SPPARKS OPERATION AND PARAVIEW VISUALIZATION† QUICK GUIDE & WALKTHROUGH

SPPARKS was developed at Sandia National Laboratories, a U.S. Department of Energy laboratory. The authors are Steve Plimption, Aidan Thompson and Alex Slepoy who can be contacted at sjplimp@sandia.gov, athomps@sandia.gov, and alexander.slepoy@nnsa.doe.gov, respectively. Funding for SPPARKS development has come from the DOE through its LDRD program. Several materials scientists at Sandia National Laboratories have also helped in the design of the code and implementation of specific models. These scientists include: Corbett Battaile, Liz Holm, Veena Tikare, Greg Wagner, Ed Webb and Xiaowang Zhou.

Brief Introduction:

"SPPARKS" is an acronym for Stochastic Parallel PARticle Kinetic Simulator. This quick guide will supply a user new to spparks with the items, direction and basic steps necessary to acquire, compile and run SPPARKS locally. This document is only meant to serve as an abbreviated guide and assumes the user has very little experience, if any, with UNIX and C++. However, if code modification is desired, a working knowledge of C++ is necessary. To that end, SPPARKS is an open-source code, meaning anyone can adjust, edit and contribute to the whole provided the addition yields a useful advance and the contributor accepts the open-source conditions for use and distribution. For full documentation on SPPARKS, its commands, features and updates, visit http://www.cs.sandia.gov/~sjplimp/spparks.html

Before getting started:

Keep in mind, SPPARKS is essentially a collection of linked C++ algorithms located in various c++ and .h files. Each c++ file is paired to a header (i.e. ".h") file for recognition and access to specific SPPARKS-derived classes and operations. Some c++ and .h pairs control lattice creation, others dictate certain kinetic monte carlo (kMc) probabilities, few simply deal with the recognition and creation of error messages while others are only concerned with simulation outputs. The full collection of files is contained in the "src" (i.e. source) folder within the SPPARKS directory. While a thorough knowledge of the C++ class system is not required to run SPPARKS, it is useful to know why and where these files exist because in addition to downloading them all, they all must be linked properly for SPPARKS to run. This linking process is essentially the local compiling or "making" of SPPARKS once the SPPARKS directory has been downloaded or acquired on a machine. Only when a modification has been made to the .cpp or .h files in the src folder will a re-compiling be necessary.

Prerequisites for Mac OS X 10.9 and later:

Beginning with Mac OS X 10.9 and 10.10, codenamed "Mavericks" and "Yosemite", respectively, Apple ceased including various compilers (e.g. gcc 4.9, g++, etc.) with their operating system. As a result, when attempting to build SPPARKS locally on either of these platforms, (and potentially any Mac platform in the future) one must install and link the necessary compiler libraries. This can be done by employing a third-party OS X package manager such as, MACPORTS or HOMEBREW. In this walkthrough, the steps needed to perform this task with HOMEBREW will be covered.

To install the needed compilers in OS X 10.9 and 10.10:

1. Open a Terminal window on your mac that is running either OS X 10.9 or a later version
2. Visit http://brew.sh and copy & paste the ruby prompt provided near the top of the page into your Terminal window
3. Once homebrew has successfully completed its installation, within the Terminal window type the following: "brew install libjpeg"

Making SPPARKS:

To compile SPPARKS so it can be run locally, complete the following steps:

1. Acquire the most recent version of SPPARKS from; http://www.cs.sandia.gov/~sjplimp/download.html
   - Simply select the circle icon to the left of "SPPARKS" under "Miscellaneous software"
   - Then click the "Download Now" button below the list
2. Unpack the tarball package in the location you would like the SPPARKS directory located
   - To recognize it, the unpacked file should be of the form "spparks(#).tar.gz"
3. After unpacking, a folder named "spparks-somedate" should be accessible containing the following items;
   - 2 text files: "license" and "README"
   - 4 folders: "doc", "example", "src" and "tools"
4. If you are not already using a bash or c-shell to access the files, open one now (this can be done by using the Terminal application with a MAC, if using a PC any unix shell should work)
5. To perform the linkings necessary to run SPPARKS, from within a Terminal do the following;
   - cd to the STUBS folder ("spparks-somedate/src/STUBS") within the SPPARKS directory
   - type "make" – a "libmpi.a" and a "mpi.o" should be created within the STUBS folder
   - now cd back to the SRC folder ("spparks-somedate/src) within the SPPARKS directory
   - type "make" – a plethora of options will be listed and the user should ascertain which *make* is best for the system being used
   - then type "make 'whatever_make_is_desired' "
   - If done properly, the process of compiling will begin, a sundry of executables will be created and at the end, an executable should be created. You will know if this has been accomplished because the last line written to the terminal should be of the form: ../spk_'whatever_your_selected_make_was'

Running SPPARKS:

SPPARKS is run by feeding an input script into the SPPARKS executable via the terminal.

If a parallel make on a mac was created, your terminal command will look similar to this;

```
% mpiexec –np # spk_mac_mpi < in.potts
```

The breakdown of the syntax is as follows:

mpiexec	= essentially says run this code in parallel
-np	= introduces the number of processors you will run on
#	= the integer value of processors you are using
spk_mac_mpi	= this is the SPPARKS build generated in the 'make' compilation
in.potts	= input file to be fed into the SPPARKS executable

Using vti_Creator.cpp to generate vti files:

"vti_Creator.cpp" is a c++ program that takes a series of dump files and generates a corresponding series of vti files for visualization in PARAVIEW. 1D, 2D or 3D arrays can be processed allowing for two parameters to be visualized within each vti file. ***Note: by default, vti_Creator.cpp labels these two parameters spin (i.e. i1) & stored energy (i.e. d1) as these were the items of greatest interest in the original creation of the code. It should be understood however, that the parameters mapped to the array in the vti files are those obtained from the 2nd and 3rd columns of the dump files used no matter what information is placed in those locations!*** To understand and adjust the information and/or order of data written to dump files see explanation of "dump" command in SPPARKS command list;
http://www.sandia.gov/~sjplimp/spparks/doc/Section_commands.html#comm

Before vti_Creator.cpp can be used, the program must be compiled and its executable must be created. To accomplish this, do the following:

1. In a terminal window, cd to the folder or directory in which vti_Creator.cpp is located
2. Type the following: % g++ vti_Creator.cpp –o createvti.out
3. If it compiles correctly you should simply get another prompt with no error messages. Additionally, an executable entitled "createvti.out" should now be located within the same directory or folder
4. Now the createvti.out executable can be used but you must ensure your dump files have the following characteristics for createvti.out to operate properly:
   - Each dumpfile in the series must have the same base filename with an iterative counter as the final character in its name (e.g. dump.0, dump.1, dump.2, etc.)
   - The iterative counters need to begin at 0
5. Place the createvti.out executable in the same folder or directory as your dumpfiles to be processed
6. In a terminal, cd to the folder or directory with the creatvti.out executable and dumpfiles if not already there.
7. In the terminal, type: % ./creatvti.out
8. You will be prompted with a series of questions to determine the base filename, size of your array and number of dump files in your series.
9. Lastly, a series of messages will scroll on the screen indicating the processing steps and creation of a vti file for each dump file supplied.

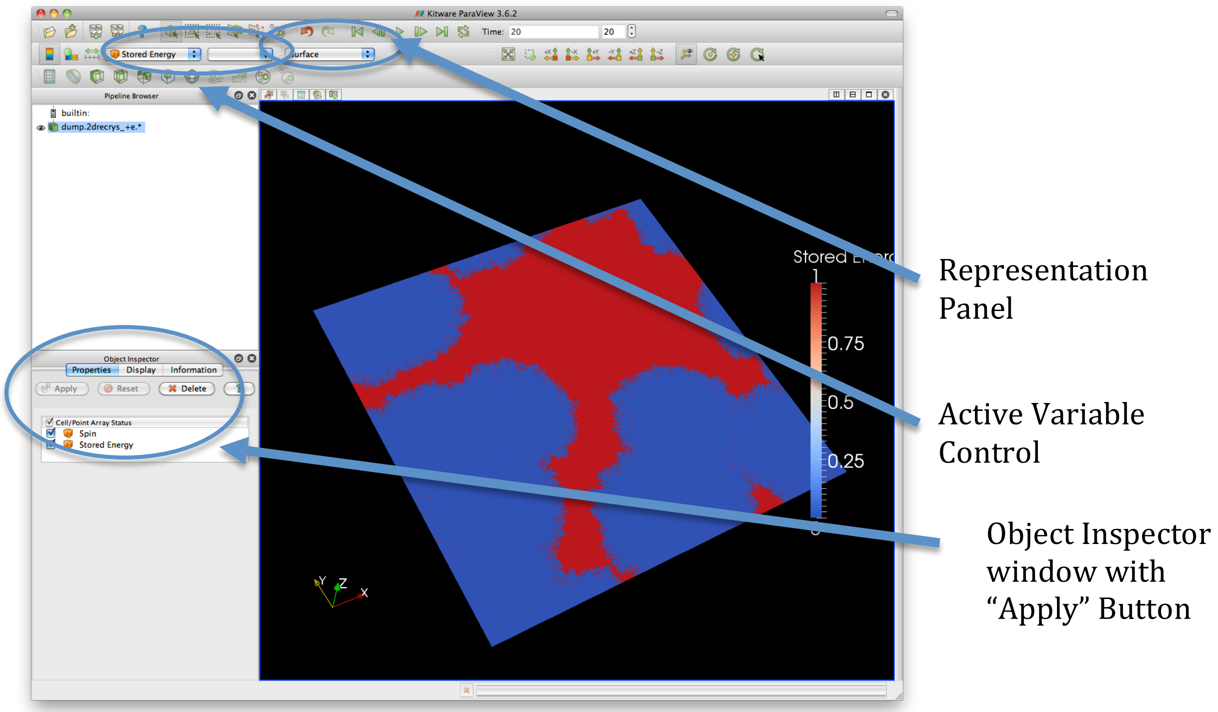
10. If all runs successfully, an “All done….” message will print to screen and a series of vti files will now be located in the file or directory in which the dump files were located.

Using PARAVIEW to visualize vti files:

PARAVIEW is an open-source downloadable code, complete with its own graphical user interface. It can be found here: http://www.paraview.org/paraview/resources/software.html. The following provides step-by-step instruction for how to visualize vti files. Additionally, a screenshot detailing notable windows and drag down menus mentioned in the steps below are included in the following page for reference.

1. Acquire PARAVIEW from the link provided above
2. Unpack the compressed PARAVIEW tarball and drag the unpacked application to Applications folder.
3. Now open PARAVIEW and using the open command from either the “File” tab in the upper left of the screen or the open icon in the upper left of the PARAVIEW window, select the vti file set desired.

   Note: If series possesses the same base filename followed by an iterative counter PARAVIEW will recognize the entire group as a set and it can be opened by selecting the set as opposed to each individual file.
4. Next, in the object inspector window and under the properties tab, click apply to assign Cell/Point Array information to all voxels in dataset
5. Now, select the visualization scheme by clicking “surface” or “surface with edges” in the representation panel drop box
6. Lastly, select the parameter to visualize by selecting “Spin” or “Stored Energy” in the active variable control panel drop box



† Disclaimer: There are many ways to do many of the actions described in this document. Detailed above is simply a streamlined and basic methodology to get the user from point A to B in terms of starting SPPARKS and visualizing a result. SPPARKS as well as PARAVIEW present a vast amount of computational and visualization options and abilities only hinted at above. Nonetheless, following the instructions above should get nearly any user started regardless of their base knowledge of UNIX, C++, Macs or PCs.