allinea

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

Application Trapped Capacity and Energy Reporting

Mark O'Connor

VP Product Management

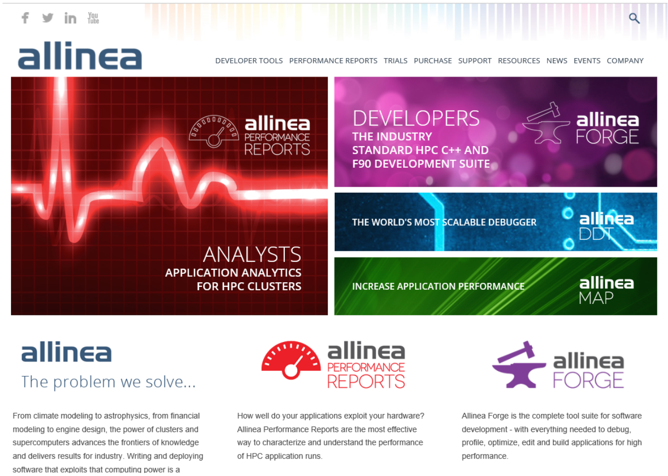
allinea FORGE

allinea DDT

allinea MAP

allinea PERFORMANCE REPORTS

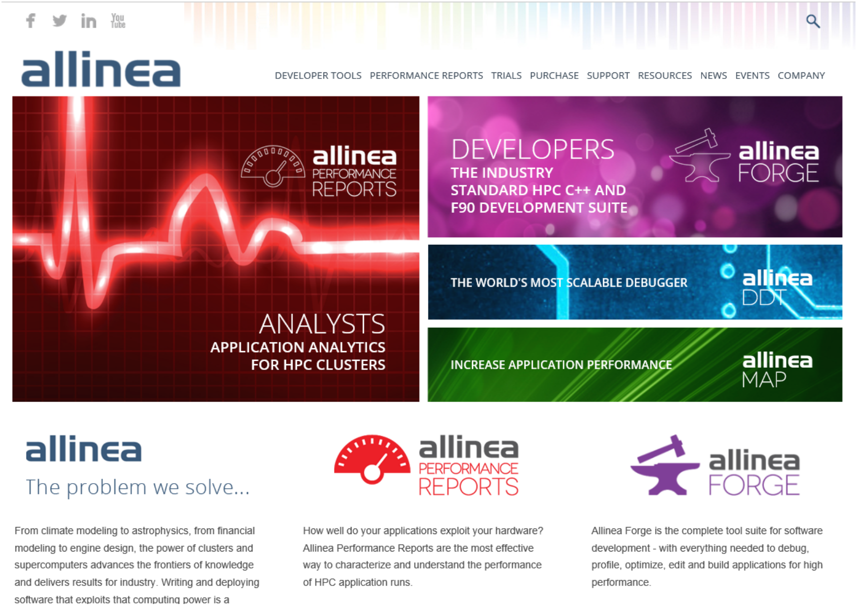
Application Trapped Capacity and Energy Reporting



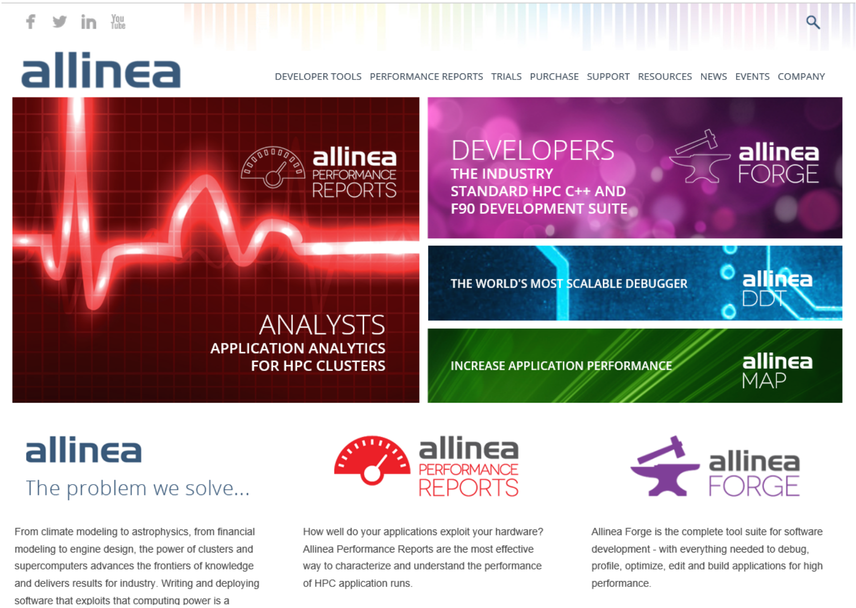
Today: Application-centric performance tuning and energy metrics

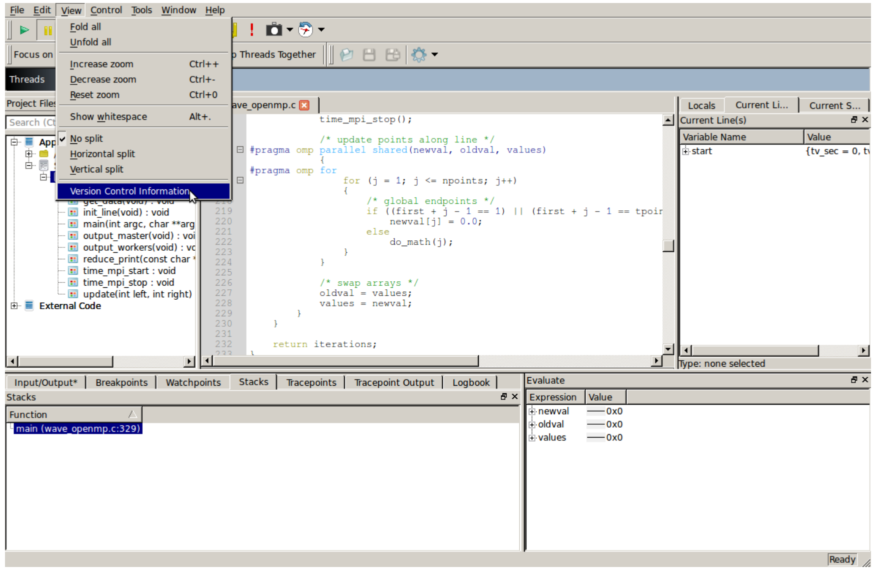
New research: Trapped Capacity Reports on a per-application basis

Future research: energy-aware scheduling, superoptimizing compilers, …

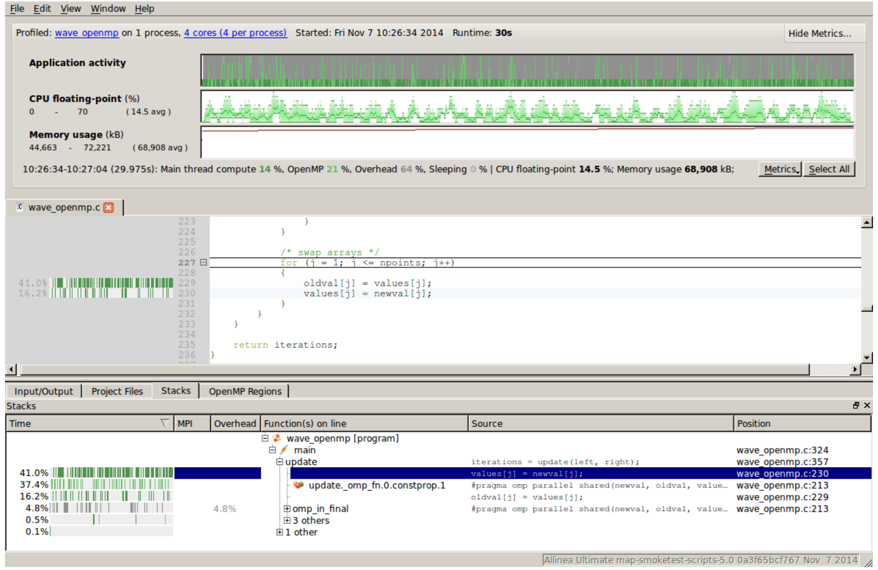


Allinea provides high-performance software tools





allinea
DDT



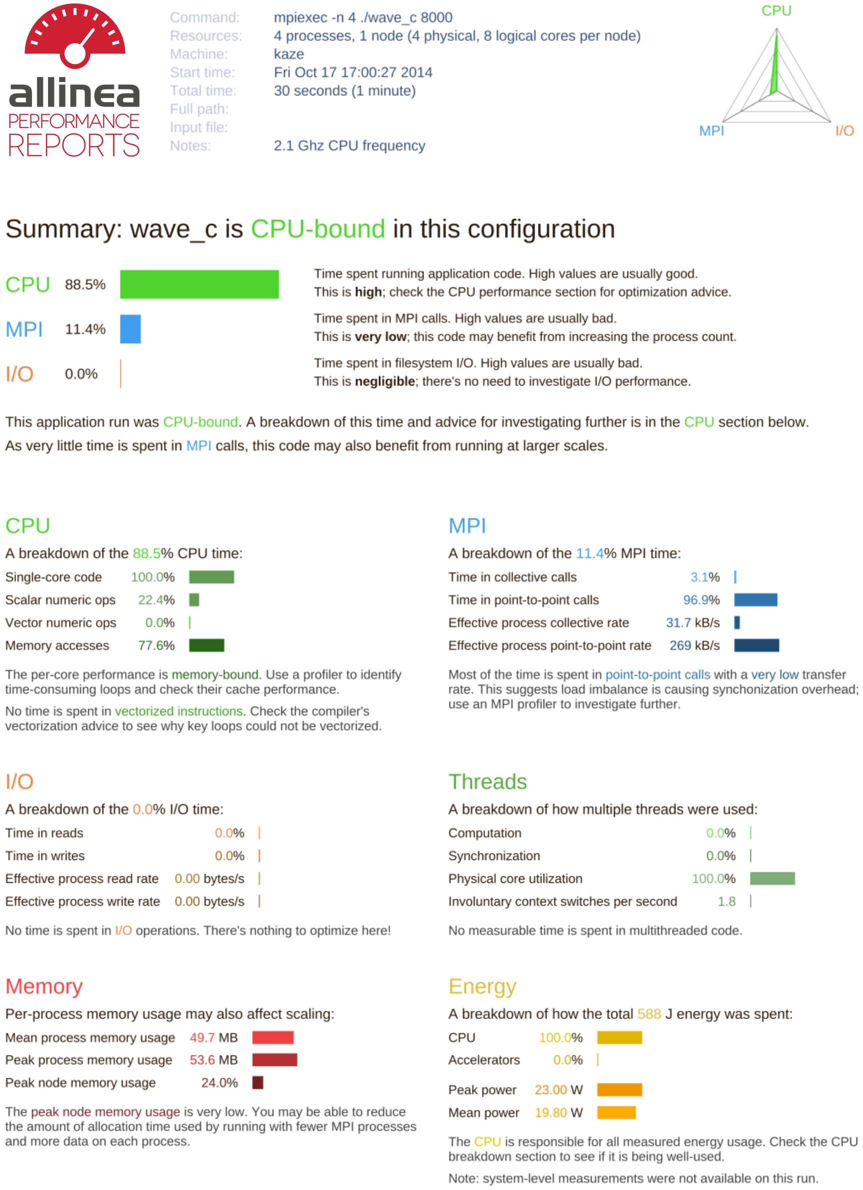
allinea
MAP

Allinea Forge

Debug and profile codes

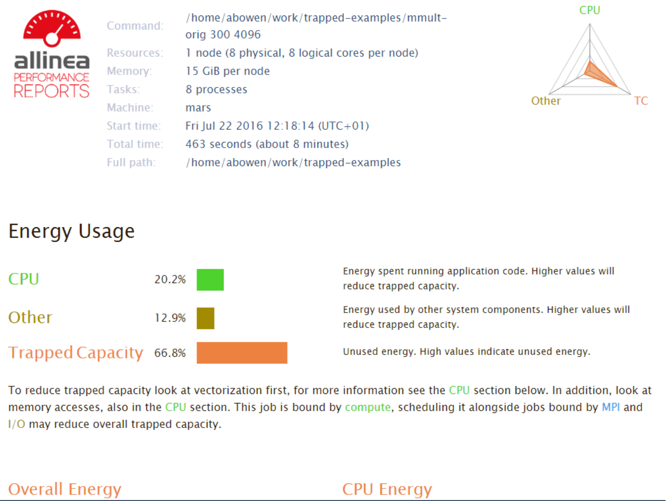
Performance Reports

Monitor and tune applications



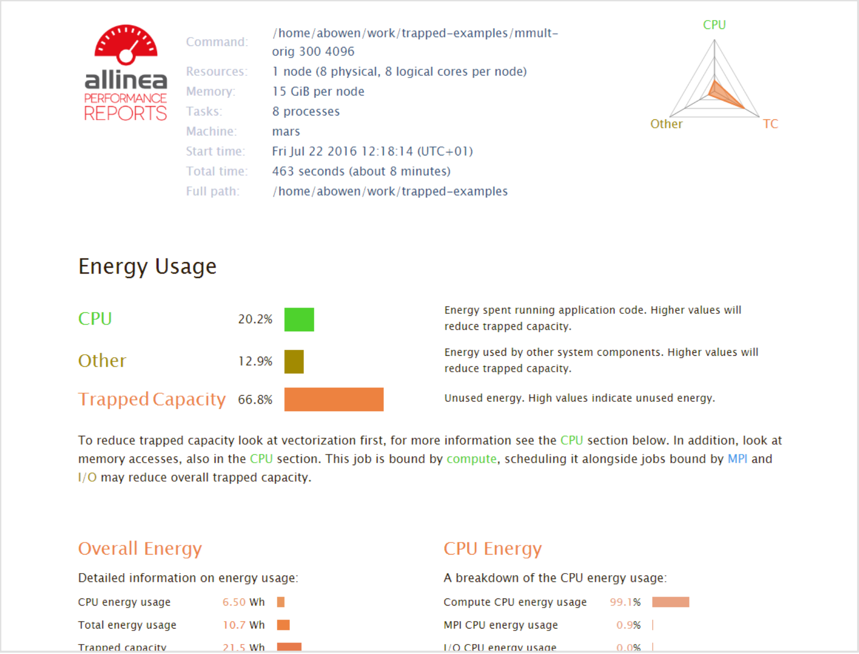
Application Trapped Capacity and Energy Reporting

Today: Application-centric performance tuning and energy metrics



New research: Trapped Capacity Reports on a per-application basis

Future research: energy-aware scheduling, superoptimizing compilers, …



allinea PERFORMANCE REPORTS

Command:	/home/abowen/work/trapped-examples/mmult-orig 300 4096
Resources:	1 node (8 physical, 8 logical cores per node)
Memory:	15 GiB per node
Tasks:	8 processes
Machine:	mars
Start time:	Fri Jul 22 2016 12:18:14 (UTC+01)
Total time:	463 seconds (about 8 minutes)
Full path:	/home/abowen/work/trapped-examples

Energy Usage

CPU	20.2%	Energy spent running application code. Higher values will reduce trapped capacity.
Other	12.9%	Energy used by other system components. Higher values will reduce trapped capacity.
Trapped Capacity	66.8%	Unused energy. High values indicate unused energy.

To reduce trapped capacity look at vectorization first, for more information see the CPU section below. In addition, look at memory accesses, also in the CPU section. This job is bound by compute, scheduling it alongside jobs bound by MPI and I/O may reduce overall trapped capacity.

Overall Energy

Detailed information on energy usage:

CPU energy usage	6.50 Wh
Total energy usage	10.7 Wh
Trapped capacity	21.5 Wh

CPU Energy

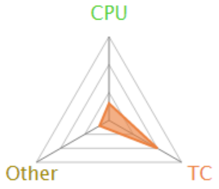
A breakdown of the CPU energy usage:

Compute CPU energy usage	99.1%
MPI CPU energy usage	0.9%
I/O CPU energy usage	0.0%

Application-centric Trapped Capacity Reports

- Show trapped capacity **per application run**
- Co-designed with Dr Imam's team in the Computational Research and Development Program at the Computing and Computational Sciences Directorate, ORNL
- Report and optimize to **minimize trapped capacity**, not just wallclock time
- Measures CPU, GPU and System power usage for each application run
- Includes actionable advice for users to help them make the most of the system's resources

Command:	/home/abowen/work/trapped-examples/mmult-orig 300 4096
Resources:	1 node (8 physical, 8 logical cores per node)
Memory:	15 GiB per node
Tasks:	8 processes
Machine:	mars
Start time:	Fri Jul 22 2016 12:18:14 (UTC+01)
Total time:	463 seconds (about 8 minutes)
Full path:	/home/abowen/work/trapped-examples



Energy Usage

CPU	20.2%	Energy spent running application code. Higher values will reduce trapped capacity.
Other	12.9%	Energy used by other system components. Higher values will reduce trapped capacity.
Trapped Capacity	66.8%	Unused energy. High values indicate unused energy.

To reduce trapped capacity look at vectorization first, for more information see the CPU section below. In addition, look at memory accesses, also in the CPU section. This job is bound by compute, scheduling it alongside jobs bound by MPI and I/O may reduce overall trapped capacity.

Overall Energy

Detailed information on energy usage:

CPU energy usage	6.50 Wh
Total energy usage	10.7 Wh
Trapped capacity	21.5 Wh
Total energy budget	32.1 Wh
Mean node power budget	250 W
Mean node power	82.8 W
Peak node power	98.0 W

The peak power draw is significantly higher than the mean, use a profiler to identify times when the power draw is low.

CPU Energy

A breakdown of the CPU energy usage:

Compute CPU energy usage	99.1%
MPI CPU energy usage	0.9%
I/O CPU energy usage	0.0%
Mean power during compute	50.5 W
Mean power during MPI	54.0 W
Mean power during I/O	0.00 W

Note: Power and energy values during MPI, I/O and compute are estimated from available information and may not be accurate.

CPU

A breakdown of the 99.2% CPU time:

Scalar numeric ops	50.8%
Vector numeric ops	0.0%
Memory accesses	49.2%

No time is spent in vectorized instructions. Vectorized instructions generally require more power than scalar computation. To reduce trapped capacity check the compiler's vectorization advice to see if key loops can be vectorized.

Significant time is spent on memory accesses. Memory operations generally require less power than computation. To reduce trapped capacity use a profiler to identify time-consuming loops and check their cache performance.

Threads

A breakdown of how multiple threads were used:

Computation	0.0%
Synchronization	0.0%
Physical core utilization	100.0%
System load	100.6%

MPI

A breakdown of the 0.8% MPI time:

Time in collective calls	2.9%
Time in point-to-point calls	97.1%
Effective process collective rate	0.00 bytes/s
Effective process point-to-point rate	33.4 MB/s

Optimising MPI communication will have little impact on the trapped capacity.

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, there's nothing to optimise here!

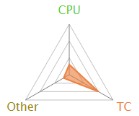
Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	210 MiB
Peak process memory usage	406 MiB
Peak node memory usage	14.0%

allinea PERFORMANCE REPORTS

Command:	/home/abowen/work/trapped-examples/mmult-orig 300 4096
Resources:	1 node (8 physical, 8 logical cores per node)
Memory:	15 GiB per node
Tasks:	8 processes
Machine:	mars
Start time:	Fri Jul 22 2016 12:18:14 (UTC+01)
Total time:	463 seconds (about 8 minutes)
Full path:	/home/abowen/work/trapped-examples



Energy Usage

CPU	20.2%	Energy spent running application code. Higher values will reduce trapped capacity.
Other	12.9%	Energy used by other system components. Higher values will reduce trapped capacity.
Trapped Capacity	66.8%	Unused energy. High values indicate unused energy.

To reduce trapped capacity look at vectorization first, for more information see the CPU section below. In addition, look at memory accesses, also in the CPU section. This job is bound by compute, scheduling it alongside jobs bound by MPI and I/O may reduce overall trapped capacity.

Overall Energy

Detailed information on energy usage:

CPU energy usage	6.50 Wh
Total energy usage	10.7 Wh
Trapped capacity	21.5 Wh
Total energy budget	32.1 Wh
Mean node power budget	250 W
Mean node power	82.8 W
Peak node power	98.0 W

The peak power draw is significantly higher than the mean, use a profiler to identify times when the power draw is low.

CPU Energy

A breakdown of the CPU energy usage:

Compute CPU energy usage	99.1%
MPI CPU energy usage	0.9%
I/O CPU energy usage	0.0%
Mean power during compute	50.5 W
Mean power during MPI	54.0 W
Mean power during I/O	0.00 W

Note: Power and energy values during MPI, I/O and compute are estimated from available information and may not be accurate.

CPU

A breakdown of the 99.2% CPU time:

Scalar numeric ops	50.8%
Vector numeric ops	0.0%
Memory accesses	49.2%

No time is spent in vectorized instructions. Vectorized instructions generally require more power than scalar computation. To reduce trapped capacity check the compiler's vectorization advice to see if key loops can be vectorized.

Significant time is spent on memory accesses. Memory operations generally require less power than computation. To reduce trapped capacity use a profiler to identify time-consuming loops and check their cache performance.

Threads

A breakdown of how multiple threads were used:

Computation	0.0%
Synchronization	0.0%
Physical core utilization	100.0%
System load	100.6%

MPI

A breakdown of the 0.8% MPI time:

Time in collective calls	2.9%
Time in point-to-point calls	97.1%
Effective process collective rate	0.00 bytes/s
Effective process point-to-point rate	33.4 MB/s

Optimising MPI communication will have little impact on the trapped capacity.

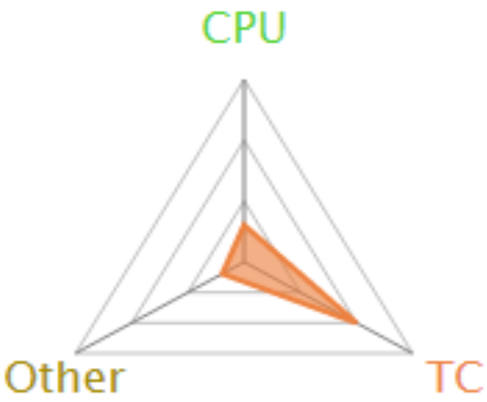
I/O

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, there's nothing to optimise here!

Trapped Capacity at a glance



Actionable advice

CPU	20.7%
Other	12.3%
Trapped Capacity	66.9%

To reduce trapped capacity look at vectoriz
memory accesses, also in the CPU section.
I/O may reduce overall trapped capacity.

Detailed breakdown

Overall Energy

Detailed information on energy usage:

CPU energy usage	6.50 Wh
Total energy usage	10.7 Wh
Trapped capacity	21.5 Wh
Total energy budget	32.1 Wh
Mean node power budget	250 W
Mean node power	82.8 W
Peak node power	98.0 W

The peak power draw is significantly higher than the mean,
profiler to identify times when the power draw is low.

Track CPU and GPU energy use

CPU Energy

A breakdown of the CPU energy usage:

Compute CPU energy usage	99.1%
MPI CPU energy usage	0.9%
I/O CPU energy usage	0.0%
Mean power during compute	50.5 W
Mean power during MPI	54.0 W
Mean power during I/O	0.00 W

Note: Power and energy values during MPI, I/O anc estimated from available information and may not

Identify inefficient applications

CPU

A breakdown of the 99.2% CPU time:

Scalar numeric ops	50.8%
Vector numeric ops	0.0%
Memory accesses	49.2%

No time is spent in vectorized instructions
instructions generally require more power
computation. To reduce trapped capacity
vectorization advice to see if key loops ca

Significant time is spent on memory acces

I/O and MPI breakdowns

I/O

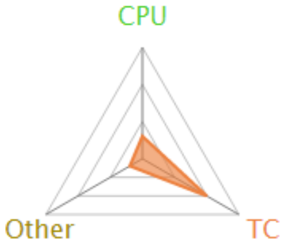
A breakdown of the 0.0% I/O time:

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, there's nothing to op

Example 1: application binary not optimized for CPU architecture

Command:	/home/abowen/work/trapped-examples/mmult-orig 300 4096
Resources:	1 node (8 physical, 8 logical cores per node)
Memory:	15 GiB per node
Tasks:	8 processes
Machine:	mars
Start time:	Fri Jul 22 2016 12:18:14 (UTC+01)
Total time:	463 seconds (about 8 minutes)
Full path:	/home/abowen/work/trapped-examples



Overall Energy

Detailed information on energy usage:

CPU energy usage	6.50 Wh
Total energy usage	10.7 Wh
Trapped capacity	21.5 Wh
Total energy budget	32.1 Wh
Mean node power budget	250 W
Mean node power	82.8 W
Peak node power	98.0 W

Energy Usage

CPU	20.2%	Energy spent running application code. Higher values will reduce trapped capacity.
Other	12.9%	Energy used by other system components. Higher values will reduce trapped capacity.
Trapped Capacity	66.8%	Unused energy. High values indicate unused energy.

To reduce trapped capacity look at vectorization first, for more information see the CPU section below. In addition, look at memory accesses, also in the CPU section. This job is bound by compute, scheduling it alongside jobs bound by MPI and I/O may reduce overall trapped capacity.

CPU

A breakdown of the 99.2% CPU time:

Scalar numeric ops	50.8%
Vector numeric ops	0.0%
Memory accesses	49.2%

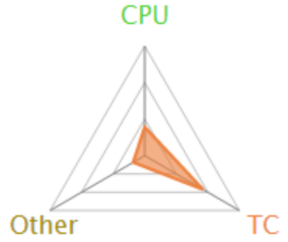
No time is spent in vectorized instructions. Vectorized instructions generally require more power than scalar computation. To reduce trapped capacity check the compiler's vectorization advice to see if key loops can be vectorized.

Significant time is spent on memory accesses. Memory operations generally require less power than computation. To reduce trapped capacity use a profiler to identify time-consuming loops and check their cache performance.

Example 1: recompiled with architecture-specific optimizations

allinea PERFORMANCE REPORTS

Command:	/home/abowen/work/trapped-examples/mmult-vec 300 256
Resources:	1 node (8 physical, 8 logical cores per node)
Memory:	15 GiB per node
Tasks:	8 processes
Machine:	mars
Start time:	Fri Jul 22 2016 15:19:11 (UTC+01)
Total time:	301 seconds (about 5 minutes)
Full path:	/home/abowen/work/trapped-examples



Overall Energy

Detailed information on energy usage:

CPU energy usage	5.43 Wh
Total energy usage	7.97 Wh
Trapped capacity	12.9 Wh
Total energy budget	20.8 Wh
Mean node power budget	250 W
Mean node power	95.6 W
Peak node power	98.0 W

Energy Usage

CPU	26.1%	Energy spent running application code. Higher values will reduce trapped capacity.
Other	12.2%	Energy used by other system components. Higher values will reduce trapped capacity.
Trapped Capacity	61.7%	Unused energy. High values indicate unused energy.

To reduce trapped capacity look at memory accesses first, for more information see the CPU section below. In addition, look at vectorization, also in the CPU section. This job is bound by compute, scheduling it alongside jobs bound by MPI and I/O may reduce overall trapped capacity.

CPU

A breakdown of the 97.2% CPU time:

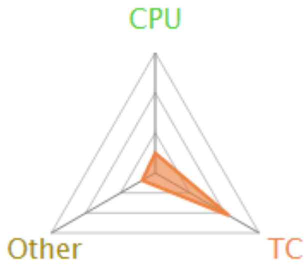
Scalar numeric ops	17.3%
Vector numeric ops	37.8%
Memory accesses	44.9%

Significant time is spent on memory accesses. Memory operations generally require less power than computation. To reduce trapped capacity use a profiler to identify time-consuming loops and check their cache performance.

Example 2: CPU idle due to inefficient OpenMP scheduling

allinea PERFORMANCE REPORTS

Command:	/home/abowen/work/trapped-examples/umult-omp-static 300 1024
Resources:	1 node (8 physical, 8 logical cores per node)
Memory:	15 GiB per node
Tasks:	1 process, OMP_NUM_THREADS was 8
Machine:	node-5
Start time:	Fri Jul 22 2016 14:39:01 (UTC+01)
Total time:	301 seconds (about 5 minutes)
Full path:	/home/abowen/work/trapped-examples



Energy Usage

CPU	16.4%	Energy spent running application code. Higher values will reduce trapped capacity.
Other	12.5%	Energy used by other system components. Higher values will reduce trapped capacity.
Trapped Capacity	71.0%	Unused energy. High values indicate unused energy.

To reduce trapped capacity look at OpenMP overhead first, for more information see the OpenMP section below. In addition, look at vectorization, see the CPU section below. This job is bound by compute, scheduling it alongside jobs bound by MPI and I/O may reduce overall trapped capacity.

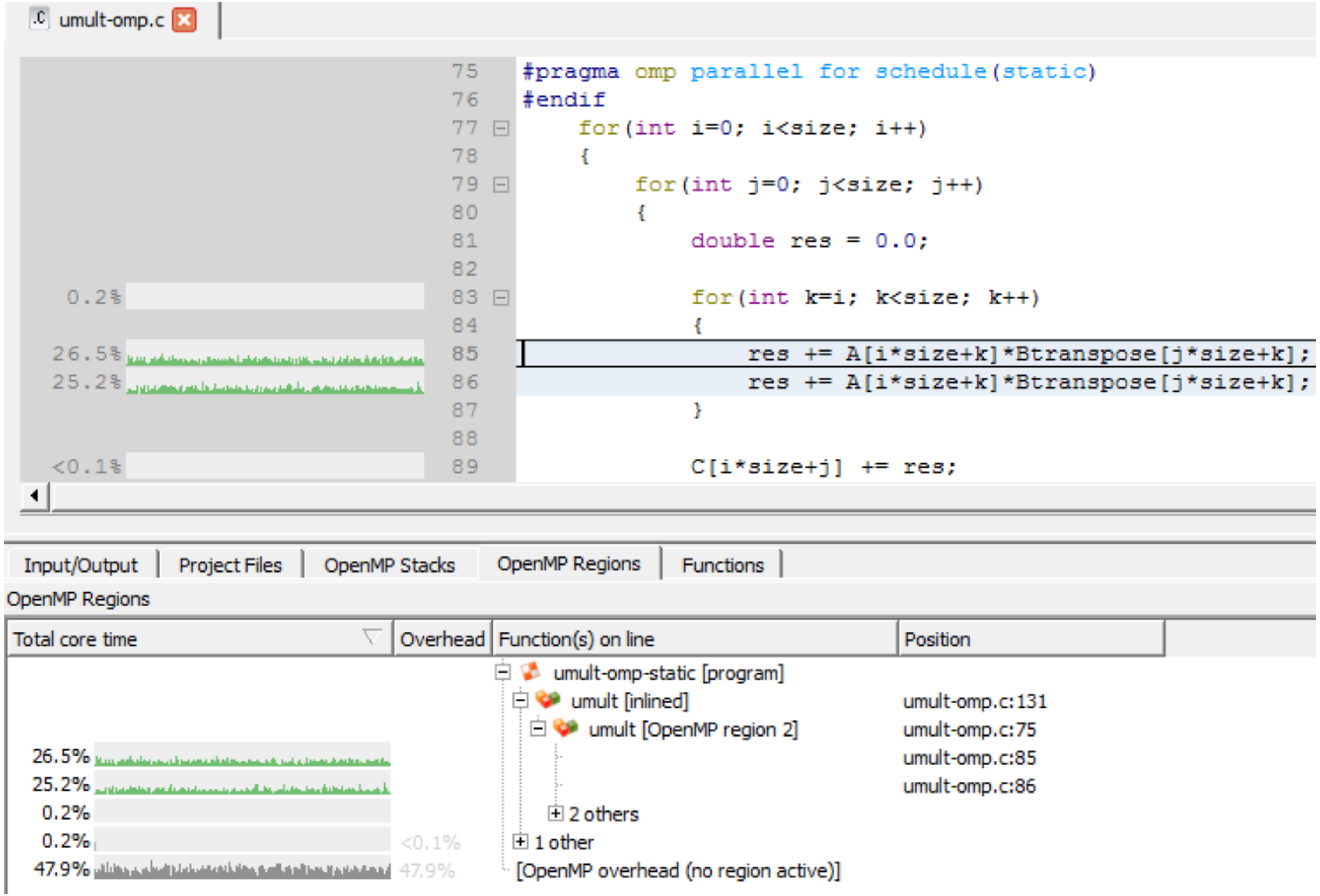
OpenMP

A breakdown of the 100.0% time in OpenMP regions:

Computation	52.0%
Synchronization	48.0%
Physical core utilization	100.0%
System load	55.4%

Significant time is spent synchronizing threads. Sleeping CPU cores require minimal power. Check which locks cause the most overhead with a profiler and improve workload balance to reduce trapped capacity.

Example 2: Allinea Forge identifies the inefficient loop



allinea FORGE

Example 2: Dynamic scheduling reduces trapped capacity

allinea PERFORMANCE REPORTS

Command:	/home/abowen/work/trapped-examples/umult-omp-dynamic 300 1024
Resources:	1 node (8 physical, 8 logical cores per node)
Memory:	15 GiB per node
Tasks:	1 process, OMP_NUM_THREADS was 8
Machine:	node-5
Start time:	Fri Jul 22 2016 14:44:34 (UTC+01)
Total time:	301 seconds (about 5 minutes)
Full path:	/home/abowen/work/trapped-examples

OpenMP

A breakdown of the 100.0% time in OpenMP regions:

Computation	99.9%
Synchronization	<0.1%
Physical core utilization	100.0%
System load	100.1%

Energy Usage

CPU	20.7%	Energy spent running application code. Higher values will reduce trapped capacity.
Other	12.3%	Energy used by other system components. Higher values will reduce trapped capacity.
Trapped Capacity	66.9%	Unused energy. High values indicate unused energy.

To reduce trapped capacity look at vectorization first, for more information see the CPU section below. In addition, look at memory accesses, also in the CPU section. This job is bound by compute, scheduling it alongside jobs bound by MPI and I/O may reduce overall trapped capacity.

CPU

A breakdown of the 100.0% CPU time:

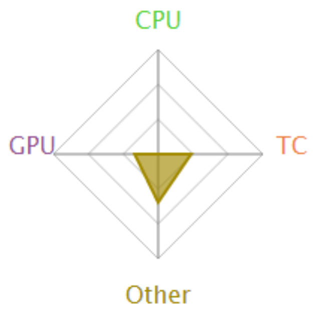
Single-core code	0.0%
OpenMP regions	100.0%
Scalar numeric ops	85.1%
Vector numeric ops	0.0%
Memory accesses	14.9%

No time is spent in vectorized instructions. Vectorized instructions generally require more power than scalar computation. To reduce trapped capacity check the compiler's vectorization advice to see if key loops can be vectorized.

Example 3: GPU accelerators unused by application

allinea PERFORMANCE REPORTS

Command:	/home/abowen/work/trapped-examples/gpu-waiting-nogpu 300 12
Resources:	1 node (12 physical, 24 logical cores per node) 2 GPUs per node available
Memory:	15 GiB per node, 11 GiB per GPU
Tasks:	1 process
Machine:	rhel-7-amd64
Start time:	Wed Jul 20 2016 11:38:37 (UTC+01)
Total time:	307 seconds (about 5 minutes)
Full path:	/home/abowen/work/trapped-examples



Energy Usage

CPU	0.0%	Energy spent running application code. Higher values will reduce trapped capacity.
GPU	22.4%	Energy spent running accelerator code. Higher values will reduce trapped capacity.
Other	46.1%	Energy used by other system components. Higher values will reduce trapped capacity.
Trapped Capacity	31.5%	Unused energy. High values indicate unused energy.

CPU metrics are not supported (no intel_rapl module)

To reduce trapped capacity look at accelerator usage first, for more information see the Accelerators section below. In addition, look at vectorization, see the CPU section below. This job is bound by compute, scheduling it alongside jobs bound by MPI, accelerator usage, and I/O may reduce overall trapped capacity.

Accelerators

A breakdown of how accelerators were used:

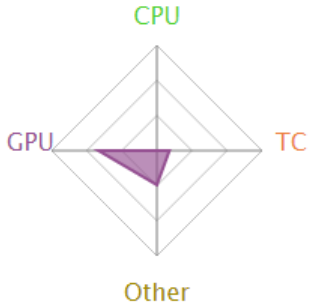
GPU utilization	0.0%
Global memory accesses	0.0%
Mean GPU memory usage	0.2%
Peak GPU memory usage	0.2%

Accelerators are available but are not used. It may be more efficient to make use of accelerators or run on nodes without them.

Example 3: Developer uses CUDA library. Are we done?

allinea PERFORMANCE REPORTS

Command:	/home/abowen/work/trapped-examples/gpu-waiting 300 12
Resources:	1 node (12 physical, 24 logical cores per node) 2 GPUs per node available
Memory:	15 GiB per node, 11 GiB per GPU
Tasks:	1 process
Machine:	rhel-7-amd64
Start time:	Wed Jul 20 2016 11:27:05 (UTC+01)
Total time:	301 seconds (about 5 minutes)
Full path:	/home/abowen/work/trapped-examples



CPU

A breakdown of the 100.0% CPU time:

Scalar numeric ops	0.0%
Vector numeric ops	0.0%
Memory accesses	0.0%
Waiting for accelerators	99.9%

Most of the time is spent waiting for accelerators. Sleeping CPU cores require minimal power. To reduce trapped capacity use asynchronous calls to overlap CPU and accelerator workloads.

Significant time is spent on memory accesses. Cache-inefficient memory operations generally require less power than computation. To reduce trapped capacity use a profiler to identify time-consuming loops and check their cache performance.

Energy Usage

CPU	0.0%	Energy spent running application code. Higher values will reduce trapped capacity.
GPU	55.5%	Energy spent running accelerator code. Higher values will reduce trapped capacity.
Other	32.6%	Energy used by other system components. Higher values will reduce trapped capacity.
Trapped Capacity	11.9%	Unused energy. High values indicate unused energy.

CPU metrics are not supported (no intel_rapl module)

To reduce trapped capacity look at time spent waiting for accelerators first, for more information see the CPU section below. In addition, look at accelerator usage, see the Accelerators section below.

Accelerators

A breakdown of how accelerators were used:

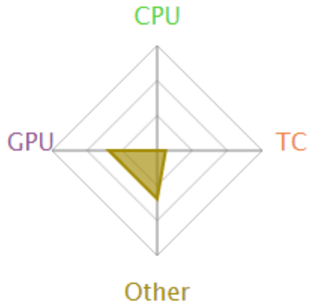
GPU utilization	48.6%
Global memory accesses	13.7%
Mean GPU memory usage	0.9%
Peak GPU memory usage	0.9%

The accelerator usage is low. Increasing the accelerator usage could reduce the trapped capacity.

Example 3: Expert helps user overlap CPU and GPU work

allinea PERFORMANCE REPORTS

Command:	/home/abowen/work/trapped-examples/gpu-waiting-async 300 12
Resources:	1 node (12 physical, 24 logical cores per node) 2 GPUs per node available
Memory:	15 GiB per node, 11 GiB per GPU
Tasks:	1 process
Machine:	rhel-7-amd64
Start time:	Wed Jul 20 2016 11:32:57 (UTC+01)
Total time:	304 seconds (about 5 minutes)
Full path:	/home/abowen/work/trapped-examples



Energy Usage

CPU	0.0%	Energy spent running application code. Higher values will reduce trapped capacity.
GPU	45.9%	Energy spent running accelerator code. Higher values will reduce trapped capacity.
Other	46.1%	Energy used by other system components. Higher values will reduce trapped capacity.
Trapped Capacity	8.0%	Unused energy. High values indicate unused energy.

CPU metrics are not supported (no intel_rapl module)

To reduce trapped capacity look at accelerator usage first, for more information see the Accelerators section below. In addition, look at vectorization, see the CPU section below. This job is bound by compute, scheduling it alongside jobs bound by MPI, accelerator usage, and I/O may reduce overall trapped capacity.

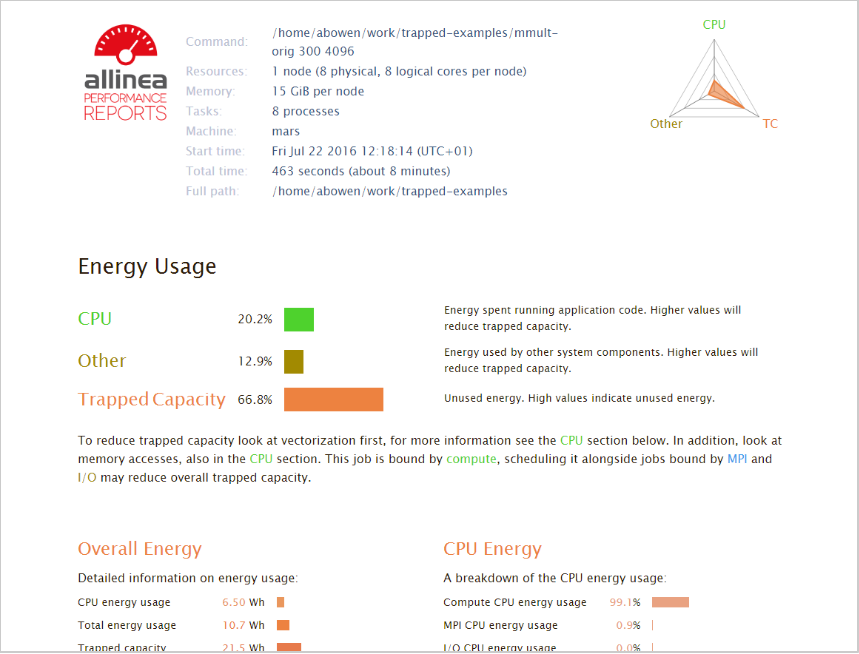
CPU

A breakdown of the 100.0% CPU time:

Scalar numeric ops	48.5%
Vector numeric ops	0.0%
Memory accesses	51.2%
Waiting for accelerators	0.4%

Significant time is spent on memory accesses. Cache-inefficient memory operations generally require less power than computation. To reduce trapped capacity use a profiler to identify time-consuming loops and check their cache performance.

No time is spent in vectorized instructions. Vectorized instructions generally require more power than scalar computation. To reduce trapped capacity check the compiler's vectorization advice to see if key loops can be vectorized.



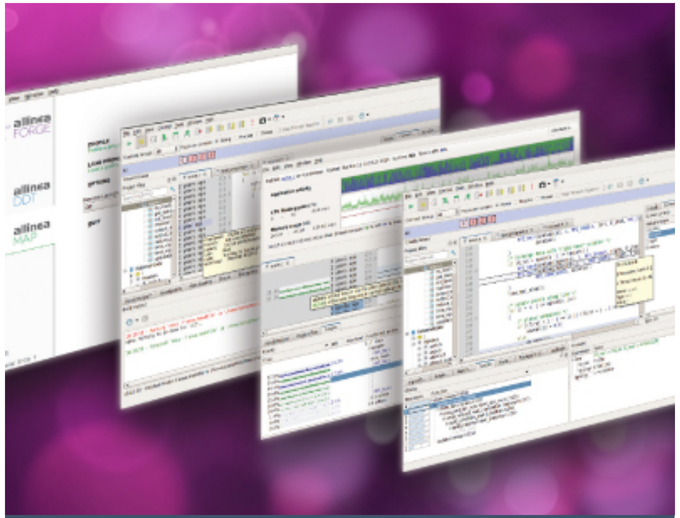
Application-centric Trapped Capacity Reports

- Show trapped capacity **per application run**
- Deployed in the Computing and Computational Sciences Directorate, ORNL
- Not currently enabled in public builds on our website
- Available to HPM16 attendees on request

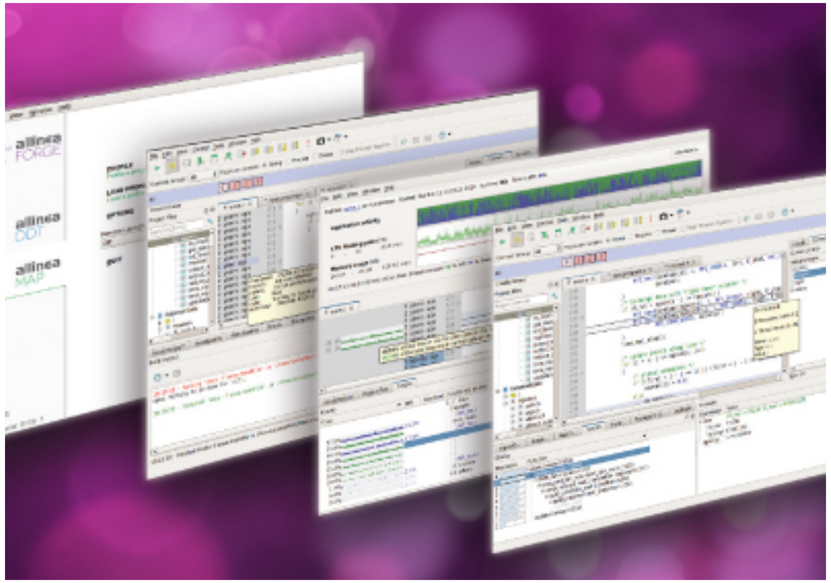
Application Trapped Capacity and Energy Reporting

Today: Application-centric performance tuning and energy metrics

New research: Trapped Capacity Reports on a per-application basis



Future research: energy-aware scheduling, superoptimizing compilers, …



Active research

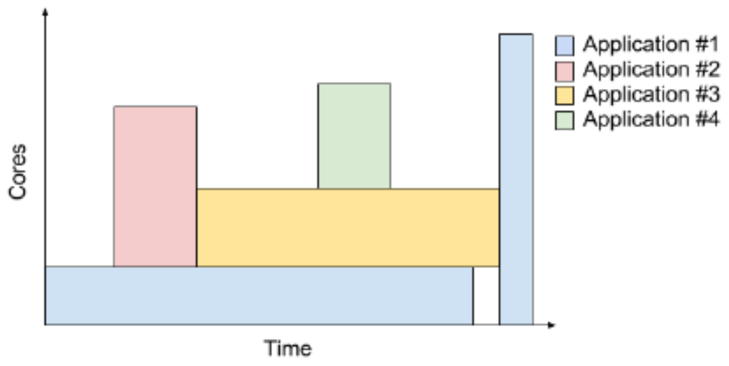
Ongoing research:

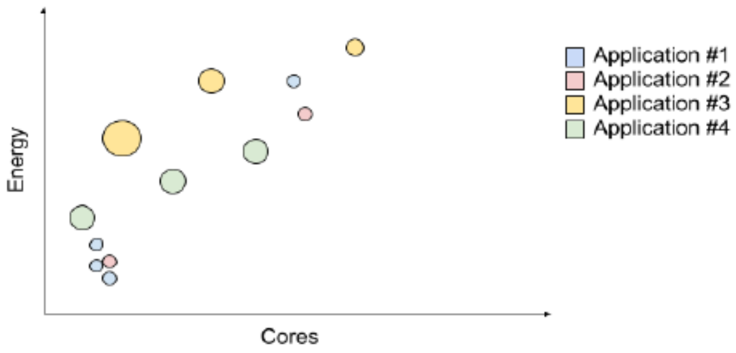
- Application Energy Dashboard
- Provide energy cost model for superoptimizing compiler
- Feed machine learning algorithm that chooses optimal compilation flags
- Data- and energy-aware scheduling via SLURM integration

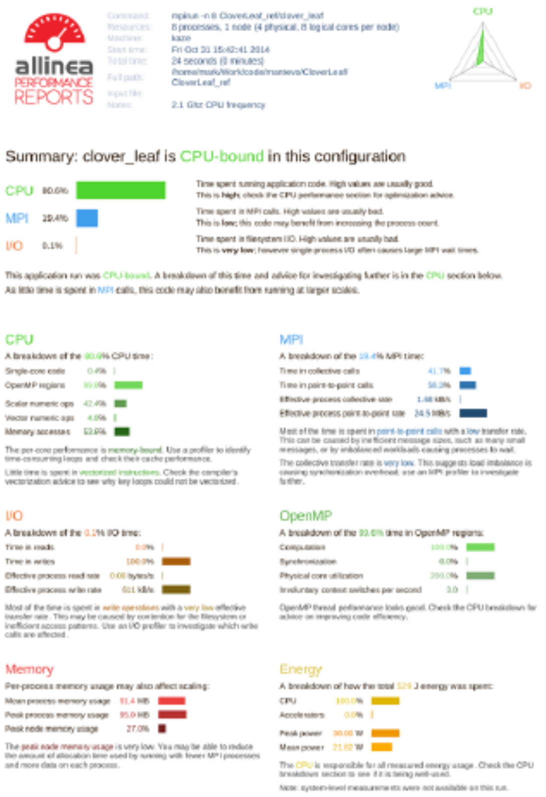
Total Software Energy Reporting and Optimization

Application Energy Dashboard

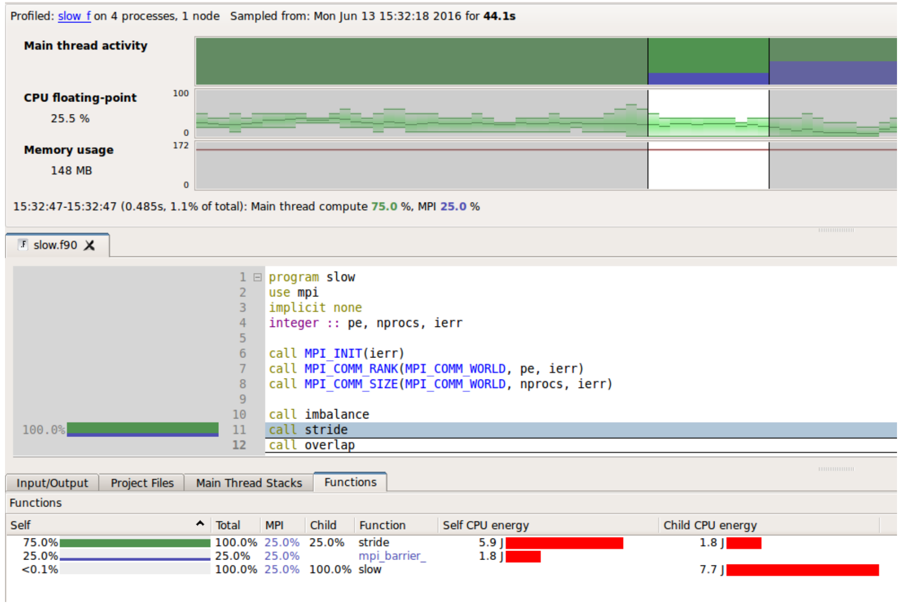
- **Collect** per-application energy metrics
- **Visualize** application runs and energy usage over time
- **Dive** into particular applications for a Performance Report

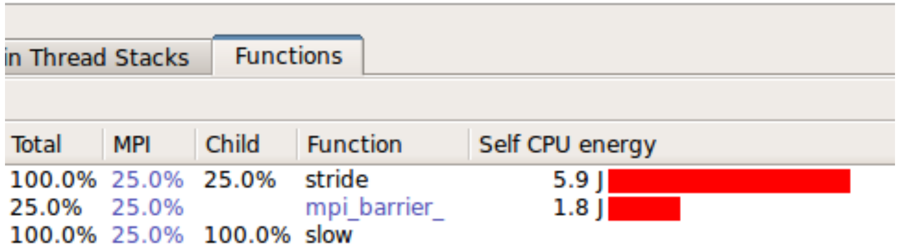






Total **S**oftware **E**nergy **R**eporting and **O**ptimization





Superoptimizing compiler

- MAP provides function-level performance data to the compiler
- EMBECOSM's stochastic superoptimizer targets most energy-costly functions
- Reruns under MAP to measure impact of changes

Machine Guided Energy Efficient Compilation

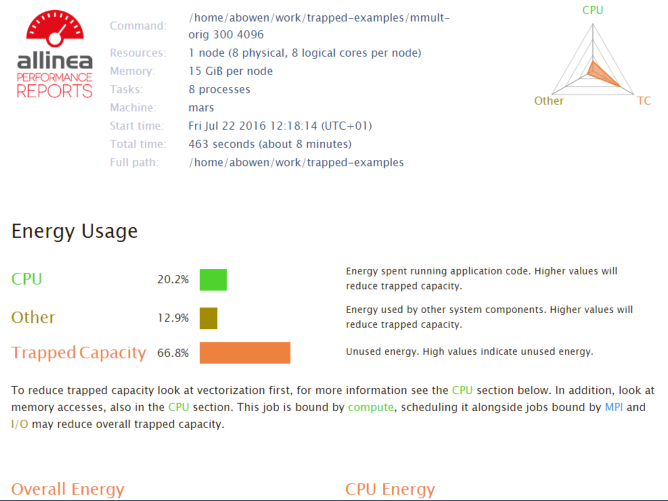
- Building on existing MAGEEC project for embedded systems
- Uses machine learning to predict compiler option combinations that boost energy efficiency
- Extending to take additional data from MAP in the input vector

Horizon 2020: Next Generation I/O for Exascale

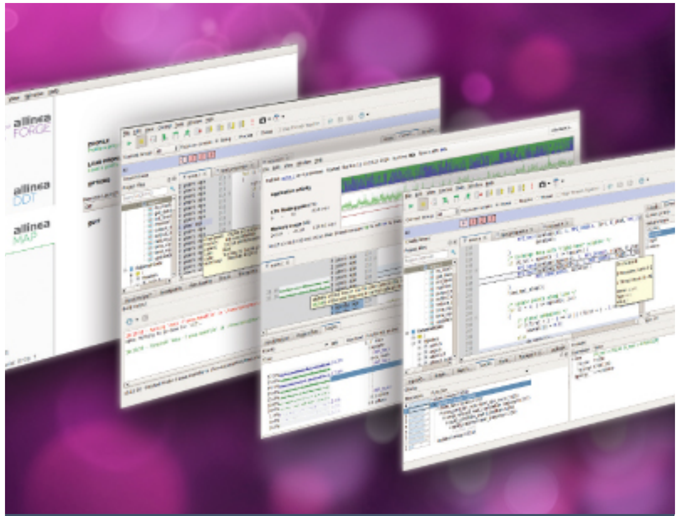
Work package overview	Target: **50% reduction in energy to solution** when compared to today's systems
	Target: 20x improved I/O performance for **real applications**
	Performance Reports will feed energy and I/O measurements into the SLURM workload scheduler
Scheduler integration	Preload data onto nodes before job starts or schedule work close to existing data location
	Schedule compute-intensive jobs on cooler parts of the system
	Limit clock speed on jobs known to be memory-bound

Application Trapped Capacity and Energy Reporting

Today: Application-centric performance tuning and energy metrics



New research: Trapped Capacity Reports on a per-application basis



Future research: energy-aware scheduling, superoptimizing compilers, …

allinea

allinea

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

Technology wish list: open cross-vendor energy APIs (that do *not* require root access)

Mark O'Connor

VP Product Management

allinea FORGE

allinea DDT

allinea MAP

allinea PERFORMANCE REPORTS