COMPARING ALLINEA'S AND INTEL'S PERFORMANCE TOOLS FOR HPC

Glenn R. Luecke
Department of Mathematics
Iowa State University
Ames, IA, 50011, USA
grl@iastate.edu

Brandon M. Groth
Department of Mathematics
Iowa State University
Ames, IA, 50011, USA
bmgroth@iastate.edu

Nathan T. Weeks
Department of Computer Science
Iowa State University
Ames, IA 50011, USA
weeks@iastate.edu

Marina Kraeva
Information Technology Services
Iowa State University
Ames, IA, 50011, USA
kraeva@iastate.edu

ABSTRACT

To efficiently use HPC machines, it is critical to optimize applications for high performance. To accomplish this, HPC developers must utilize performance tools to find and correct performance problems within large, complex scientific applications. Allinea and Intel offer vendor-supported performance tools that are regularly updated to capture important performance metrics on the latest hardware. In this paper, the authors evaluated and compared Allinea's MAP performance tool and Intel's performance tools to aid in application optimization. The authors found that Allinea's MAP provided useful performance metrics necessary to diagnose and fix performance problems using an intuitive, easy-to-use user interface. Intel's performance tools provided a more detailed and customizable view of application performance, at the expense of a more complicated user interface. The comparison presented in this paper will help HPC developers decide which performance tool is best for them.

Keywords: MAP, Performance Reports, Trace Analyzer, VTune

1 INTRODUCTION

Training application developers to write high performance applications is increasingly important for today's commercial, government and research organizations. This is challenging since HPC architectures and programming models are rapidly changing. Being able to evaluate an application's performance without special tools can be a difficult and time-consuming task. Performance tools are needed to evaluate application performance and identify bottlenecks, but if the tools are difficult to use, then few will use them. The purpose of this study is to evaluate and compare Allinea's and Intel's performance tools not only for the functionality needed to optimize applications, but also for ease-of-use.

Allinea offers two tools: Allinea MAP and Allinea Performance Reports. MAP and Performance Reports are currently being used at many universities and US government labs such as Oak Ridge National Laboratory, Los Alamos National Laboratory, and the National Energy Research Scientific Computing Center (NERSC). Intel offers Intel Parallel Studio, which contains the following performance tools used in this study: Intel Profile Function and Loop Execution Time, Intel Trace Analyzer and Collector, and Intel VTune Amplifier. Many organizations use Parallel Studio because it contains Intel's compilers, performance analysis tools, and their optimized libraries.

Some other tools available are TAU (University of Oregon), HPCToolkit (Rice University), Scalasca (Forschungszentrum Jülich), and Vampir (Dresden University of Technology). While TAU, HPCToolkit, and Scalasca are freely available, vendors and the HPC support staff at Iowa State University currently do not support them. Thus, we did not include these tools in this study.

In 2010, Marowka reported on the functionality and ease-of-use of the Intel Thread Profiler for Windows (Marowka 2010). In 2004, Collette et al. published a summary and comparison of many debuggers and performance tools used in HPC at the time (Collette, Corey, and Johnson 2004). Furthermore, Appelbe et al. in 1996 gave a summary of current software as well as recommendations for future improvements (Appelbe and Bergmark 1996). In addition, Moore et al. published a review of performance analysis tools for MPI in 2001 (Moore et al. 2001). The authors are not aware of any recent study dealing with Allinea's or Intel's performance tools.

This paper is organized as follows. Section 2 discusses our methodology and introduces the epiSNP bioinformatics code used in this study. Sections 3 and 4 discuss Allinea's MAP and Performance Reports. Sections 5, 6, and 7 discuss Intel's Profile Function and Loop Execution Time (PFLET), Trace Analyzer and Collector, and VTune Amplifier, respectively. Sections 8 contains the summary and conclusions.

2 METHODOLOGY

The evaluations in this study are for serial programs, and programs using MPI, OpenMP, or both MPI+OpenMP. The following evaluation categories represent what the authors consider to be most important when using a performance tool in an HPC educational environment:

- Was the tool easy to use when compiling and running applications?
- Is the GUI easy to use?
- Does the tool clearly present profiling data for lines, functions, and loops?
- Does the tool clearly present CPU, memory, and I/O data?
- Can the tool detect MPI and OpenMP load-balancing problems?
- Can the tool handle long-running jobs?

The evaluation was performed with a bioinformatics Fortran application comprised of 2000+ lines of code, called epiSNP, written by Ma (Ma et al. 2008) and optimized by Weeks (Weeks et al. 2016) aided by Allinea MAP and Intel's PFLET. There are several versions of the optimized epiSNP, including a serial version and a hybrid MPI+OpenMP version. For this study, we used the optimized serial epiSNP with PFLET and the optimized MPI+OpenMP epiSNP for all other tools. epiSNP was launched on 4 nodes with two 8-core sockets using 1 MPI rank per socket and 8 OpenMP threads per MPI rank. All runs were performed on Iowa State University's CyEnce Cluster (see http://www.hpc.iastate.edu/systems). The MPI+OpenMP epiSNP ran with this configuration had an approximate runtime of 7 hours with the 4.4GB data set used in Weeks (Weeks et al. 2016).

The following lists the versions of the software used for this study:

- Allinea Forge version 6.0
- Allinea Performance Reports version 6.0
- Intel Parallel Studio XE 16.0

Allinea MAP and Intel VTune can be used with accelerators, however we did not evaluate this feature for the following reasons. Allinea Forge requires an additional license, which we did not have at Iowa State University. Furthermore, we were not able to run VTune with the Intel Xeon Phi on CyEnce, even though we carefully followed the Intel documentation.

3 ALLINEA MAP

Allinea MAP is a performance tool that collects statistical samples for each line of code in an application. MAP offers profiling support for code written in Fortran, C, and C++.

To use MAP on an MPI+OpenMP Fortran program, called prog.f90, the only requirement is to compile the application with the "-g" compiler option to add symbolic debugging information to the executable:

```
mpiifort -g -qopenmp prog.f90
map ./a.out
```

The map command will launch the MAP Run Window via X11 forwarding and produce a .map file after the job has completed. When using MPI or OpenMP, one will have to provide the appropriate configuration before submitting the job. MAP can also be run without a GUI from a job script, by prefixing the run command with "`map --profile`", for p MPI processes:

```
map --profile mpirun -np p ./a.out
```

At the end of execution with "−−profile", a .map file will be generated. Once a .map file has been created, it can be used to examine MAP performance data without rerunning the application. This is accomplished by selecting Load Profile Data File in the Start Menu or by issuing:

```
map ./<MAP_file>.map
```

The “−−profile” option is useful for long-running jobs, since the profile data will be lost if the interactive X session is interrupted by network connection loss.

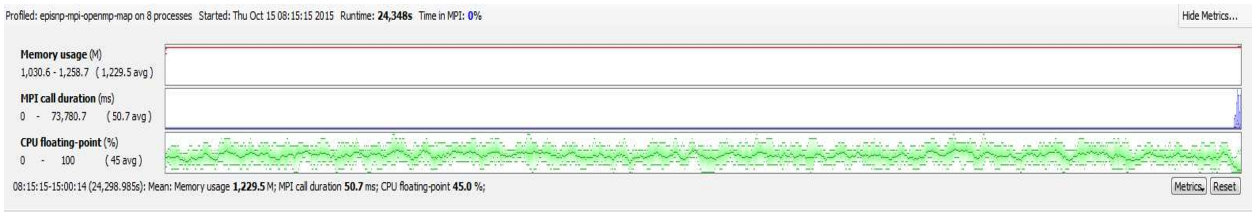
The GUI is laid out as follows:

- Top panel (Metric View): View a time line of several metrics (Figure 1).
- Middle panel (Source Code View): View statistical timing information for each line of the source code (Figure 2).
- Bottom panel (Stacks View): View a top-down tree of longest running lines/functions of code in an application (Figure 3).

In the Metric View, users can zoom in on a certain time interval by performing a "click-and-drag" on the metric’s desired time frame. The other views will update to use this interval of time for execution time percentages and hotspots. Load imbalance between MPI processes or OpenMP threads can also be inferred by using the Metrics View.

MAP also has the ability to edit and recompile the source code within the GUI. It has built-in text editing options such as undo, redo, copy, paste, etc. The MAP GUI will not update to any changes until the edited code has been saved, compiled, and profiled.

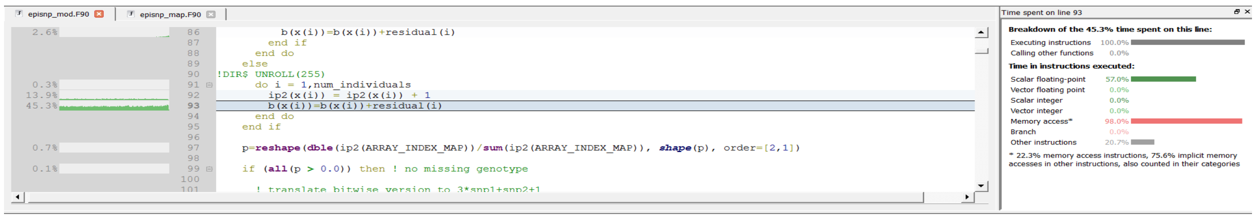
The MAP GUI uses an X connection. Alternatively, users can download the Allinea Forge Remote Client to connect to remote clusters via SSH without X11 forwarding. The Remote Client provides a more responsive GUI by avoiding the latency of X11 forwarding. This is especially useful when there is a high-latency connection between the workstation and the cluster.


Profiled: episnp-mpi-openmp-map on 8 processes Started: Thu Oct 15 08:15:15 2015 Runtime: **24,348s** Time in MPI: **0%**

Memory usage (M)	1,030.6 - 1,258.7 (1,229.5 avg)
MPI call duration (ms)	0 - 73,780.7 (50.7 avg)
CPU floating-point (%)	0 - 100 (45 avg)

08:15:15-15:00:14 (24,298.985s): Mean: Memory usage **1,229.5** M; MPI call duration **50.7** ms; CPU floating-point **45.0** %;

Figure 1: MAP Metric View with default metrics.



```
  2.6%   86           b(x(i))=b(x(i))+residual(i)
         87         end if
         88       end do
         89     else
         90 !DIR$ UNROLL(255)
  0.3%   91       do i = 1,num_individuals
 13.9%   92         ip2(x(i)) = ip2(x(i)) + 1
 45.3%   93         b(x(i))=b(x(i))+residual(i)
         94       end do
         95     end if
         96
  0.7%   97     p=reshape(dble(ip2(ARRAY_INDEX_MAP))/sum(ip2(ARRAY_INDEX_MAP)), shape(p), order=[2,1])
         98
  0.1%   99     if (all(p > 0.0)) then ! no missing genotype
        100
        101       ! translate bitwise version to 3*snp1+snp2+1
```

Time spent on line 93

Breakdown of the 45.3% time spent on this line:

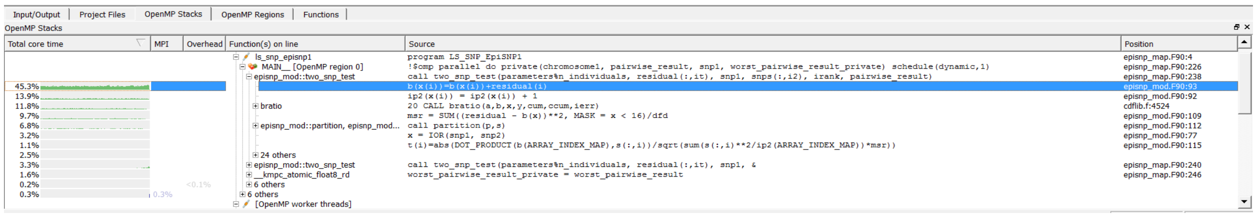
Executing instructions	100.0%
Calling other functions	0.0%

Time in instructions executed:

Scalar floating-point	57.0%
Vector floating point	0.0%
Scalar integer	0.0%
Vector integer	0.0%
Memory access*	98.0%
Branch	0.0%
Other instructions	20.7%

* 22.3% memory access instructions, 75.6% implicit memory accesses in other instructions, also counted in their categories

Figure 2: MAP Source Code View and Selected Line View (right).



OpenMP Stacks

Total core time	MPI	Overhead	Function(s) on line	Source	Position
			ls_snp_episnp1	program LS_SNP_EpiSNP1	episnp_map.F90:4
			MAIN__ [OpenMP region 0]	!$omp parallel do private(chromosome1, pairwise_result, snp1, worst_pairwise_result_private) schedule(dynamic,1)	episnp_map.F90:226
			episnp_mod::two_snp_test	call two_snp_test(parameters%n_individuals, residual(:,it), snp1, snps(:,i2), irank, pairwise_result)	episnp_map.F90:238
45.3%				b(x(i))=b(x(i))+residual(i)	episnp_mod.F90:93
13.9%				ip2(x(i)) = ip2(x(i)) + 1	episnp_mod.F90:92
11.8%			bratio	20 CALL bratio(a,b,x,y,cum,ccum,ierr)	cdflib.f:4524
9.7%				msr = SUM((residual - b(x))**2, MASK = x < 16)/dfd	episnp_mod.F90:109
6.8%			episnp_mod::partition, episnp_mod...	call partition(p,s)	episnp_mod.F90:112
3.2%				x = IOR(snp1, snp2)	episnp_mod.F90:77
1.1%				t(i)=abs(DOT_PRODUCT(b(ARRAY_INDEX_MAP),s(:,i))/sqrt(sum(s(:,i)**2/ip2(ARRAY_INDEX_MAP))*msr))	episnp_mod.F90:115
2.5%			24 others		
3.3%			episnp_mod::two_snp_test	call two_snp_test(parameters%n_individuals, residual(:,it), snp1, &	episnp_map.F90:240
1.6%			__kmpc_atomic_float8_rd	worst_pairwise_result_private = worst_pairwise_result	episnp_map.F90:246
0.2%		<0.1%	6 others		
0.3%	0.3%		6 others		
			[OpenMP worker threads]		

Figure 3: MAP OpenMP Stack.

3.1 Evaluation

The user can choose between launching the GUI interface on the cluster using a standard job script or using the Remote Client to profile an application. The authors found that the remote client was helpful in keeping the GUI responsive between workstation and remote cluster. With X11 forwarding, the authors experienced slow response times to any actions performed on the GUI. Whereas, with the remote client, the response time dramatically improved.

The authors found that the initial data view was easy to navigate and the time interval feature of the Metric View was especially useful. With statistical changes occurring in all three views for a particular time frame, this can help pinpoint local bottlenecks in source code, as well as what may be causing the bottleneck based on performance metrics. Furthermore, a user can find load imbalance in their application by using this technique. Also, the pre-made groups in the drop-down menu provide good coverage of important metrics such as vectorization, memory usage, MPI communication, and OpenMP synchronization.

The authors felt that MAP was easy to use, as well provided the functionality needed to optimize programs. However, MAP lacks the ability to track cache misses, whereas Intel VTune does have this functionality (see section 7).

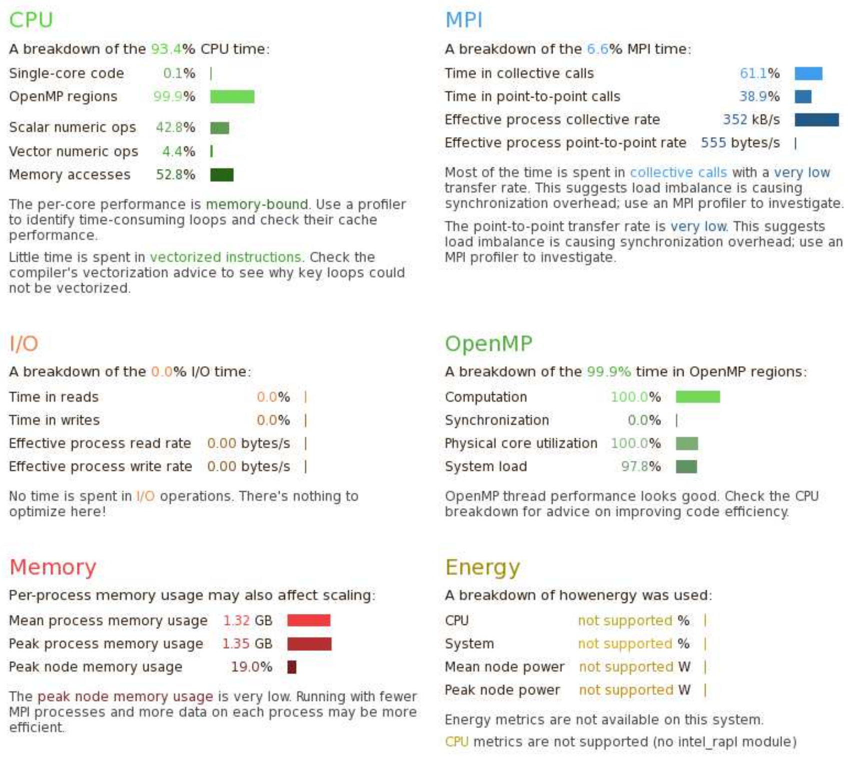
4 ALLINEA PERFORMANCE REPORTS

Allinea Performance Reports (Reports) is a low-overhead tool that produces a one-page report summarizing CPU, MPI, I/O, OpenMP, memory, and energy performance information. In addition, Performance Reports gives high-level suggestions for improving performance. Reports can be used on applications written in C, C++, and Fortran. Performance Reports does not offer performance data at the line, function, or loop levels.

To run Performance Reports, one adds "`perf-report`" before the run command, e.g.:

```
perf-report mpirun -np p ./a.out
```

The user does not have to recompile with the `"-g"` option, as Performance Reports doesn't collect data at the source-code level. After running, Performance Reports generates text and HTML files containing identical information, but the HTML file has accompanying graphics. The text file can be opened with any text editor, while the HTML file can be opened with any web browser. For each of these breakdowns in Figure 4, a message follows telling how well the area performed and gives advice on improving performance, as well what to look for when profiling.



CPU

A breakdown of the 93.4% CPU time:

Single-core code	0.1%
OpenMP regions	99.9%
Scalar numeric ops	42.8%
Vector numeric ops	4.4%
Memory accesses	52.8%

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.

Little time is spent in vectorized instructions. Check the compiler's vectorization advice to see why key loops could not be vectorized.

MPI

A breakdown of the 6.6% MPI time:

Time in collective calls	61.1%
Time in point-to-point calls	38.9%
Effective process collective rate	352 kB/s
Effective process point-to-point rate	555 bytes/s

Most of the time is spent in collective calls with a very low transfer rate. This suggests load imbalance is causing synchronization overhead; use an MPI profiler to investigate.

The point-to-point transfer rate is very low. This suggests load imbalance is causing synchronization overhead; use an MPI profiler to investigate.

I/O

A breakdown of the 0.0% I/O time:

Time in reads	0.0%
Time in writes	0.0%
Effective process read rate	0.00 bytes/s
Effective process write rate	0.00 bytes/s

No time is spent in I/O operations. There's nothing to optimize here!

OpenMP

A breakdown of the 99.9% time in OpenMP regions:

Computation	100.0%
Synchronization	0.0%
Physical core utilization	100.0%
System load	97.8%

OpenMP thread performance looks good. Check the CPU breakdown for advice on improving code efficiency.

Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	1.32 GB
Peak process memory usage	1.35 GB
Peak node memory usage	19.0%

The peak node memory usage is very low. Running with fewer MPI processes and more data on each process may be more efficient.

Energy

A breakdown of howenergy was used:

CPU	not supported %
System	not supported %
Mean node power	not supported W
Peak node power	not supported W

Energy metrics are not available on this system.
CPU metrics are not supported (no intel_rapl module)

Figure 4: Performance Report Breakdowns.

4.1 Evaluation

Allinea Performance Reports is different from other tools included in this survey. To start, there is only one step necessary to use Performance Reports - prefixing the command to be profiled. Users do not have to recompile the application to use Performance Reports. There is no learning curve for viewing the output as it comes in text or a viewable web page. Also, the authors found the optimization advice given for each category to be helpful. Another benefit of Performance Reports is having an idea of the bottlenecks in a user's application before using a full-featured performance tool like Allinea MAP (section 3) or Intel VTune (section 7). Knowing whether an application is memory, compute, or communication bound makes it easier to focus on the most important performance metrics when using a profiler that provides more detailed performance information.

5 INTEL PROFILE FUNCTION OR LOOP EXECUTION TIME

The Intel compiler contains a performance tool called "Profile Function or Loop Execution Time" (PFLET), which is a serial profiler. Information provided by PFLET can help one to decide which code portions to optimize and which to parallelize.

To profile with PFLET, issue

```
ifort -profile-loops=all prog.f90
./a.out
```

Here, "`-profile-loops=all`" enables the compiler to time function calls and loops in the program. Also, the "`-profile-loops-report=2`" option uses additional instrumentation to record loop iteration counts and reports min, max, and average iteration counts.

After the execution ends, PFLET will generate up to two .dump files (one for functions and one for loops), as well as .xml files containing the same data. Using any text editor, users can open the .dump files to view the results (see Table 1). The output is formatted into a table, where each row corresponds to a single function or loop labeled as file:line in the last column.

Table 1: PFLET Dump File Output.

time(abs)	time(%)	self(abs)	self(%)	loop_entries	function	function_file:line	loop_file:line
5745418040135	52.1	5745418040135	52.1	1080869265	episnp_mod_mp_two_snp_test_..0	episnp_mod.F90:57	episnp_mod.F90:92
1535455501662	13.9	1535455501662	13.9	518803997	episnp_mod_mp_two_snp_test_..0	episnp_mod.F90:57	episnp_mod.F90:109
493391216573	4.5	493391216573	4.5	1080869265	episnp_mod_mp_two_snp_test_..0	episnp_mod.F90:57	episnp_mod.F90:77
446037342805	4.0	446037342805	4.0	1080869265	episnp_mod_mp_two_snp_test_..0	episnp_mod.F90:57	episnp_mod.F90:92
351345228991	3.2	351345228991	3.2	1353462421	bpser_	cdflib.f:2307	cdflib.f:2397
201603090096	1.8	201603090096	1.8	518813997	episnp_mod_mp_partition_	episnp_mod.F90:125	episnp_mod.F90:369
66765839894	0.6	57666794224	0.5	721888762	bgrat_	cdflib.f:2714	cdflib.f:2775
46256515042	0.4	46256515042	0.4	518803997	episnp_mod_mp_two_snp_test_..0	episnp_mod.F90:57	episnp_mod.F90:109

5.1 Evaluation

PFLET is designed to only profile loops and functions, and not at the statement level. The text output is easy to interpret, as it doesn't require using a GUI. The authors consider the lack of statement-level profiling to be major drawback, especially for applications with large function or loop bodies. Furthermore, PFLET was not designed to use OpenMP. For these reasons, this tool's utility is limited to identifying loops or functions in a serial program that may benefit from parallelization or other optimizations.

6 INTEL TRACE ANALYZER AND COLLECTOR

Intel Trace Analyzer and Collector (ITAC) is a graphical tool used for understanding MPI application behavior. ITAC can help identify load imbalances and investigate communication correctness using traces.

To compile and collect performance data for ITAC, issue:

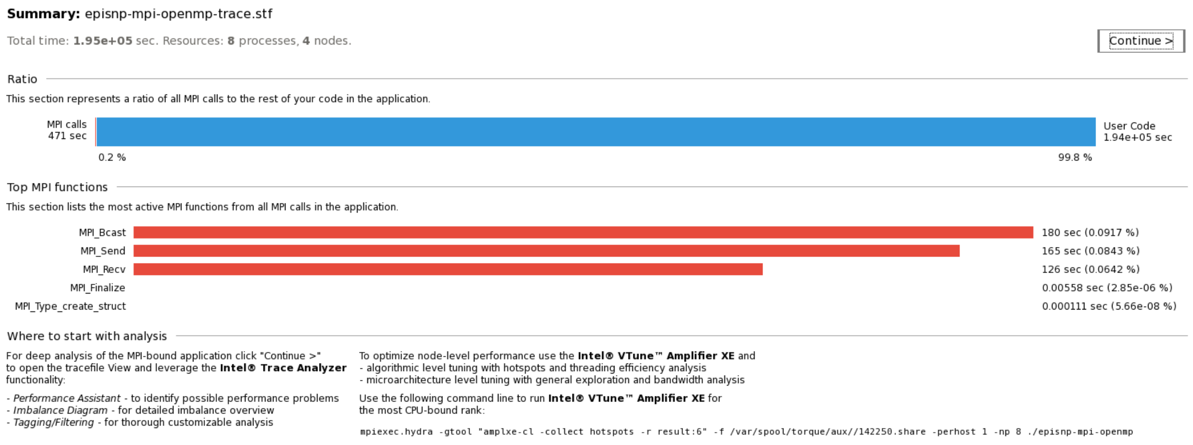
```
mpiifort -g -qopenmp prof.f90
mpirun -trace -np p ./a.out
```

The "`-trace`" option will create trace files, one per MPI process, containing information through the entire execution of the application, including MPI communication. To launch the ITAC GUI, issue:

```
traceanalyzer ./a.out.stf
```

where .stf is a Structured Trace File (STF) generated by "`-trace`". On start up, ITAC will open the Summary Page for the application, which contains a Ratio bar and a list of longest-running MPI calls (see Figure 5). This allows one to quickly see if the application is compute-bound or communication-bound.

Continuing from the Summary Page, the ITAC GUI opens with the Flat Profile tab on the left. The Load Balance tabs shows ratios of user-code and MPI communication for each MPI process. This data can be viewed in a series of bar graphs or pie charts per MPI process (see Figure 6). There is also a Call Tree and Call Graph in the Flat Profile, where both show different MPI calls executed throughout program. However, we did not find these features to be useful for our application.The Event Timeline displays MPI process activity as MPI communication (red) and user code (blue), with black lines connecting processes that are communicating. Through the Event Timeline, by right-clicking on an MPI process and navigating through a series of windows, users can determine the current line of code.


Summary: episnp-mpi-openmp-trace.stf

Total time: **1.95e+05** sec. Resources: **8** processes, **4** nodes.

Ratio

This section represents a ratio of all MPI calls to the rest of your code in the application.

Top MPI functions

This section lists the most active MPI functions from all MPI calls in the application.

MPI_Bcast	180 sec (0.0917 %)
MPI_Send	165 sec (0.0843 %)
MPI_Recv	126 sec (0.0642 %)
MPI_Finalize	0.00558 sec (2.85e-06 %)
MPI_Type_create_struct	0.000111 sec (5.66e-08 %)

Where to start with analysis

For deep analysis of the MPI-bound application click "Continue >" to open the tracefile View and leverage the **Intel® Trace Analyzer** functionality:

- *Performance Assistant* - to identify possible performance problems
- *Imbalance Diagram* - for detailed imbalance overview
- *Tagging/Filtering* - for thorough customizable analysis

To optimize node-level performance use the **Intel® VTune™ Amplifier XE** and
- algorithmic level tuning with hotspots and threading efficiency analysis
- microarchitecture level tuning with general exploration and bandwidth analysis

Use the following command line to run **Intel® VTune™ Amplifier XE** for the most CPU-bound rank:

```
mpiexec.hydra -gtool "amplxe-cl -collect hotspots -r result:6" -f /var/spool/torque/aux//142250.share -perhost 1 -np 8 ./episnp-mpi-openmp
```

Figure 5: ITAC Summary Page.

6.1 Evaluation

Once past the Summary Page, the features of ITAC are hidden behind blank panels. The window is filled with a large amount of unused blank space, with essentially no data displayed. Furthermore, many of the drop-down "arrows" require two clicks for data to be shown. Another issue occurs when using the Application and MPI tables in the Function Profile. By clicking on the first arrow, it makes the second arrow (MPI) move below a list of all MPI ranks. Because of this, the authors consider profiling a job with many MPI processes to be difficult. These are all GUI design flaws that slow user interaction with the profile data.

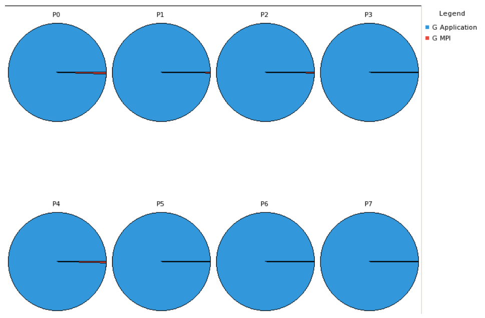


Figure 6: ITAC Load Balance Graph.

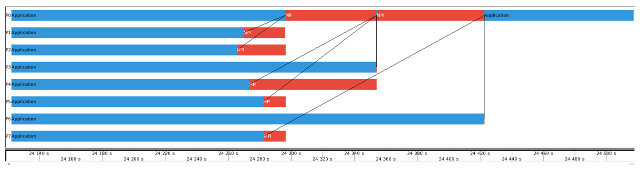


Figure 7: ITAC Event Timeline Chart.

The authors found the Load Balance Graph (Figure 6) and the Event Timeline Chart (Figure 7) to be the most useful in identifying and locating MPI communication problems. The Load Balance Function Profile helps one identify poorly-performing MPI ranks by looking at how long they spend doing computation and communication. The Event Timeline then helps identify when and where the problem occurs. When finding a performance problem with the Event Timeline, one needs to find the source code causing the problem. The authors were unable to access this data, despite following the documentation instructions and found the process of accessing the source code to be unintuitive and tedious. Furthermore, the documentation says that OpenMP regions are viewable from the Event Timeline, but we were unable to activate them.

7 INTEL VTUNE AMPLIFIER

Intel VTune Amplifier XE (VTune) is a performance tool that gathers performance data of serial or shared memory applications using threads. VTune offers support for applications written in C, C++, C#, Fortran, Java, assembly, as well as applications using accelerators such as the Intel Xeon Phi coprocessor and GPUs with OpenCL. Furthermore, VTune supports OpenMP, Intel Threading Building Blocks, and Intel Cilk Plus. VTune supports MPI, however it only stores performance data for MPI processes without communication. To profile communication among MPI processes, MAP or ITAC can be used (see Section 3 or Section 6).

Intel has chosen to use the command “`amplexe`” instead of the common name Vtune for job submission. To profile a serial or pure OpenMP application, the VTune GUI can launch the application for data collection (`amplxe-gui`). To profile MPI or MPI+OpenMP applications with VTune, one must use the command line interface to collect the data (`amplxe-cl`). To compile and collect Basic Hotspots performance data on serial or pure OpenMP code, issue

```
ifort -g -qopenmp prog.f90
amplxe-cl -quiet -collect hotspots ./a.out
```

For MPI+OpenMP jobs, this changes to:

```
mpiifort -g -qopenmp prog.f90
mpirun -np p amplxe-cl -quiet -collect hotspots -result-dir my_dir ./a.out
```

The “`-result-dir`" option is required for MPI profiling, as VTune holds each MPI rank’s profile data in different directories. For long-running jobs it is recommended to use the “`-quiet`” option. This will disable VTune’s status I/O.

Besides Basic Hotspots, VTune offers many different collection options. These collection options are broken into three groups: Algorithm Analysis, Microarchitecture Analysis, and Platform Analysis. The different collections for each of these groups are as follows:

- Algorithm Analysis: Basic Hotspots, Advanced Hotspots, Concurrency, Locks and Waits, HPC Performance Characterization.
- Microachitecture Analysis: General Exploration, Memory Access, SGX Hotspots, TSX Hotspots, TSX Exploration.
- Platform Analysis: CPU/GPU Concurrency, System Overview.

A feature of interest of relevant interest to HPC is the HPC Performance Characterization. This option collects performance metrics relevant to HPC applications such as CPU usage, FPU usage, GFLOPS, and memory bandwidth information. As of Parallel Studio 2016, this was a preview feature that was not available on our cluster. Intel has since added this into Parallel Studio 2017, but we were unable to test it.

After the collection job has completed, analysis can be performed by using the VTune GUI or command line regardless how the data was collected. For programs on remote servers, the command line version may be faster than the GUI due to X11 forwarding. However, not all of the data that VTune collects can be reported through command line. To view the collection results of rank "`<rank>`" issue

```
amplxe-gui my_dir.<rank>
```

To use the command line with an appropriate report type, issue

```
amplxe-cl -report <report-type> -result-dir my_dir.<rank>
```

Since most people will be using the GUI to analyze the performance data, we continue to use amplxe-gui.

The sections within the Summary pane show execution time, function hotspots, OpenMP execution time, OpenMP hotspots, and OpenMP load balancing. For additional information on each of these sections, see the Intel VTune User Guide.

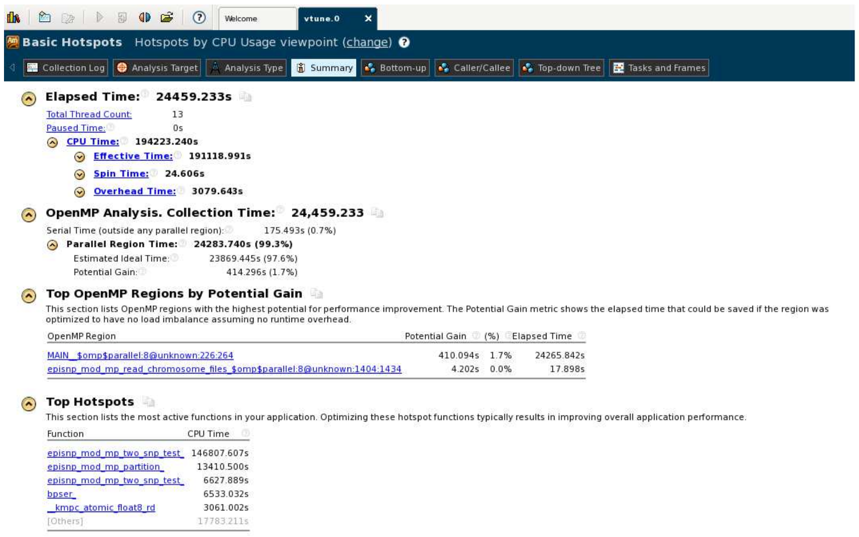
In addition to the Summary pane, there are other windows that use a different format. These windows are:

- Bottom-up: Displays self timings for lines, functions, and loops.
- Caller/Callee: Displays parent and child function calls.
- Top-down Tree: Displays both self timings and total timings.
- Tasks and Frames: Displays a time line for tasks (logical unit of work) and frames (period of time between beginning and end points).

The Bottom Up pane only displays self timings, while the Top Down pane displays both self and total timings. For this reason, the Top Down pane is sufficient for code analysis (Figure 9). Furthermore, by right-clicking and selecting "View Source", one can see the function call site or the largest bottleneck contained within the function if the call site is a call leaf. The order of this tree is controlled by selecting a Call Stack metric, which is in the rightmost drop down menu. The main metrics are CPU Time (default), Wait Time, Wait Count, Spin Time, Context Switch Time, and Task Time. There are also more specialized metrics based on these main metrics, as well as specific hardware event collection metrics.

7.1 Evaluation

Intel VTune is a performance tool that allows users to find problems with computation, memory, and thread performance. It has a steep learning curve because the tool offers many options to the user and the GUI is



Basic Hotspots Hotspots by CPU Usage viewpoint (change)

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Caller/Callee | Top-down Tree | Tasks and Frames

Elapsed Time: 24459.233s

Total Thread Count:	13
Paused Time:	0s
CPU Time:	**194223.240s**
Effective Time:	**191118.991s**
Spin Time:	**24.606s**
Overhead Time:	**3079.643s**

OpenMP Analysis. Collection Time: 24,459.233

Serial Time (outside any parallel region):	175.493s (0.7%)
Parallel Region Time:	**24283.740s (99.3%)**
Estimated Ideal Time:	23869.445s (97.6%)
Potential Gain:	414.296s (1.7%)

Top OpenMP Regions by Potential Gain

This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that could be saved if the region was optimized to have no load imbalance assuming no runtime overhead.

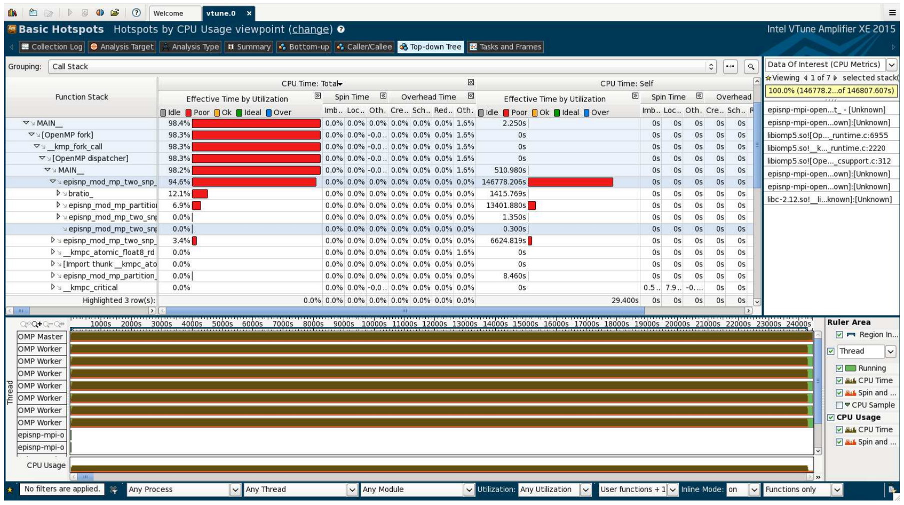
OpenMP Region	Potential Gain	(%)	Elapsed Time
MAIN__ompparallel:8@unknown:226:264	410.094s	1.7%	24265.842s
episnp_mod_mp_read_chromosome_files_ompparallel:8@unknown:1404:1434	4.202s	0.0%	17.898s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	CPU Time
episnp_mod_mp_two_snp_test_	146807.607s
episnp_mod_mp_partition_	13410.500s
episnp_mod_mp_two_snp_test_	6627.889s
bpser_	6533.032s
__kmpc_atomic_float8_rd	3061.002s
[Others]	17783.211s

Figure 8: VTune Summary.



Basic Hotspots Hotspots by CPU Usage viewpoint (change) Intel VTune Amplifier XE 2015

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Caller/Callee | Top-down Tree | Tasks and Frames

Grouping: Call Stack

Function Stack	CPU Time: Total								CPU Time: Self					
	Effective Time by Utilization	Spin Time			Overhead Time				Effective Time by Utilization	Spin Time			Overhead	
	Idle Poor Ok Ideal Over	Imb..	Loc..	Oth.	Cre..	Sch..	Red..	Oth.	Idle Poor Ok Ideal Over	Imb..	Loc..	Oth.	Cre..	Sch..
MAIN__	98.4%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	1.6%	2.250s	0s	0s	0s	0s	0s
[OpenMP fork]	98.3%	0.0%	0.0%	-0.0..	0.0%	0.0%	0.0%	1.6%	0s	0s	0s	0s	0s	0s
__kmp_fork_call	98.3%	0.0%	0.0%	-0.0..	0.0%	0.0%	0.0%	1.6%	0s	0s	0s	0s	0s	0s
[OpenMP dispatcher]	98.3%	0.0%	0.0%	-0.0..	0.0%	0.0%	0.0%	1.6%	0s	0s	0s	0s	0s	0s
MAIN__	98.2%	0.0%	0.0%	-0.0..	0.0%	0.0%	0.0%	1.6%	510.980s	0s	0s	0s	0s	0s
episnp_mod_mp_two_snp_	94.6%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	146778.206s	0s	0s	0s	0s	0s
bratio_	12.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	1415.769s	0s	0s	0s	0s	0s
episnp_mod_mp_partitio	6.9%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	13401.880s	0s	0s	0s	0s	0s
episnp_mod_mp_two_snp	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	1.350s	0s	0s	0s	0s	0s
episnp_mod_mp_two_snp	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.300s	0s	0s	0s	0s	0s
episnp_mod_mp_two_snp_	3.4%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	6624.819s	0s	0s	0s	0s	0s
__kmpc_atomic_float8_rd	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	1.6%	0s	0s	0s	0s	0s	0s
[Import thunk __kmpc_ato	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0s	0s	0s	0s	0s	0s
episnp_mod_mp_partition_	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	8.460s	0s	0s	0s	0s	0s
__kmpc_critical	0.0%	0.0%	0.0%	-0.0..	0.0%	0.0%	0.0%	0.0%	0s	0.5..	7.9..	-0....	0s	0s
Highlighted 3 row(s):	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	29.400s	0s	0s	0s	0s	0s

Data Of Interest (CPU Metrics)
Viewing 1 of 7 selected stack(
100.0% (146778.2...of 146807.607s)
episnp-mpi-open...t_ - [Unknown]
episnp-mpi-open...own]:[Unknown]
libiomp5.so![Op..._runtime.c:6955
libiomp5.so!__k..._runtime.c:2220
libiomp5.so![Ope..._csupport.c:312
episnp-mpi-open...own]:[Unknown]
episnp-mpi-open...own]:[Unknown]
libc-2.12.so!__li...known]:[Unknown]

No filters are applied. | Any Process | Any Thread | Any Module | Utilization: Any Utilization | User functions + 1 | Inline Mode: on | Functions only

Figure 9: VTune Top-Down Tree Window.

often not intuitive. The VTune's Basic Hotspots analysis presents performance data that is usually sufficient to optimize a program.

Once data collection has been completed, users can further choose between the GUI or the command line to view the data analysis. The authors found the Summary and Top-down Tree windows to provide the most useful performance data. The Summary window does a good job covering generic metrics for a program, including interesting items such as parallel Potential Gain. Furthermore, it offers an easy-to-read load balance chart for OpenMP, which many users will find useful. Statistical timing information for functions and lines of code can be viewed in the Top-down and Bottom-Up Windows. The table format needs improvement as it often contains many columns of data that are empty.There are options to view the source code within VTune by using the "View Source" (via right-click) for particular functions and lines. The authors feel it should be easier to see the source code along with the performance data.

For jobs that have long execution times, the Hardware Event-Based Sampling collection types may lead to significant amounts of data collected. For example, profiling epiSNP with the Advanced Hotspots analysis resulted in large 64GB directories per MPI rank, with four MPI processes used for the job, one per node. When trying to open files this large, the GUI would become unresponsive. It was much simpler using the command line report feature and forgoing the graphics than dealing with the slow response times of the GUI.

8 SUMMARY AND CONCLUSIONS

Using the hybrid MPI+OpenMP version of epiSNP, the authors evaluated Allinea and Intel performance tools for functionality and ease-of-use. Table 2 contains the tool Functionality Summary.For our functionality criteria, the tool that scored the best was MAP, as it contained the features needed for application optimization. MAP has an Yes* for detecting MPI load imbalance because the Metric View can imply a load imbalance, but does not offer direct detection like in ITAC. Intel Trace Analyzer and Collector (ITAC) scores poorly in this category, as it lacks detailed source code support, memory metrics, and I/O metrics. However, the authors felt that ITAC's MPI functionality, specifically the Event Timeline Chart and the Load Balance Graph, were especially useful.

Table 2: Functionality Summary.

Category	MAP	Reports	PFLET	ITAC	VTune
Profiling Lines?	Yes	No	No	No	Yes
Profiling Functions?	Yes	No	Yes	Yes	Yes
Profiling Loops?	Yes	No	Yes	No	No
CPU Performance Data?	Yes	Yes	Yes	Yes	Yes
Memory Performance Data?	Yes	Yes	No	No	Yes
I/O Performance Data?	Yes	Yes	No	No	Yes
Detect MPI Load Imbalance?	Yes*	No	No	Yes	No
Detect OpenMP Load Imbalance?	Yes	Yes	No	No	Yes

To evaluate the performance of an MPI application using Intel's performance tools, one must use two tools: ITAC for MPI communication and either VTune or PFLET to pinpoint bottlenecks in source code.Whereas, Allinea MAP is a single tool that provides both performance data for MPI communication and source-code-level profiling information. The authors consider this an advantage of MAP over ITAC or VTune. Furthermore, the authors consider MAP's user interface to be more intuitive than either ITAC's or VTune's dissimilar interfaces.

Allinea also has a high-level, easy-to-use performance tool called Performance Reports. This tool does not require recompilation. Combined, the ITAC and VTune summary pages provide similar information compared to Performance Reports, but this requires the application to be run twice (once for each tool). In addition, Performance Reports provides suggestions for performance enhancements, whereas Intel's tools do not. The authors feel that these suggestions are valuable and hope that other tools will implement similar analyses to aid in program optimization. However, to find performance problems after using Performance Reports, additional runs with other tools requiring source code compilation will likely be needed.

GUI responsiveness is another important consideration. The Remote Client for MAP provides a significant improvement in usability, as it reduces communication lag compared to X11 forwarding. It also reduces the load on the login node of the cluster, as image rendering is moved to the user's workstation. The Ease-of-Use Summary (Table 3) contains the authors impressions of using the various tools. In this table, we gave two ratings: Satisfactory (S) and Needs Improvement (NI). Here, Performance Reports and Intel Performance Function or Loop Execution Time (PFLET) do not use GUIs, so they were categorized as N/A.

Table 3: Ease-of-Use Summary. S stands for Satisfactory and NI stands for Needs Improvement.

Category	MAP	Reports	PFLET	ITAC	VTune
Navigation of GUI	S	N/A	N/A	NI	S
Data Presentation	S	S	NI	S	NI
Handles Long-running Jobs	S	S	S	NI	NI

ACKNOWLEDGMENTS

The research reported in this paper is partially supported by the HPC@ISU equipment at Iowa State University, some of which has been purchased through funding provided by NSF under MRI grant number CNS 1229081 and CRI grant number 1205413. Personnel time was fully funded by Iowa State University.

REFERENCES

Appelbe, B., and D. Bergmark. 1996. "Software Tools for High Performance Computing: Survey and Recommendations". *Scientific Programming* vol. 5, pp. 239–249.

Collette, M., B. Corey, and J. Johnson. 2004, December. "High Performance Tools & Technologies". Technical report, Lawrence Livermore National Laboratory.

Ma, L., H. B. Runesha, D. Dvorkin, J. R. Garbe, and Y. Da. 2008. "Parallel and serial computing tools for testing single-locus and epistatic SNP effects of quantitative traits in genome-wide association studies". *BMC Bioinformatics* vol. 9 (1), pp. 315.

Marowka, A. 2010, July. "A Study of the Usability of Multicore Threading Tools". *International Journal of Software Engineering and Its Applications* vol. 4 (3).

Moore, S., D. Cronk, K. London, and J. Dongarra. 2001, September. "Review of Performance Analysis Tools for MPI Parallel Programs". *Recent Advances in Parallel Virtual Machine and Message Passing Interface*.

Weeks, N. T., G. R. Luecke, B. M. Groth, M. Kraeva, L. Ma, L. M. Kramer, J. E. Koltes, and J. M. Reecy. 2016, July. "High-performance epistasis detection in quantitative trait GWAS". *The International Journal of High Performance Computing Applications* .

AUTHOR BIOGRAPHIES

GLENN R. LUECKE received his Ph.D. in mathematics from the California Institute of Technology. He is currently Professor of Mathematics and director of HPC education and training at Iowa State University.

BRANDON GROTH is a research assistant in the Department of Mathematics HPC group at Iowa State University. His research interests include scientific computing, mathematical applications in the sciences, and numerical analysis.

NATHAN WEEKS is currently pursuing a Ph.D. in Computer Science at Iowa State University. His research interests include parallel computing, software application optimization, and bioinformatics.

MARINA KRAEVA received her Ph.D. in Computer Science from the State Technical University of Novosibirsk, Russia in 1999 and joined the High Performance Computing group at Iowa State University.