ParaTools ThreadSpotter Analysis of HELIOS

ParaTools, Inc.
2836 Kincaid St.
Eugene, OR 97405
(541) 913-8797
info@paratools.com

1 Introduction

High performance computing requires good programming techniques to ensure good scalability, and effective use of compute resources. Performance analysis tools help software developers improve efficiency and increase software performance for efficient use of HPC systems. ParaTools ThreadSpotter conducts performance analysis by collecting data on data movement and reuse.

1.1 *ParaTools ThreadSpotter*

ParaTools ThreadSpotter differs from other profiling tools in its ability to analyze, interpret, and prioritize performance bottlenecks and relate these back to the developer in plain English. It is easy to use since it operates on un-modified binaries and captures sparse memory fingerprints through periodic sampling. It presents its findings and recommendations in an HTML report that shows key memory problems alongside the application source code and callgraph, making it easy to see where improvements may be introduced into the code. It can detect a range of issues including wasted cache space, false sharing of cache lines, and a lack of data reuse, all of which squander precious l1 and l2 cache and are exacerbated by increasing the number of threads per core. No other tool has these capabilities.

ParaTools enhanced ThreadSpotter by making it easier to work with MPI applications. This short tutorial walks through the execution of a basic ThreadSpotter Analysis using the CREATE-AV Helios application running v7-tram on lightning.

2. Preparing and collecting performance data

ThreadSpotter works best with the Intel compiler. The important modules to load for building on lightning are the following modules: create, PrgEnv-intel, intel/14.0.2.144. I executed the following when preparing to build and run Helios on lightning with ThreadSpotter:

```
module load create
module swap PrgEnv-cray PrgEnv-intel
module swap intel intel/14.0.2.144
module swap cray-mpich cray-mpich/7.2.6
```

Depending on what modules you already have loaded these may or may not be the appropriate commands to run. Here is a full list of modules loaded for the Helios binary in this analysis:

```
module list
Currently Loaded Modulefiles:
  1) modules/3.2.10.3                    16)/app/startup/login2.module
  2) eswrap/1.3.3-1.020200.1278.0        17) create
  3) switch/1.0-1.0502.60522.1.61.ari    18) cray-libsci/13.2.0
  4) craype-network-aries         19) udreg/2.3.2-1.0502.10518.2.17.ari
  5) intel/14.0.2.144             20) ugni/6.0-1.0502.10863.8.29.ari
  6) craype/2.4.2              21) pmi/5.0.10-1.0000.11050.0.0.ari
  7) cray-mpich/7.2.6          22) dmapp/7.0.1-1.0502.11080.8.76.ari
  8) pbs/12.2.404.152084       23) gni-headers/4.0-1.0502.10859.7.8.ari
  9) craype-ivybridge          24) xpmem/0.1-2.0502.64982.5.3.ari
 10) java/jdk1.8.0_51          25) dvs/2.5_0.9.0-1.0502.2188.1.116.ari
 11) /app/startup/shell.module 26) alps/5.2.4-2.0502.9774.31.11.ari
 12) /app/startup/alias.module 27) rca/1.0.0-2.0502.60530.1.62.ari
 13) costinit                  28) atp/1.8.3
 14) /app/startup/login.module 29) PrgEnv-intel/5.2.82
 15) /app/startup/set_ACCOUNT.module
```

Prior to building run:

```
source $CREATE_HOME/av/externals/ptoolsrte-
0.5.2alpha5_sles11sp1_x86_64/etc/ptoolsrte.bashrc.intel14.0
.2.144_mpich7.2.6
```

Next add -g so that HELIOS is built with symbols and ThreadSpotter will be able to resolve line numbers and provide more meaningful information. To do this add "-g" to Intel compiler family flags in `cmake/setup.cmake` and `cmake/avcore.cmake`. Next build HELIOS in the usual fashion. For the binary used in this study, this was done with the following commands:

```
mkdir build
cd build
../configure -t ../config/install/lightning.xml -v -i .
```

It is important to verify that ThreadSpotter and TAU are properly installed on the system and to update your path to include the appropriate directory. At the time of this tutorial these packages were installed on lightning and the appropriate directories to add to the path were:

`$PET_HOME/pkgs/threadspotter/bin (or /app/comenv/pkgs/threadspotter/bin)`

`$PET_HOME/pkgs/tau/craycnl/bin (or /app/comenv/pkgs/tau/craycnl/bin)`

The latest stable version of ParaTools ThreadSpotter is installed on many of the DSRC systems. ParaTools ThreadSpotter can be used by adding `$PET_HOME/pkgs/threadspotter-VERSION/bin` to `$PATH`. Likewise, TAU is typically in `$PET_HOME/pkgs/tau-VERSION/craycnl/bin` (or `$PET_HOME/pkgs/tau-VERSION/x86_64/bin` on a SGI systems).

2.1 Collecting Data

To collect data the directories above must be listed in the path when Helios is executed. ThreadSpotter is invoked using extensions to TAU. The flags to invoke threadspotter are -ptts options together with the --tau option. The .pbs script used to collect this data is shown below:

```
#!/bin/bash
#PBS -A <PROJECTNAME>
#PBS -l walltime=05:10:00
#PBS -N sphere
#PBS -q standard
#PBS -j oe
#PBS -l application=helios
#PBS -l select=32:ncpus=24:mpiprocs=8
#PBS -V

echo $PBS_O_WORKDIR
cd $PBS_O_WORKDIR

# too paths here
export PATH=$PATH:$PET_HOME/pkgs/threadspotter/bin:$PATH
export PATH=$PATH:$PET_HOME/pkgs/tau/craycnl/bin:$PATH

# Check -aprun-num matches csi "-p" flag
export TAU_EXEC_FLAGS="-T helios -ptts -ptts-num=256 -ptts-sample-flags=-g1"
# YOUR PATH HERE
/p/home/drmackay/HELIOS-v7-r820/build/bin/csi --tau -p 256
```

It is not necessary to collect ThreadSpotter data for a long run. The data collected here was run for 7 time steps, but 3 or 4 should be sufficient. So adjust nsteps to an appropriate number. ThreadSpotter will cause time dilation and the code will run slower. By default ThreadSpotter will collect data and write a .smp file for each MPI rank. This file is then processed into a report (.tsr file), and this can then be processed into an html report. The ThreadSpotter options used for Tau in this case handles the conversion of the .smp files into .html viewable reports.

3. Viewing reports and performance data

When the job completes there will be a new directory named ptts. The directory ptts contains the data for each MPI rank. Typically a developer will open the reports by opening ./ptts/index.html in a web browser such as firefox. I did not find a web browser on lightning so the data was moved to thunder and the data was viewed by going to the ptts directory and entering: `firefox index.html`. The resulting view is shown in Figure 1.

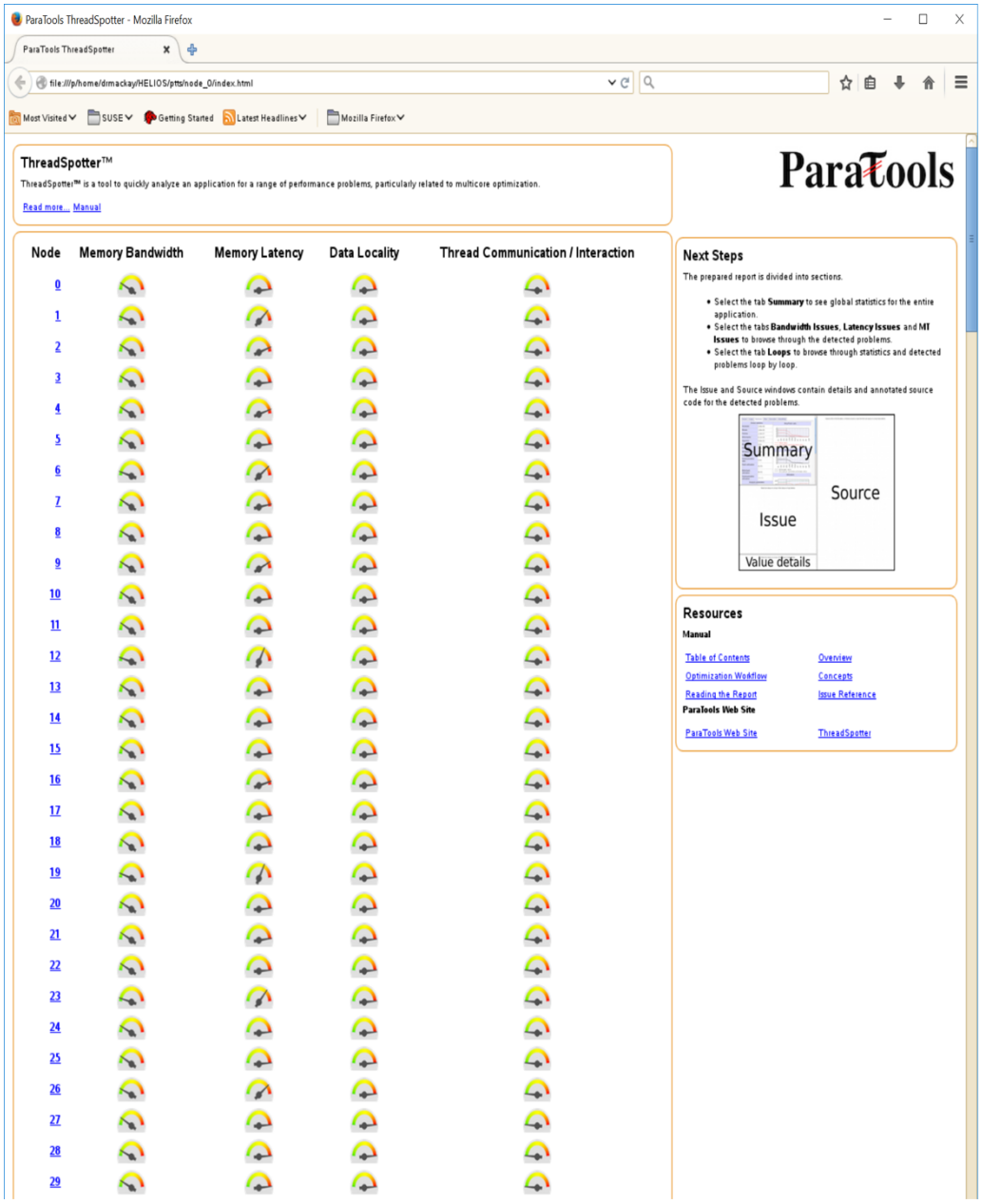

Figure 1: View of index.html produced by ThreadSpotter.

Please notice there is a line for each MPI rank. In addition, there are four columns: Memory Bandwidth, Memory Latency, Data Locality and Thread Communication/Interaction. There is a mini dashboard for each column and each row. Scanning the first view it is evident that Memory Bandwidth is not critical bottleneck for any of the MPI ranks. Data Locality is consistently high on the dashboard as a bottleneck for all MPI ranks. The importance of Memory Latency varies across the MPI ranks. To see the specifics the developer simply clicks

on the rank number in the far left column to see the data for a specific rank number. In this case I select mpi rank 13, the resulting view is shown in Figure 2.

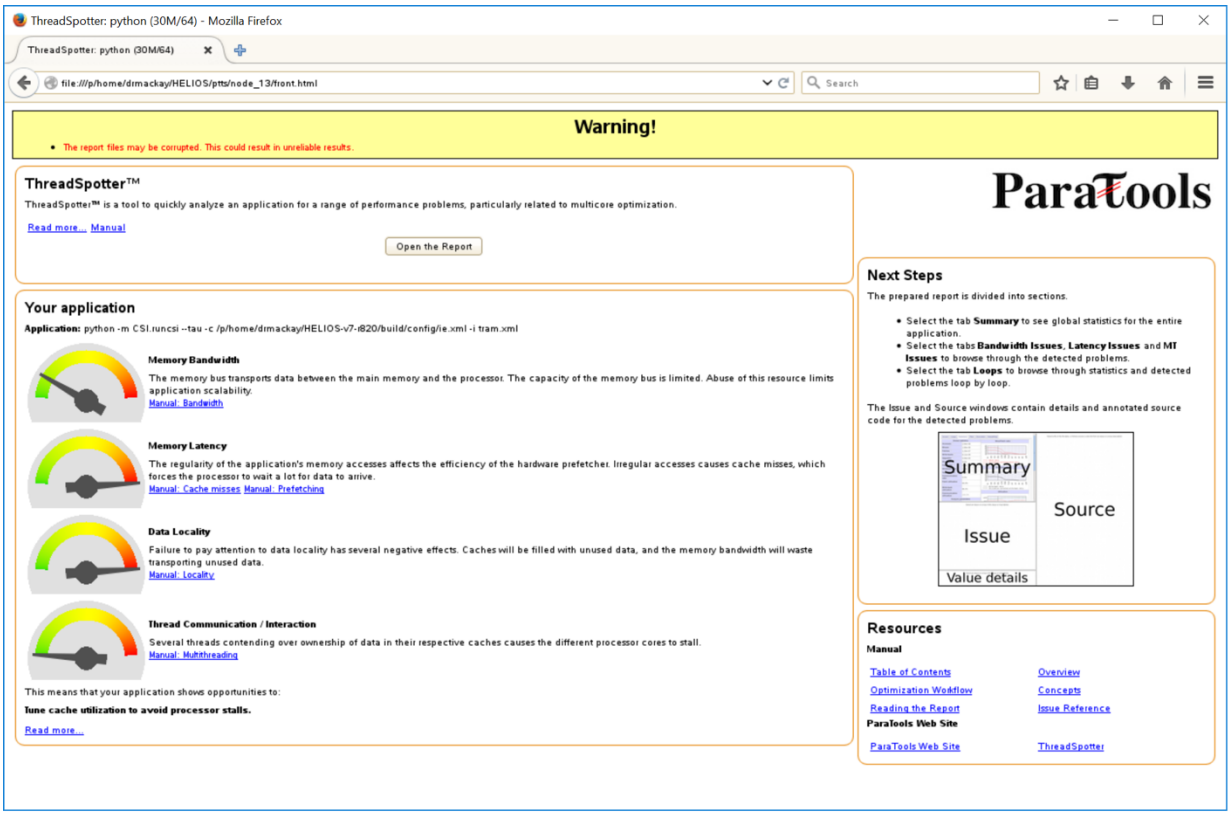

Figure 2: MPI Rank 13 summary report.

Selecting the button “Open the Report” provides a list of all the issues identified by ParaTools ThreadSpotter. This list provides an issue number for reference as well as issue type and information about bandwidth, fetches, write-backs, fetch utilization and write-back utilization. This is shown below in Figure 3.

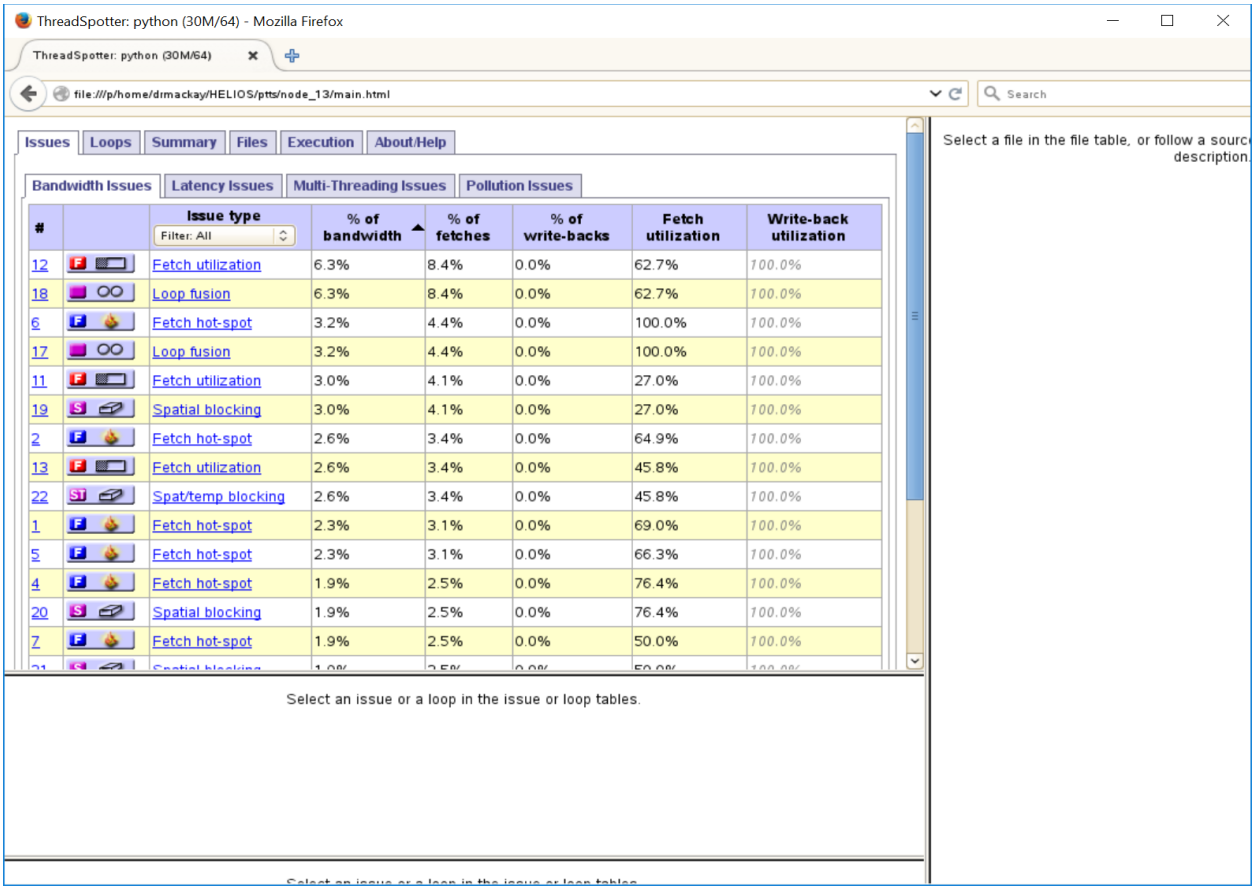

Figure 3: Node 13 initial issue view.

Normally at this point we would examine some of the issues reported. In this tutorial we want to illustrate the best use of cache analysis. The default settings for ParaTools ThreadSpotter is to provide a report based on L3 cache model. On the lightning system it is felt an l2 cache analysis model will provide better hints for tuning. The l2 cache analysis will provide tuning tips that will carry over to tuning for Intel many core Xeon Phi systems as well. So we will show how to generate an l2 cache model report. The information collected about data movement and usage is sufficient for generating an l2 cache report. It is not necessary to collect data again. To generate the desired report we set the following environment variables: `TAU_TS_REPORT_FLAGS="--level 2 -r lru"`. This instructs the ThreadSpotter analysis to consider l2 cache movement using a least recently used replacement policy for the cache. To the runtime option we add: `-ptts-post` which tells Tau to just run post-process analysis on the existing .smp files. The batch file to generate level 2 cache analysis with a least recently used analysis policy is shown below:

```
PBS -A <project id>
#PBS -l walltime=xx:yy:zz
#PBS -N sphere
#PBS -q standard
#PBS -j oe
#PBS -l application=helios
#PBS -l select=32:ncpus=24:mpiprocs=8
#PBS -V

echo $PBS_O_WORKDIR
cd $PBS_O_WORKDIR

# YOUR PATH HERE
export PATH=$PATH:$PET_HOME/pkgs/threadspotter/bin:$PATH
export PATH=$PATH:$PET_HOME/pkgs/tau/craycnl/bin:$PATH

# Check -aprun-num matches csi "-p" flag
export TAU_TS_REPORT_FLAGS="--level 2 -r lru"
export TAU_EXEC_FLAGS="-T helios -ptts -ptts-num=256 -ptts-sample-
flags=-g1 -ptts-post"
/p/home/drmackay/HELIOS-v7-r820/build/bin/csi --tau -p 256
```

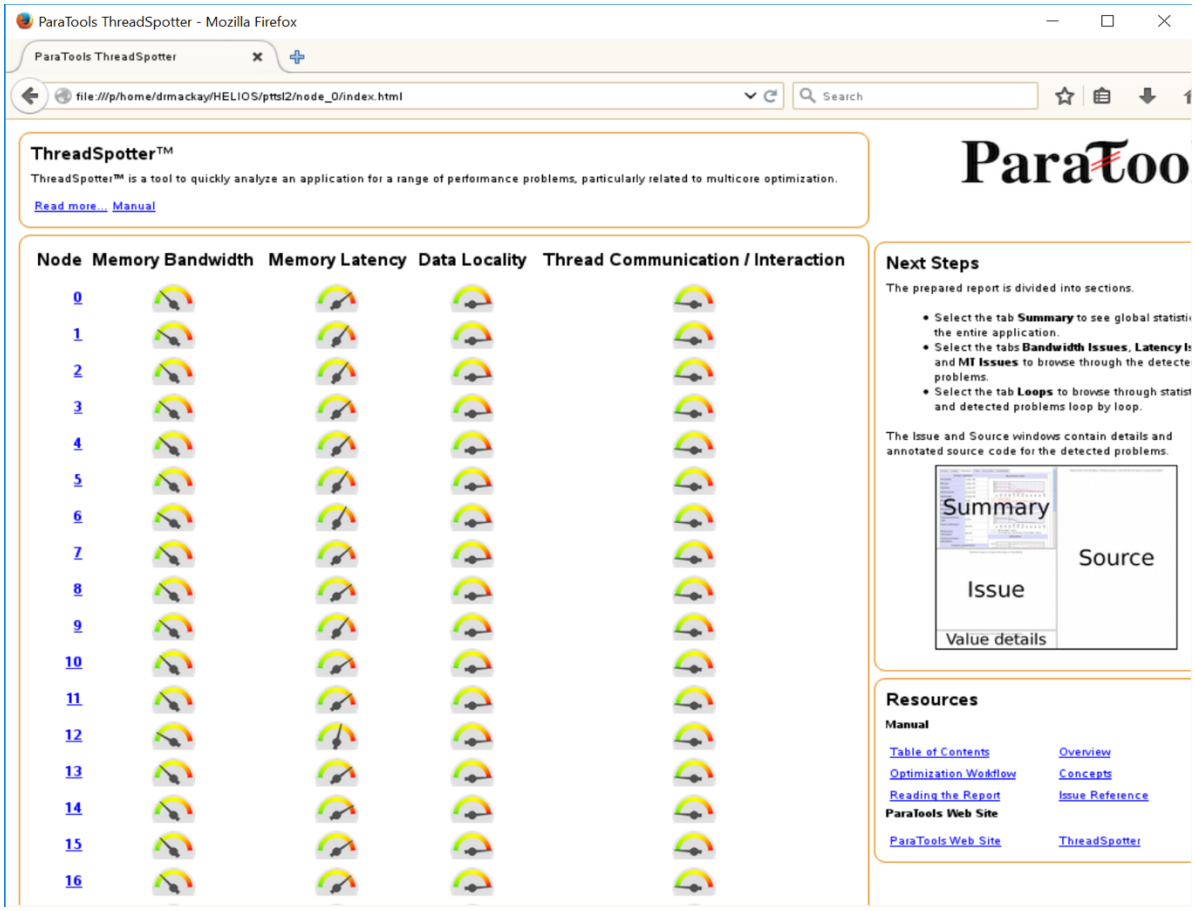

Figure 4: Analysis for l2 cache with Least Recently Used policy.

The index report generated is very similar to the original L3 cache report. A noticeable exception is that the impact of memory latency is not as high. Data locality remains the predominant bottleneck (See Figure 4).

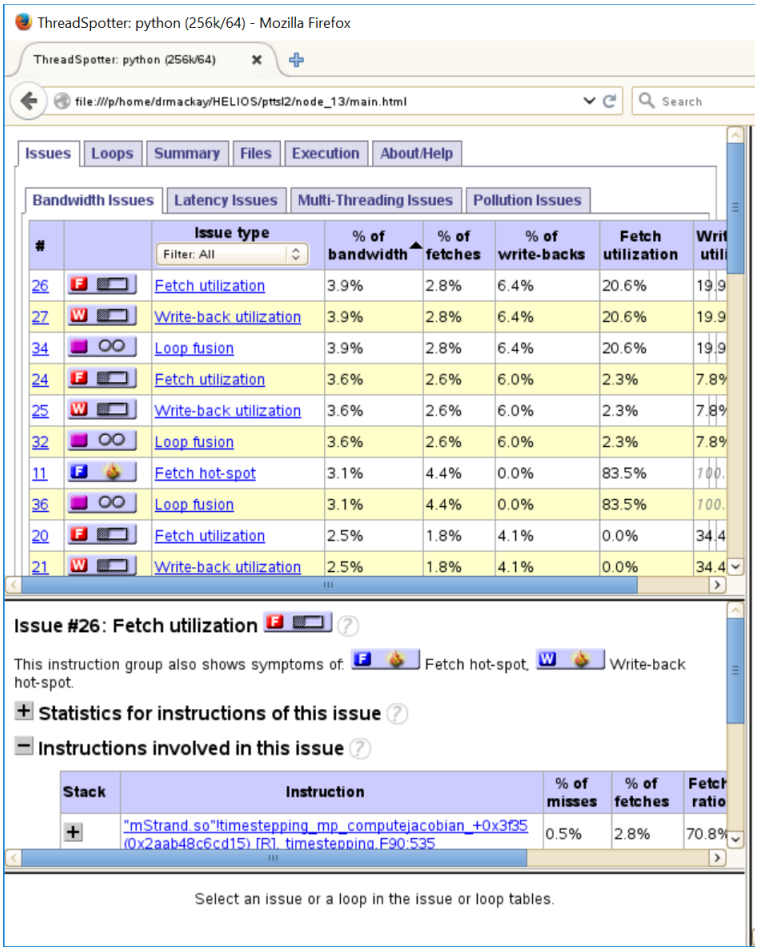

Figure 5: Details of issue 26 - Fetch Utilization (Node 13).

Once again node 13 is selected for more detailed analysis. We immediately select "Open the Report" button as was previously shown; this document skips a redundant screenshot of that step. Now we drill down into one of the issues. Click on issue 26 on the top left hand side opening up details of issue 26 – Fetch Utilization. The details for issue 26 open up as shown in Figure 5.

The right-hand pane is now filled with the source code. The pane below the list provides details such as statistics and instructions involved. You may need to click the box in the bottom pane to expand the details displayed. To reverse this, click on the box and the details will be hidden and return to a box with the heading next to it. It is important to note that there is not just a single issue associated with the code segments. Fetch Utilization is listed as the highest issue identified. The report shows that Fetch-hotspot is also associated with this line of code. Looking at line 535 in the annotated source code there are additional boxes associated with line 535 – the boxes associated with line 535 represent Fetch Utilization, Fetch Hotspot and loop fusion. If

you click on the box of each line of 535 the associated issue will be highlighted in the bottom pane on the left-hand side. Clicking those boxes on line 535 reveals that the top three issues in this report; issues 26, 27 and 34 all involve line 535 (Alternatively you can click on the number in the far-left column and see that each of the top three issues involves the same line). If you click on the circle containing a question mark (?) next to the Issue #26 Fetch Utilization line in the lower pane – a new window will open up explaining what fetch utilization means in ThreadSpotter reports. Fetch utilization percentages are a percentage of a cache line that is used by the code operations. For example if I were striding through a linear one dimensional array by 2 (array[0], array[2], array[4], . . .) using only half the elements in the array. The computer will bring in the full cache line which includes array[1], array[3], . . . but those data items are not actually used – so I am only utilizing half of the data I am bringing into cache; this is a 50% fetch utilization. To improve fetch utilization data might be restructured so that the storage pattern matches the data access patterns.

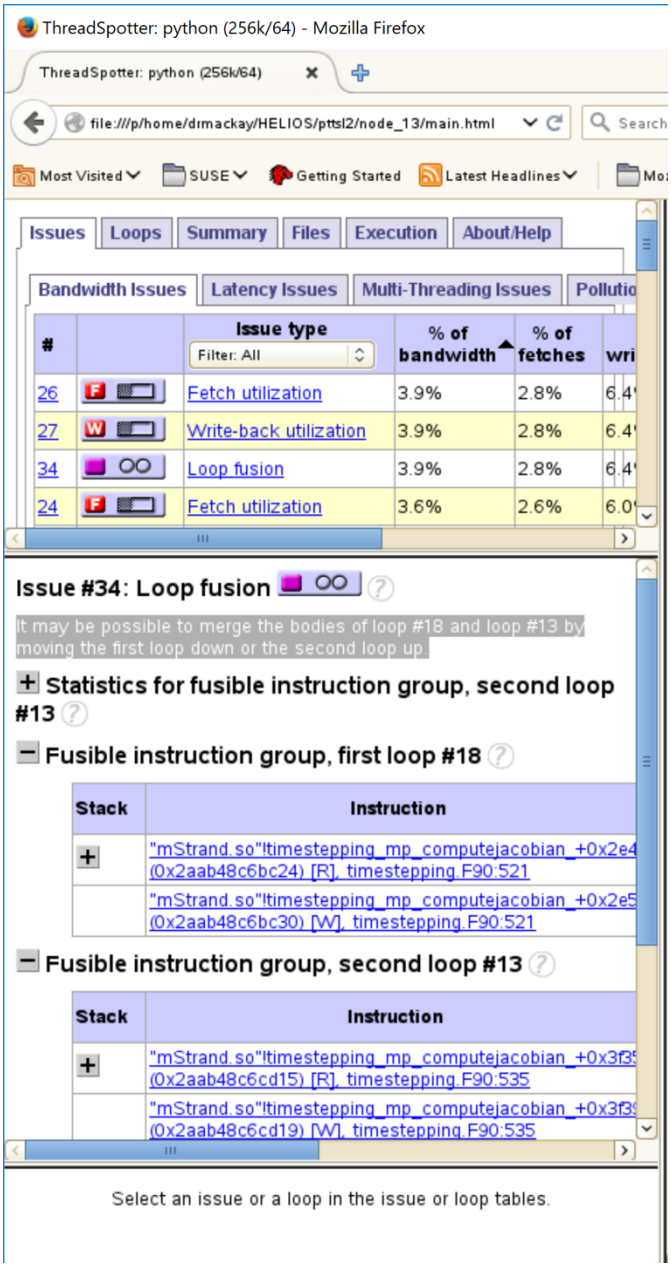

Figure 6: Issue 34 -loop fusion report Node 13.

In Figure 6 the view is shown when the loop fusion box (purple icon) is clicked in the right-hand pane on line 535. This particular issue suggests the developer explore methods to fuse the loop in file timestepping.F90 line 535 and line 521. All three of the issues – 26, 27, 34 relate to data reuse around line 535. Suggestions to improve performance are to structure the code to reuse data while it is in cache or even better while it is in the processor registers. The loops in lines 521 and 535 each stride through the same array and each loop brings it back into cache. By the

time a loop is completed the beginning of an array or the data at the beginning of the array may be flushed from cache and need to be fetched in again for the next loop. By exploring loop fusion or restructuring the code to work in blocks or chunks of data it may be possible to perform the operations of each associated loop on the data while it is the processor registers or in the close l2 cache. This ordering of data use is the predominant suggestion ParaTools ThreadSpotter analysis reveals. Restrictions on listing source code prevent more detailed analysis presentation in this document. The top 6 issues reported in this ThreadSpotter analysis all relate to the code segments between lines 517 and line 535.

4. Summary

ThreadSpotter has been enhanced to work with multi-langauge codes such as HELIOS and work well in an MPI environment. This tutorial demonstrates how ThreadSpotter can be run on HELIOS to generate performance analysis reports. Navigation of the performance analysis reports are demonstrated to view the various issues reported. The method to find additional information about any topic in the help can be found for clicking on the circle question mark {?) next to any issue and additional information explaining what the issue means and possible ideas for tuning the software will be opened up.