

THE ALLINEA DDT DEBUGGER AND MAP PERFORMANCE PROFILER



RYAN HULGUIN
Applications Engineer
ARM

Ryan.Hulguin@arm.com

May 3, 2017
Argonne, IL

AS OF DECEMBER 2016, ALLINEA IS PART OF ARM

Our objective:

Remain the trusted leader in cross platform HPC tools

The same successful team...

- We will continue to work with our customers, partners and you!

... is stronger than ever...

- We can now respond quicker and deliver our roadmap faster

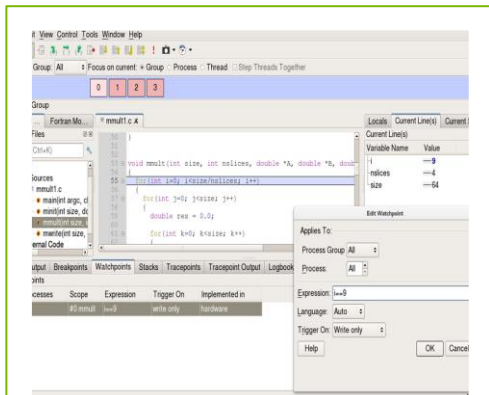
... as committed as ever...

- We remain 100% committed to providing cross-platforms tools for HPC

... and looking forward to the future.

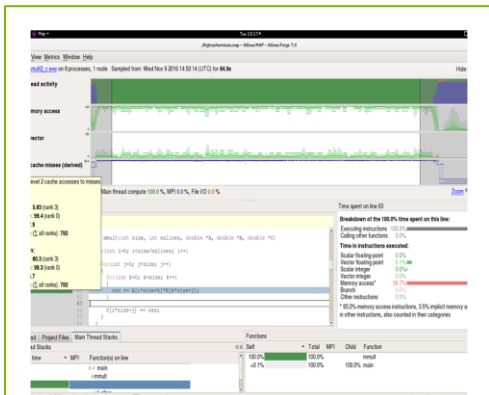
- We are working with vendors to support the next generations of systems.

ALLINEA TOOLKITS SAVE USERS' AND DEVELOPERS' TIME



The screenshot shows the Alinea Forge debugging environment. On the left, there's a source code editor with a C program. In the center, a 'Locals' window displays variable values like 'result', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'. On the right, a 'Breakpoints' window shows a breakpoint set on line 55. The bottom of the window has a 'Process' tab and a 'Logbook'.

Alinea Forge (debugging)



The screenshot displays the Alinea Forge profiling interface. It features a main timeline showing CPU activity (green), memory accesses (blue), and cache misses (red). Below the timeline, there are detailed views for 'Time spent on line 53' and 'Breakdown of the 100.0% time spent on this line'. The 'Time in instructions executed' section lists various instructions and their execution counts. A table at the bottom shows the 'Functions on line' with columns for 'Line', 'Function', 'Total', 'MPI', and 'Child'.

Alinea Forge (profiling)

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

The total wallclock time was spent as follows:

- CPU 4.8%
- MPI 41.3%
- I/O 53.9%

Time spent running application code. High values are usually good. This is low, it may be worth improving I/O performance first.

Time spent in MPI calls. High values are usually bad. This is average, check the MPI breakdown for advice on reducing it.

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: 4.5%
- Vector numeric ops: 0.1%
- Memory accesses: 95.0%
- Other: 0.0%

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

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance. This suggests a significant load imbalance is causing synchronization overhead. You can investigate this further with an MPI profiler.

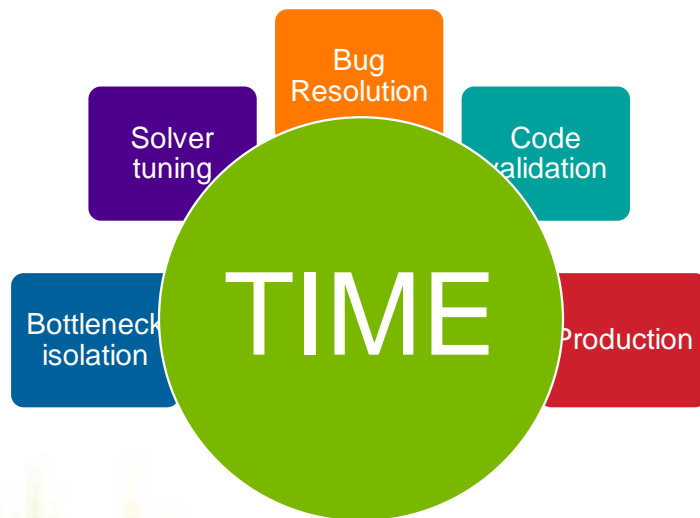
No time was spent in vectorized instructions. Check the compiler's vectorization advice to see why key loops could not be vectorized.

I/O

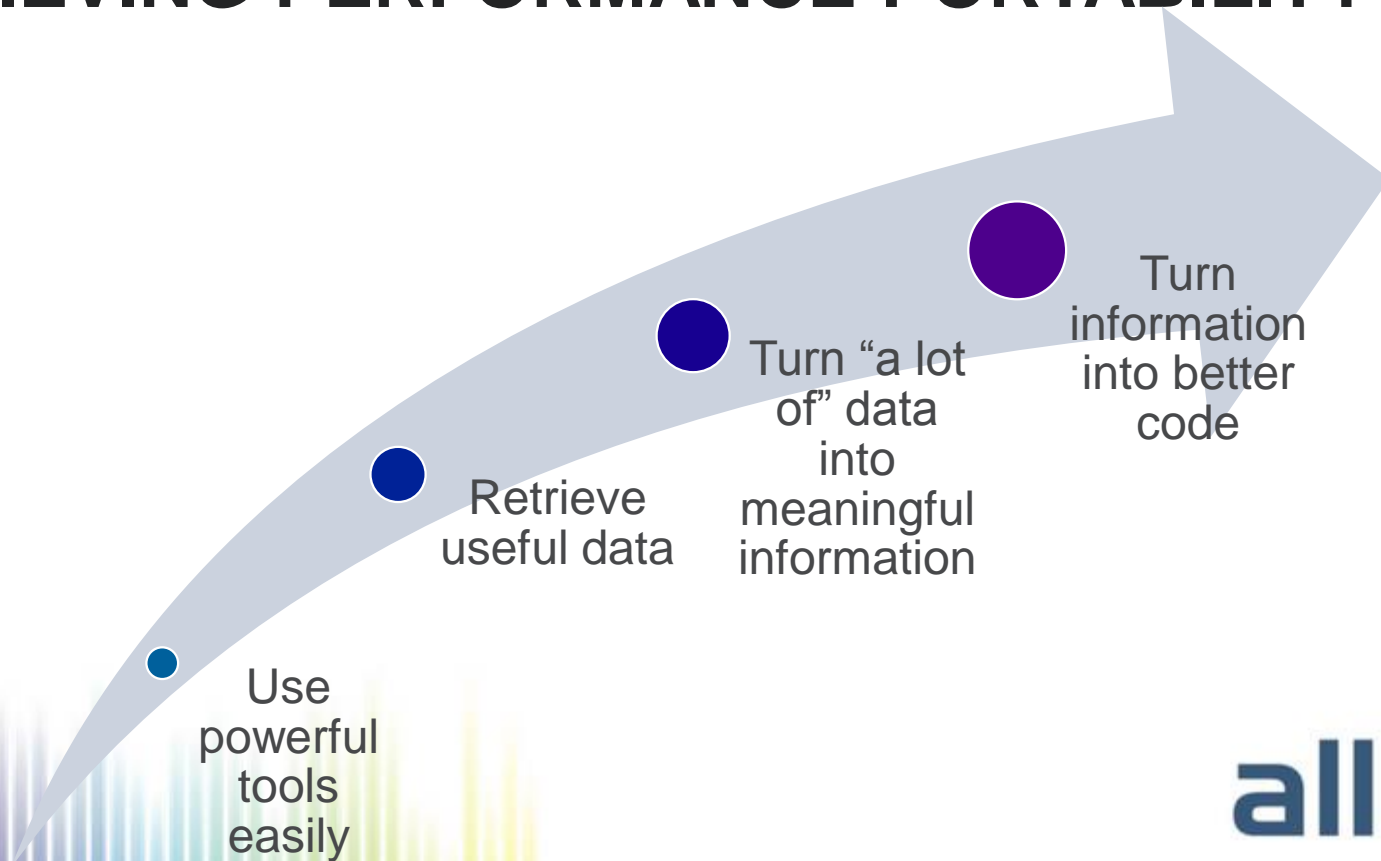
Memory

Alinea Performance Reports

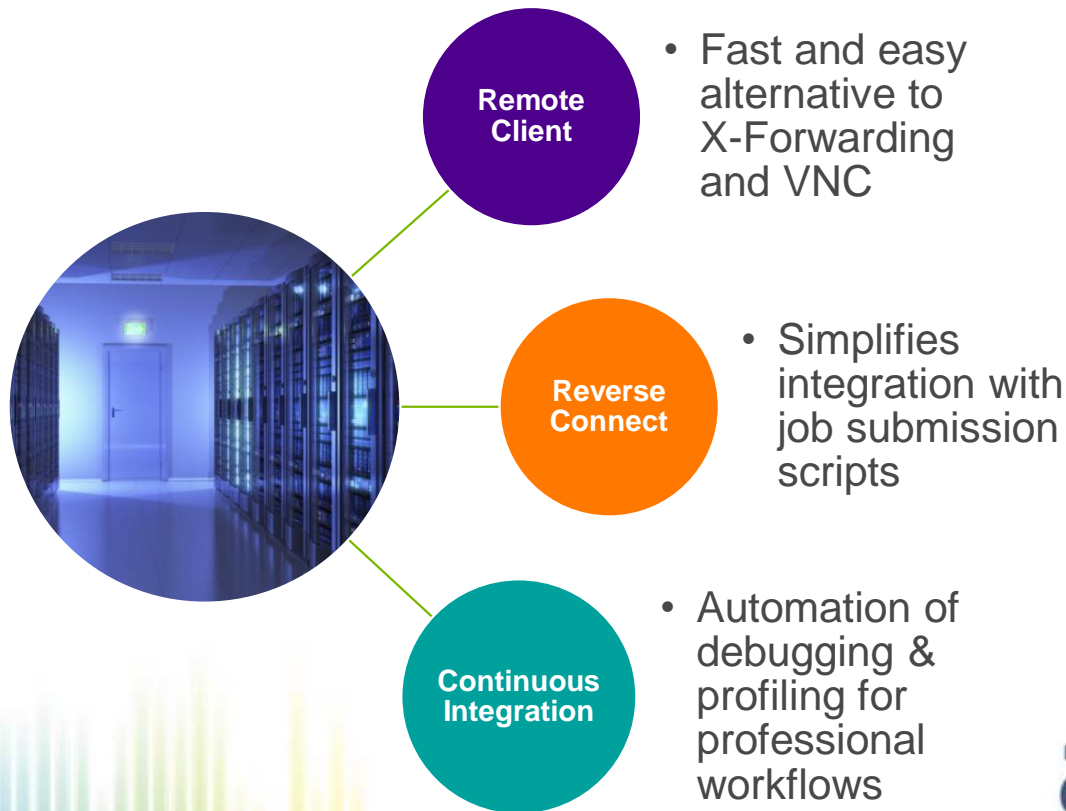
We do tools for a single reason:
help people save their time.



ACHIEVING PERFORMANCE PORTABILITY

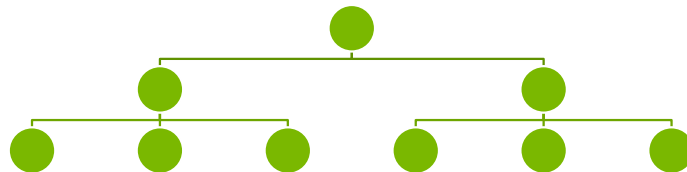


USING POWERFUL TOOLS MORE EASILY

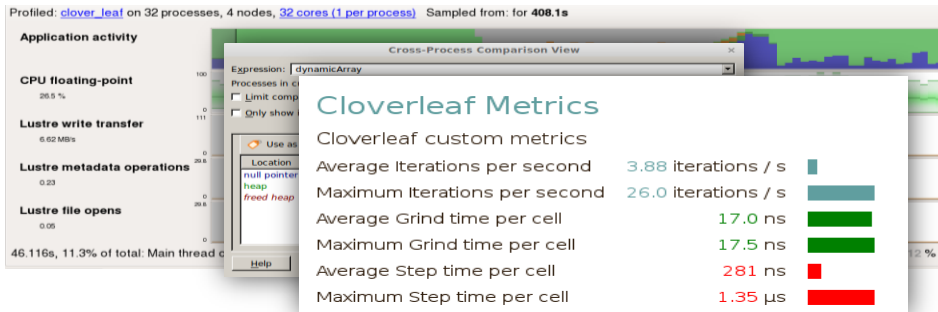


GENERATING USEFUL AND MEANINGFUL INFORMATION

Scalable &
Portable



Data collection

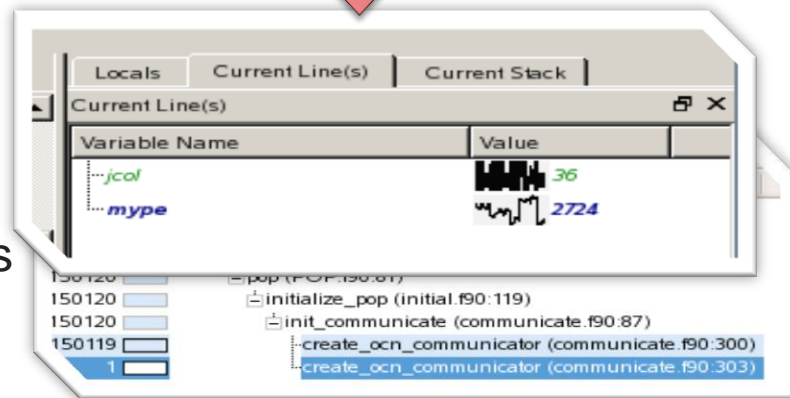
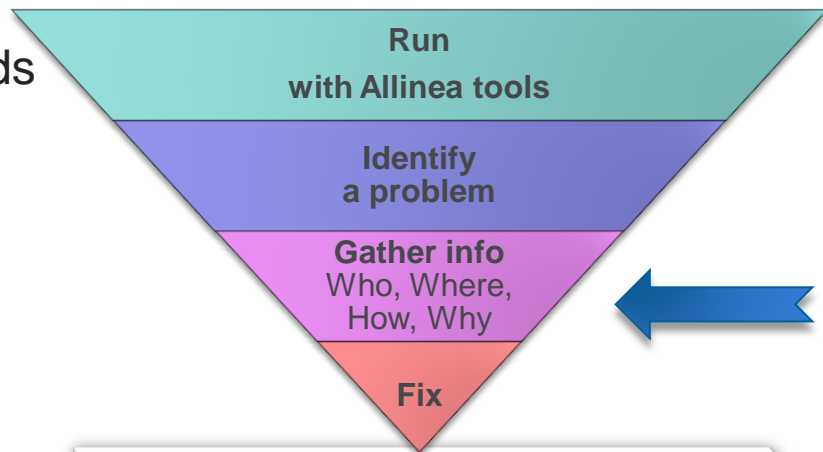


Data
processing



ALLINEA DDT – THE DEBUGGER

- Who had a rogue behavior ?
 - Merges stacks from processes and threads
- Where did it happen?
 - leaps to source
- How did it happen?
 - Diagnostic messages
 - Some faults evident instantly from source
- Why did it happen?
 - Unique “Smart Highlighting”
 - Sparklines comparing data across processes



PREPARING CODE FOR USE WITH DDT

- As with any debugger, code must be compiled with the debug flag typically `-g`
- It is recommended to turn off optimization flags i.e. `-O0`
- Leaving optimizations turned on can cause the compiler to *optimize out* some variables and even functions making it more difficult to debug

SEGMENTATION FAULT

- In this example, the application crashes with a segmentation error outside of DDT.

```
Terminal - rhulguin@ryanlinux:/media/sf_VM_share/Training_Codes/1_2_cstartmpi.f90
File Edit View Terminal Tabs Help
#2 0x7FC085B8E66F
#0 0x7FEF17094467
#1 0x7FEF17094AAE
#2 0x7FEF1637F66F
#3 0x4017EB in func3 at cstartmpi.f90:103
#4 0x4014B8 in cstartmpi at cstartmpi.f90:62
#0 0x7F585EDF6467
#1 0x7F585EDF6AAE
#2 0x7F585E0E166F
-----
mpirun noticed that process rank 12 with PID 18305 on node
remotemachine exited on signal 11 (Segmentation fault).
-----
[rhulguin@ryanlinux f90]$
```

- What happens when it runs under DDT?

SEGMENTATION FAULT IN DDT

The screenshot displays the DDT (Data Display Tool) interface. The main window shows the source code of a Fortran program named 'cstartmpi.f90'. The code defines a subroutine 'func3' that allocates a 2D array 'tab' and iterates over it. A segmentation fault occurred at line 103, where the program attempted to access an invalid memory address. The 'Program Stopped' dialog box provides details about the fault, including the signal 'SIGSEGV' and the reason: 'address not mapped to object (attempt to access invalid address)'. The dialog also offers options to 'Continue' or 'Pause' the program.

```
95
96  subroutine func3(my_rank)
97    integer, allocatable :: tab(:, :)
98    integer :: x, y, my_rank
99
100    allocate(tab(0:12, 0:12))
101    do i=0, 11
102      do while (y /= 12)
103        tab(x, y) = (x+1)*(y+1)
104        y = y+my_rank+1
105      end do
106    end do
107    deallocate(tab)
108  end subroutine func3
109
110  subroutine print_arg(arg)
111    character(128) :: arg
```

Variable Name	Value
tab	
x	4198128
y	0

Program Stopped

Processes 0-15:

Process stopped in func3 (cstartmpi.f90:103) with signal SIGSEGV (Segmentation fault).

Reason/Origin: address not mapped to object (attempt to access invalid address)
Your program will probably be terminated if you continue.
You can use the stack controls to see what the process was doing at the time.

Always show this window for signals

- DDT takes you to the exact line where Segmentation fault occurred, and you can pause and investigate

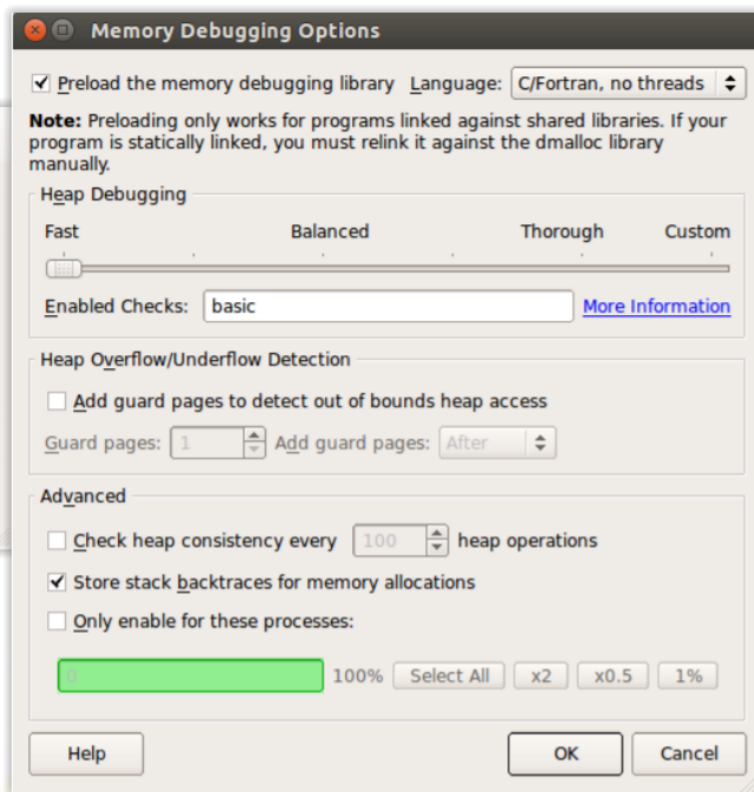
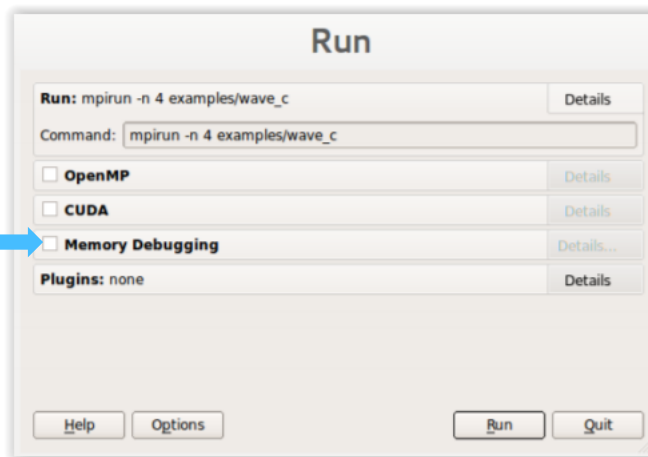
INVALID MEMORY ACCESS

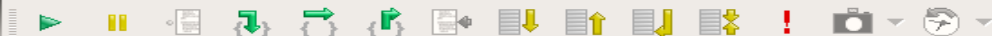
```
f cstartmpi.f90 x
96  subroutine func3(my_rank)
97      integer, allocatable :: tab(:, :)
98      integer :: x, y, my_rank
99
100     allocate(tab(0:12, 0:12))
101     do i=0, 11
102         do while (y /= 12)
103             tab(x, y) = (x+1)*(y+1)
104             y = y+my_rank+1
105         end do
106     end do
107     deallocate(tab)
108 end subroutine func3
109
110 subroutine main
111     character(12) :: rank_str
```

Variable Name	Value
tab	([0] = ([0] = -158...
x	4198128
y	0

- The array `tab` is a 13x13 array, but the application is trying to write a value to `tab(4198128,0)` which causes the segmentation fault.
- `i` is not used, and `x` and `y` are not initialized

ADVANCED MEMORY DEBUGGING





Current Group: All Focus on current: Group Process Thread Step Threads Together

All 24576 processes (0-24575) Paused: 17223 Playing: 7353 Finished: 0
 Currently selected: 260 (on nid00194, pid 9481, main thread IWP 9481)

Create Group

Project Files

Search (Ctrl+K)

- VolumeTrav
- wave.c
- weird.c
- WholeGeom
- Writer.cc
- wspace.c
- XdrFileWrite
- XdrMemRead
- XdrMemWrite
- XdrReader.c
- XdrWriter.cc
- XmlAbstract
- xyzpart.c
- External Code

```

551 ikvsortii(ntsamples, allpicks);
552
553
554 /* Select the final splitters. Set the boundaries to
555 for (i=1; i<npses; i++)
556     mypicks[i] = allpicks[i*ntsamples/npses];
557 mypicks[0].key = IDX_MIN;
558 mypicks[npses].key = IDX_MAX;
559
560
561 WCOREPOP; /* free allpicks */
562
563 STOPTIMER(ctrl, ctrl->AuxTmr2);
564 STARTTIMER(ctrl, ctrl->AuxTmr3);
565
    
```

Locals Current Line(s) Current Stack

Variable Name	Value
allpicks	0x2aab8055e010
i	2245
mypicks	0x2a6f8f0
npses	24575
ntsamples	1818550

Input/Output Breakpoints Watchpoints Stacks Tracepoints Tracepoint Output Logbook

Stacks

Processes	Threads	Function
17223	17223	main (main.cc:37)
17223	17223	SimulationMaster::SimulationMaster (SimulationMaster.cc:63)
17223	17223	SimulationMaster::Initialise (SimulationMaster.cc:154)
17223	17223	hemelb::geometry::GeometryReader::LoadAndDecompose (GeometryReader.cc:188)
17223	17223	hemelb::geometry::GeometryReader::OptimiseDomainDecomposition (GeometryReader.cc:188)
17223	17223	hemelb::geometry::decomposition::OptimisedDecomposition::OptimisedDecomposition
17223	17223	hemelb::geometry::decomposition::OptimisedDecomposition::CallParmetris (OptimisedDecomposition.cc:90)
17223	17223	ParMETIS_V3_PartGeomKway (gkmetis.c:90)
17223	17223	libparmetis_Coordinate_Partition (xyzpart.c:58)
17223	17223	libparmetis_PseudoSampleSort (xyzpart.c:556)

Type: none selected

Evaluate

Expression	Value
i*ntsamples	-212322546

Type: int
 Range: from -2147259746 to -12282046
 49/17223 processes equal

Debugging at Scale Requires Powerful Visual Representations

The screenshot shows a debugger interface with the following components:

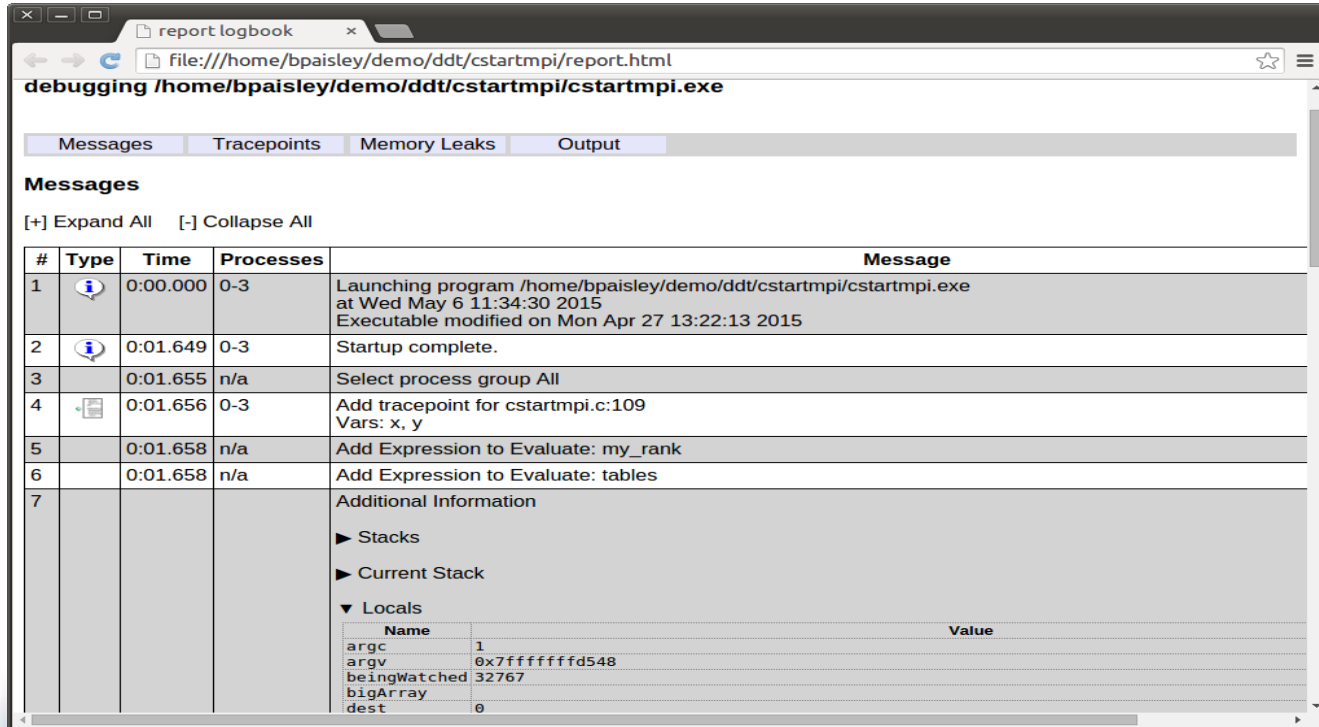
- Threads:** Four threads are visible, with thread 3 selected.
- Project Files:** A tree view showing the application code structure, including 'Sources' and 'External Code'.
- Code Editor:** Displays the source code for 'wave_openmp.c'. A breakpoint is set at line 227, which is highlighted in red. The code includes a loop for processing points and a swap operation for arrays.
- Locals:** A table showing the current line's local variables:

Variable Name	Value
oldval	0x7ffff4b7a010
values	0x7ffff4b7a010
- Stacks:** A table showing the call stack for the selected thread:

Threads	Function
1	main (wave_openmp.c:354)
1	update (wave_openmp.c:227)
3	omp_in_final
- Evaluate:** A table for evaluating expressions:

Expression	Value
newval	0x7ffff4b7a010
oldval	0x7ffff4b7a010
values	0x7ffff4b7a010

ENABLE LARGE SCALE DEBUGGING AND REGRESSION TESTING WITH OFFLINE DEBUGGING



The screenshot shows a web browser window titled "report logbook" with the address bar displaying "file:///home/bpaisley/demo/ddt/cstartmpi/report.html". The main content area is titled "debugging /home/bpaisley/demo/ddt/cstartmpi/cstartmpi.exe" and features a navigation bar with tabs for "Messages", "Tracepoints", "Memory Leaks", and "Output". The "Messages" tab is active, showing a list of messages with columns for "#", "Type", "Time", "Processes", and "Message".

Messages

[+] Expand All [-] Collapse All

#	Type	Time	Processes	Message												
1		0:00.000	0-3	Launching program /home/bpaisley/demo/ddt/cstartmpi/cstartmpi.exe at Wed May 6 11:34:30 2015 Executable modified on Mon Apr 27 13:22:13 2015												
2		0:01.649	0-3	Startup complete.												
3		0:01.655	n/a	Select process group All												
4		0:01.656	0-3	Add tracepoint for cstartmpi.c:109 Vars: x, y												
5		0:01.658	n/a	Add Expression to Evaluate: my_rank												
6		0:01.658	n/a	Add Expression to Evaluate: tables												
7				Additional Information ▶ Stacks ▶ Current Stack ▼ Locals <table border="1"><thead><tr><th>Name</th><th>Value</th></tr></thead><tbody><tr><td>argc</td><td>1</td></tr><tr><td>argv</td><td>0x7fffffff548</td></tr><tr><td>beingWatched</td><td>32767</td></tr><tr><td>bigArray</td><td></td></tr><tr><td>dest</td><td>0</td></tr></tbody></table>	Name	Value	argc	1	argv	0x7fffffff548	beingWatched	32767	bigArray		dest	0
Name	Value															
argc	1															
argv	0x7fffffff548															
beingWatched	32767															
bigArray																
dest	0															

FIVE GREAT THINGS TO TRY WITH ALLINEA DDT

Input/Output	Breakpoints	Watchpoints	Tracepoints	Tracepoint Output	Stacks (All)
Tracepoint Output					
Tracepoint	Process	Values logged			
vhone 90-65	916, rank0 12,14,17,23,23,12	mype	210-3527	jscl	2-4B mod pay
vhone 90-61	940, rank0 12,14,17,23,23,12	ks	1	lmax	par
vhone 90-65	940, rank0 12,14,17,23,23,12	mype	210-3527	jscl	2-4B mod pay
vhone 90-61	950, rank0 12,14,17,23,23,12	ks	1	lmax	par
vhone 90-65	918, rank0 12,14,17,23,23,12	mype	210-3527	jscl	2-4B mod pay
vhone 90-61	986, rank0 12,14,17,23,23,12	ks	1	lmax	par
vhone 90-65	984, ra 12,14				
vhone 90-61	980, ra 17,14				

The scalable print alternative

```

for (i = 0; i < SIZE M; i++)
  for (j = 0; j < SIZE O; j++)
    C[i][j] = 0;

for (i = 0; i < SIZE M; i++)
  for (j = 0; j < SIZE N; j++)
    for (k = 0; k < SIZE O; k++)
      C[i][j] += A[i][k] * B[k][j];
  
```



Stop on variable change

```

hello.c x
This file is newer than your program. Please recompile then restart yo
43 else
44     test=-1;
45 }
46
47 void func3()
48 {
49     void* i = (void*) 1;
50     while(i++ || !i)
51         free((void*)i);
52
53     portability 'i' is of type 'void *'. When using void pointers in calcula
54     Left click to add a breakpoint on line 50
55 {
56     ty;
57     ty;
58     in
  
```

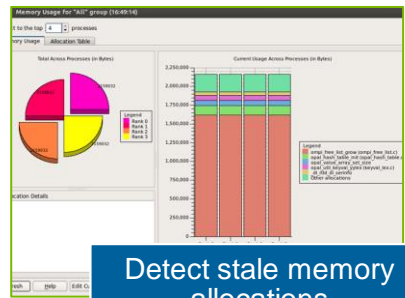
Static analysis warnings on code errors

```

&& !strcmp(argv[i], "crash")) {
0;
5; *(char **)argv[i]);
ll se

Program Stopped
Processes 0-3:
Memory error detected in main (hello.c:118):
null pointer dereference or unaligned memory access
Note: the latter may sometimes occur spuriously if gval
enabled
Tip: Use the stack list and the local variables to explore
current state and identify the source of the error.
r, "I
= 1;
ist.s
= 0;
  
```

Detect read/write beyond array bounds



Detect stale memory allocations

THE UNCOMFORTABLE TRUTH ABOUT APPLICATIONS



allinea
DDT

allinea
MAP

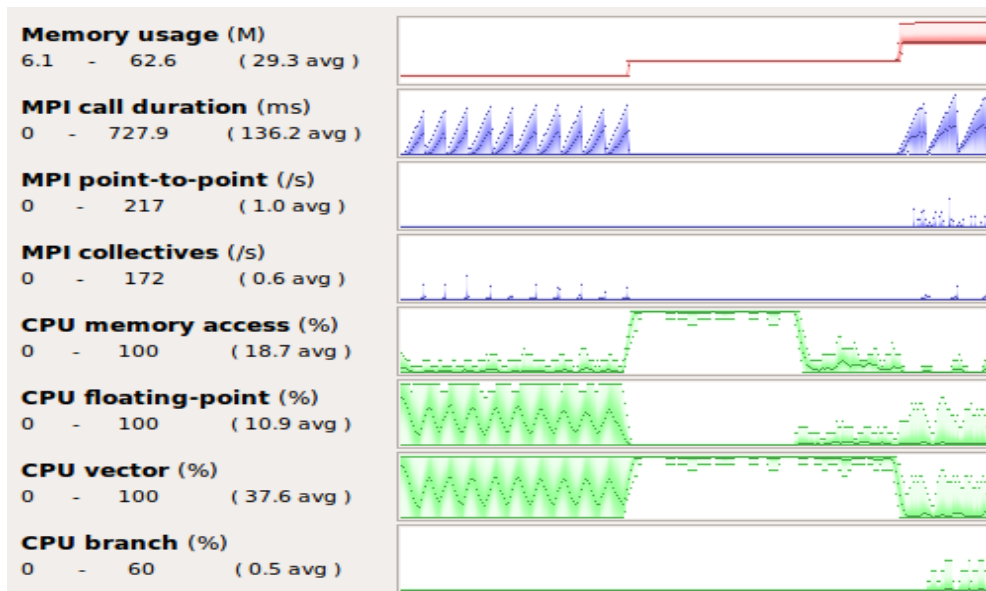
Optimized for modern CPUs



Hey, at least it compiles



GLEAN DEEP INSIGHT FROM OUR SOURCE-LEVEL PROFILER



Track memory usage across the entire application over time

Spot MPI and OpenMP imbalance and overhead

Optimize CPU memory and vectorization in loops

Detect and diagnose I/O bottlenecks at real scale

ALLINEA MAP – THE PROFILER



Small data files



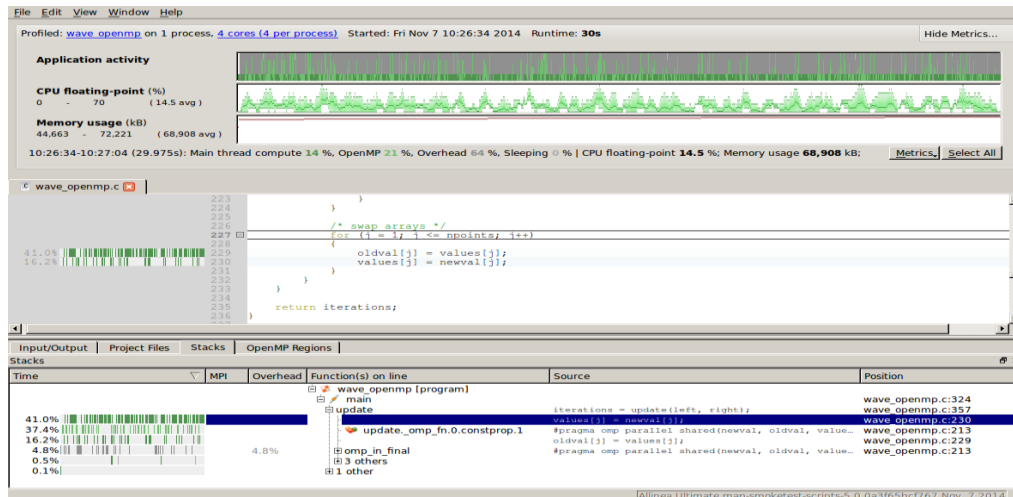
<5% slowdown



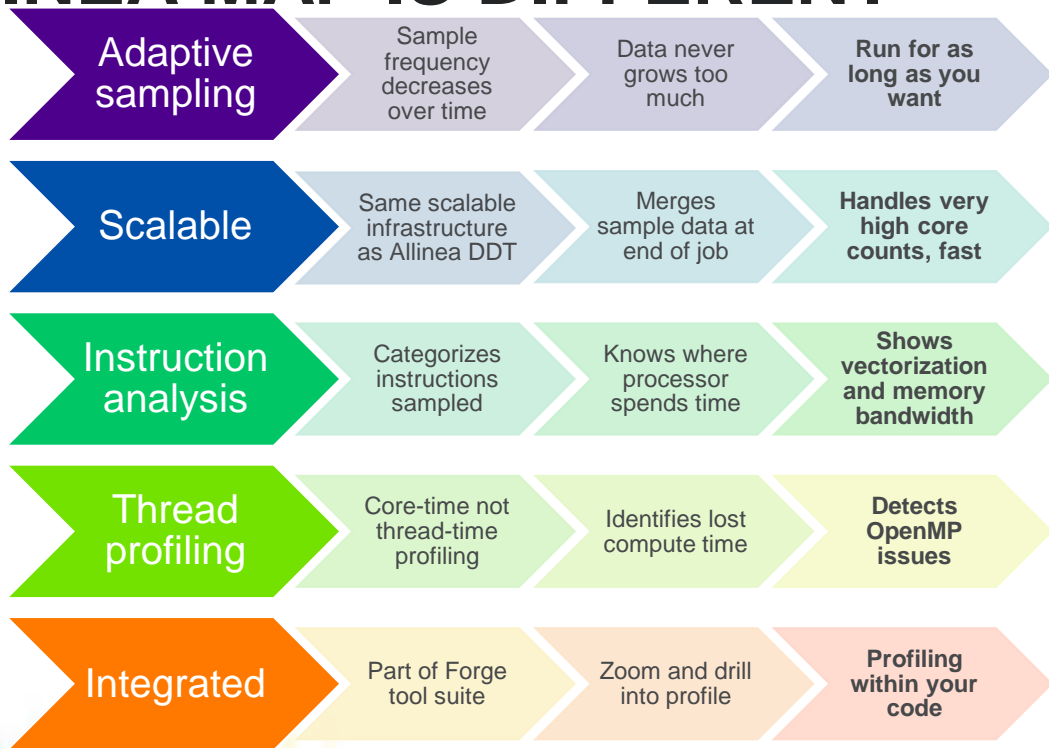
No instrumentation



No recompilation



HOW ALLINEA MAP IS DIFFERENT



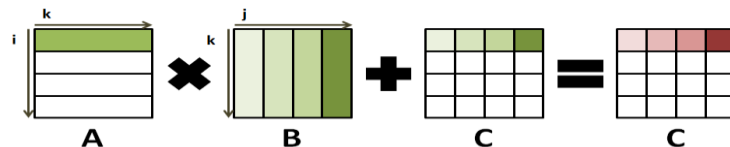
PREPARING CODE FOR USE WITH MAP

- To see the source code, the application should be compiled with the debug flag typically `-g`
- It is recommended to *always* keep optimization flags on when profiling

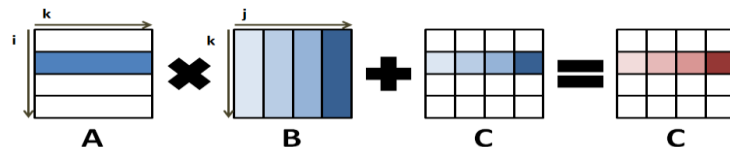
MATRIX MULTIPLICATION EXAMPLE

$$C = A \times B + C$$

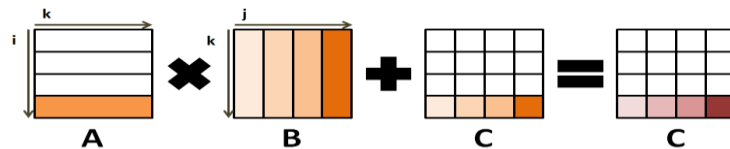
Master process



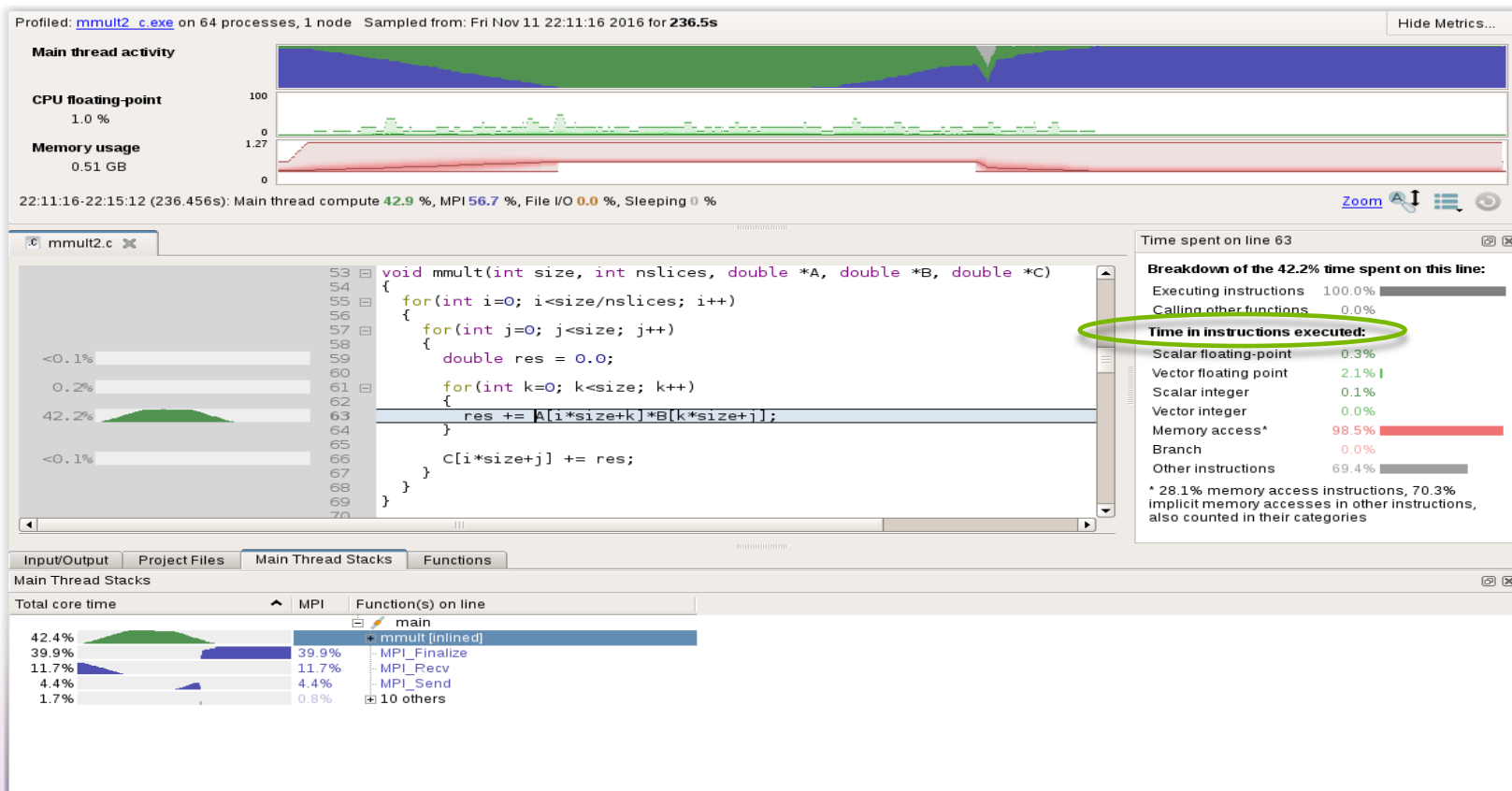
Slave process 1



Slave process n-1



MATRIX MULTIPLICATION PROFILE



ENABLING VECTORIZATION

The compiler is unable to vectorize efficiently because of the following line in C:

```
res += A[i*size+k]*B[k*size+j];
```

and in F90:

```
res=A(i*size+k)*B(k*size+j)+res
```

rewrite mmult to have

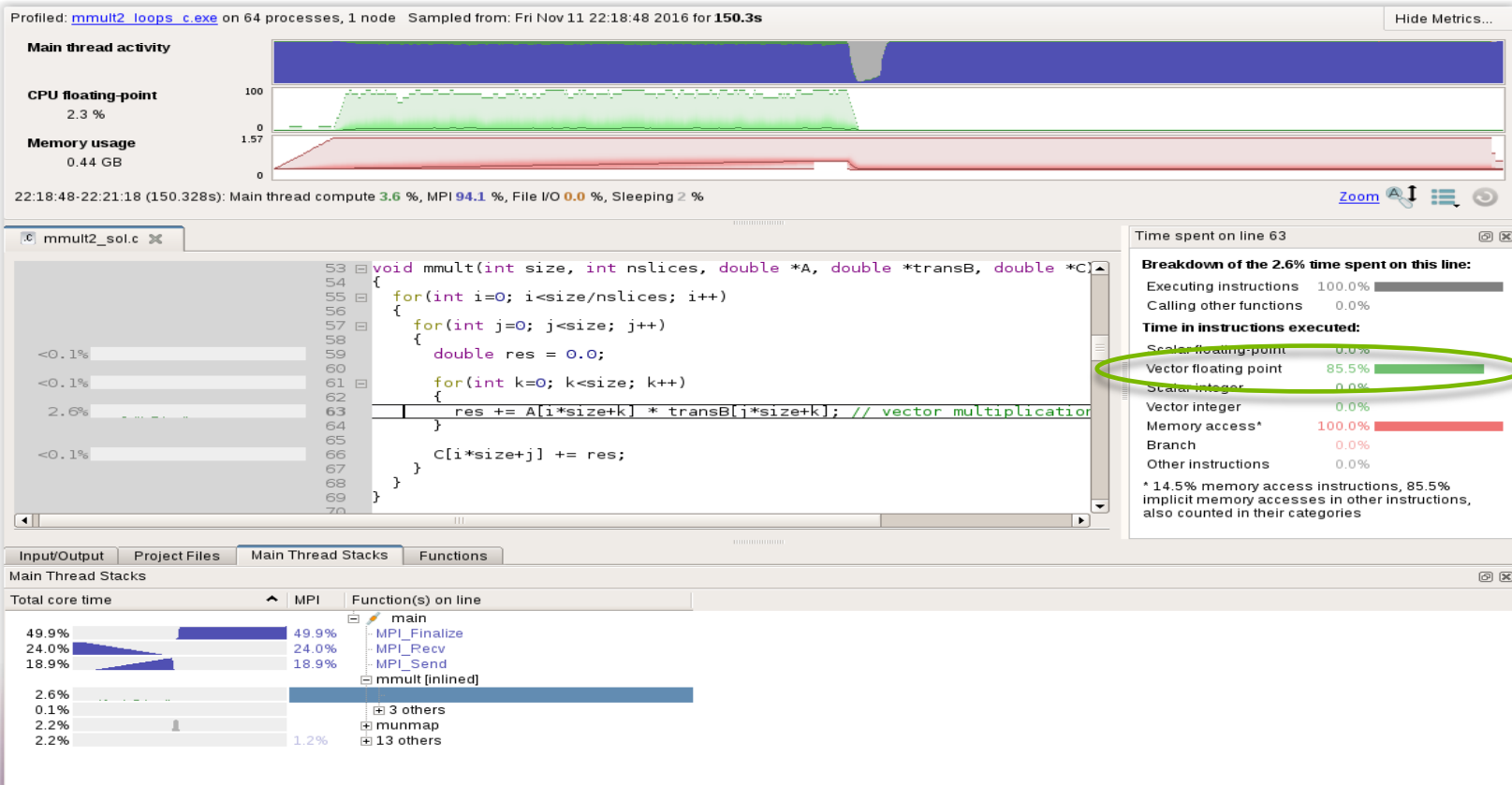
in C:

```
res += A[i*size+k]*transB[j*size+k];
```

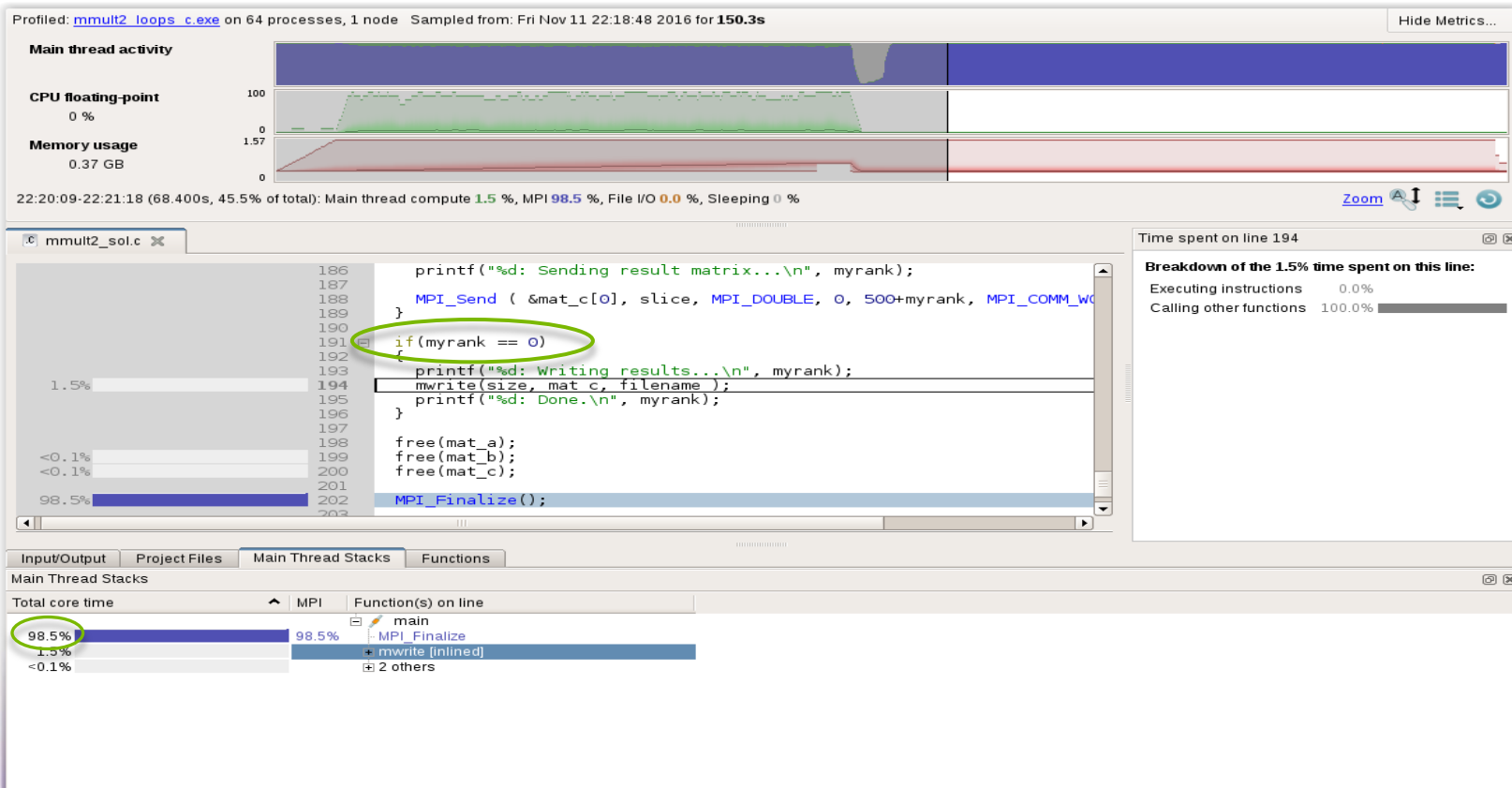
and in F90:

```
res=A(i*size+k)*transB(j*size+k)+res
```

IMPROVING DATA LAYOUT AND ACCESS PATTERN



SERIAL BOTTLENECK



INEFFICIENT I/O

```
if(myrank == 0)
{
    printf("%d: Receiving result matrix...\n", myrank);
    [...]
}
else
{
    printf("%d: Sending result matrix...\n", myrank);
    [...]
}

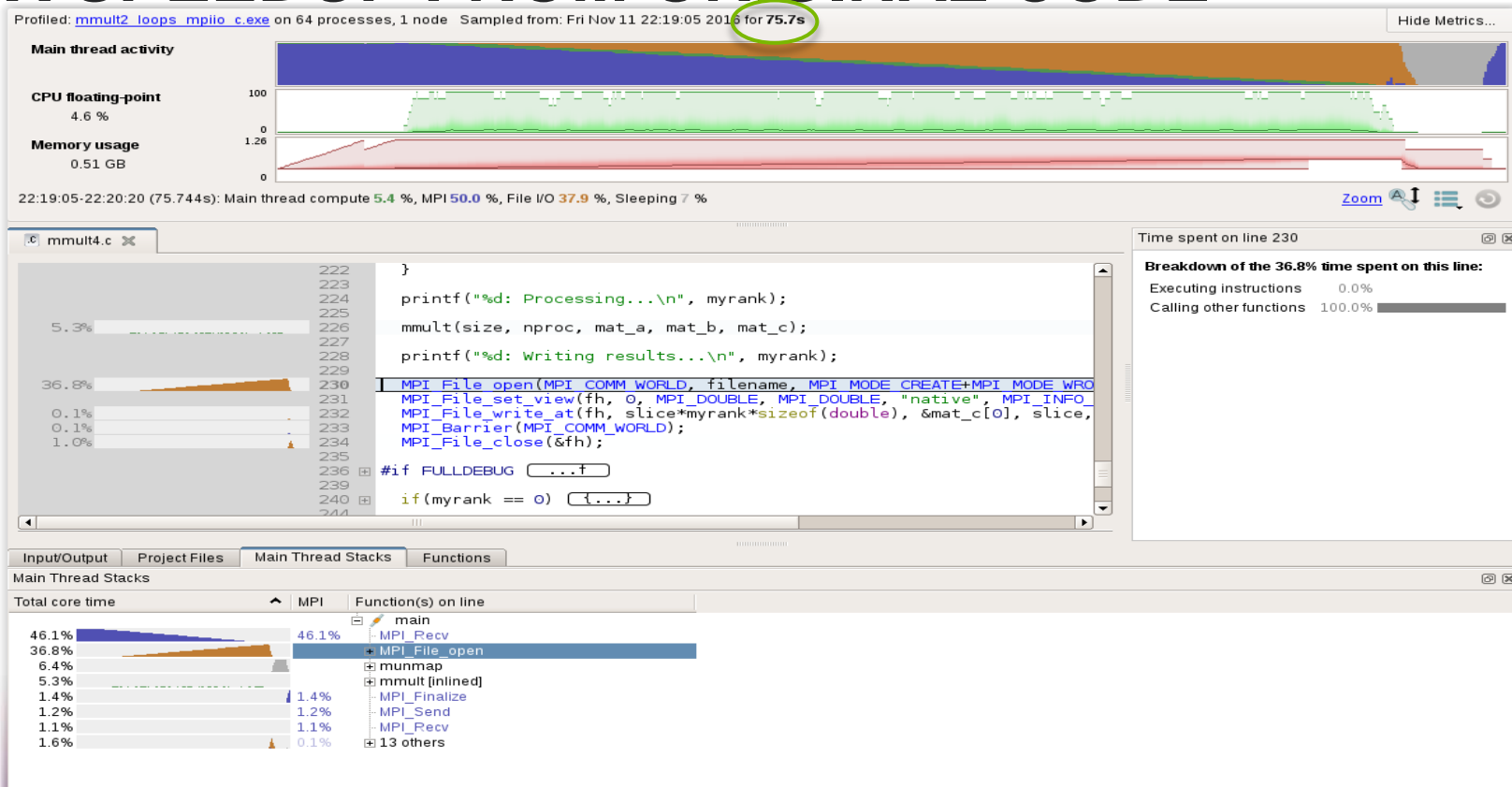
if(myrank == 0)
{
    printf("%d: Writing results...\n", myrank);
    mwrite(size, mat_c, filename);
}
```

IMPROVE SCALABILITY OF I/O ROUTINES

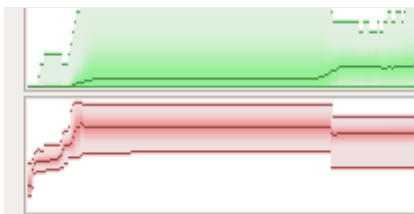
```
printf("%d: Writing results...\n", myrank);
```

```
MPI_File_open(MPI_COMM_WORLD, filename,  
MPI_MODE_CREATE+MPI_MODE_WRONLY, MPI_INFO_NULL, &fh);  
MPI_File_set_view(fh, 0, MPI_DOUBLE, MPI_DOUBLE, "native",  
MPI_INFO_NULL);  
MPI_File_write_at(fh, slice*myrank*sizeof(double), &mat_c[0],  
slice, MPI_DOUBLE, &st);  
MPI_Barrier(MPI_COMM_WORLD);  
MPI_File_close(&fh);
```

3X SPEEDUP FROM ORIGINAL CODE



SIX GREAT THINGS TO TRY WITH ALLINEA MAP

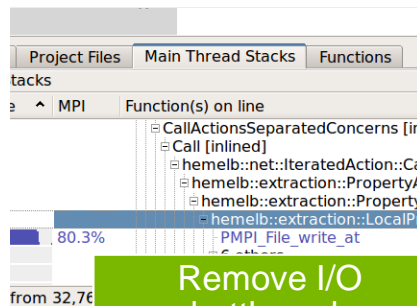


compute 76 %. MPI 24 %. File

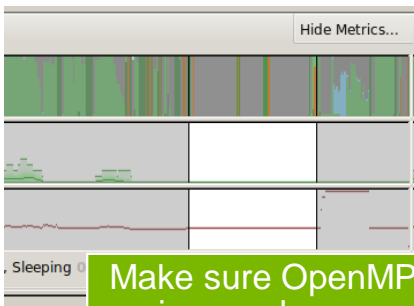
Find the peak memory use

```
30 ! late to the party
31 do j=1,20*nprocs; a
32 end if
33
34 if (pe /= 0) then
35 call MPI_SEND(a, si
36 else
37 do from=1,nprocs-1
38 call MPI_RECV(b,
39 do j=1,50; b=sqrt
40 print *, "Answer f
41 end do
42 end if
43 end do
44 call MPI_BARRIER(MPI CO
```

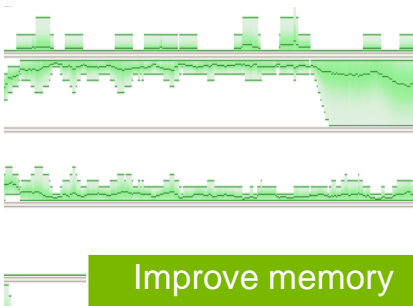
Fix an MPI imbalance



Remove I/O bottleneck



Make sure OpenMP regions make sense



Improve memory access

```
size, nproc, mat a
A[i*size+k]*B[k*s
```

Restructure for vectorization

PUBLIC ROADMAP: 7.1 (JULY 2017)



Debugging

- Fast C++ mem debug
- Improved STL support

Profiling

- Subset of ranks
- GPU source-code
- Metric histogram preview

Platform

- IBM Spectrum PMIx
- OpenPOWER
- ARMv8

PUBLIC ROADMAP: 8.0 (NOVEMBER 2017)



Debugging

- Python support
- AddressSanitizer plugin

Profiling

- Python support
- Improved performance on Intel KNL

Platform

- CUDA 9
- OpenPOWER 9

GETTING STARTED ON MIRA/COOLEY/THETA

- (Optional) Install local client on your laptop
 - www.allinea.com/products/forge/downloads
 - Linux – installs full set of tools
 - Windows, Mac – just a remote client to the remote system
 - Run the installation and software
 - “Connect to remote host”
 - Hostname:
 - username@mira.alcf.anl.gov
 - username@cooley.alcf.anl.gov
 - username@theta.alcf.anl.gov

 - Remote installation directory: /soft/debuggers/ddt

 - Click Test
- You are now ready to debug on Mira/Vesta/Cetus – or debug and profile on Cooley/Theta

ENABLING ALLINEA TOOLS

- On the machines that use Softenv, modify ~/.soft to include +ddt
- On the machines that use modules, load the forge module
module load forge/7.0

STATIC LINKING EXTRA STEPS

- To enable advanced memory debugging features, you must link explicitly against our memory libraries
- Simply add the link flags to your Makefile, or however appropriate
lflags = -L/soft/debuggers/ddt/lib/64 -Wl,--undefined=malloc -ldmalloc -Wl,--allow-multiple-definition
- In order to profile, static profiler libraries must be created with the command
make-profiler-libraries --libtype=static

Instructions to link the libraries will be provided after running the above command

SAMPLE USAGE COMMANDS

- Mira

```
ddt --connect --mpiargs="--block $COBALT_PARTNAME" --processes=128 -procs-  
per-node=1 ./myProgram.exe
```

- Theta

```
rpn=64
```

```
ddt aprun -n $((COBALT_JOBSIZE*rpn)) -N $rpn -d $depth -j 1 -cc depth  
./myProgram.exe
```

```
ALLINEA_OFFLINE_LICENCE_TIMEOUT=1000000 map aprun -n  
$((COBALT_JOBSIZE*rpn)) -N $rpn -d $depth -j 1 -cc depth ./myProgram.exe
```

QUESTIONS?

THANK YOU

www.anl.gov