SANDIA REPORT

SAND2006-3515
Unlimited Release
Printed June 2006

Porting Salinas to the Windows Platform

Garth Reese, Sandia National Laboratories
Christopher Riley Wilson, Tactical Staffing Resources

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Sandia National Laboratories

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.doe.gov/bridge

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online ordering:	http://www.ntis.gov/ordering.htm

SAND2006-3515
Unlimited Release
Printed June 2006

Porting Salinas to the Windows Platform

Garth M. Reese and Christopher Riley Wilson
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185

Abstract

The ASC program has enabled significant development of high end engineering applications on massively parallel machines. There is a great benefit in providing these applications on the desktop of the analysts and designers, at least insofar as the small models may be run on these platforms, thus providing a tool set that spans the application needs.

This effort documents the work of porting Salinas to the WINDOWS™ platform. Selection of the tools required to compile, link, test and run Salinas in this environment is discussed. Significant problems encountered along the way are listed along with an estimation of the overall cost of the port.

This report may serve as a baseline for streamlining further porting activities with other ASC codes.

Contents

Appendix

Chapter 1

Introduction

The Department of Energy ASCI program began development of massively parallel applications in 1998. Sandia has been actively developing these applications, both within the SIERRA framework, and others. The primary focus of the development is the “big iron” type machines – machines with many processors that run huge analysis programs.

To a large extent, this effort has been very successful. For example, many analyses for weapon systems are now performed using these codes. The state of the art in analysis has grown tremendously, and verified analysis is now viable for issues which could never previously be performed. International recognition for these unique capabilities has also been demonstrated (as in the 2002 Gordon Bell award for **Salinas**).[1] However, significant issues have developed in the entire analysis process.

We find that many analysts use a smaller, serial code on their WINDOWS™ workstation. Small design models are developed and analyzed. As the models grow they may get too large for the desktop. The current process requires the analyst to throw out all the existing models, and develop and new mesh and model in an entirely different format. Many aspects of the analysis change, from the pre-processor, through analysis to the post-processing. As a consequence much effort is lost, and time is wasted. Most importantly, the analysts must now be expert in both analysis codes. This provides a tremendous burden to analysts, and is a significant barrier to adoption of ASCI codes throughout our analyst community.

To allay that concern, we began a port of **Salinas** to the WINDOWS™ platform. This will allow analysts to consistently use the same tool from initial design through final analysis. It will explore aspects of workflow that are difficult to evaluate without this port. By this we mean to determine if a windows port will provide sufficient capability to meet the needs of the analysis community.

The **Salinas** port is seen as a trail blazing activity for other SIERRA based codes. We anticipate that many of the hurdles that we have to pass for this port will be similar for SIERRA tools. Note however, that the windows port for SIERRA tools is significantly more complex for a large variety of reasons.

The port from the Unix environment to WINDOWS™ is not expected to be trivial. There will be many code changes required. The build environment is impacted, and perhaps most importantly, the run environment is quite different. However, because **Salinas** has a serial version running on the Unix platform we anticipate that the porting effort will be reasonable. We are engaging the SIMBA[2] team to assist in the development of a graphical user environment – clearly a task for which the **Salinas** team is not well suited.

Chapter 2

Requirements

The following goals were established in the porting process.

- The port must not be too costly.

  We estimate the total time to complete this port, including machine setup and configuration, to take about 3 weeks for one person. This assumes one week for initial configuration, one week porting TPLs, and one week compiling and linking salinas.

- The executable must run without access to any special libraries or DLLs.
- A user should be able to install the new executable using a simple process such as *unzip*.
- We must be able to test the port.
- The configuration must be easily archived, so that the build can be repeated on a different machine with the same tools.
- The focus is on maintenance and support, rather than on performance. This is a convenience port, rather than a high performance port. As such, decisions like compiler selection, will be slanted towards support.
- Eventually we want to access the executables and related files using a graphical user interface, such as that provided by SIMBA. However, this is beyond the scope of the current effort.

Chapter 3

Initial Configuration

3.1 Compiler Selection

The primary requirement in choosing a Windows compiler suite is that it have both Fortran and C++ compilers. Many C++ compilers exist on the Windows platform, but there are far fewer Fortran compilers. The four primary candidates we considered were the following:

- Absoft
- Intel
- PGI
- Open Watcom

We chose to use the Intel v9.1 compilers for a few reasons. First, **Salinas** has already been compiled using the Intel compilers on multiple systems. This will cause us to encounter fewer compiler incompatibilities, even though operating system incompatibilities still exist. Intel compilers offer a robust set of supported language features and also generally produce better optimized executables than the other candidates, though performance is a secondary concern at this point in the porting process.

Though we do not plan on using it directly, the Intel compilers use various libraries and header files from Microsoft Visual C++. Since this compiler is relatively inexpensive compared to the Intel compilers, this is not seen as a large deterrent.

3.2 Make Environment Selection

While our primary build system uses the Unix make utility, we also support building with the *SCons* utility. *SCons* is a python based tool that is very versatile and reliable. Since make is not available on the Windows platform, *SCons* will be used.[*]

Since the configuration files are stored in text format, they can easily be archived in our cvs repository, and they allow us to replicate our builds on any similar architecture. The only requirement is that the same compilers are present on the machine as well as a version of python.

3.3 Hardware Configuration

No unusual hardware was required. The port was done on a standard HP XW4200 Workstation with 3.8GHz/800 Intel Pentium 4 processor and 2GB of RAM.

[*] One requirement to our port is that the software be easily installed on the analyst's machines. This precludes the use of Cygwin for the run time environment which effectively requires the installation of a new shell.

Chapter 4

Setting up the Environment

Once the compiler version issues were overcome (see the Compiler Version Incompatibilities section 6.1), the installation and configuration was very easy. Microsoft Visual Studio was installed first. The only additional configuration performed after initial installation was to set up the environment variables. Microsoft provides a script called *vsvars32.bat*, which allows you to run its compilers from the command line, but it must be done every time you compile. All the script really does is set some environment variables, so these values were added to the static environment using the Advanced tab under System Properties.

After Visual Studio was installed, the Intel compilers came next. They were very easy to install and even updated the environment variables automatically, so no additional work was required.

Note that if using the *SCons* build tool, you must make sure the external environment is propagated to the build environment as this is not the default behavior.

Chapter 5

Compiling Salinas

Salinas had been ported to many different platforms before this port was started. It has also been ported with the Intel compilers, so there were not that many difficulties when compiling the Salinas source files. Minor code changes were necessary in various files, which won't be described here. The larger issues are discussed below.

5.1 Missing Header Files

The most notable code changes were required because WINDOWS™ has no equivalent header file to *unistd.h*. As a result, we had to replace the file ”read” and ”write” operations with the equivalent stream implementation. This was a little time consuming, but not very difficult because streams have their own read and write operations, which map fairly well to the *unistd* versions.

Windows also has no *strings.h* header file. The primary uses for Salinas from this file were the case insensitive string comparisons ”strcasecmp” and ”strncasecmp.” Windows does have equivalent versions of these functions with different names. The following lines were added to Salinas to resolve the issue:

```
#define strcasecmp(a,b) _stricmp(a,b)
#define strncasecmp(a,b,c) _strnicmp(a,b,c)
```

5.2 Reading and Writing Text Files

On Unix-like systems, each line of a text file is terminated with a newline character (\n). However, Windows text files have a carriage return character (\r) in addition to a newline character. When this two character combination is read from a file, an automatic translation is performed, so that only a newline is returned to the reading program. Only one byte of data is reported as read, even though in reality two bytes were read from the file. This discontinuity causes serious problems when using the commands ”fseek” and ”ftell”, which seek to a specific point in the file or report the current location in the file. Simple uses like seeking to the start of the file work fine, but any seeking to an offset from the current location in the file does not work.

Salinas only had two instances of *fseek* that had to be removed. Both were overcome by more careful bookkeeping while parsing the file. The other uses of *fseek* are either to the beginning of the file or to a location directly obtained from a call to *ftell* (as opposed to an offset from this location).

5.3 Building Third Party Libraries (TPLs)

Most of the TPLs we build rely on a Make system, which is not available on Windows. However, we were able to use our *SCons* build tool to compile most of them. Since it was already set up for our Salinas build, it was fairly easy to write a single configuration file for each TPL that described which files to compile. Then, *SCons* used the same build options we had already set up for the Salinas build. Once the individual files were compiled, *SCons* was able to pack them into a library so we can link with them.

The *netcdf* library presented the only major exception to the *SCons* build strategy. Because *netcdf* had already been ported to Windows, we chose to use the existing Visual Studio build environment. Although Visual Studio was used to handle the bookkeeping tasks, the files were still compiled using the Intel compilers. During installation, the Intel compilers can integrate themselves into Visual Studio. We simply had to open their project file (.sln) into Visual Studio, and configure it to use the Intel compilers to produce a static library.

Chapter 6

Major Issues in the Process

6.1 Compiler Version Incompatibilities

We started this port after a new version of the Microsoft compiler had recently been released. Because Visual Studio 2005 was new, the Intel 9.0 compilers were not yet compatible with it. The fact that most retailers stopped selling older versions of the Microsoft compilers when 2005 was released compounded the issue. We were able to start our initial port using a beta version of Intel's 9.1 compiler, which did support the new Microsoft Visual Studio.

The final version of the Intel 9.1 compilers has since been released and the transition from beta to release was without issue.

6.2 Test Suite

After the initial port was completed, and we had a Salinas executable, we needed to test for its correct operation. The original Unix test suite relies heavily on *csh* scripts, which are not supported on Windows. There are some implementations of csh-like environments, but we were unsure if these would work and whether or not they were worth the effort to try.

We considered the option of running our tests using the Sierra based SNtools runtest utility. However, at the time of this port, the SNtools have not been ported to Windows, so this was not a viable alternative.

We decided to use the Linux-like *Cygwin* environment for our testing. We definitely did not want our main port to rely on *Cygwin*, but it was acceptable to use it to run our test suite. Here, we found no trouble running our *csh* scripts and the test suite performed as it does on all other platforms.

Chapter 7

User Interface

As with Unix-like platforms, a standard command line interface is available through Windows' DOS-like shell. However, most Windows users prefer Graphical User Interfaces over the command line, so we are pursuing a relationship with the SIMBA[2] team to provide a new interface. SIMBA provides a GUI wrapper for traditionally command line based applications. This should make Salinas easier to use in the WINDOWS™ environment. The interaction is illustrated in Figure 7.1.

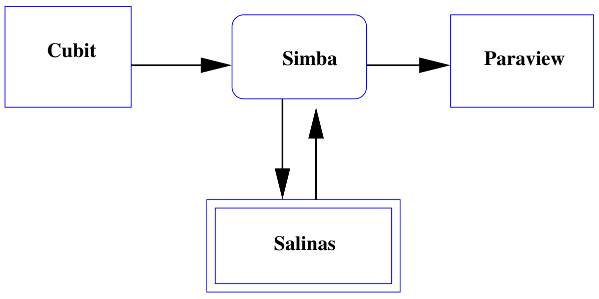

Figure 7.1. GUI interaction model

References

[1] Bhardwaj, M., Pierson, K., Reese, G., Walsh, T., Day, D., Alvin, K., Peery, J., Farhat, C., and Lesoinne, M., "Salinas: A Scalable Software for High-Performance Structural and Solid Mechanics Simulations," in *Supercomputing*, Baltimore, MD, November 2002, Gordon Bell Award winner.

[2] Whiteside, R., "Computer Sciences and Information Technologies - SIMBA," Tech. Rep. http://www.ca.sandia.gov/8900/simba.php, Sandia National Laboratories.

Appendix A

Summary of Effort

Table A.1 summarizes the time spent in each of the individual efforts to port **Salinas** to the WINDOWS™ platform.

Table A.1. Cost of Porting

Activity	**Cost**
Hardware Selection	1 day
Compiler Selection	1 day
Make environment	2 days
Hardware	$2000
Computer setup	1 hour
Configuration	4 days
Building TPLS	2 days
Compiling Salinas	2 days
Number of Code changes	100 lines
Linking Salinas	1 day

DISTRIBUTION:

1	0380	Morgan, Harold S, 1540
1	0847	Wilson, Peter J, 1520
1	0847	Baca, Thomas J, 1523
3	0847	Bitsie, Fernando, 1523
1	0847	Freymiller, James Eban, 1523
1	0557	Griffith, Daniel, 1523
1	0847	Holzmann, Wil A, 1523
1	0847	Kmetyk, Lubomyra N, 1523
1	0847	Miller, A Keith, 1523
1	0847	Rice, Amy E, 1523
1	0557	Simmermacher, Todd W, 1523
1	0847	Tipton, D Gregory, 1523
1	0847	Fulcher, Clay W G, 1526
1	0847	Redmond, James M, 1526
1	0382	Bova, Steven W, 1541
1	0382	Domino, Stefan Paul, 1541
1	0382	Gianoulakis, Steven E, 1541
1	0382	Glass, Micheal W, 1541
1	0382	Lober, Randy, 1541
1	0382	Lorber, Alfred A, 1541
1	0382	Okusanya, Tolulope O, 1541
1	0382	Subia, Samuel, 1541
1	0380	Alvin, Kenneth F, 1542
1	0380	Blanford, Mark L, 1542
1	0380	Crane, Nathan K, 1542
1	0380	Fortier, Harrison, 1542

1	0380	Gilmartin, William B, 1542
1	0380	Gullerud, Arne S, 1542
1	0380	Hales, Jason D, 1542
1	0380	Heinstein, Martin W, 1542
1	0380	Key, Samuel W, 1542
1	0380	Koteras, J Richard, 1542
1	0380	Olivas, Bryan Ray, 1542
1	0380	Perschbacher, Brent M, 1542
1	0380	Pierson, Kendall Hugh, 1542
1	0380	Porter, Vicki L, 1542
1	0380	Reese, Garth M, 1542
1	0380	Spencer, Benjamin W, 1542
1	0380	Walsh, Timothy Francis, 1542
5	0380	Wilson, Christopher Riley, 1542
1	0382	Aragon, Kathy, 1543
1	0382	Belcourt, Kenneth, 1543
1	0382	Brown, Kevin, 1543
1	0382	Espen, Peter Karl, 1543
1	0382	Sjaardema, Gregory D, 1543
1	0382	Stewart, James R, 1543
1	0382	Williams, Alan B, 1543
1	9042	Bhutani, Nipun, 8774
1	9042	Jew, Michael D, 8774
1	9042	Kistler, Bruce, 8774
1	9042	Kletzli, Daniel, 8774
1	9042	Revelli, Vera, 8774
1	9159	Whiteside, Bob, 8964
2	9018	Central Technical Files, 8945-1
2	0899	Technical Library, 9616