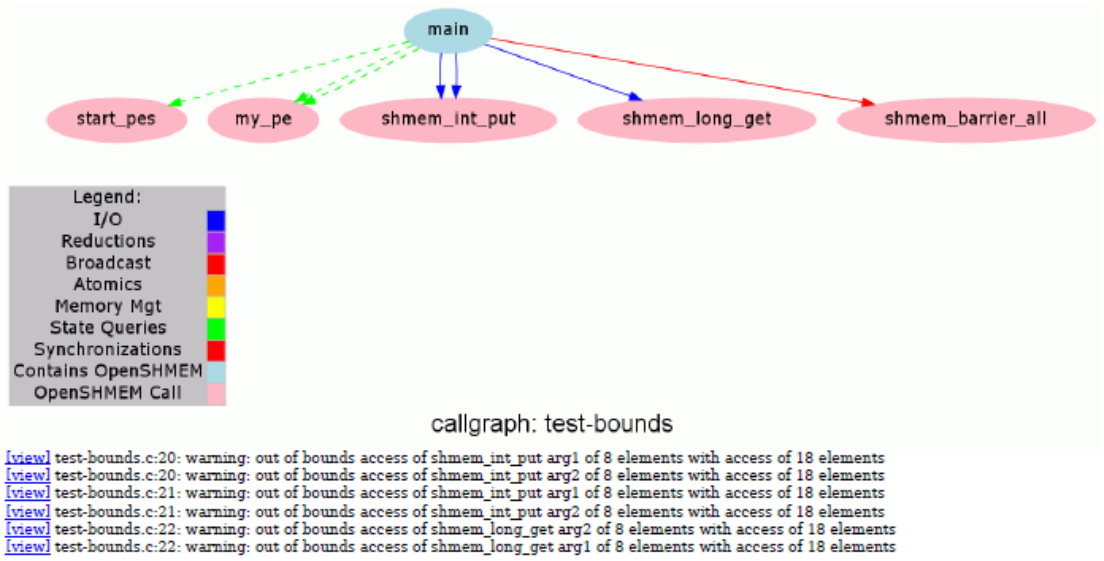

Figure 5: Callgraph with the OpenSHMEM Analyzer warning messages

The messages contain the line number and the source file name of the OpenSHMEM call that is triggering the warning.

3.3 Analyses available with the OpenSHMEM Analyzer

The OpenSHMEM Analyzer is able to perform the following analyses:

- check the ordering of OpenSHMEM initialization and finalization calls;
- check if there is an OpenSHMEM initialization call in the program;
- check if there is more than one OpenSHMEM initialization call;
- check if OpenSHMEM calls are using symmetric memory variables;
- check for out-of-bounds access for arrays in OpenSHMEM calls;
- check for out-of-bounds strided access of arrays;
- check if one of the arguments of the OpenSHMEM calls evaluates to NULL;
- perform constant propagation and common sub-expression elimination to simplify the arguments of an OpenSHMEM call (useful for checking if some arguments evaluate to constants);
- check if pointers to symmetric data are allocated with shmalloc calls;
- check if any variables used in expressions of pointer arithmetic with symmetric variables are initialized;
- check if global pointers to symmetric data are initialized;
- check if pointers to symmetric data are aliased with other pointers that can have side-effects to the OpenSHMEM call;
- check if the OpenSHMEM calls arguments are of the correct types or storage allocation for OpenSHMEM typeless calls.

3.4 The OpenSHMEM Analyzer Warning Messages

The following sections show examples of warning messages which the Analyzer tool can output.

3.4.1 Multiple OpenSHMEM initialization calls

```
int main(...)  {                    void sub (...) {
    ...                                 ...
    start_pes();                        start_pes();
    sub(...);                       }
}
```

```
warning: more than one OpenSHMEM initialization call found
```

3.4.2 Non-symmetric variable used in OpenSHMEM call

```
int sub(...) {
  long target, source;
  ...
  shmem_long_get(target, source, ...);
}
```

```
badget.c:65: warning: non-symmetric variable in arg2 of shmem_long_get
```

3.4.3 Out-of-bounds accesses in OpenSHMEM call

```
#define N 8
#define M 8

int sub (...) {
  ...
  static int targ[N], src[N];
  ...
  shmem_int_put(targ, src, N + M, 2);
  ...
}
```

```
test-bounds.c:20: warning: out of bounds access of shmem_int_put arg1 of 8
elements
```

3.4.4 Out-of-bounds accesses in OpenSHMEM call determined with constant propagation (must use -O3)

```
#define N 8
#define M 8

int sub (...) {
  int len = N;
  static int targ[N], src[N];
  ...
  for(i = 0; i < M; i++) len++;
  ...
  shmem_int_put(targ, src, len, 2);
  ...
}
```

```
test-bounds-constprog.c:23: warning: out of bounds access of shmem_int_put
arg1 of 8 elements with access of 16 elements
```

3.4.5 Out-of-bounds accesses in OpenSHMEM call with strided access

```
int src2[N];
...
int sub(...) {
  ...
  shmem_iget32(dest2, src2, 1, 2, N, npes - 1);
  ...
}
```

```
test_shmem_get_globals.c:297: warning: out of bounds access of shmem_iget32
arg2 of 7 elements with access of 14 elements
```

3.4.6 Pointer is initialized with the wrong memory allocator (non-symmetric)

```
int sub(...) {
  ...
  float *y;
  ...
  y = (float *) malloc((n_local1 - n_local0 + 2) * sizeof(float));
  ...
  shmem_float_put(&y[n_local0-1+1], &y[n_local1-1], 1, _my_pe() + 1);
  ...
}
```

```
shmem_heap.c:35: warning: variable arg1 of call shmem_float_put is
initialized with malloc
```

3.4.7 Global Pointer is not initialized

```
float *y;

int sub(...) {
  ...
  shmem_float_put(&y[n_local0-1+1], &y[n_local1-1], 1, _my_pe() + 1);
  ...
}
```

```
shmem_heap-global.c:20: warning: global variable arg1 of call shmem_float_put
is uninitialized
```

3.4.8 Wrong storage type for typeless OpenSHMEM call

```
long lfrom;

int sub(void) {
  ...
  shmem_get32(lget, lfrom, N, 4);
  ...
}
```

```
test-types.c:23: warning: wrong storage class of shmem_get32 arg2 of 8 bytes
```

4 Acknowledgements

This work is supported by the United States Department of Defense and used resources of the Extreme Scale Systems Center located at the Oak Ridge National Laboratory and the HPCTools Group at the University of Houston.

Appendices

A Download and Install the OpenSHMEM Analyzer

A.1 Download Location

The project website for the OpenSHMEM Analyzer is:

`http://www.openshmem.org/OSA`

A.2 Installation

The OpenSHMEM Analyzer is available in the following ways:

A.2.1 Pre-built Executable

For immediate use, a tarball of the OpenSHMEM Analyzer, called

`openuh-3.0.38-x86_64-bin.tar.bz2` [1]

can be downloaded via the project website above. Then

1. extract the contents of the tarball to a directory, call it `prefix`;
2. prepend the directory `prefix/openuh-3.0.38/bin` to your `PATH` environment variable.

A.2.2 Source

The full source code of the OpenUH compiler containing the OpenSHMEM Analyzer can also be downloaded from a repository via the project website above.

Configuration is via the common GNU Autotools `configure` command, so you can build *in situ* or in a separate build directory. The OpenSHMEM Analyzer will be built when the `--enable-osa` configure flag is used. Then do the usual `make`/`make install` sequence.

A.3 Tool Infrastructure Documentation

A paper describing the OpenUH compiler that is the infrastructure of the OpenSHMEM Analyzer is in [1].

[1] 3.0.38 is the version at time of writing but will change in the future.

B Ongoing and Future Work

We plan to develop a stand-alone user interface to present an interactive callgraph and control flow graph that are aware of OpenSHMEM calls and develop a triage between these different views. The idea is that there will be a data flow view that originates from OpenSHMEM calls and enables the user to see the use-define chains that can help to keep track of the use and definition of pointers in OpenSHMEM calls. This will help the user evaluate the placement of a particular call and how it is related to the rest of the application more intuitively. In addition, we could develop a view to see how symmetric variables are accessed in the entire application, since symmetric variables can affect many procedures (e.g. global variables, interprocedural pointers). These sorts of views are important to understand the overall side effects of the application.

The OpenSHMEM Analyzer is moving toward a proper infrastructure to perform parallel data flow analysis is needed, and apply the state-of-the-art and how this relates to the context of OpenSHMEM. As a first step, we will explore the concept of program slicing at the process level, which is the process for separating the different control flow graphs from different processes. The idea is to simplify the control flow graphs of a program per process and correctly denote synchronizations across them. Through graph analysis we can classify the different process control-flowgraphs into subsets that can be used to represent optimizations and where classical program optimizations can be applied per OpenSHMEM task.

We plan to explore how to integrate better the OpenSHMEM Analyzer to the OpenSHMEM library in a way that is more compiler friendly. We need to make sure the library implementation allows the compiler to analyze and optimize it together with the application. This will include the inlining of all OpenSHMEM calls, their specialization to the application context, constant propagation/dead code elimination, and elimination of redundant runtime checks. The OpenSHMEM Analyzer could perform checks at compile time and define a set of assertions that we can enforce at runtime to make sure the library is run in the right context, reducing runtime checks and overheads [4, 3].

We will also explore how to integrate dynamic information to the OpenSHMEM Analyzer. This will mean integrating it with performance tools such as TAU and the OpenSHMEM Tracer. We will combine the OpenSHMEM Analyzer instrumentation with the OpenSHMEM wrappers of these tools to gather calculate frequently executed paths in the control graph and callgraph or values of variables at a given point of execution. This will help toward feedback and present to the user a dynamic callgraph and control flow graph, that shows the execution code coverage and the frequently accessed code path that can be specialized.

References

[1] Barbara Chapman, Deepak Eachempati, and Oscar Hernandez. Experiences Developing the OpenUH Compiler and Runtime Infrastructure. *International Journal of Parallel Programming*, pages 1–30, 2012.

[2] Oscar Hernandez, Siddhartha Jana, Swaroop Pophale, Stephen Poole, Jeffery Kuehn, and Barbara Chapman. The OpenSHMEM Analyzer. In *The proceedings of the Sixth Conference on Partitioned Global Address Space Programming Models, Santa Barbara, CA, USA*, October 2012.

[3] Swaroop Pophale, Oscar Hernandez, Stephen Poole, and Barbara Chapman. Static Analyses for Unaligned Collective Synchronization Matching for OpenSHMEM. In *Proceedings of the Seventh Conference on Partitioned Global Address Space Programming Model*, PGAS '13, 2013.

[4] Swaroop Pophale, Oscar Hernandez, Stephen Poole, and Barbara Chapman. Extending the OpenSHMEM Analyzer to Perform Synchronization and Multi-valued Analysis. In Stephen Poole, Oscar Hernandez, and Pavel Shamis, editors, *OpenSHMEM and Related Technologies. Experiences, Implementations, and Tools*, volume 8356 of *Lecture Notes in Computer Science*, pages 134–148. Springer International Publishing, 2014.