



High performance tools to debug, profile, and analyze your applications

Weather and Climate Models: Preparing Development Workflows for Exascale

Florent Lebeau
flebeau@allinea.com



Outline

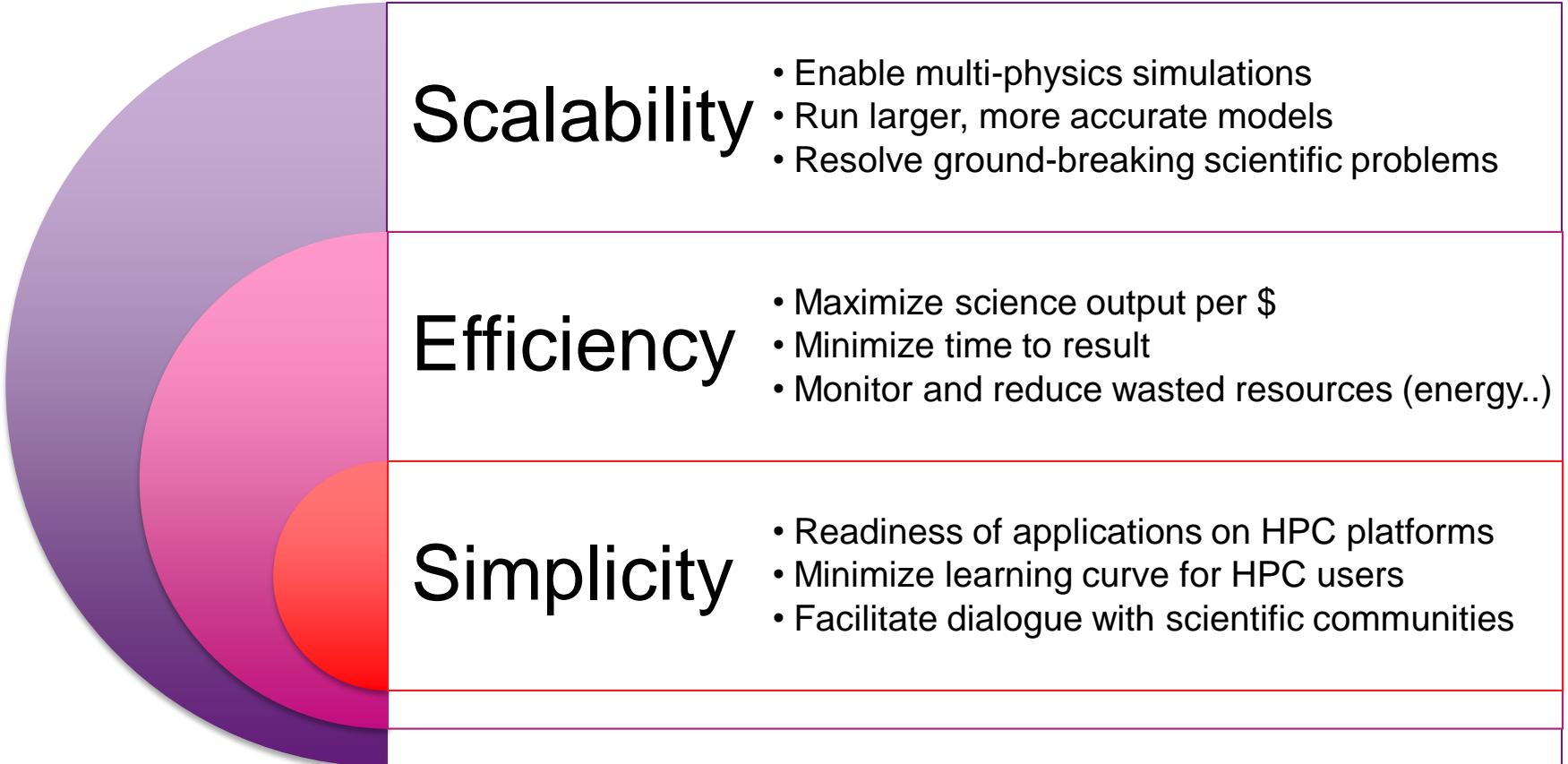
- How to Handle Increasingly Complex Models?
- Allinea's Tool Solution
- Automate Fault Detection of Weather and Climate Models
- What is coming next

Weather and Forecasting models



allinea

As the complexity increases, the demands are evolving



Allinea's vision

- **Helping maximize HPC production**
 - Reduce HPC systems operating costs
 - Resolve cutting-edge challenges
 - Promote Efficiency (as opposed to Utilization)
 - Transfer knowledge to HPC communities
- **Helping the HPC community design the best applications**
 - Reach highest levels of performance and scalability
 - Improve scientific code quality and accuracy



Automation script example

```
#!/bin/bash -l
# Job submission file name
jobfile=test_jacobi_mpi_omp_gnu.sub
# Load environment
module load compiler/gnu mpi/openmpi_gnu
module load allinea/perf-report
# Compile
make clean && make

# Job submission file configuration
cat << EOF > $jobfile
#!/bin/bash -l
#SBATCH --job-name='test_jacobi_mpi_omp_gnu'
#SBATCH --time=00:05:00
#SBATCH --ntasks=128
#SBATCH -ntasks-per-node=2
export OMP_NUM_THREADS=16
srun ./jacobi_omp_mpi_gnu.exe
EOF

# Submit
sbatch $jobfile

# Check results
[...]
```

Compile

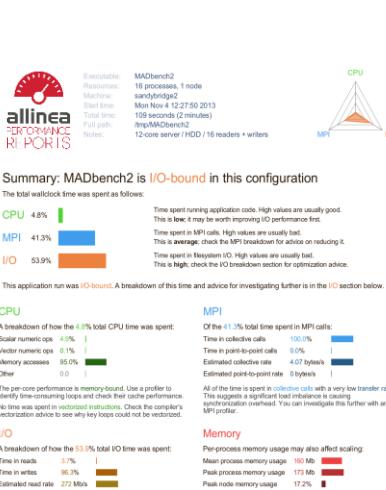
Execute

Test

Performance monitoring

```
perf-report -o jacobi_omp_mpi_gnu_perf.csv \
srun ./jacobi_omp_mpi_gnu.exe
```

- -o specifies the name and format of the output
 - Html
 - Txt
 - CSV



Name	Value
Executable	jacobi_omp_mpi_gnu.exe
Command	srun ./jacobi_omp_mpi_gnu.exe
Processes	120
Nodes	64
Physical cores per node	16
Logical cores per node	32
Memory per node (GiB)	32
Machine	mars
Started on	Wed Sep 28 17:04:42 2016
Total time (s)	1534

Energy efficiency monitoring

Energy

A breakdown of how the 3.6 Wh was used:

CPU	62.9%	
System	37.1%	
Mean node power	92.4 W	
Peak node power	94 W	

Significant energy is wasted during MPI communications. It may be more efficient to use fewer nodes with more data on each node.

Significant time is spent with the CPU clock frequency c



```
fail=0
# --- check energy usage
f=jacobi_omp_mpi_gnu_perf.csv
tot_energy=`grep "Total energy"|awk -F, '{print $2}'``
if [ "$t" > "3000" ] ; then
    ((fail++))
    echo "Test has failed: Total energy =${tot_energy}"
else
    echo "Test has succeeded"
fi
```

Efficiency monitoring

CPU

A breakdown of the 99.8% CPU time:

Scalar numeric ops	3.1%	
Vector numeric ops	2.7%	
Memory accesses	94.2%	███████
Waiting for accelerators	0.0%	

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.

I/O

A breakdown of how the 53.9% total I/O time was spent:

Time in reads	3.7%	
Time in writes	96.3%	███████
Estimated read rate	272 Mb/s	███████
Estimated write rate	7.06 Mb/s	

Most of the time is spent in **write operations**, which have a very low **transfer rate**. This may be caused by contention for the filesystem or inefficient access patterns. Use an I/O profiler to investigate which write calls are affected.

MPI

A breakdown of the 0.2% MPI time:

Time in collective calls	17.6%	█
Time in point-to-point calls	82.4%	███████████
Effective process collective rate	0.00 bytes/s	
Effective process point-to-point rate	285 MB/s	███████████

Most of the time is spent in **point-to-point calls** with an average transfer rate. Using larger messages and overlapping communication and computation may increase the effective transfer rate.

Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	96.0 MB	██████████
Peak process memory usage	108 MB	███████████
Peak node memory usage	6.0%	█

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

Accelerators

A breakdown of how accelerators were used:

GPU utilization	78.3%	███████████
Global memory accesses	70.9%	███████████
Mean GPU memory usage	31.5%	███████
Peak GPU memory usage	38.7%	███████

Significant time is spent in **global memory accesses**. Try modifying kernels to use shared memory instead and check for bad striding patterns.

GPU utilization is acceptable. Use the NVIDIA Visual Profiler to improve kernel performance.

OpenMP

A breakdown of the 99.5% time in OpenMP regions:

Computation	58.9%	██████████
Synchronization	41.1%	███████
Physical core utilization	100.0%	████████████████████
System load	99.7%	████████████████████

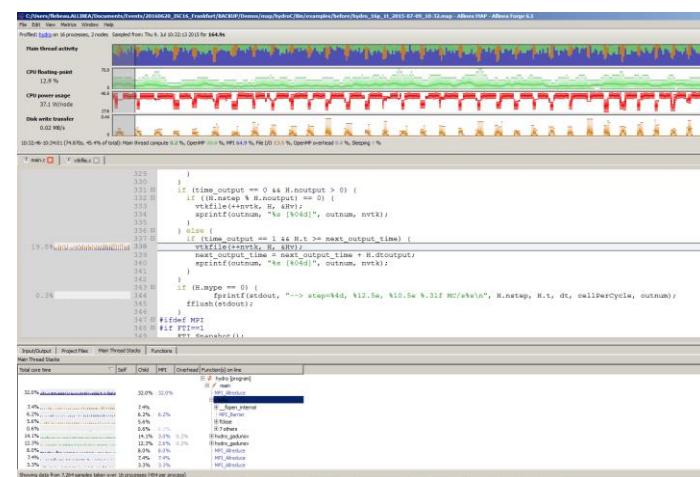
Significant time is spent **synchronizing** threads in parallel regions. Check the affected regions with a profiler.

This may be a sign of overly fine-grained parallelism (OpenMP regions in tight loops) or workload imbalance.

Automate profiling

```
map --profile -o jacobi_omp_mpi_gnu_perf.map \
    srun ./jacobi_omp_mpi_gnu.exe
perf-report -o jacobi_omp_mpi_gnu_perf.csv \
    jacobi_omp_mpi_gnu_perf.map
```

- -o specifies the name of the output
- The output can be turned into a report with Allinea Performance Reports for pre-processing
- The output can be open for afterwards for further investigation:
 - On the login node using X forwarding with Allinea MAP
 - Or copied locally and using the remote client
 - Linux, Windows and MacOS X builds
 - <http://www.allinea.com/products/forge/download>



Automate debugging

```
ddt --offline --output=jacobi_omp_mpi_gnu_debug.txt \
--trace-at _jacobi.F90:83,residual \
srun ./jacobi_omp_mpi_gnu.exe
```

- `--offline` enable non-interactive debugging
- `--output` specifies the name and output of the non-interactive debugging session
 - Html
 - Txt

Automate debugging

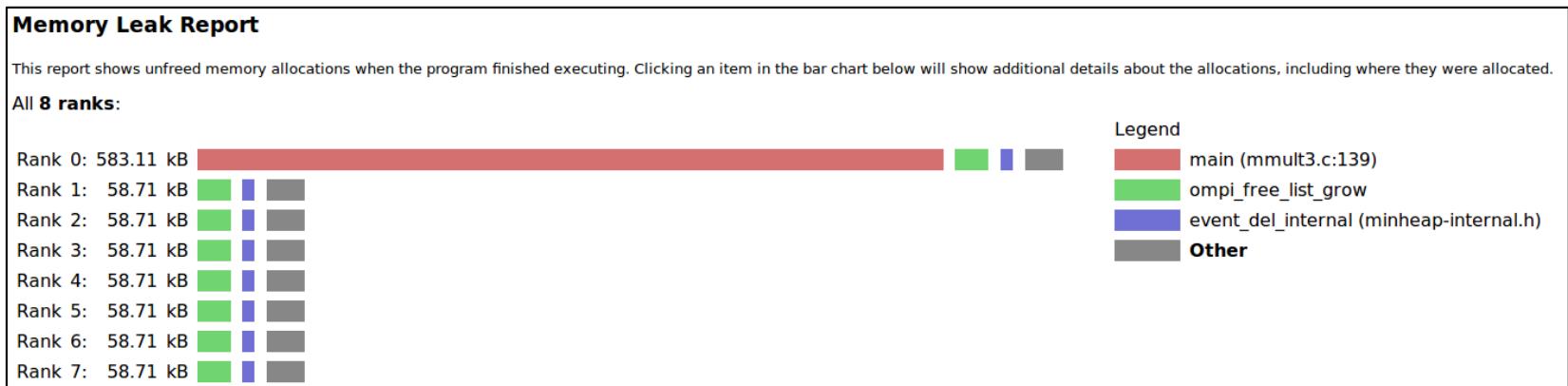
#	Time	Tracepoint	Processes	Values
1	21:18.172	jacobi_mpi_omp_gnu.exe (_jacobi.f90:83)	0-127	residual: 2.57



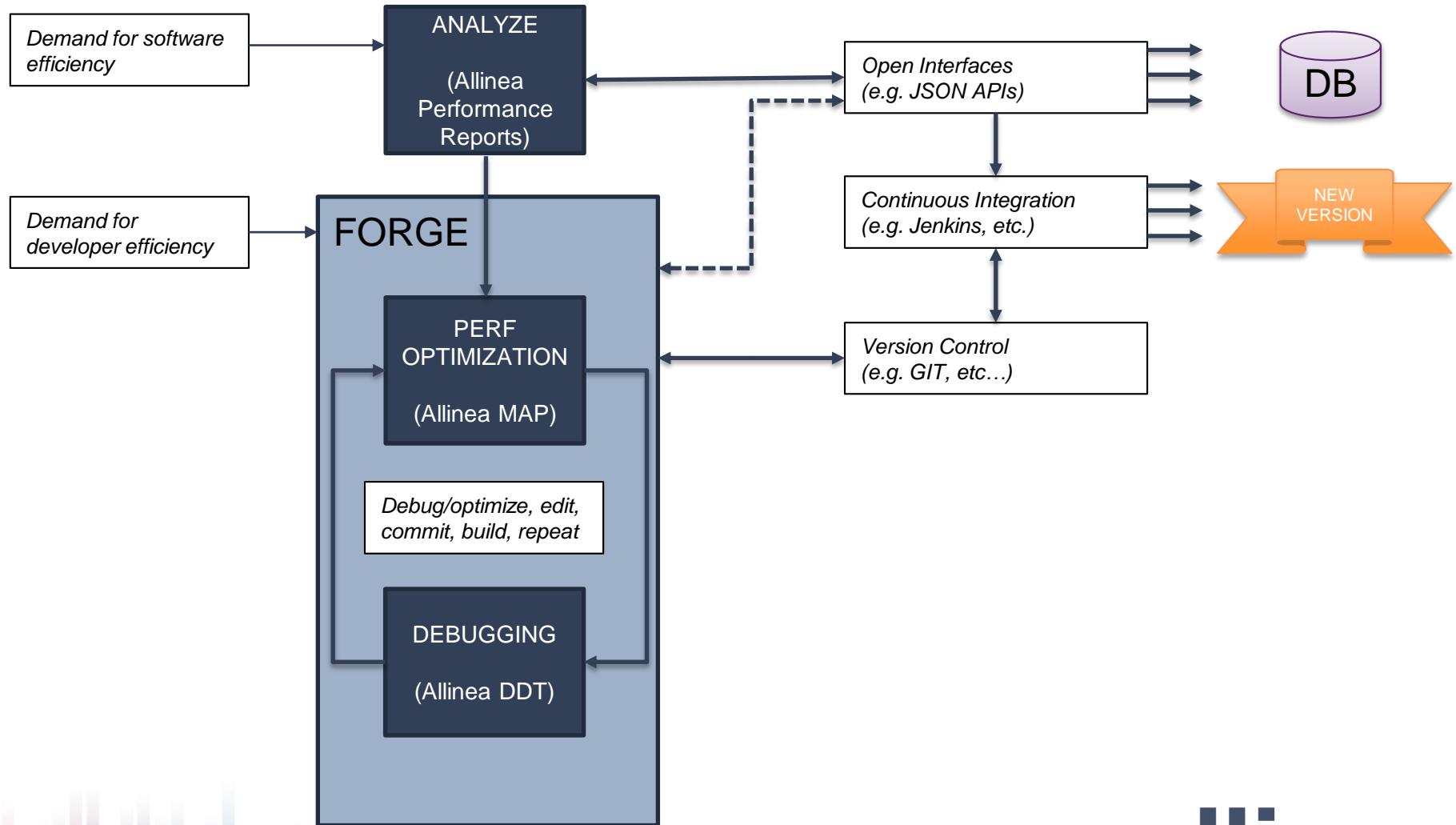
```
fail=0
# --- check DDT tracepoint (residual)
f=jacobi_omp_mpi_gnu_debug.txt
resid=`grep ^tracepoint $f |awk -Fresidual: '{print $2}' |tail -1 |cut -c2-5`
if [ "$resid" != "2.57" ] ; then
    ((fail++))
    echo "Test has failed resid=$resid"
else
    echo "Test has succeeded"
```

Automate debugging

- Other available options:
 - `--trace-changes`: set a tracepoint on the variable introduced by the latest commit (git, svn, mercurial)
 - `--break-at`: set a breakpoint
 - `--mem-debug`: check for memory defects and leaks
 - `--check-bounds`: check for out of bounds array accesses



Development process workflow



allinea

What is coming next?

MAP

- Profile selected ranks only
- Toggle between absolute times and percentages
- Workflow integration: export function-level performance data to CI tools (Jenkins, Bamboo etc)

Performance Reports

- **Custom metrics**
- Profile selected ranks only
- Toggle between absolute times and percentages
- Workflow integration: export all metrics data to CI tools (Jenkins, Bamboo etc)



High performance tools to debug, profile, and analyze your applications

Thank you !

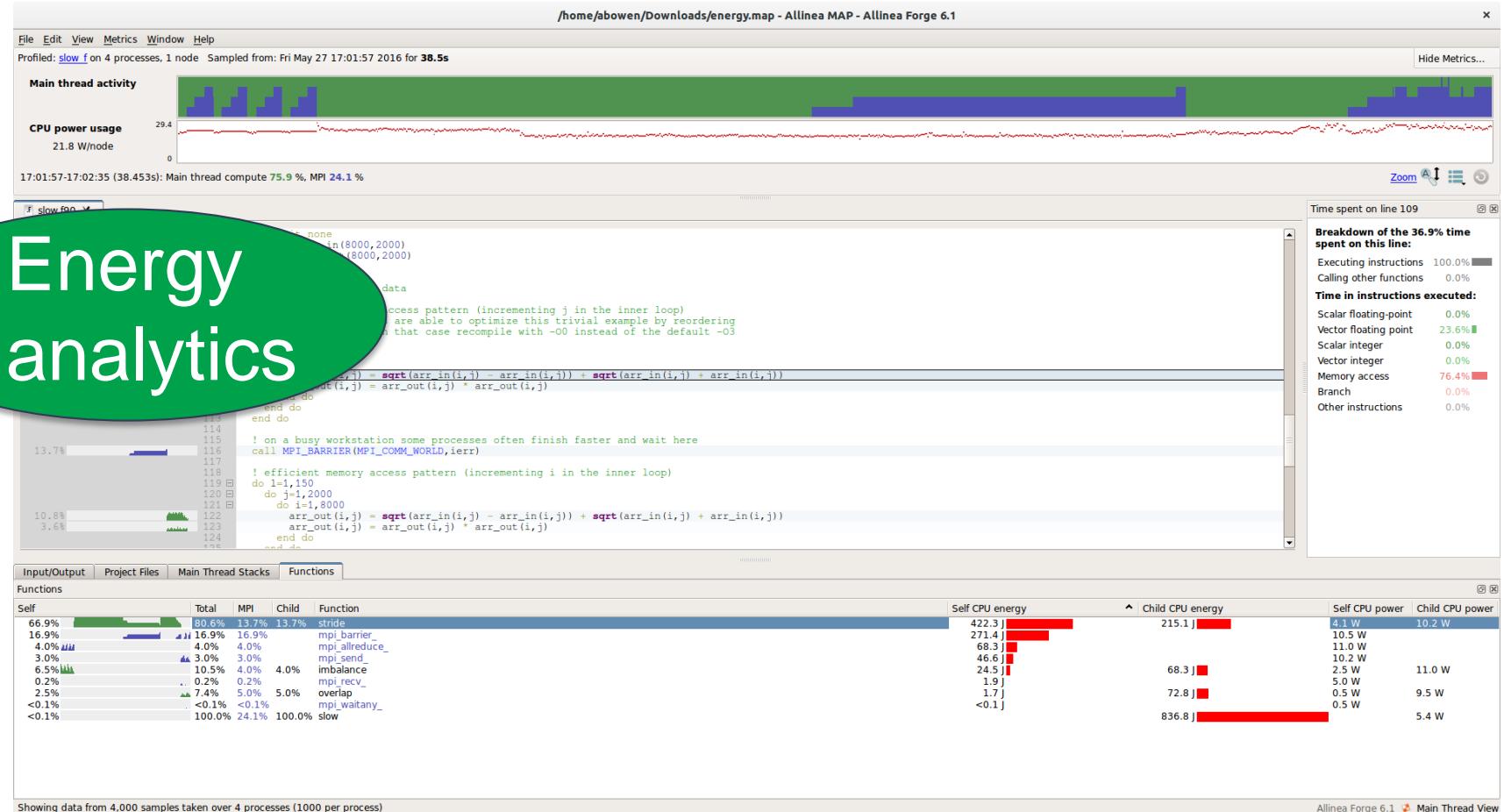
Technical Support team :

Sales team :

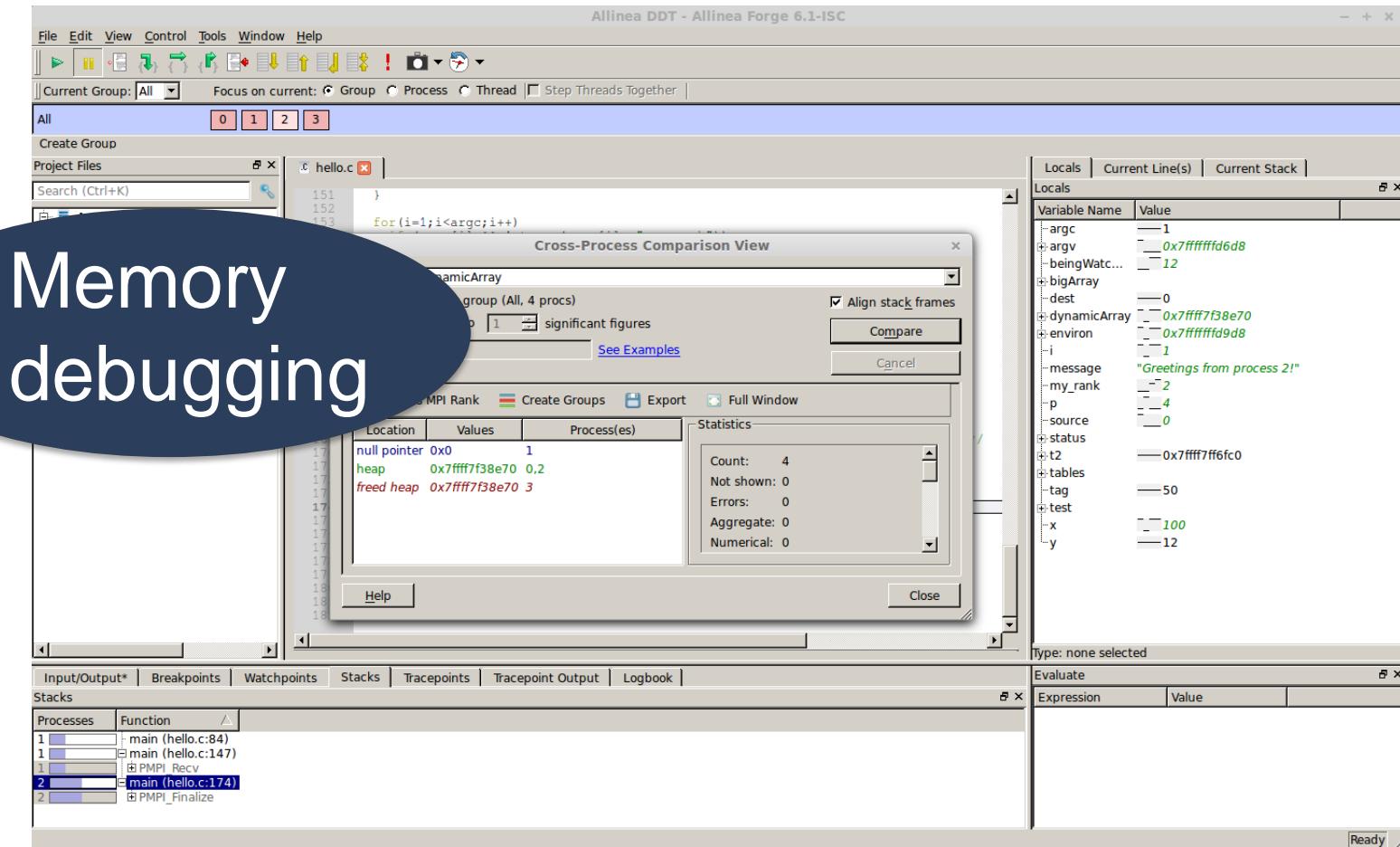
support@allinea.com

sales@allinea.com





allinea



allinea

Offline debugging

6		0:05.230	n/a	Allinea Forge - Allinea DDT Debugging : mpirun -n 4 ./examples/wave_c 10 MPI implementation : Auto-Detect (Open MPI) * number of processes : 4 * number of nodes : 1 Memory debugging enabled : Yes * setting : Balanced * check bounds : 1 after																									
7		0:05.230	0-3	Play																									
8		0:05.295	0-3	Process stopped at breakpoint in do_math (wave.c:145).																									
Additional Information																													
				<table border="1"> <thead> <tr> <th>Threads</th> <th>Function</th> <th>Source</th> <th>Variables</th> </tr> </thead> <tbody> <tr> <td></td><td>main (wave.c:334)</td><td>iterations = update(left, right);</td><td> <table border="1"> <tr><td>Rank 0, thread 1</td></tr> <tr><td>Name</td><td>Value</td></tr> <tr><td>allt</td><td><optimized out></td></tr> <tr><td>argc</td><td>2</td></tr> <tr><td>argv</td><td>0x7fffffff858</td></tr> <tr><td>calculation_rate</td><td><optimized out></td></tr> <tr><td>iterations</td><td><optimized out></td></tr> <tr><td>left</td><td>-2 (from -2 to 2)</td></tr> <tr><td>right</td><td>1 (from -2 to 3)</td></tr> </table></td></tr></tbody> </table>	Threads	Function	Source	Variables		main (wave.c:334)	iterations = update(left, right);	<table border="1"> <tr><td>Rank 0, thread 1</td></tr> <tr><td>Name</td><td>Value</td></tr> <tr><td>allt</td><td><optimized out></td></tr> <tr><td>argc</td><td>2</td></tr> <tr><td>argv</td><td>0x7fffffff858</td></tr> <tr><td>calculation_rate</td><td><optimized out></td></tr> <tr><td>iterations</td><td><optimized out></td></tr> <tr><td>left</td><td>-2 (from -2 to 2)</td></tr> <tr><td>right</td><td>1 (from -2 to 3)</td></tr> </table>	Rank 0, thread 1	Name	Value	allt	<optimized out>	argc	2	argv	0x7fffffff858	calculation_rate	<optimized out>	iterations	<optimized out>	left	-2 (from -2 to 2)	right	1 (from -2 to 3)
Threads	Function	Source	Variables																										
	main (wave.c:334)	iterations = update(left, right);	<table border="1"> <tr><td>Rank 0, thread 1</td></tr> <tr><td>Name</td><td>Value</td></tr> <tr><td>allt</td><td><optimized out></td></tr> <tr><td>argc</td><td>2</td></tr> <tr><td>argv</td><td>0x7fffffff858</td></tr> <tr><td>calculation_rate</td><td><optimized out></td></tr> <tr><td>iterations</td><td><optimized out></td></tr> <tr><td>left</td><td>-2 (from -2 to 2)</td></tr> <tr><td>right</td><td>1 (from -2 to 3)</td></tr> </table>	Rank 0, thread 1	Name	Value	allt	<optimized out>	argc	2	argv	0x7fffffff858	calculation_rate	<optimized out>	iterations	<optimized out>	left	-2 (from -2 to 2)	right	1 (from -2 to 3)									
Rank 0, thread 1																													
Name	Value																												
allt	<optimized out>																												
argc	2																												
argv	0x7fffffff858																												
calculation_rate	<optimized out>																												
iterations	<optimized out>																												
left	-2 (from -2 to 2)																												
right	1 (from -2 to 3)																												
	0-3 4 update (wave.c:211)	do_math(j);	<table border="1"> <tr><td>Rank 0, thread 1</td></tr> <tr><td>Name</td><td>Value</td></tr> <tr><td>iterations</td><td><optimized out></td></tr> <tr><td>j</td><td>102 (from 101 to 102)</td></tr> <tr><td>left</td><td>-2 (from -2 to 2)</td></tr> <tr><td>now</td><td><aggregate value></td></tr> <tr><td>right</td><td>1 (from -2 to 3)</td></tr> <tr><td>stop</td><td>0</td></tr> </table>	Rank 0, thread 1	Name	Value	iterations	<optimized out>	j	102 (from 101 to 102)	left	-2 (from -2 to 2)	now	<aggregate value>	right	1 (from -2 to 3)	stop	0											
Rank 0, thread 1																													
Name	Value																												
iterations	<optimized out>																												
j	102 (from 101 to 102)																												
left	-2 (from -2 to 2)																												
now	<aggregate value>																												
right	1 (from -2 to 3)																												
stop	0																												

	0-3 4 do_math (wave.c:145)	newval[i] = (2.0 * values[i]) - oldval[i]					------------------	-----------------------		Rank 0, thread 1			Name	Value		i	102 (from 101 to 102)	
	▶ Current Stack																	
10		0:07.795	0-3	Play														
11		0:07.815	0-3	Process stopped at breakpoint in do_math (wave.c:145).														