Dividing and Conquering Meshes within the NIST Fire Dynamics Simulator (FDS) on Multicore Computing Systems

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Donald Charles Collins

December 2015

Acknowledgements

A big thankyou to my wife, Julia, and my two amazing children, Samuel and Eleanora, for providing encouragement as they endured my seemingly endless academic pursuits.

Abstract

The National Institute for Standards and Technology (NIST) Fire Dynamics Simulator (FDS) provides a computational fluid dynamics model of a fire, which can be visualized by using NIST Smokeview (SMV). Users must create a configuration file (*.fds) that describes the environment and other characteristics of the fire scene so that the FDS software can produce the output file (*.smv) needed for visualization.The processing can be computationally intensive, often taking between several minutes and several hours to complete. In many cases, a user will create a file that is not optimized for a multicore computing system. By dividing meshes within the fds file, the author was able to substantially reduce the computing time.

Preface

All results are a product of the experiments performed by the author, Donald Charles Collins on a custom-built 32-core server which is the property of the University of Tennessee Electrical Engineering and Computer Science (EECS) Department. All simulations were performed by the author.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and General Information

About NIST Fire Dynamics Simulator (FDS) and Smokeview (SMV)

Since 2000, Fire Dynamics Simulator (FDS) has been provided to the public by the National Institute for Standards and Technology (NIST). The latest version, 6.3.0, was provided on October 1, 2015. This is the version that the author used to perform the experiments in this paper.

FDS is a computational fluid dynamics application that is written in FORTRAN. It provides the option to use the Message Passing Interface (MPI) in order to take advantage of a multicore system or a multi-node network. Its companion software, Smokeview (SMV) is used to visualize the output of the FDS application. Both are provide by NIST for free to the public. It is a primary visualization tool for engineers and specialists within the fields of Fire Protection Engineering and Fire Investigation.

Chapter 2

Literature Review

The main source for information dealing with utilizing multiple cores within Fire Dynamics Simulator comes from section 3.1.2 of the NIST FDS User's Guide.

Meshes in FDS

A mesh in FDS represents the three-dimensional bounds of the volume or "box" in which the fire scene exists. It has boundaries that range from x0 to x1, y0 to y1 and z0 to z1. There are also values that represent the number of cells contained within each range. A cell is a smaller volume with the larger mesh volume. Here is an example of a mesh line in the input file (*.fds):

```
&MESH IJK=24,12,24, XB=1.1,3.5,3.6,4.8,0.0,2.4 /
```

The "&MESH" is a descriptor that identifies the line as a mesh line. In this case, the entire fire scene dimensions are contained within this one mesh line. If you were to run FDS on this line, it would use a single core to process the SMV visualization.

The above mesh line contains an "XB" variable with a sextuple of real numbers. The first two numbers, 1.1 and 3.5, represent the range from x0 to x1. The next set, 3.6 and 4.8, represent the range from y0 to y1. The final set represents the range from z0 to

z1. You can think of x and y ranges as being the length and width of a room, whereas the z range would be the height of that room.

A mesh is divided into cells, which are smaller volumes whose dimensions are determined by the “IJK” value on the “&MESH” line. As seen above, the “IJK” variable holds a triad of integers 24, 12 and 24. These integers represent the number of cells within the x, y and z ranges, respectively. For instance, the x range is 3.5-1.1 = 2.4. Since the first “IJK” value is 24, then the range 2.4 contains 24 cells.

A fire scene simulation can be divided into multiple meshes. Figures 1, 2 and 3 show how a room appears in Smokeview for 1, 8 and 64 mesh divisions respectively.

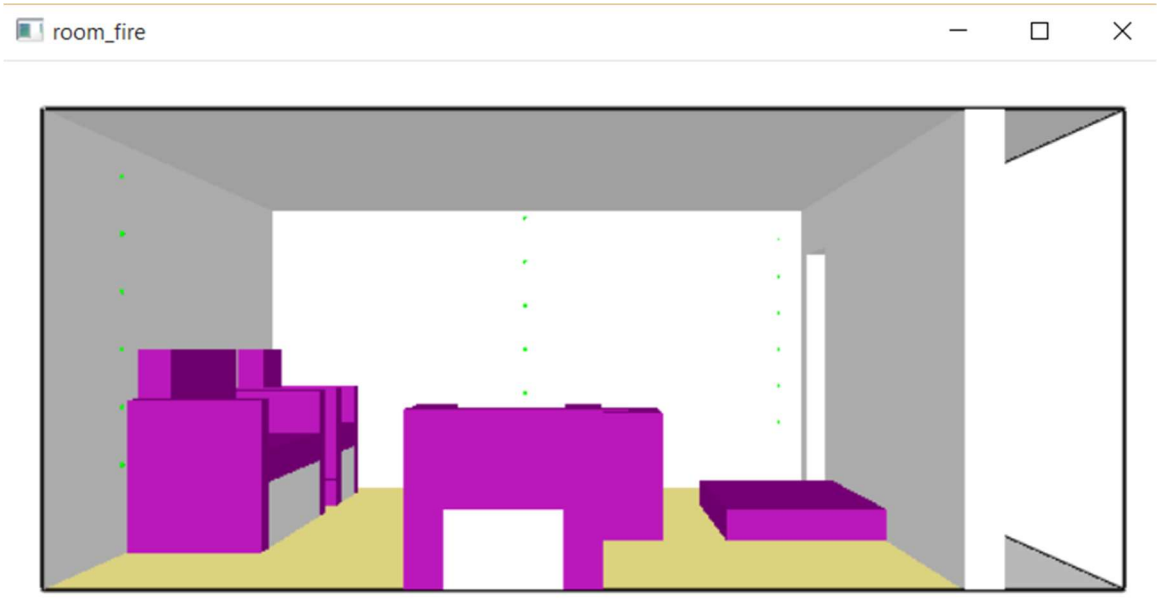

Figure 1: Room with a Single Mesh

Figure 2: Room with 8 Meshes (2x2x2)

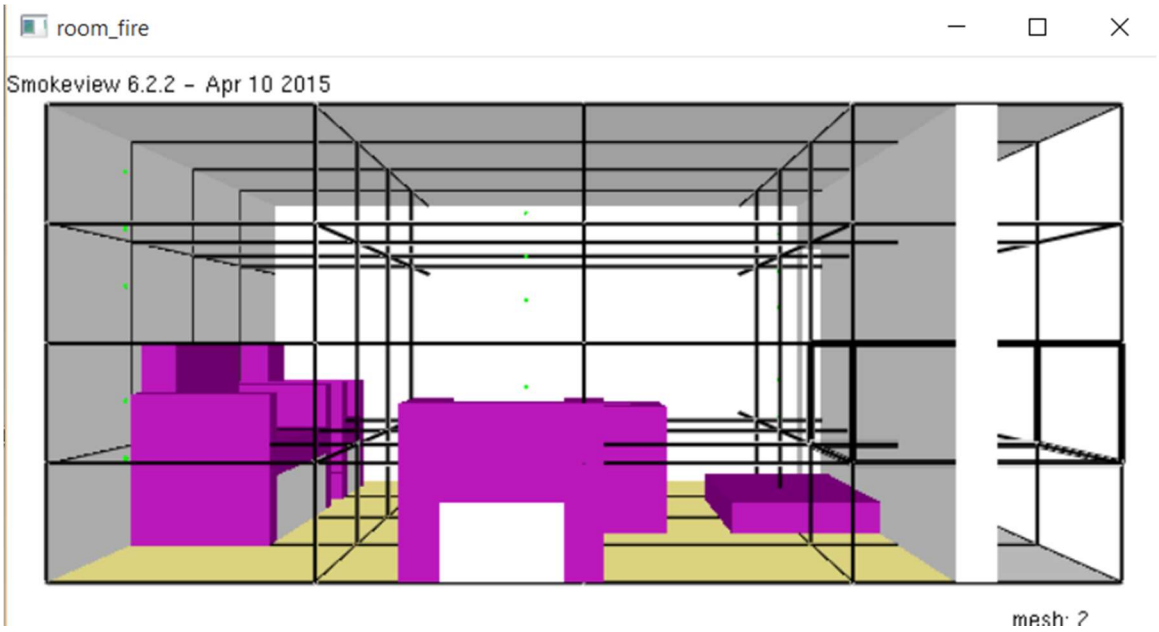

Figure 3: Room with 64 Meshes (4x4x4)

The Message Passing Interface (MPI) within FDS

Fire Dynamics Simulator (FDS) provides an option to use the Message Passing Interface (MPI) in order to take advantage of multiple cores on the same computer. It will default to at least one MPI process per core as long as there are at least as many meshes defined in the FDS input file (*.fds) as there are processing cores on the computing machine.

Preparing FDS Meshes for MPI

In order to employ MPI on FDS, you'll need to ensure that there are at least as many "&MESH" lines in the input file as there are cores on the computing machine. If the number of mesh lines exceeds the number of cores, each mesh will need to be assigned to an MPI process by adding the "MPI_PROCESS" variable to the line as follows:

```
&MESH IJK=6,3,6, XB= 1.1, 1.7, 3.6, 3.9, 0, 0.6, MPI_PROCESS=0 /
```

It is important to ensure that each line is preceded by a line whose MPI_PROCESS variable value is the same size or less than the current line. For example, this order is permissible:

```
&MESH IJK=6,3,6, XB= 1.1, 1.7, 3.9, 4.2, 1.2, 1.8, MPI_PROCESS=0 /
&MESH IJK=6,3,6, XB= 1.1 , 1.7, 3.9, 4.2, 1.8, 2.4, MPI_PROCESS=0 /
&MESH IJK=6,3,6, XB= 1.1 , 1.7, 4.2, 4.5, 0, 0.6, MPI_PROCESS=1 /
&MESH IJK=6,3,6, XB= 1.1, 1.7, 4.2, 4.5, 0.6, 1.2, MPI_PROCESS=1 /
```

This order is not:

```
&MESH IJK=6,3,6, XB= 1.1, 1.7, 3.9, 4.2, 1.2, 1.8, MPI_PROCESS=0 /
&MESH IJK=6,3,6, XB= 1.1 , 1.7, 3.9, 4.2, 1.8, 2.4, MPI_PROCESS=1 /
&MESH IJK=6,3,6, XB= 1.1 , 1.7, 4.2, 4.5, 0, 0.6, MPI_PROCESS=0 /
&MESH IJK=6,3,6, XB= 1.1, 1.7, 4.2, 4.5, 0.6, 1.2, MPI_PROCESS=1 /
```

A Python script (mesh-slicer.py) was written by the author of this paper and has been provided in the appendix. To use the script, just fill in the variables with the information from the single FDS input file's &MESH line and then run the script. Python 3 must be installed on your machine in order to run the script. The output will be multiple "&MESH" lines based on the parameters given by the user. These can be copied and pasted into the FDS input file, replacing the single "&MESH" line.

In order for the meshes to be divisible, the "IJK" values must all produce an integer value when they are divided. The FDS software requires this. For instance, if you need to divide a mesh into 8 volumes, you would divide each axis by 2. This means each "IJK" value would have to be divided by 2 as well. In this case, it is crucial to have each "IJK" value be an even number before the division takes place.

The Python code was written with the assumption that the user is providing the correct input data for the fire simulation. The Python code does not change the cell size or overall scene dimensions that are provided by the user.

Running MPI for FDS

If a single-core instance of FDS were to be run, the command line input would look similar to this:

```
PROMPT> fds myInputFile.fds
```

In this example, “fds” is the Fire Dynamics Simulator executable which takes the “myInputFile.fds” as input. In order to run an mpi-enabled program, a command line input similar to the following is needed:

```
PROMPT> mpiexec –n 32 –localonly fds myInputFile.fds
```

In this example, “mpiexec” is the command that executes the message passing interface (MPI) protocol on fds. The –n flag specifies the number of cores to use, based on the integer that follows. Note that this number shall not exceed the actual number of cores that the machine has. It can, however, specify a number smaller than the maximum. For instance, on a 32-core machine, you can specify 8 cores if you wish. The FDS file must be formatted to accept this. The “-localonly” flag tells mpiexec that only the local machine is being used.

Chapter 3

Materials and Methods

Computing Environment

Hardware

The computer that was used to run the simulations was a custom-built 32-core server which is the property of the University of Tennessee Electrical Engineering and Computer Science (EECS) Department. Table 1 lists the computer specifications for the server that ran the simulations.

Table 1: Computer Specifications

COMPONENTS	SPECIFICATIONS
Processors	Dual AMD Opteron 6274 2.2 GHz 16-Core
L1 Instruction Cache	8 x 64 KB 2-way set associative shared
L1 Data Cache	16 x 16 KB 4-way set associative data
Memory	120 GB @ 1333 MHz
Front Side Bus	1333 MHz
Hard Drives	5 x 300 GB

Software

The software that was used to run the simulations was FDS-SMV v. 6.3.0, otherwise known as Fire Dynamics Simulator with SmokeView. The software is freely available online. Information can be found on the NIST website (NIST, 2015). Table 2 lists the involved software.

Table 2: Software Specifications

COMPONENTS	SPECIFICATIONS
Operating System	Windows 7 Professional, SP-1
Simulation Software	FDS-SMV v. 6.3.0

Chapter 4

Results and Discussion

Small Fire Scene Simulation – Whirlpool Fire

Overview of the Whirlpool Fire Simulation

The input file for this example is located in the "Examples" directory for the Fire Dynamics Simulator, version 6.3.0. Figure 4 shows a visual 3d representation of the fire scene. This is the Smokeview visualization of the FDS output produced from the simulation experiment.

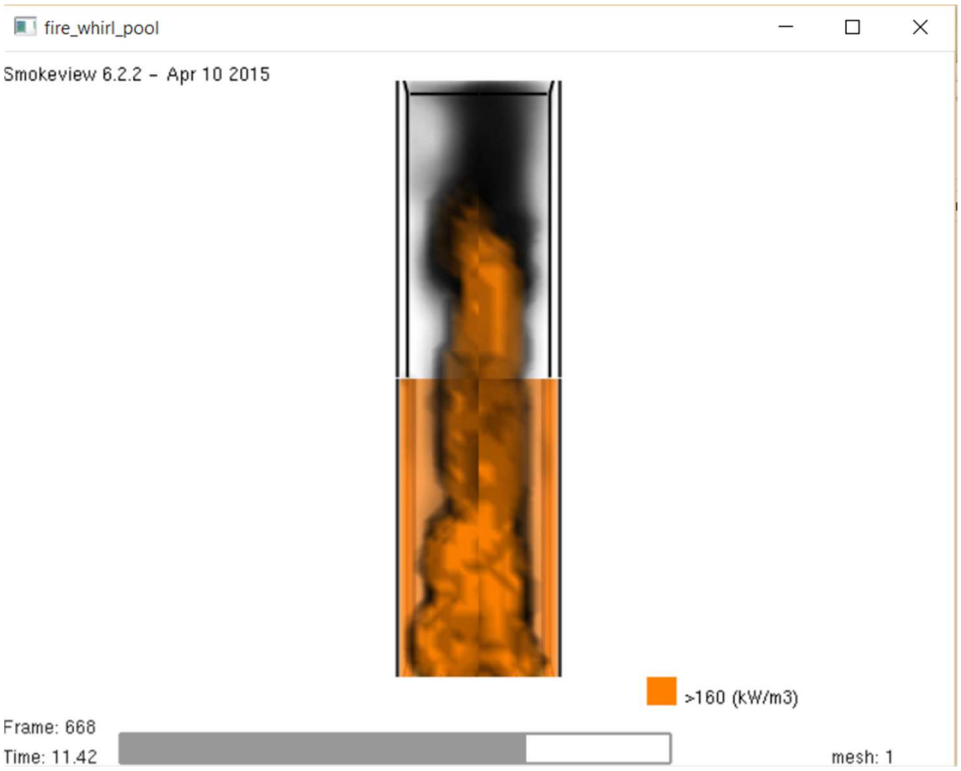

Figure 4: Whirlpool Fire Simulation Visualization

Along with the visible light visualization, the output files contain data which represent the movement of heat throughout the scene, whether through conduction, convection or radiation. Figure 5 shows an example of the visualization of thermal transfer within the scene.

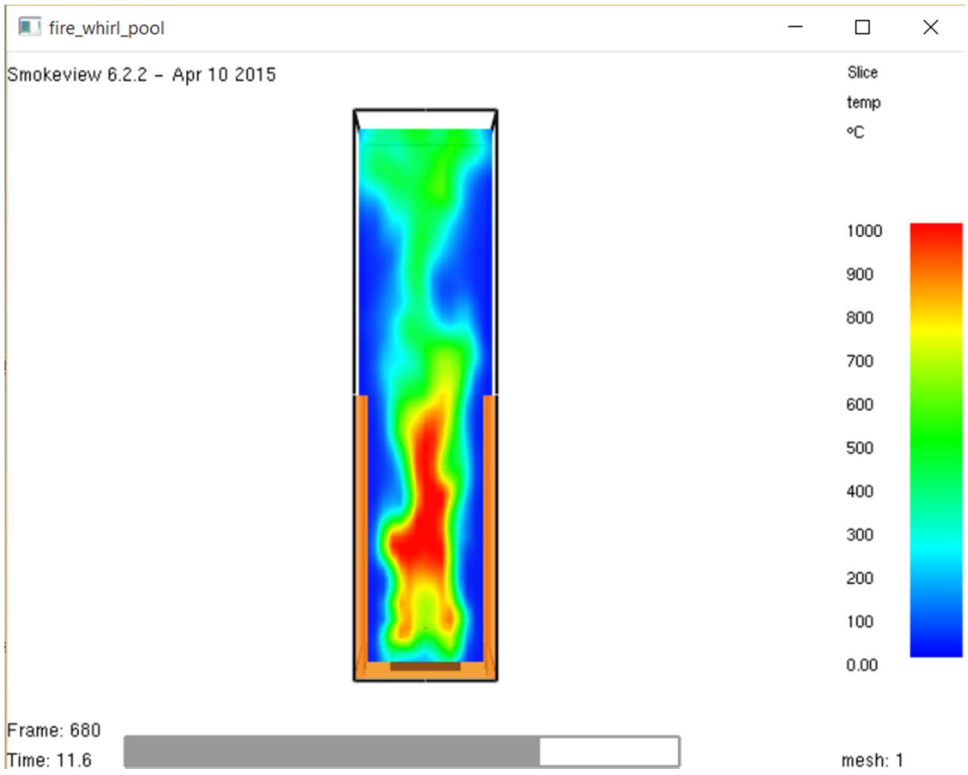

Figure 5: Whirlpool Fire Simulation Heat Map

The Effect of Mesh Counts on Simulation Time

Six different Whirlpool Fire simulations were performed, each with a greater number of meshes than the previous simulation. In order to affirm the results, each simulation was performed three times and then averaged.

The first set of simulations utilized a single mesh that was already included in the Whirlpool Fire input file that is available in the Examples folder of FDS 6.3.0 for Windows 7. The average elapsed total wall clock time for completion of this simulation was 2694 seconds, or roughly 45 minutes.

The second set of simulations was performed with a modified input file in which the original mesh was split into four smaller meshes. The number four is effectively the number of volumes that would be cut from a mesh that was divided by two along its x and y axes, but not its z axis. Splitting the mesh was accomplished with a custom Python script (mesh-slicer.py) that was written by the author of this paper (See appendix).

The average elapsed time for this set of simulations was 918 seconds, or roughly 15 minutes. This ran in a third of the time that it took the simulation that utilized the original single-mesh input file. This represents a speedup of 3, or a 300% increase in speed over the single-mesh time.

The third set of simulations was performed with a modified input file in which the original mesh was split into eight smaller meshes. The number eight is

effectively the number of volumes that would be cut from a mesh that was divided by two along each axis.

The elapsed time for this simulation was 842 seconds, or roughly 14 minutes. This is less than a third of the time that it took the simulation that utilized the original single-mesh input file. This was a speedup of 3.2, or a 320% increase in speed over the single-mesh time. This is also an improvement over the mesh that was split into 4 volumes.

The fourth set of simulations was performed with a modified input file in which the original mesh was split into 16 smaller meshes. The number four is effectively the number of volumes that would be cut from a mesh that was divided by four along its x axis and 2 along its y and z axes.

The average elapsed time for this set of simulations was 706 seconds, or roughly 12 minutes. This ran in under a third of the time that it took the simulation that utilized the original single-mesh input file. This was a speedup of 3.75, or a 375% increase in speed over the single-mesh time.

The fifth set of simulations was performed with a modified input file in which the original mesh was split into 32 smaller meshes. The number 32 is effectively the number of volumes that would be cut from a mesh that was divided by four along its x and y axes and by two along its z axis.

The average elapsed time for this set of simulations was 727 seconds, or roughly 12 minutes. This ran in under a third of the time that it took the

simulation that utilized the original single-mesh input file. This was a speedup of 3.75, or a 375% increase in speed over the single-mesh time.

The sixth and final simulation was performed with a modified input file in which the original mesh was split into 64 smaller meshes. The number 64 is effectively the number of volumes that would cut from a mesh that was divided by four along each axis. The previously mentioned Python script was used to split the original mesh into 64 smaller meshes. It completed in 838 seconds, or roughly 14 minutes. This represents a speedup of 3.2, or a 320% increase in speed over the single-mesh time.

With the Whirlpool simulations, the results improved dramatically after more than one core was introduced, but then the speedup leveled off after the 16-core simulations. The results of all six simulations are summarized in Figure 6 and Figure 7.

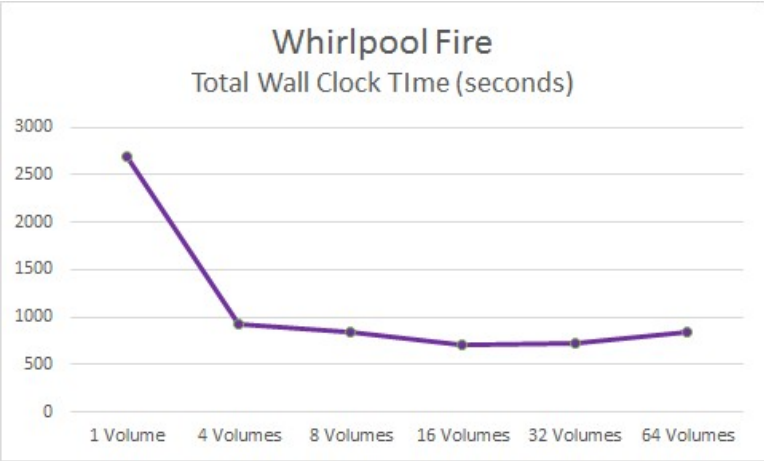

Figure 6: Whirlpool Fire Simulation Times

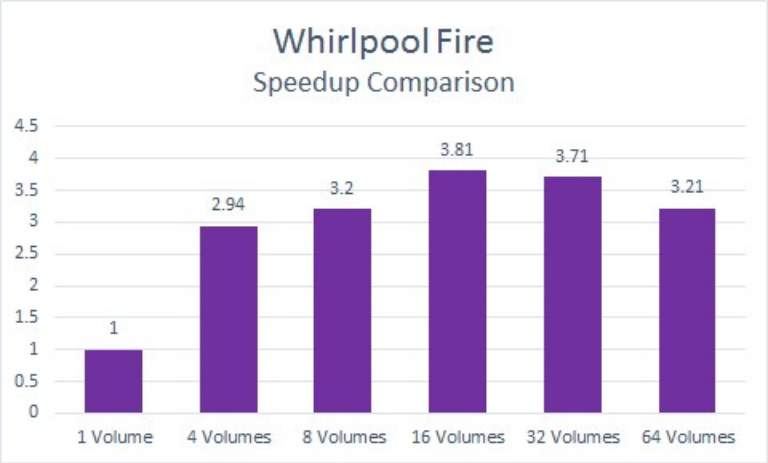

Figure 7: Whirlpool Fire Simulation Speedup Comparison

The Effect of Mesh Counts on Smokeview File Size

Splitting up the meshes had a significant effect on the size of the resulting Smokeview file. As can be seen in Figure 8, the file sizes for the six simulations increased dramatically with the number of mesh divisions. The 64-mesh simulation produced a file that was 195 Kilobytes, as opposed to the single-mesh simulation, which produced a 16 Kilobyte file size. The former is 12 times larger than the latter.

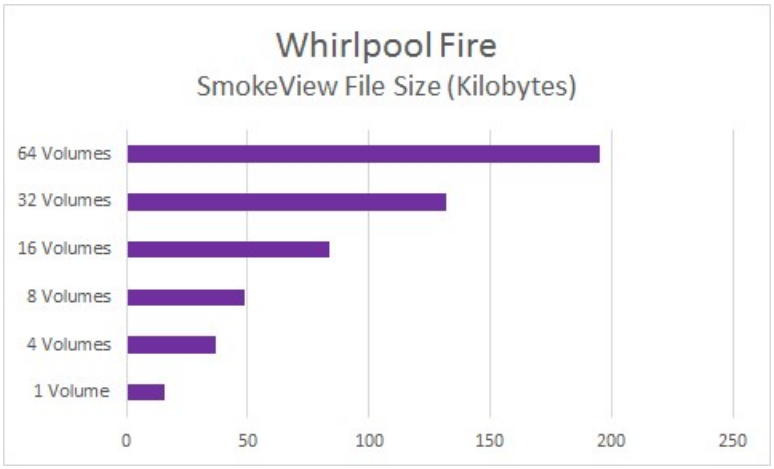

Figure 8: Whirlpool Fire Smokeview File Sizes

Comparing Total Step Time and Total Wall Clock Time

In all six simulations, there was little difference between the Total Step Time and the Total Wall Clock Time. This is shown in Figure 9. The program is efficiently processing data with very little latency between steps.

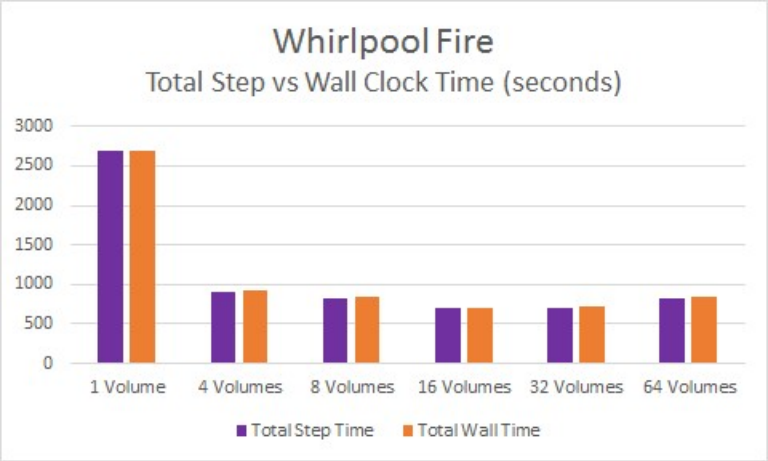

Figure 9: Whirlpool Fire Simulation Total Step vs. Wall Clock Time

Medium Fire Scene Simulation – Couch Fire

Overview of the Couch Fire Simulation

The input file for this example is located in the "Examples" directory for the Fire Dynamics Simulator, version 6.3.0. Figure 10 shows a visual 3d representation of the fire scene. This is the Smokeview visualization of the FDS output produced from the simulation experiment.

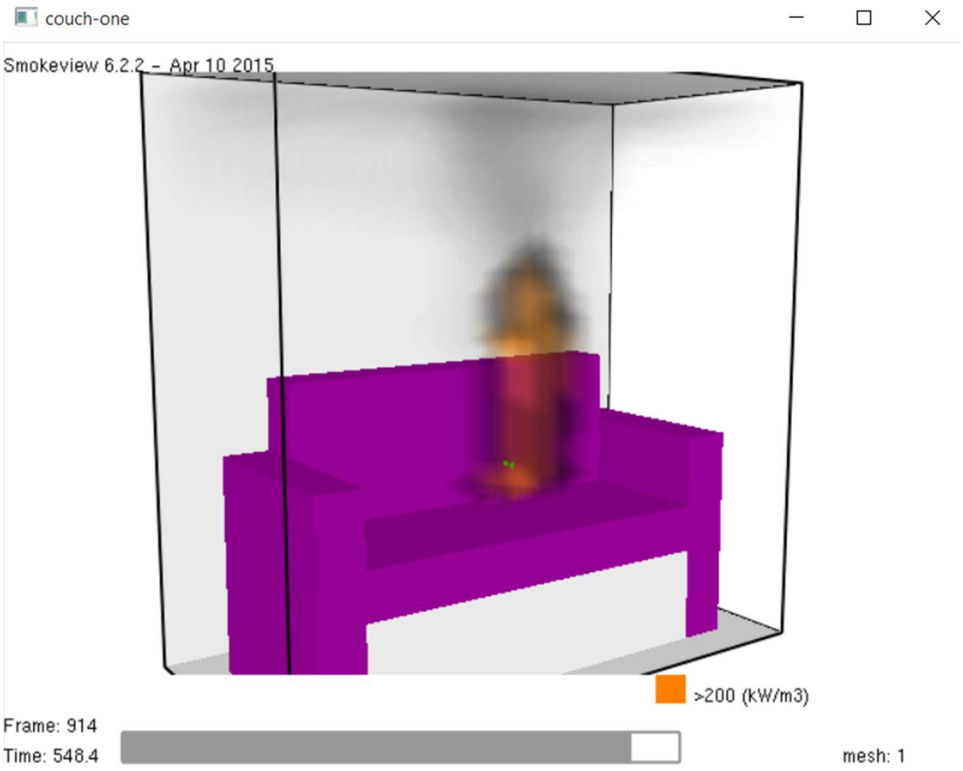

Figure 10: Couch Fire Simulation Visualization

Along with the visible light visualization, the output files contain data which represent the movement of heat throughout the scene, whether through conduction, convection or radiation. Figure 11 shows an example of the visualization of thermal transfer within the scene.

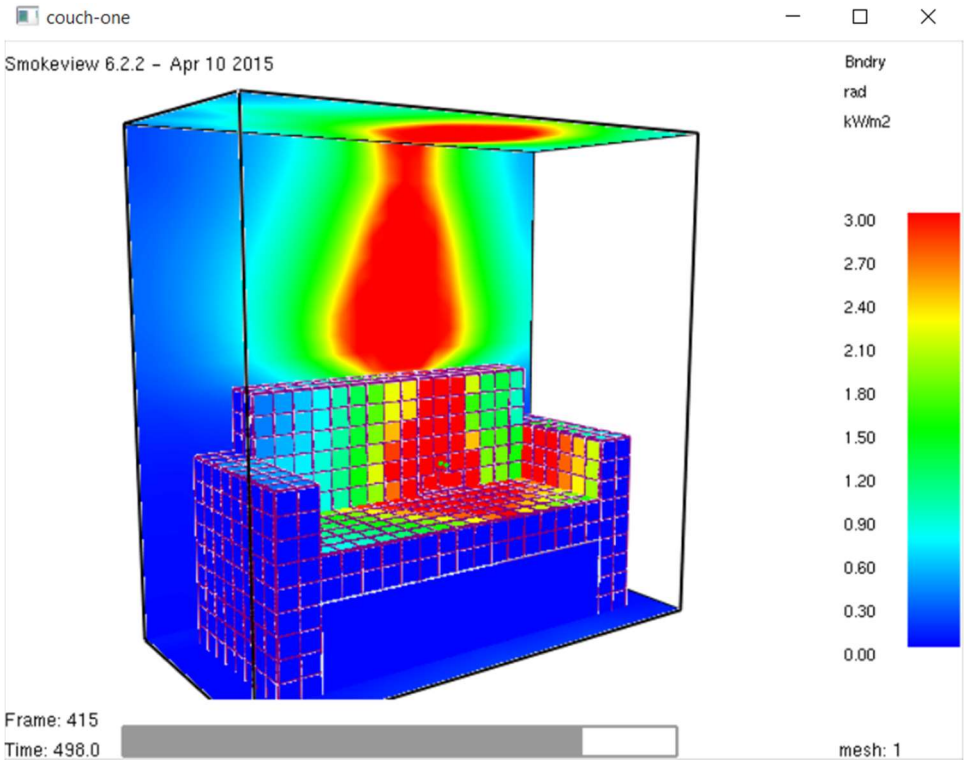

Figure 11: Couch Fire Simulation Heat Map

The Effect of Mesh Counts on Simulation Time

Six different Couch Fire simulations were performed, each with a greater number of meshes than the previous simulation. In order to affirm the results, each simulation was performed three times and then averaged.

The first simulation utilized a single mesh that was slightly modified from the one included in the Couch Fire input file that is available in the Examples folder of FDS 6.3.0 for Windows 7. The modification was a change in the x-range value from 10 to 12 in order to ensure that the length of the x-range would remain an integer value even after being divided by two or four. The "IJK" variable requires this.

The first set of simulations utilized a single mesh that was already included in the Couch Fire input file that is available in the Examples folder of FDS 6.3.0 for Windows 7. The average elapsed total wall clock time for completion of this simulation was 4466 seconds, or roughly 1 hour and 15 minutes.

The second set of simulations was performed with a modified input file in which the original mesh was split into four smaller meshes. The number four is effectively the number of volumes that would be cut from a mesh that was divided by two along its x and y axes, but not its z axis. Splitting the mesh was accomplished with a custom Python script (mesh-slicer.py) that was written by the author of this paper (See appendix).

The average elapsed time for this set of simulations was 1732 seconds, or roughly 29 minutes. This ran in well under half the time that it took the simulation that utilized the original single-mesh input file. There was a speedup of 2.58, or a 258% increase in speed over the single-mesh time.

The third set of simulations was performed with a modified input file in which the original mesh was split into eight smaller meshes. The number eight is effectively the number of volumes that would be cut from a mesh that was divided by two along each axis.

The elapsed time for this simulation was 1958 seconds, or roughly 33 minutes. This is still less than half the time that it took the simulation that utilized the original single-mesh input file. There was a speedup of 2.28, or a 228% increase in speed over the single-mesh time.

The fourth set of simulations was performed with a modified input file in which the original mesh was split into 16 smaller meshes. The number four is effectively the number of volumes that would be cut from a mesh that was divided by four along its x axis and two along its y and z axes.

The average elapsed time for this set of simulations was 1319 seconds, or roughly 22 minutes. This ran in under a third of the time that it took the simulation that utilized the original single-mesh input file. There was a speedup of 3.39, or a 339% increase in speed over the single-mesh time.

The fifth set of simulations was performed with a modified input file in which the original mesh was split into 32 smaller meshes. The number 32 is

effectively the number of volumes that would be cut from a mesh that was divided by four along its x and y axes and by two along its z axis.

The average elapsed time for this set of simulations was 1018 seconds, or roughly 17 minutes. This ran in under a quarter of the time that it took the simulation that utilized the original single-mesh input file. There was a speedup of 4.39, or a 439% increase in speed over the single-mesh time.

The sixth and final set of simulations was performed with a modified input file in which the original mesh was split into 64 smaller meshes. The number 64 is effectively the number of volumes that would cut from a mesh that was divided by four along each axis. The previously mentioned Python script was used to split the original mesh into 64 smaller meshes. It completed in 1156 seconds, or roughly 19 minutes. This represents a speedup of 3.86, or a 386% increase in speed over the single-mesh time.

With the Couch Fire simulations, the results improved dramatically after more than one core was introduced, but then the speedup leveled off after the 16-core simulations. The results of all five simulations are summarized in Figure 12 and Figure 13.

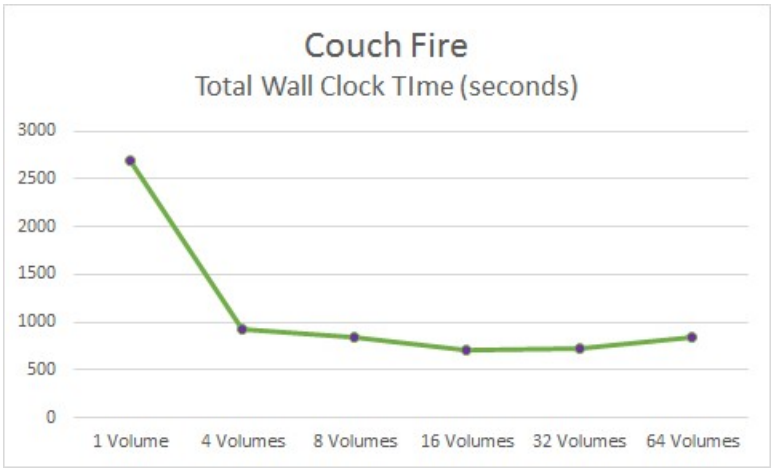

Figure 12: Couch Fire Simulation Times

Figure 13: Couch Simulation Speedup Comparison

The Effect of Mesh Counts on Smokeview File Sizes

Splitting up the meshes had a significant effect on the size of the resulting Smokeview file. As can be seen in Figure 14, the file sizes for the six simulations increased dramatically with the number of mesh divisions. The 64-mesh simulation produced a file that was 327 Kilobytes, as opposed to the single-mesh simulation, which produced a 151 Kilobyte file size. The former is more than double the size of the latter.

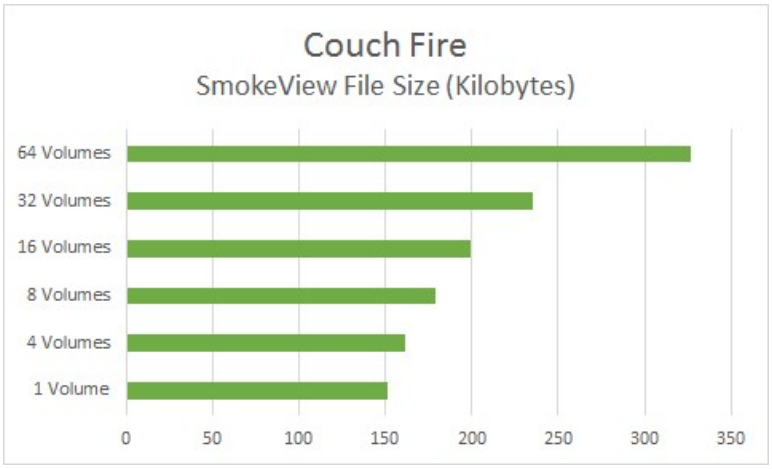

Figure 14: Couch Fire Smokeview File Sizes

Comparing Total Step Time and Total Wall Clock Time

In all six simulations, there was little difference between the Total Step Time and the Total Wall Clock Time. This is shown in Figure 15. The program is efficiently processing data with very little latency between steps.

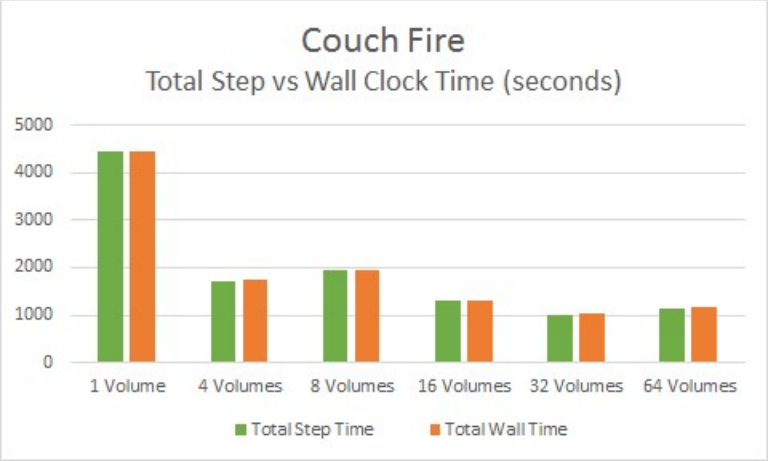

Figure 15: Couch Fire Simulation Total Step vs. Wall Clock Time

Large Fire Scene Simulation – Room Fire

Overview of the Room Fire Simulation

The input file for this example is located in the "Examples" directory for the Fire Dynamics Simulator, version 6.2, but was not included in the 6.3.0 version. Figure 16 shows a visual 3d representation of the fire scene. This is the

Smokeview visualization of the FDS output produced from the simulation experiment.

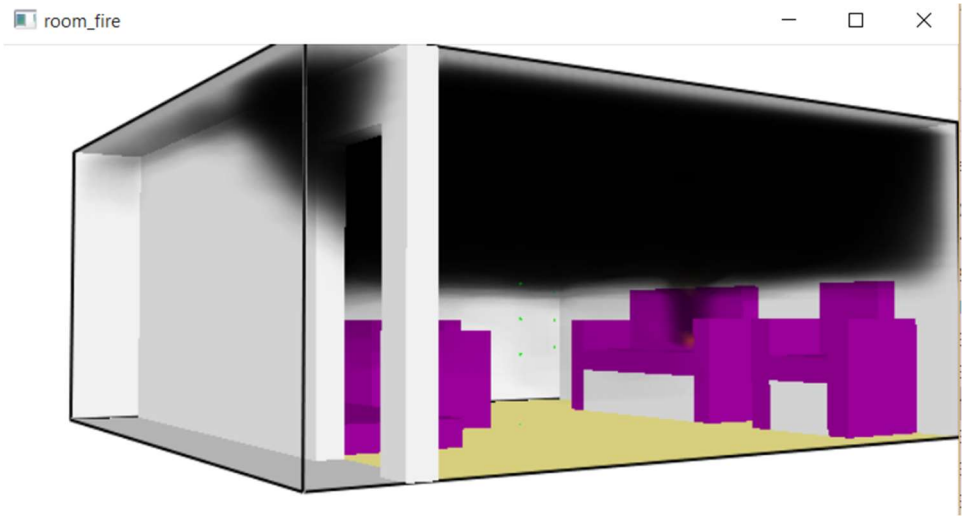

Figure 16: Room Fire Simulation Visualization

Along with the visible light visualization, the output files contain data which represent the movement of heat throughout the scene, whether through conduction, convection or radiation. Figure 17 shows an example of the visualization of thermal transfer within the scene.

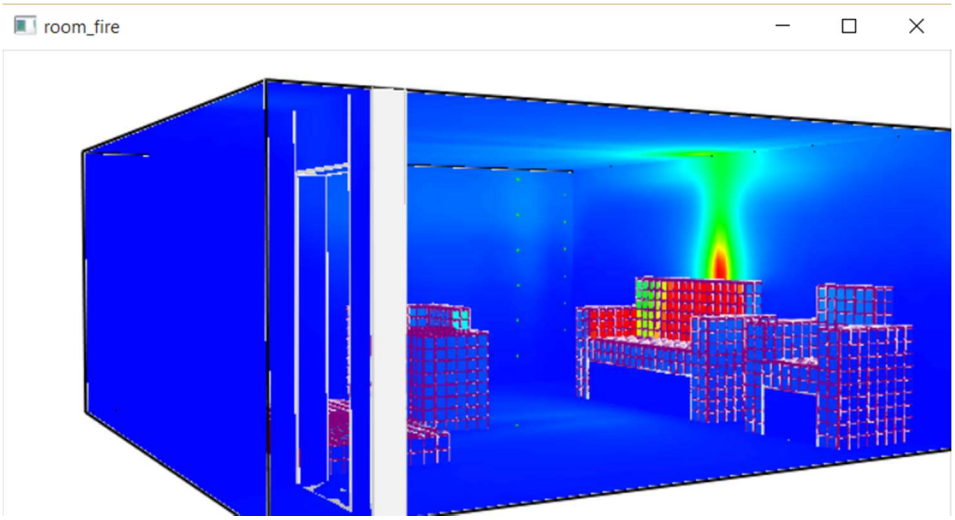

Figure 17: Room Fire Simulation Heat Map

The Effect of Mesh Counts on Simulation Time

Six different Room Fire simulations were performed, each with a greater number of meshes than the previous simulation. In order to affirm the results, each simulation was performed three times and then averaged.

The first set of simulations utilized a single mesh that was already included in the Room Fire input file that is available in the Examples folder of

FDS 6.2 versions for Windows 7. This Example was not included in FDS 6.3.0. The average elapsed total wall clock time for completion of this simulation was 10410 seconds, or roughly 2 hours and 53 minutes.

The second set of simulations was performed with a modified input file in which the original mesh was split into four smaller meshes. The number four is effectively the number of volumes that would be cut from a mesh that was divided by two along its x and y axes, but not its z axis. Splitting the mesh was accomplished with a custom Python script (mesh-slicer.py) that was written by the author of this paper (See appendix).

The average elapsed time for this set of simulations was 3136 seconds, or roughly 53 minutes. This ran in less than a third of the time that it took the simulation that utilized the original single-mesh input file. There was a speedup of 3.32, or a 332% increase in speed over the single-mesh time.

The third set of simulations was performed with a modified input file in which the original mesh was split into eight smaller meshes. The number eight is effectively the number of volumes that would be cut from a mesh that was divided by two along each axis.

The elapsed time for this simulation was 4812 seconds, or roughly 1 hour and 20 minutes. This is still less than half the time that it took the simulation that utilized the original single-mesh input file. There was a speedup of 2.16, or a 216% increase in speed over the single-mesh time.

The fourth set of simulations was performed with a modified input file in which the original mesh was split into 16 smaller meshes. The number four is effectively the number of volumes that would cut from a mesh that was divided by four along its x axis and two along its y and z axes.

The average elapsed time for this set of simulations was 2571 seconds, or roughly 43 minutes. This ran in a fourth of the time that it took the simulation that utilized the original single-mesh input file. There was a speedup of 4, or a 400% increase in speed over the single-mesh time.

The fifth set of simulations was performed with a modified input file in which the original mesh was split into 32 smaller meshes. The number 32 is effectively the number of volumes that would be cut from a mesh that was divided by four along its x and y axes and by two along its z axis.

The average elapsed time for this set of simulations was 2385 seconds, or roughly 40 minutes. This ran in under a quarter of the time that it took the simulation that utilized the original single-mesh input file. There was a speedup of 4.36, or a 436% increase in speed over the single-mesh time.

The sixth and final set of simulations was performed with a modified input file in which the original mesh was split into 64 smaller meshes. The number 64 is effectively the number of volumes that would be cut from a mesh that was divided by four along each axis. The previously mentioned Python script was used to split the original mesh into 64 smaller meshes. It completed in 2832 seconds, or

roughly 47 minutes. This represents a speedup of 3.68, or a 368% increase in speed over the single-mesh time.

With the Room Fire simulations, the results improved dramatically after more than one core was introduced, but then the speedup leveled off after the 16-core simulations. The results of all six simulations are summarized in Figure 18 and Figure 19.

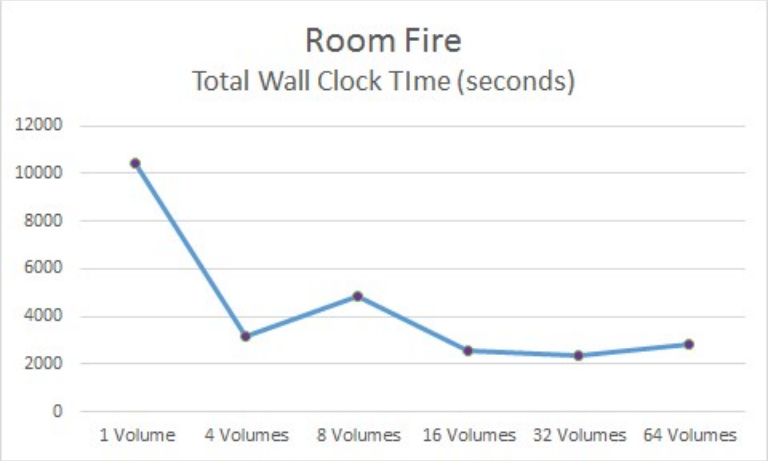

Figure 18: Room Fire Simulation Times

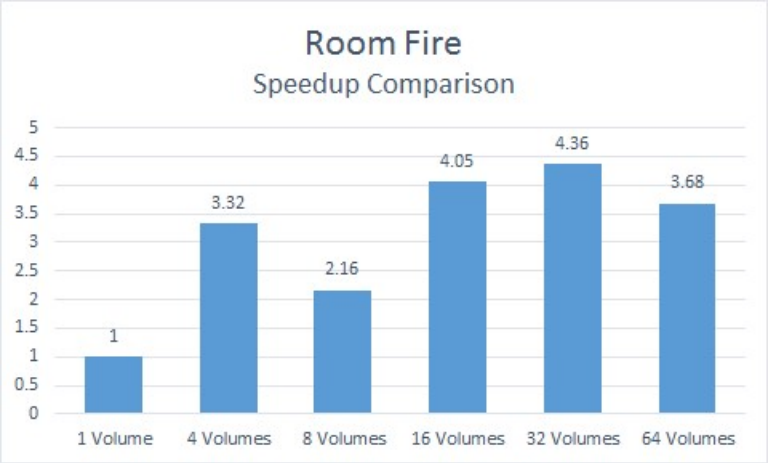

Figure 19: Room Fire Simulation Speedup Comparison

The Effect of Mesh Counts on Smokeview File Size

Splitting up the meshes had a significant effect on the size of the resulting Smokeview file. As can be seen in Figure 20, the file sizes for the six simulations increased dramatically with the number of mesh divisions. The 64-mesh simulation produced a file that was 670 Kilobytes, as opposed to the single-mesh simulation, which produced a 463 Kilobyte file size. The former is nearly 45% larger than the latter.

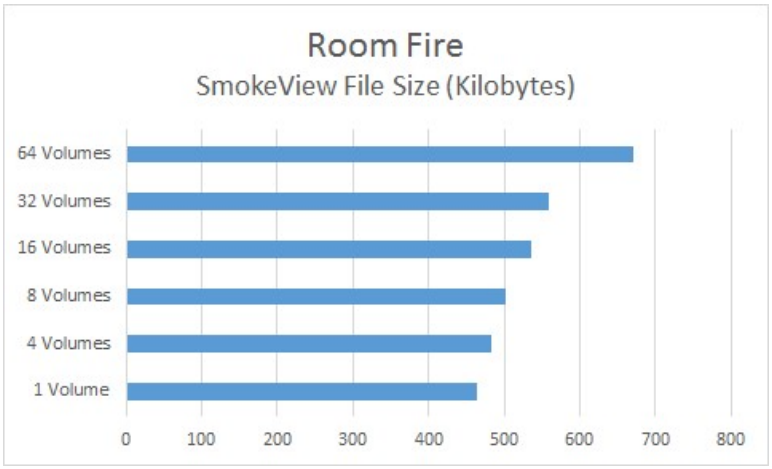

Figure 20: Room Fire Smokeview File Sizes

Comparing Total Step Time and Total Wall Clock Time

In all six simulations, there was little difference between the Total Step Time and the Total Wall Clock Time.

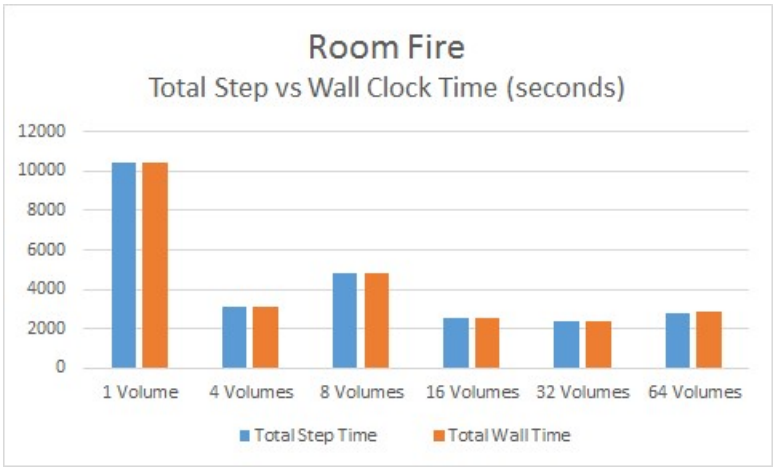

Figure 21: Room Fire Simulation Total Step vs. Wall Clock Time

Chapter 5

Conclusions and Recommendations

Results

Dividing a single mesh into four or more volumes on a multicore machine will significantly shorten the wall clock time for the simulation. This can be an improvement of upwards of 400% in some situations. Ensuring that there is at least one mesh per core will significantly improve the performance of a simulation over that of a single mesh calculation. The speedup comparison can be seen in Figure 22. Note that a "Vol." represents a volume that was cut out of a single mesh and is itself a new smaller mesh. The best times for the Couch and Room simulations occurred when the number of volumes equaled the number of cores on the 32-core machine. The overall speedup comparison can be seen in Figure 23.

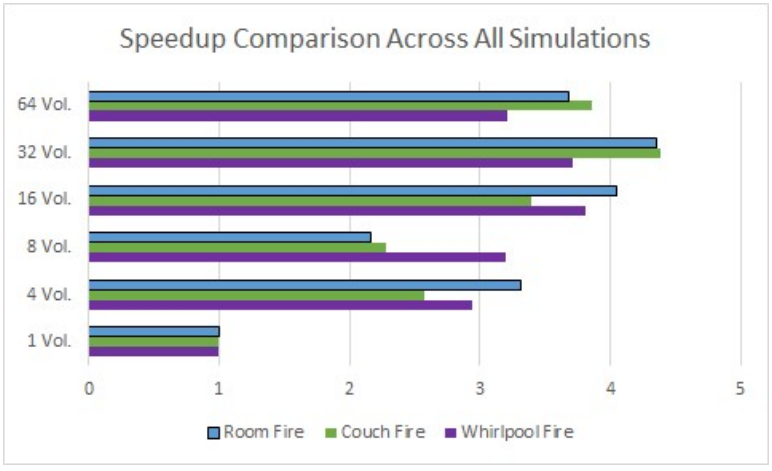

Figure 22: Speedup Comparison Across all Simulations

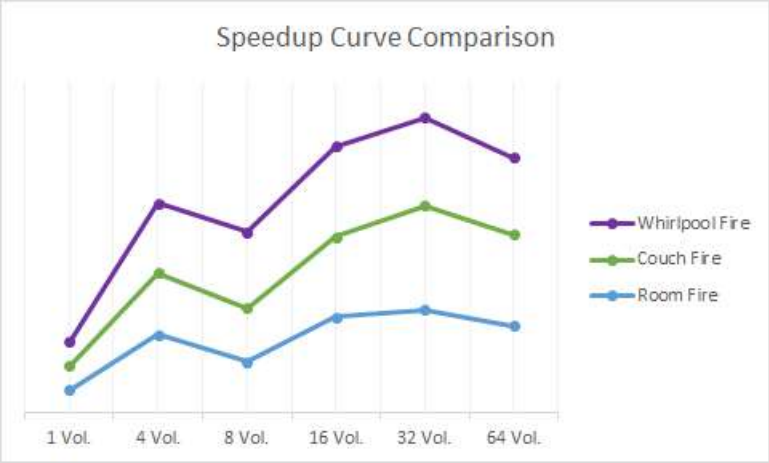

Figure 23: Speedup Curve Comparison Across All Simulations

Recommendations

Automatic Mesh Divisions

If you use `mpiexec -n <cores>` on an FDS input file (*.fds) with a single mesh, as in Figure 24, it will not use more than one mpi process. It does not divide up the mesh for you. This will actually fail and not allow FDS to continue, as can be seen in Figure 25. The user must specify multiple meshes in the fds file in order to take advantage of multicore processing with FDS. It is recommended that the software automatically divide up the meshes once variables have been assigned by the user. The Python script (mesh-slicer.py) written by the author does this type of mesh division (See Appendix).

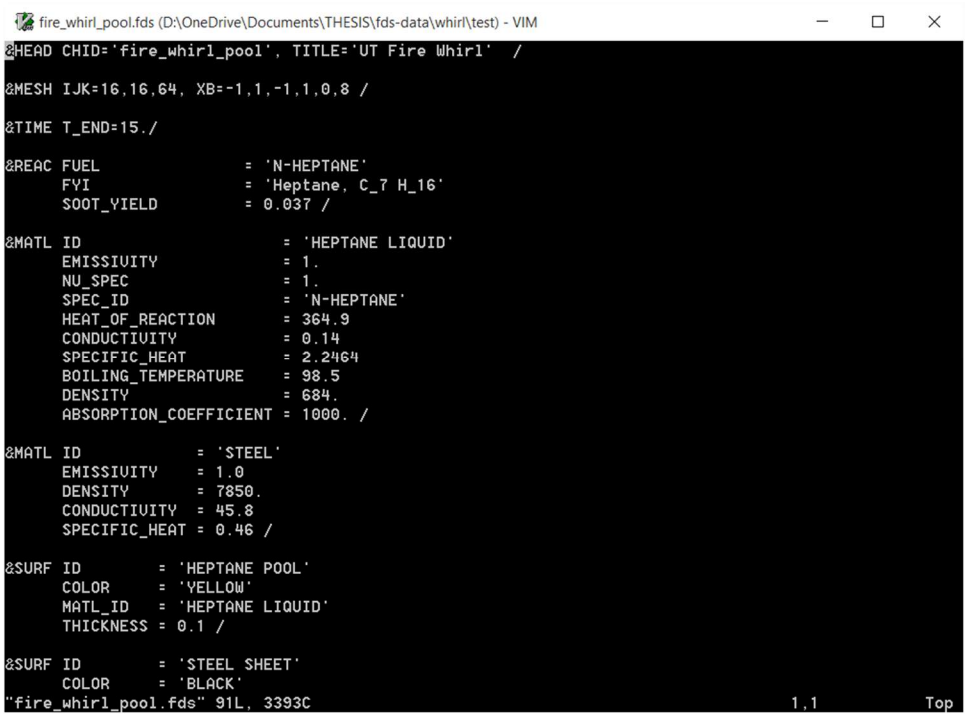

Figure 24: An fds File with a Single Mesh

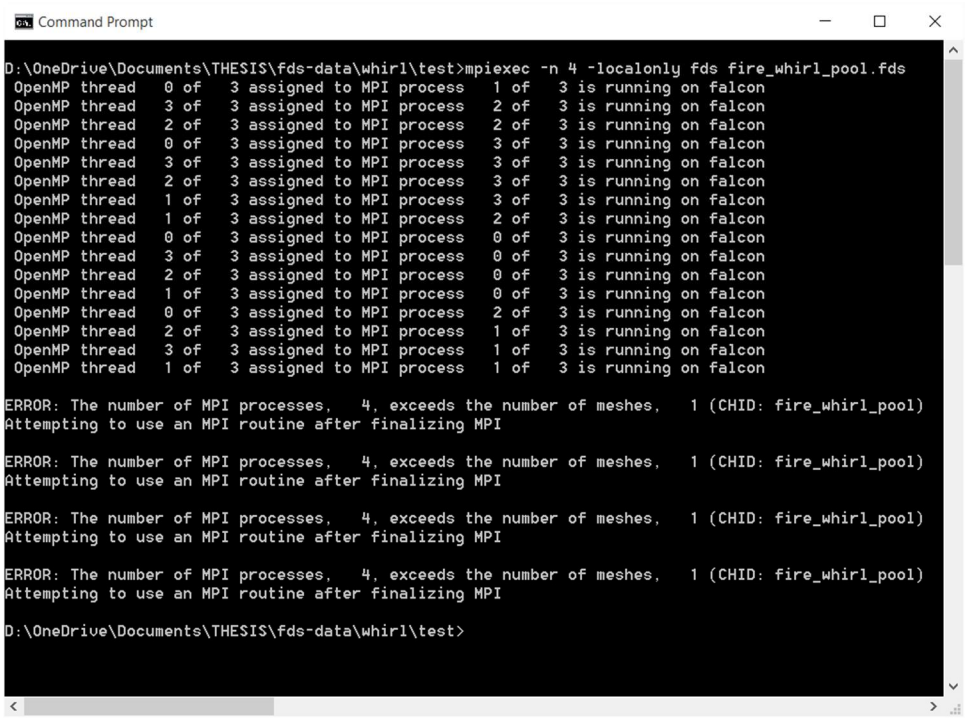

Figure 25: Errors Due to an Inadequate Number of Meshes per Process

Optimized Core Assignment

If you have multiple meshes in the FDS file, as seen in Figure 26, and run `mpiexec -n <cores>`, it will assign each mesh to an mpi process on a single core. However, if there are more meshes than cores, it will assign each individual mesh to a single core until it reaches the last core, as seen in Figure 27. All of the following meshes will be assigned to the final mpi process on the final core, which creates an imbalanced situation. It is recommended that FDS automatically assign the appropriate number of meshes (or volumes) to each mpi process. The Python script (mesh-slicer.py) does this type of balancing (See Appendix).

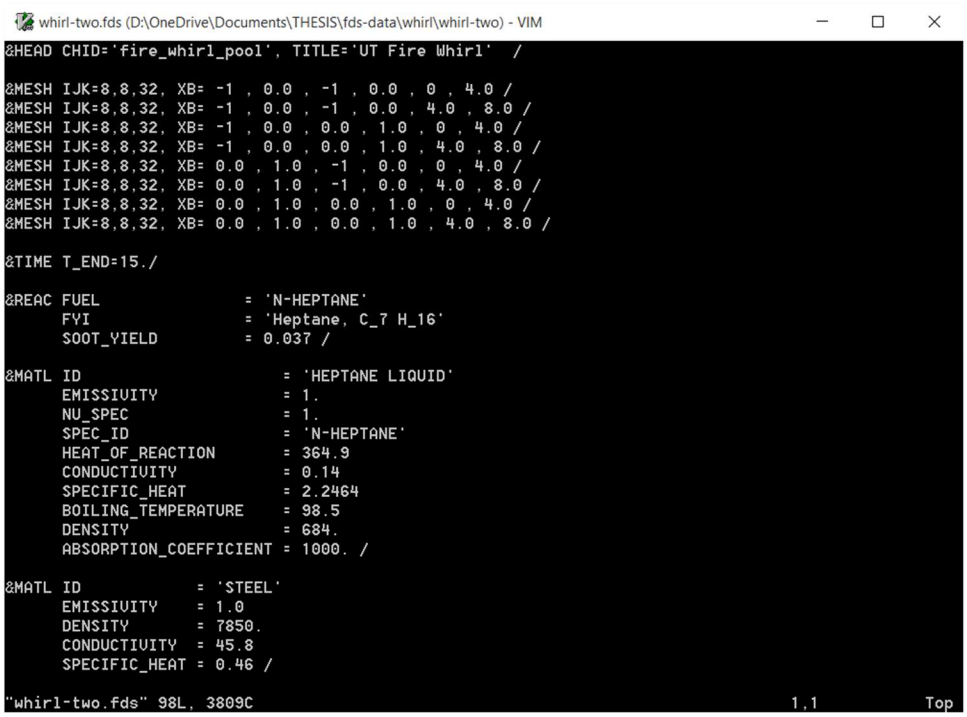

Figure 26: An fds file with 8 Meshes

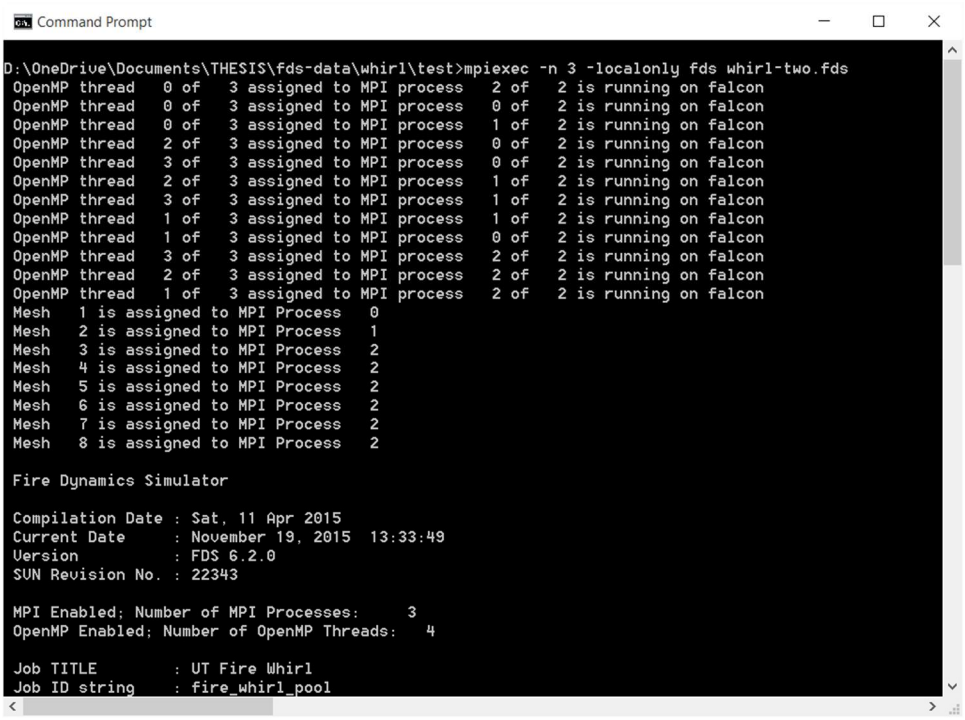

Figure 27: Imbalanced Process Assignment

References

Advanced Micro Devices (AMD). (2015, November 16). *AMD Opteron™ 6200 Series Processor Quick Reference Guide.* Retrieved from Advanced Micro Devices (AMD): https://www.amd.com/documents/opteron_6000_qrg.pdf

McGrattan, K., Hostikka, S., Floyd, J., Baum, H., & Rehm, R. (2007). *Fire Dynamics Simulator (Version 5) Technical Reference Guide.* Washington, DC: NIST Special Publication 1018-5.

McGrattan, K., Hostikka, S., McDermott, R., Floyd, J., Weinschenk, C., & Overholt, K. (2015). *Fire Dynamics Simulator User's Guide* (Sixth ed.). National Institute for Standards and Technology (NIST).

NIST. (2015, November 20). *FDS and SmokeView*. Retrieved from NIST Engineering Laboratory: http://www.nist.gov/el/fire_research/fds_smokeview.cfm

Appendix

Python Script for Creating &MESH lines for the FDS file

```
#-------------------------------------------------------------------------------
# mesh_slicer.py
#
# Written by DC Collins
# University of Tennessee
# EECS Department
#
# This script takes a user's inputs for a Fire Dynamics Simulator file
# (*.fds) mesh and divides them up into multiple meshes for use with a
# multicore system. Formatting works with FDS 6.3.0.
#
# The Python code was written with the assumption that the user is providing
# the correct input data for fire simulation. The Python code does not change
# the cell size that is provided by the user.
#
# Developed for Python 3 on November 3, 2015
#
# Use:
#
# (1) Enter your variables directly into the Python script near the top.
# (2) Run the script
# (3) Copy the output and paste it into your FDS file as &MESH lines
#
# Variables:
#
# volumes - Number of &MESH lines you want to create.
# cores - Number of processing cores you intend to use on the computer.
# Ix, Jy, Kz - These are for the IJK variable on the &MESH line.
# xmin, xmax - Beginning and ending x-axis locations on the &MESH line.
# ymin, ymax - Beginning and ending y-axis locations on the &MESH line.
# zmin, zmax - Beginning and ending z-axis locations on the &MESH line.
#
# Example:
#
# This mesh line in an FDS file:
#
# &MESH IJK=52,54,24, XB=0.0,5.2,-0.8,4.6,0.0,2.4 /
#
# would be represented like this in the code:
#
# &MESH IJK= Ix, Jy, Kz, XB = xmin, xmax, ymin, ymax, zmin, zmax /
#
#-------------------------------------------------------------------------------

import math

#-------------------------------------------------------------------------------
# The user must type in the values for each of the variables in this section
#-------------------------------------------------------------------------------

# Volumes must be divisble by 2

volumes = 32
cores = 32

# This is the IJK part of the mesh line

Ix = 16
Jy =  16
Kz = 64

# This is the XB part of the mesh line
```

```
xmin = -1
xmax = 1
ymin = -1
ymax = 1
zmin = 0
zmax = 8

#-------------------------------------------------------------------------------
# Dividing the mesh into the requested number of volumes
#-------------------------------------------------------------------------------

# Determine how to divide up the volumes

if volumes == 1:
        xElements = 1
        yElements = 1
        zElements = 1

elif volumes == 2:
        xElements = 2
        yElements = 1
        zElements = 1

elif volumes == 4:
        xElements = 2
        yElements = 2
        zElements = 1

elif volumes == 8:
        xElements = 2
        yElements = 2
        zElements = 2

elif volumes == 16:
        xElements = 4
        yElements = 2
        zElements = 2

elif volumes == 32:
        xElements = 4
        yElements = 4
        zElements = 2

elif volumes == 64:
        xElements = 4
        yElements = 4
        zElements = 4

else:
        print("Please choose volumes that are between 1 and 64")

        # Default to 8 volumes

        xElements = 2
        yElements = 2
        zElements = 2

#-------------------------------------------------------------------------------
# Dividing up the IJK values so that they match the mesh divisions
#-------------------------------------------------------------------------------

IxNew = Ix / xElements
JyNew = Jy / yElements
KzNew = Kz / zElements

xrange = round(xmax - xmin, 2)
yrange = round(ymax - ymin, 2)
zrange = round(zmax - zmin, 2)
```

```
xprev=xmin
yprev=ymin
zprev=zmin


#-------------------------------------------------------------------------------
# These nested while loops increment each volume and print out a mesh line
#-------------------------------------------------------------------------------

i = round(xmin + abs(xrange/xElements), 3)
j = round(ymin + abs(yrange/yElements), 3)
k = round(zmin + abs(zrange/zElements), 3)

proc = 0
pCounter = 0
volsPerProc = volumes/cores

while i <= xmax:

        while j <= ymax:

                while k <= zmax:
                        if pCounter >= volsPerProc:
                                pCounter = 0
                                proc = proc + 1

                        if volsPerProc >= 1:
                                print ("&MESH IJK=", IxNew, ",", JyNew, ",", KzNew, "XB=",
                                round(xprev,3), ",", round(i,3), ",",
                                round(yprev,3), ",", round(j,3), ",",
                                round(zprev,3), ",", round(k,3), " MPI_PROCESS =", proc,
"/")
                        else:
                                print ("&MESH IJK=", IxNew, ",", JyNew, ",", KzNew, "XB=",
                                round(xprev,3), ",", round(i,3), ",",
                                round(yprev,3), ",", round(j,3), ",",
                                round(zprev,3), ",", round(k,3),"/")

                        zprev=k
                        k=round(k+abs(zrange/zElements),3)
                        pCounter = pCounter + 1

                k=round(zmin + abs(zrange/zElements),3)
                zprev=zmin
                yprev=j
                j=round(j+abs(yrange/yElements),3)
        j=round(ymin + abs(yrange/yElements),3)
        yprev=ymin
        xprev=i
        i=round(i+abs(xrange/xElements),3)



elements=8

xmin = -1
xmax = 1
ymin = -1
ymax = 1
zmin = 0
zmax = 8

xrange = round(xmax - xmin, 2)
yrange = round(ymax - ymin, 2)
zrange = round(zmax - zmin, 2)

#print ("&MESH IJK=24,10,24, XB=",
```

```
#       xmin, ",", xmax, ",",
#       ymin, ",", ymax, ",",
#       zmin, ",", zmax)

#print (xrange, yrange, zrange)

xprev=xmin
yprev=ymin
zprev=zmin

i = round(xmin + abs(xrange/elements), 3)
j = round(ymin + abs(yrange/elements), 3)
k = round(zmin + abs(zrange/elements), 3)

#print(i, j, k)

while i <= xmax:

        while j <= ymax:

                while k <= zmax:
                        print ("&MESH IJK=16,16,64, XB=",
                          round(xprev,3), ",", round(i,3), ",",
                          round(yprev,3), ",", round(j,3), ",",
                          round(zprev,3), ",", round(k,3), "/")
                        #print(xprev, i, yprev, j, zprev, k)
                        zprev=k
                        k=round(k+abs(zrange/elements),3)

                k=round(zmin + abs(zrange/elements),3)
                zprev=zmin
                yprev=j
                j=round(j+abs(yrange/elements),3)
        j=round(ymin + abs(yrange/elements),3)
        yprev=ymin
        xprev=i
        i=round(i+abs(xrange/elements),3)
```

Vita

Donald Charles Collins spent his childhood in Bolivar, Tennessee. After graduating from high school at Bolivar Central in 1989, Donald joined the United States Navy and was trained as an Aviation Electronics Technician and Aircrewman on the C-9B aircraft. In 1994, he left active duty and moved to Knoxville, Tennessee where he met his wife Julia. For several years, Donald worked his way up from a computer technician to a systems administrator on high performance systems. In 2004, he received his Bachelor of Science degree in Information Technology from American Intercontinental University. In 2014, Donald received a second Bachelor of Science in Computer Engineering from the University of Tennessee. Over the past several years, Donald has been serving as a commissioned Information Professional officer in the United States Navy Reserve. In his civilian career, he has worked with URS Corporation, the Innovative Computing Laboratory (ICL) and the National Institute for Computational Science (NICS). Currently, Donald is providing computer engineering consulting services at Oak Ridge National Laboratory (ORNL).