allinea

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

Large scale heterogeneous applications made easy with Allinea

ECMWF, 27/10/2016

Florent Lebeau
flebeau@allinea.com

allinea FORGE allinea DDT allinea MAP allinea PERFORMANCE REPORTS

Weather and Forecasting models today

Scalability

Efficiency

Simplicity

allinea

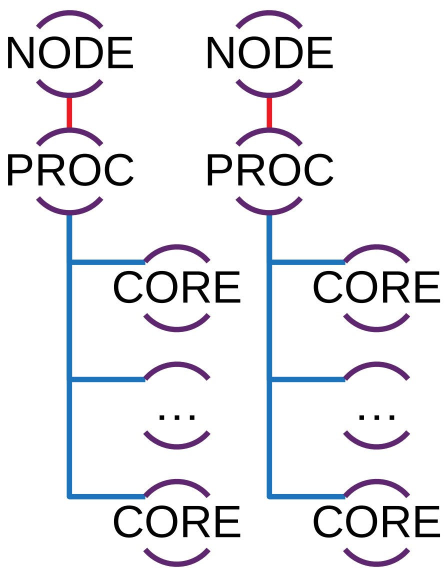
Towards more vertical & horizontal scalability

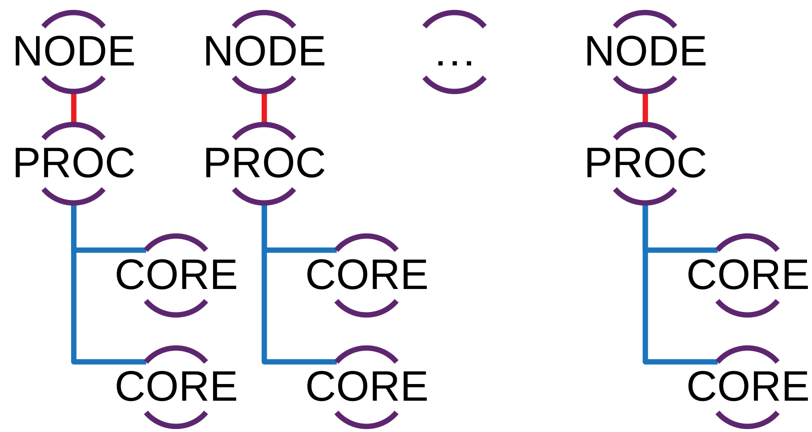
INTEL KNIGHTS LANDING

NVIDIA GPUS

NEXT-GEN INTEL XEON

ARM v8





Heterogeneous Systems

- Use many processor architectures
  - x86_64 + GPUs
  - x86_64 + KNL
  - OpenPower + GPUs
  - ARMv8 + GPUs
  - …

- Goal:
  - Increase compute power with specialised processors
  - Improve energy efficiency

- Parallel Programming Languages:
  - MPI
  - OpenMP
  - CUDA
  - OpenACC
  - …

allinea

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

Where to find Allinea's tools

Over 65% of Top 100 HPC systems

- From small to very large tools provision

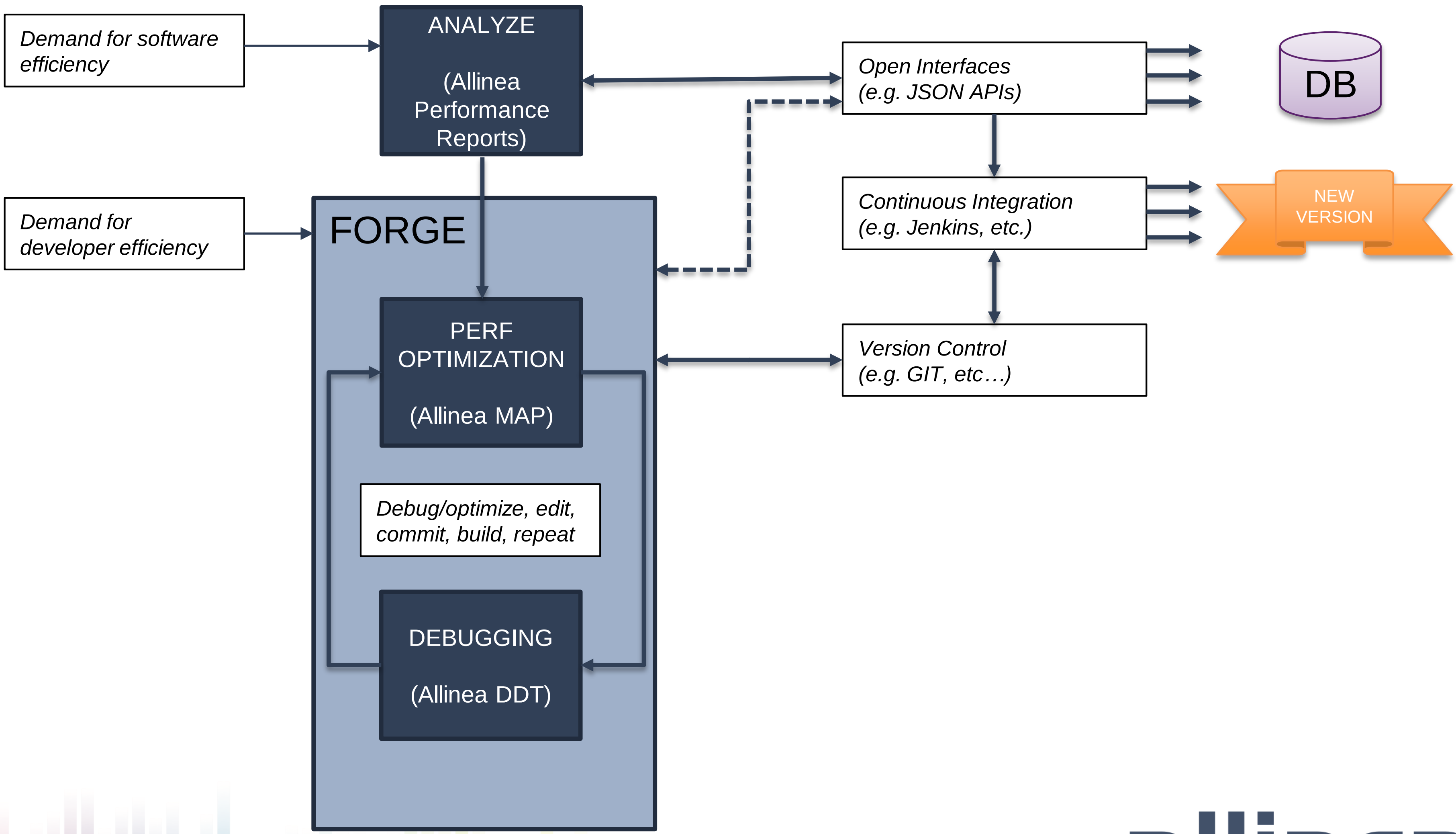
8 of the Top 10 HPC systems

- From 1,000 to 700,000 core tools usage

Future leadership systems

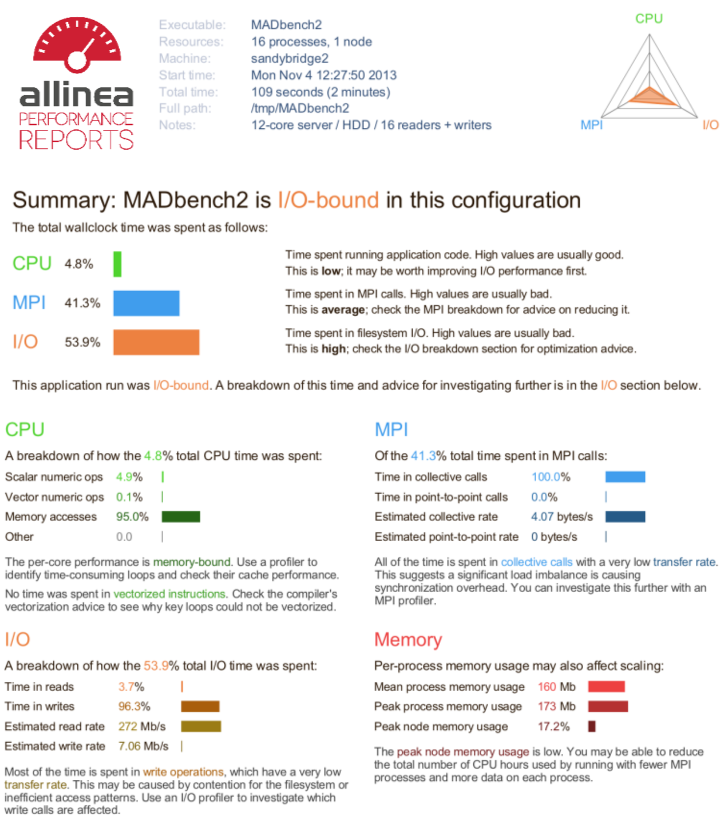
- Millions of cores usage

Development process workflow



allinea

Analyse with Allinea Performance Reports



allinea PERFORMANCE REPORTS

Executable:	MADbench2
Resources:	16 processes, 1 node
Machine:	sandybridge2
Start time:	Mon Nov 4 12:27:50 2013
Total time:	109 seconds (2 minutes)
Full path:	/tmp/MADbench2
Notes:	12-core server / HDD / 16 readers + writers

CPU
MPI
I/O

Summary: MADbench2 is I/O-bound in this configuration

The total wallclock time was spent as follows:

CPU	4.8%	Time spent running application code. High values are usually good. This is **low**; it may be worth improving I/O performance first.
MPI	41.3%	Time spent in MPI calls. High values are usually bad. This is **average**; check the MPI breakdown for advice on reducing it.
I/O	53.9%	Time spent in filesystem I/O. High values are usually bad. This is **high**; check the I/O breakdown section for optimization advice.

This application run was I/O-bound. A breakdown of this time and advice for investigating further is in the I/O section below.

CPU

A breakdown of how the 4.8% total CPU time was spent:

Scalar numeric ops	4.9%
Vector numeric ops	0.1%
Memory accesses	95.0%
Other	0.0

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

No time was spent in vectorized instructions. Check the compiler's vectorization advice to see why key loops could not be vectorized.

MPI

Of the 41.3% total time spent in MPI calls:

Time in collective calls	100.0%
Time in point-to-point calls	0.0%
Estimated collective rate	4.07 bytes/s
Estimated point-to-point rate	0 bytes/s

All of the time is spent in collective calls with a very low transfer rate. This suggests a significant load imbalance is causing synchronization overhead. You can investigate this further with an MPI profiler.

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.

Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	160 Mb
Peak process memory usage	173 Mb
Peak node memory usage	17.2%

The peak node memory usage is low. You may be able to reduce the total number of CPU hours used by running with fewer MPI processes and more data on each process.

- Very simple start-up
- No source code needed
- Fully scalable, very low overhead
- Rich set of metrics
- Powerful data analysis

allinea

Increase the efficiency of your jobs

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.

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 waiting for memory accesses. Reducing the CPU clock frequency could reduce overall energy usage.

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.

allinea

Allinea Forge: the toolkit for HPC developers

ACCESSIBLE

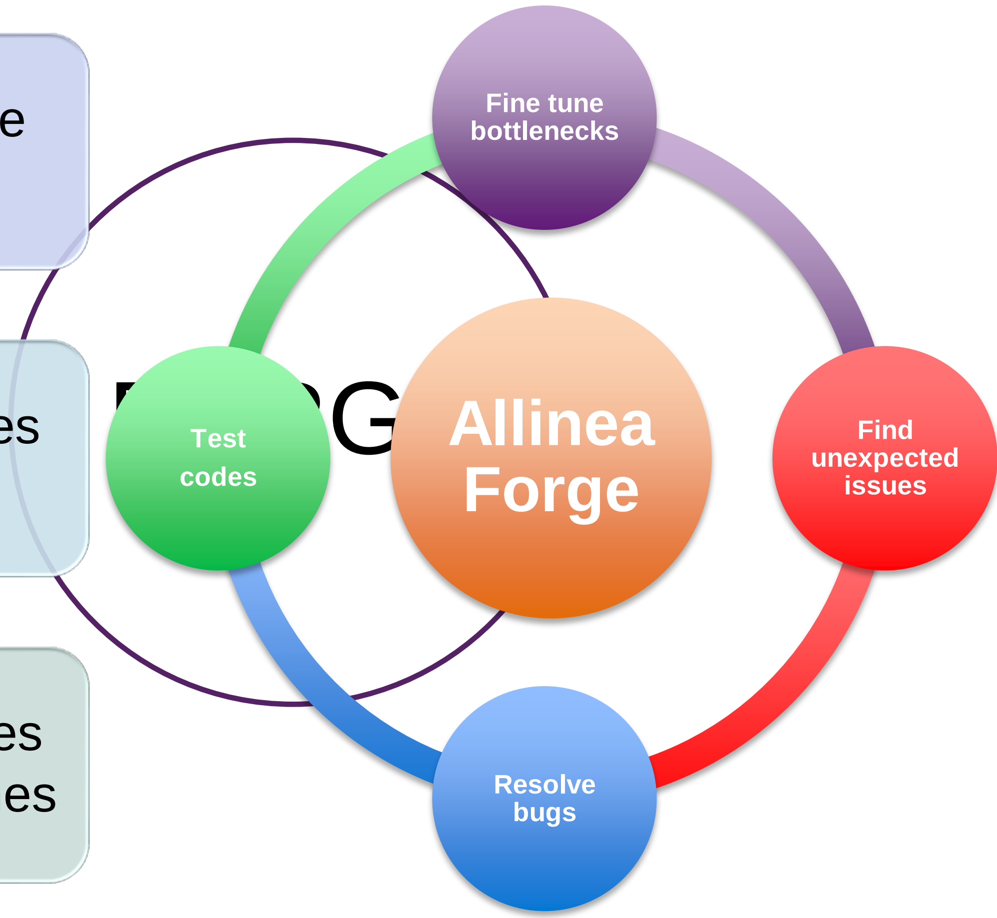
- Unique single interface
- Easy to start and use

POWERFUL

- State of the art features
- Fully scalable

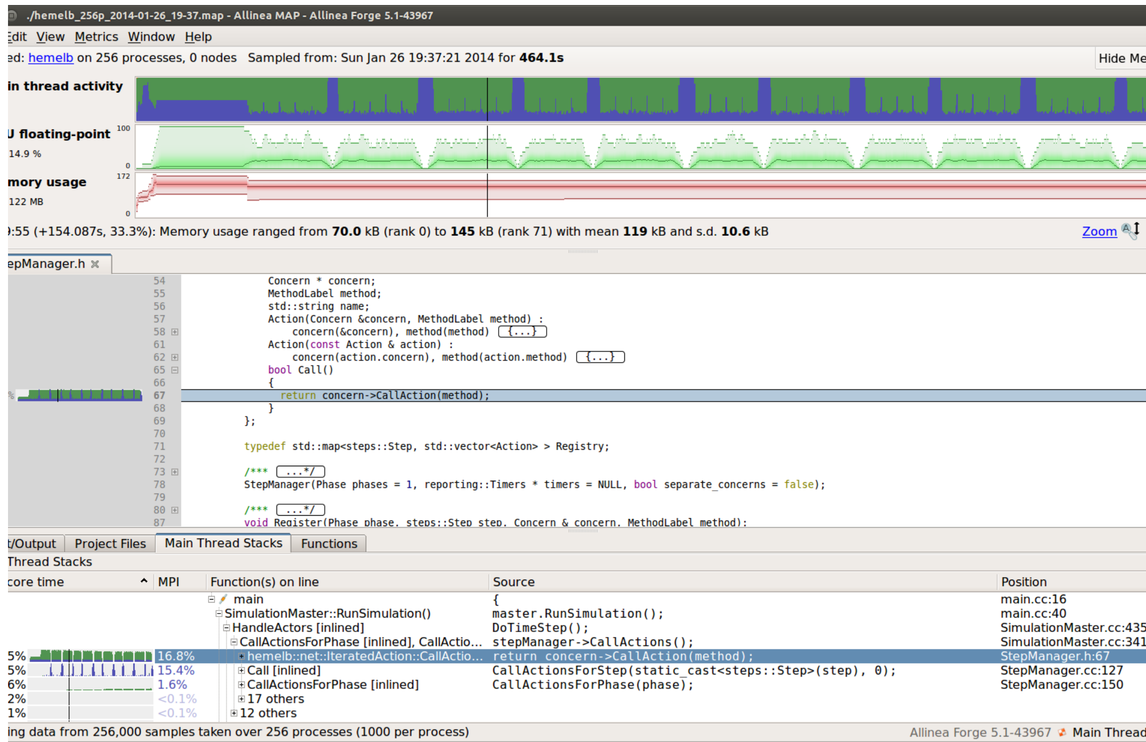
INNOVATIVE

- Tackles new challenges
- For latest IBM machines



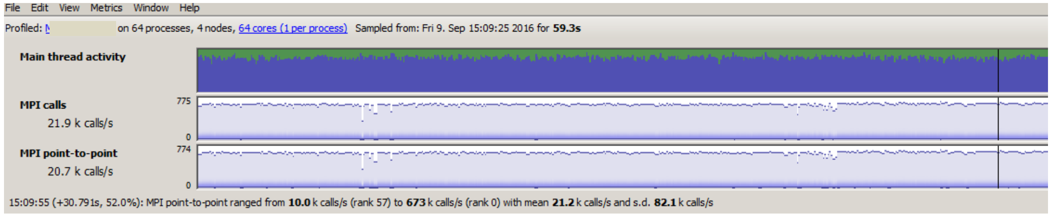
Allinea MAP – the profiler

- No instrumentation
- Low overhead
- Scalable
- Small data files

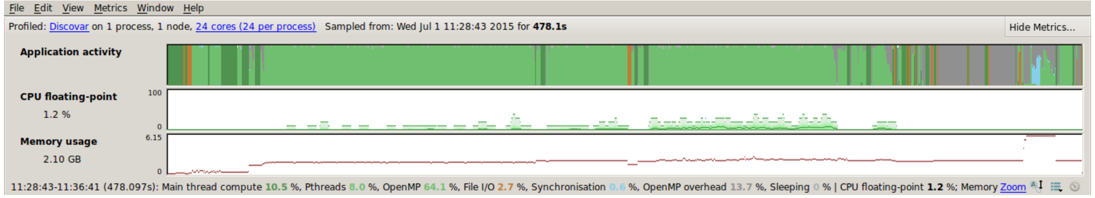


Quickly spot application bottlenecks

- Find patterns of MPI and OpenMP imbalance

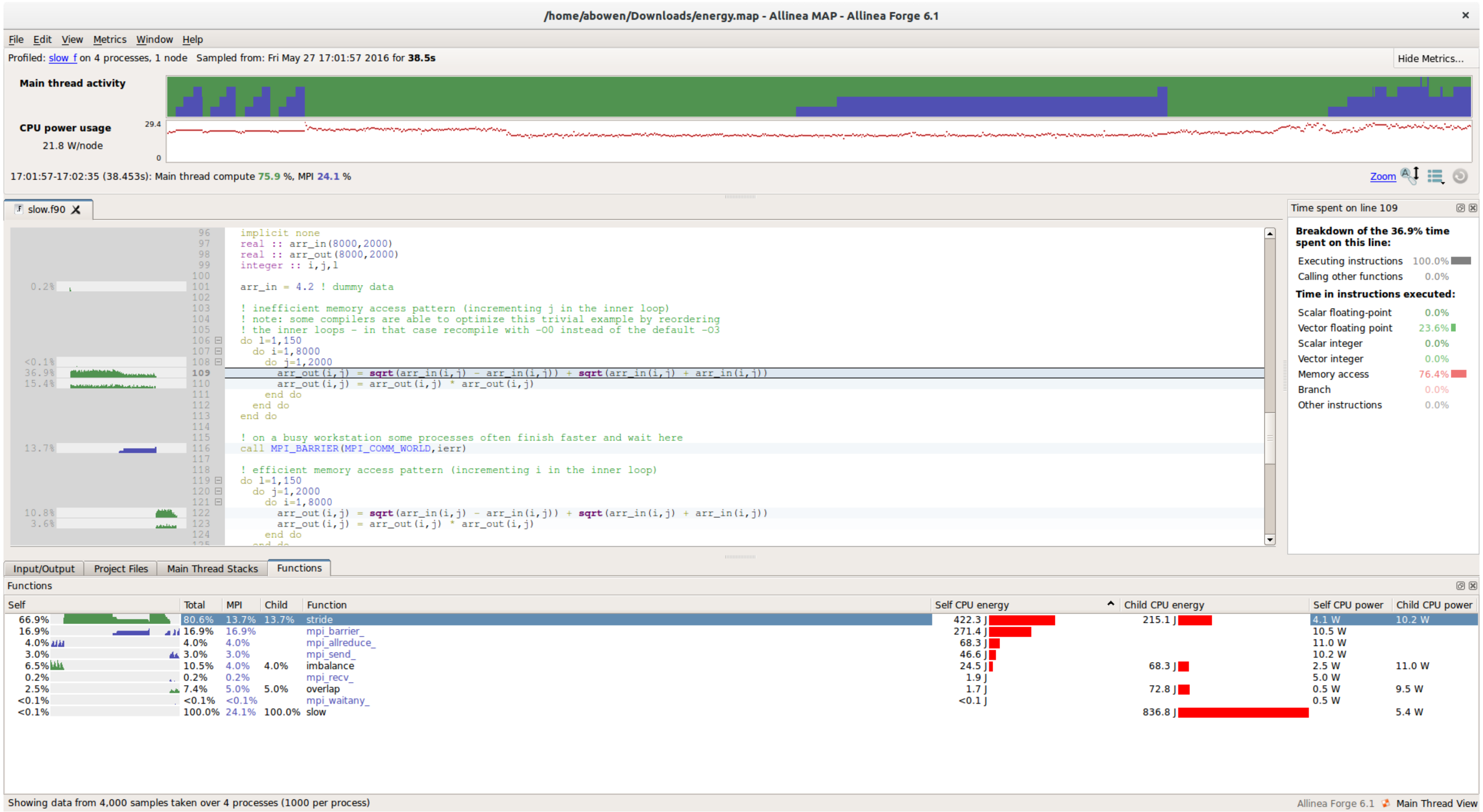


- Offload compute intensive parallel regions to be offloaded to a coprocessor or accelerator



allinea

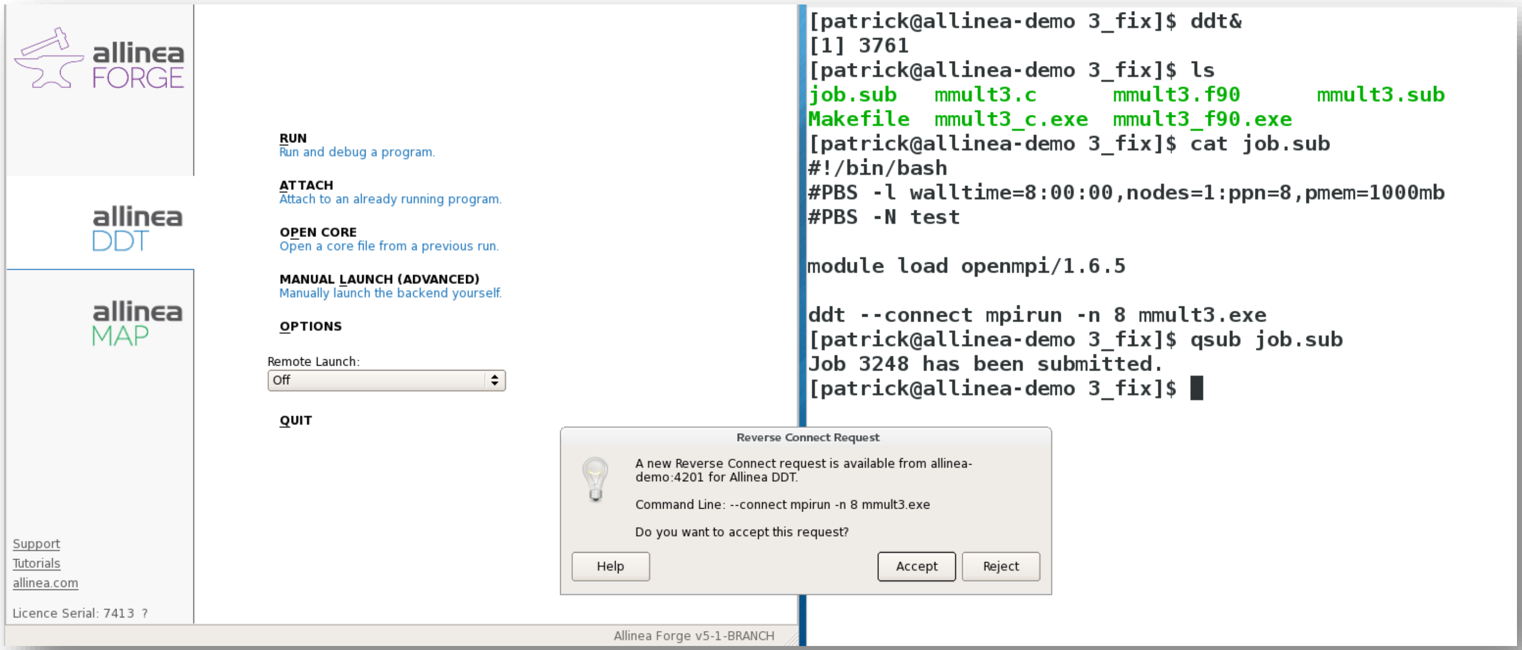
Energy analytics



allinea

Allinea DDT – the debugger

- Easily debug executable in your workflow with the “reverse connect” mechanism

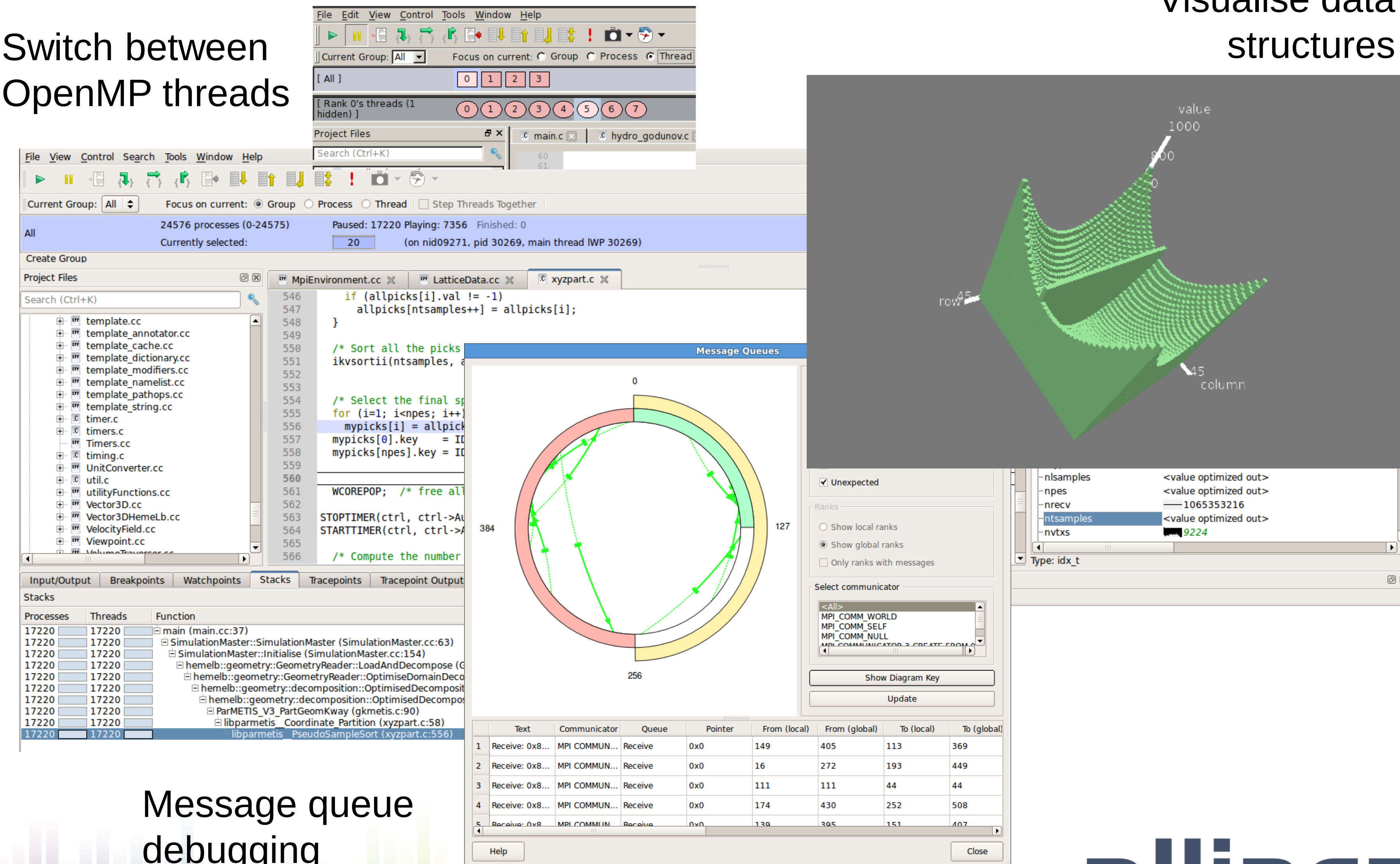


allinea

Debug large scale application

Switch between OpenMP threads

Visualise data structures

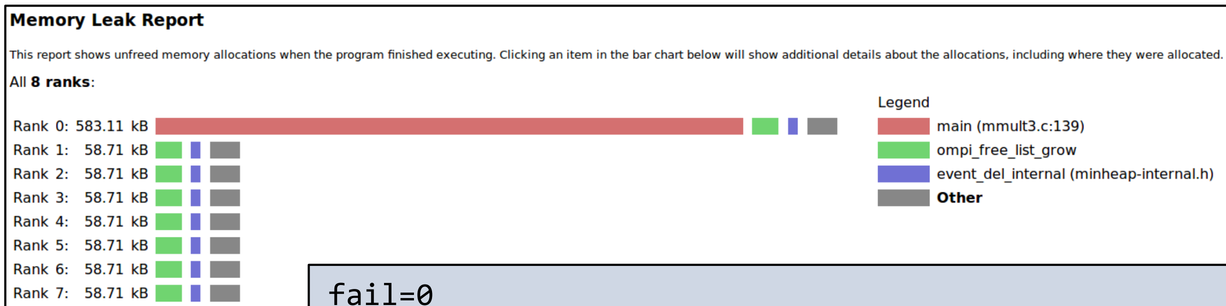


Message queue debugging

allinea

Automate debugging with offline mode

#	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”
```

allinea

Conclusion

- Analyse application efficiency and understand behaviour with Allinea Performance Reports

- Develop faster by debugging and optimising large-scale applications with Allinea Forge

- Available for latest architectures:
  - x86_64
  - KNL
  - CUDA 8.0
  - ARMv8
  - OpenPower

allinea

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

Thank you!

Any question, please ask.

Technical Support team : flebeau@allinea.com or support@allinea.com
Sales team : sales@allinea.com or marcin@allinea.com

allinea FORGE allinea DDT allinea MAP allinea PERFORMANCE REPORTS