Debugging and Profiling with Arm Forge

Ryan Hulguin
Applications Engineer
ryan.hulguin@arm.com

ALCF Simulation, Data, and Learning Workshop
February 28, 2018

Argonne NATIONAL LABORATORY

Agenda

- Introduction to Arm Forge
- Basic Arm DDT Example
- Memory Debugging Arm DDT Example
- Offline Mode DDT Example
- Basic Arm MAP Profiling Example
- Using Arm MAP to Improve Performance
- Questions
- Hands on Examples

Arm in HPC Tools

arm Allinea Studio

For Arm

Arm Forge

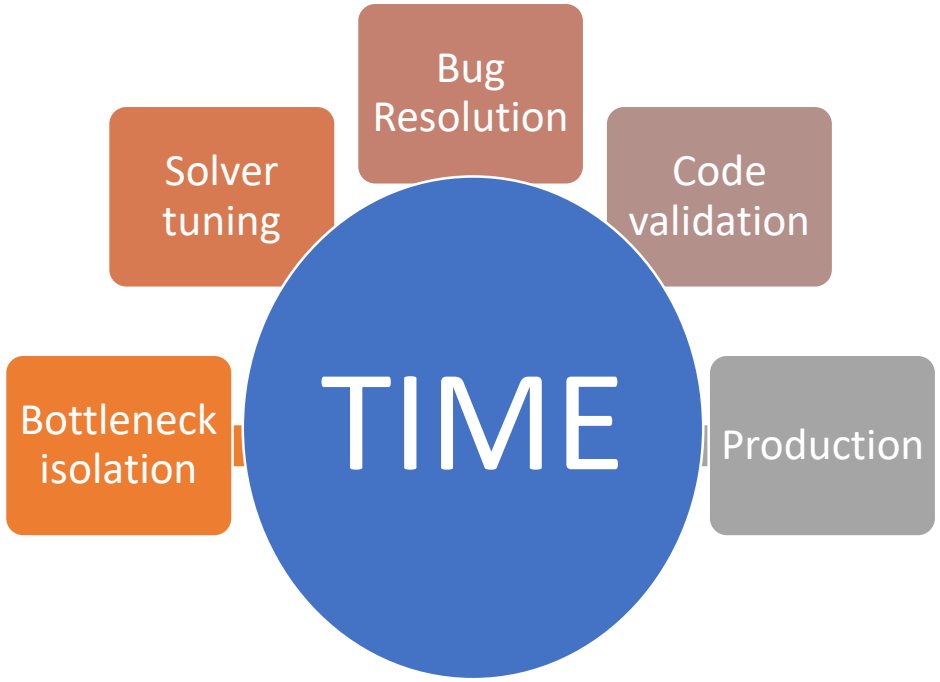
Arm MAP

Arm DDT

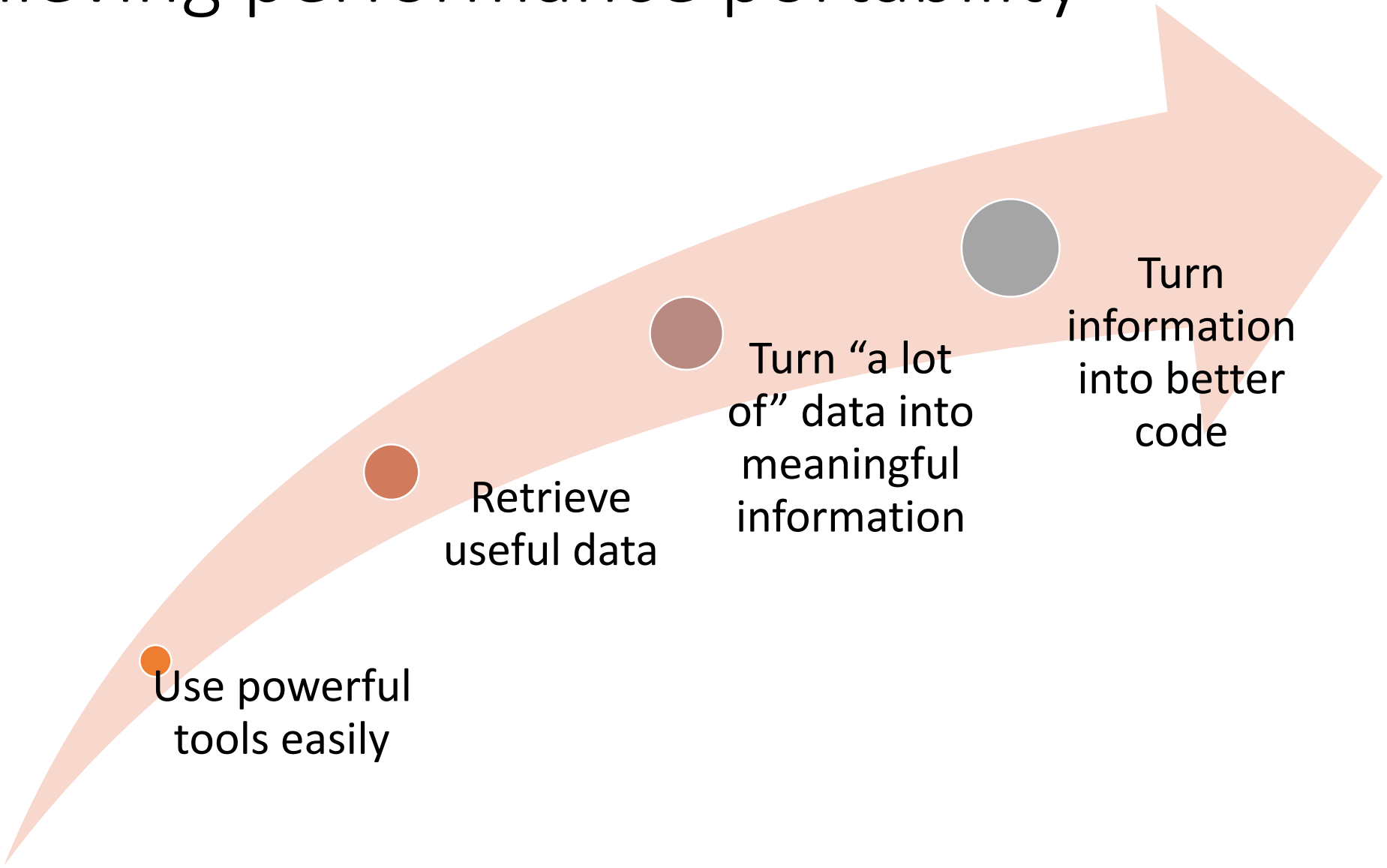
Arm Performance Reports

For Cross-platform

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



Achieving performance portability



Using powerful tools more easily

Remote Client

- Fast and easy alternative to X-Forwarding and VNC

Reverse Connect

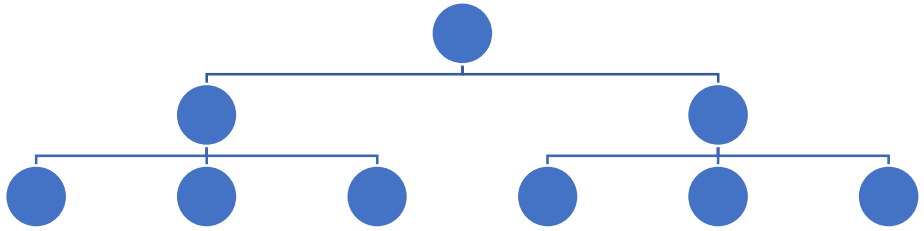
- Simplifies integration with job submission scripts

Continuous Integration

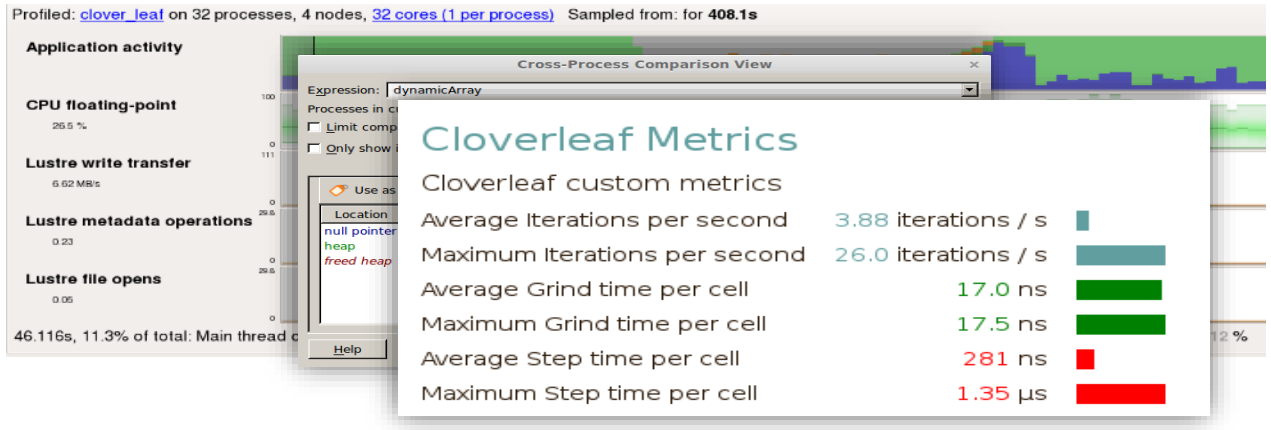
- Automation of debugging & profiling for professional workflows

Generating useful and meaningful information

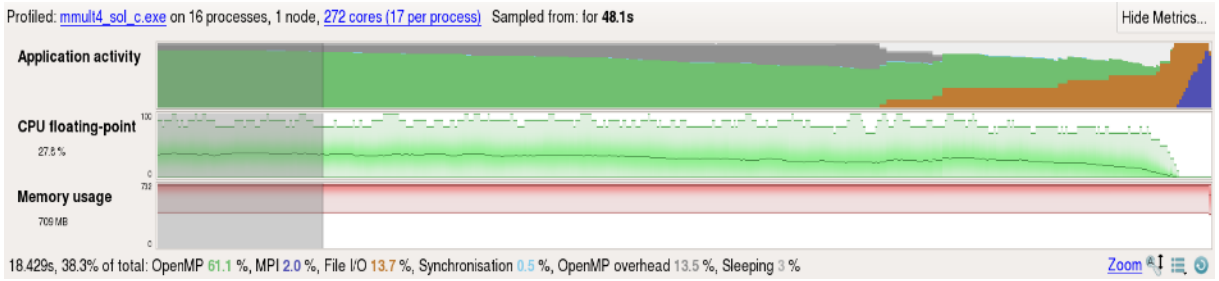
Scalable & Portable



Data collection

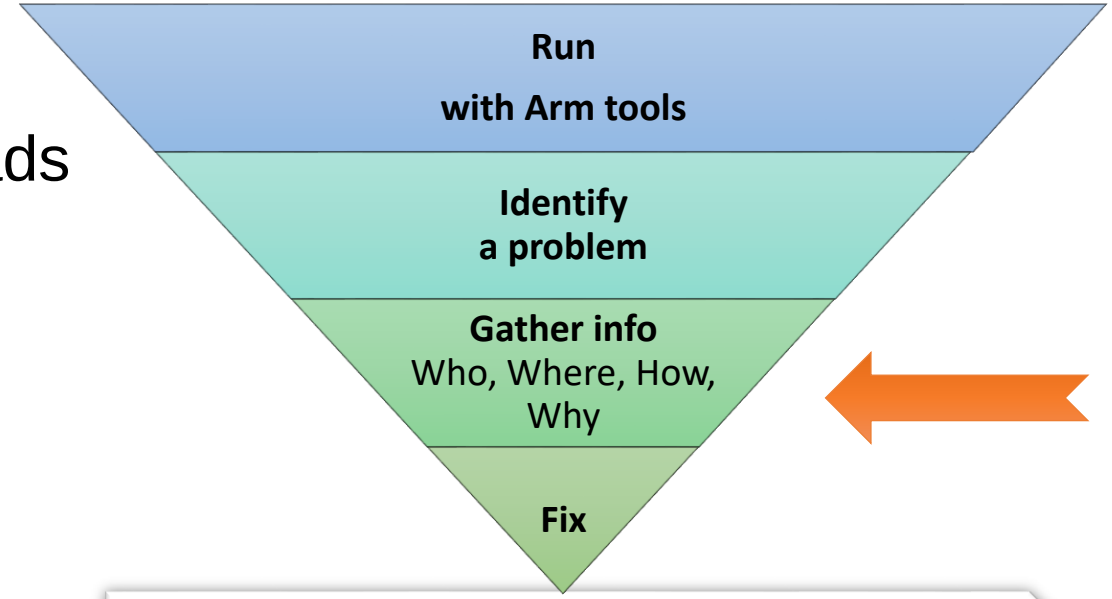


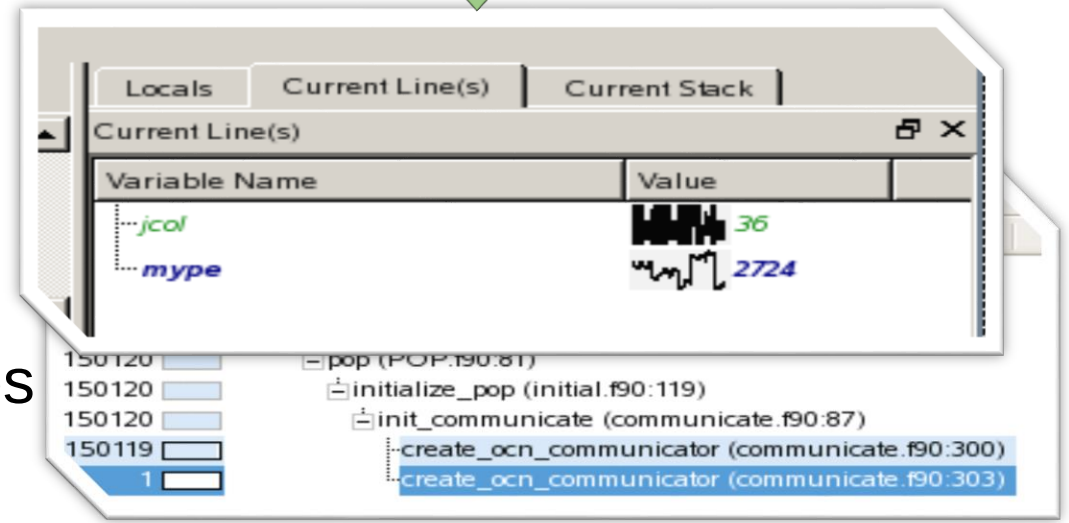
Data processing



Arm 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

Basic Arm DDT Example

Argonne NATIONAL LABORATORY

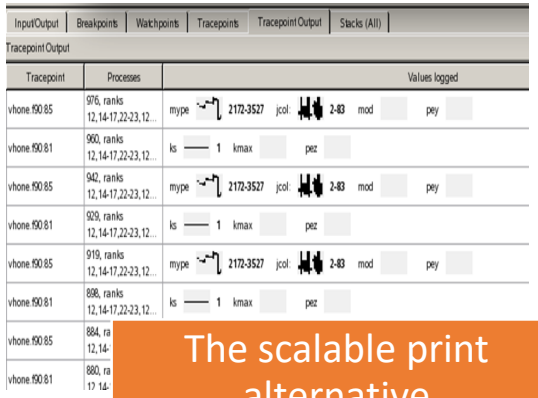
Memory Debugging Arm DDT Example

Argonne NATIONAL LABORATORY

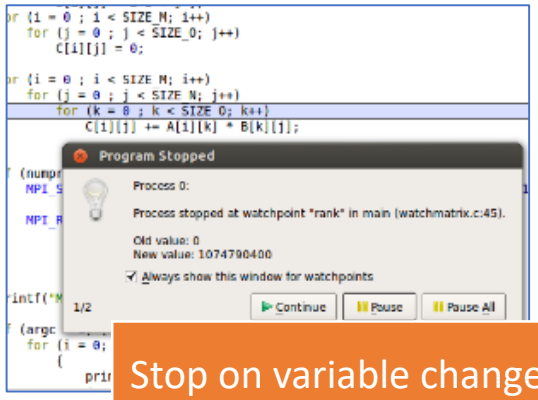
Offline Mode DDT Examples

Argonne NATIONAL LABORATORY

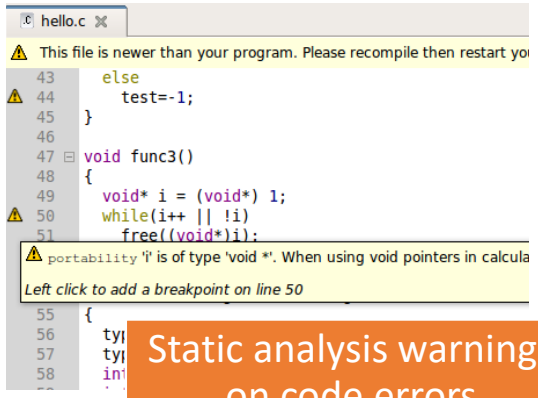
Five great things to try with Arm DDT



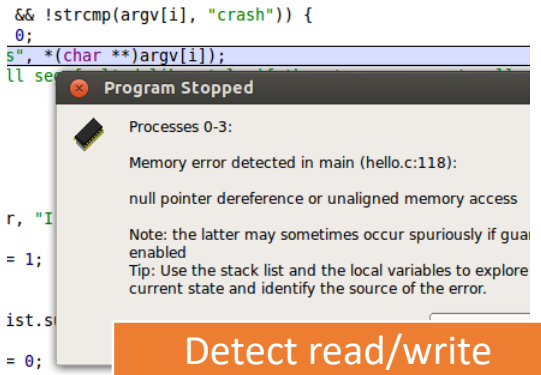
The scalable print alternative



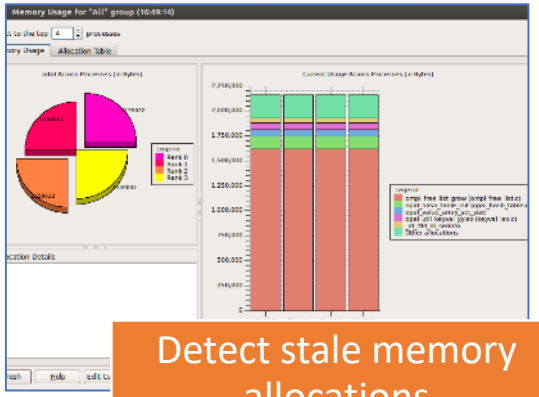
Stop on variable change



Static analysis warnings on code errors

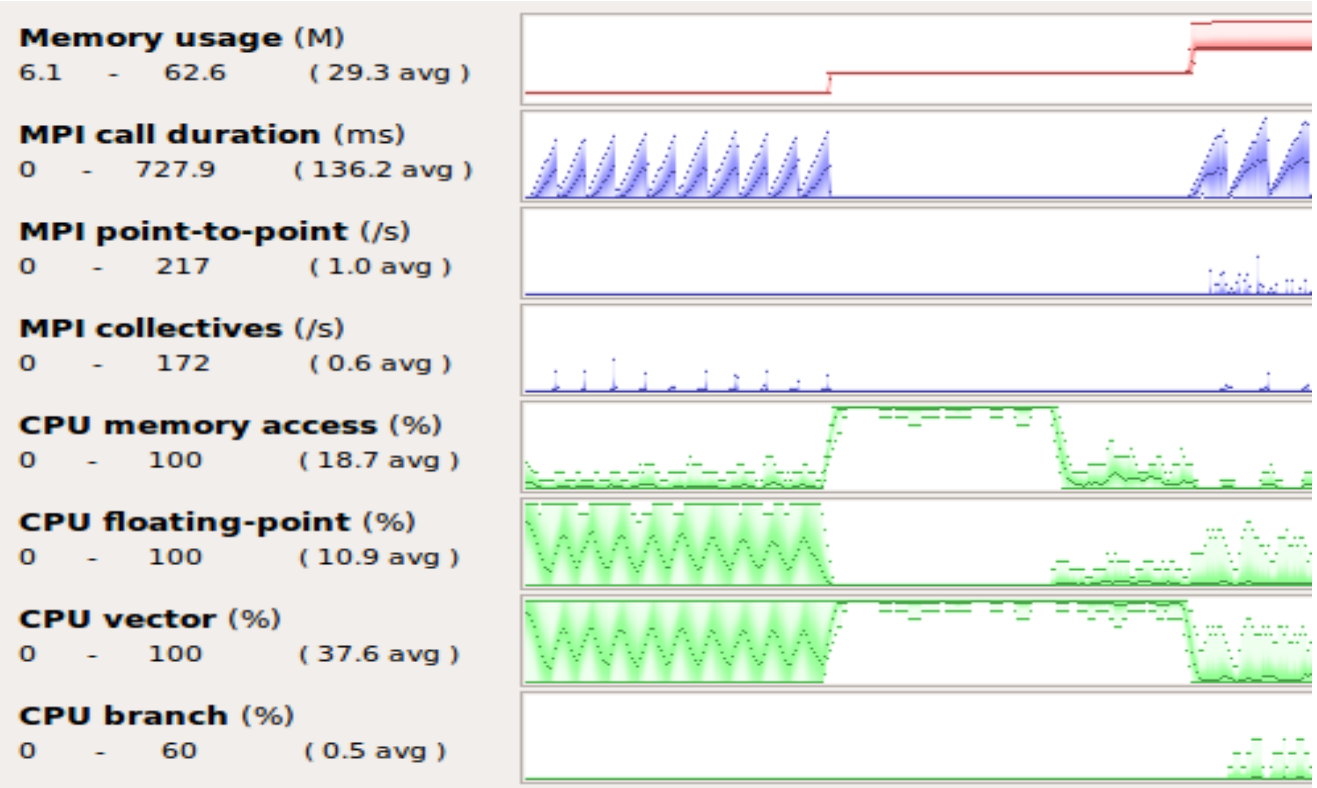


Detect read/write beyond array bounds



Detect stale memory allocations

Glean Deep Insight from our Source-Level Profiler



Track memory usage across the entire application over time

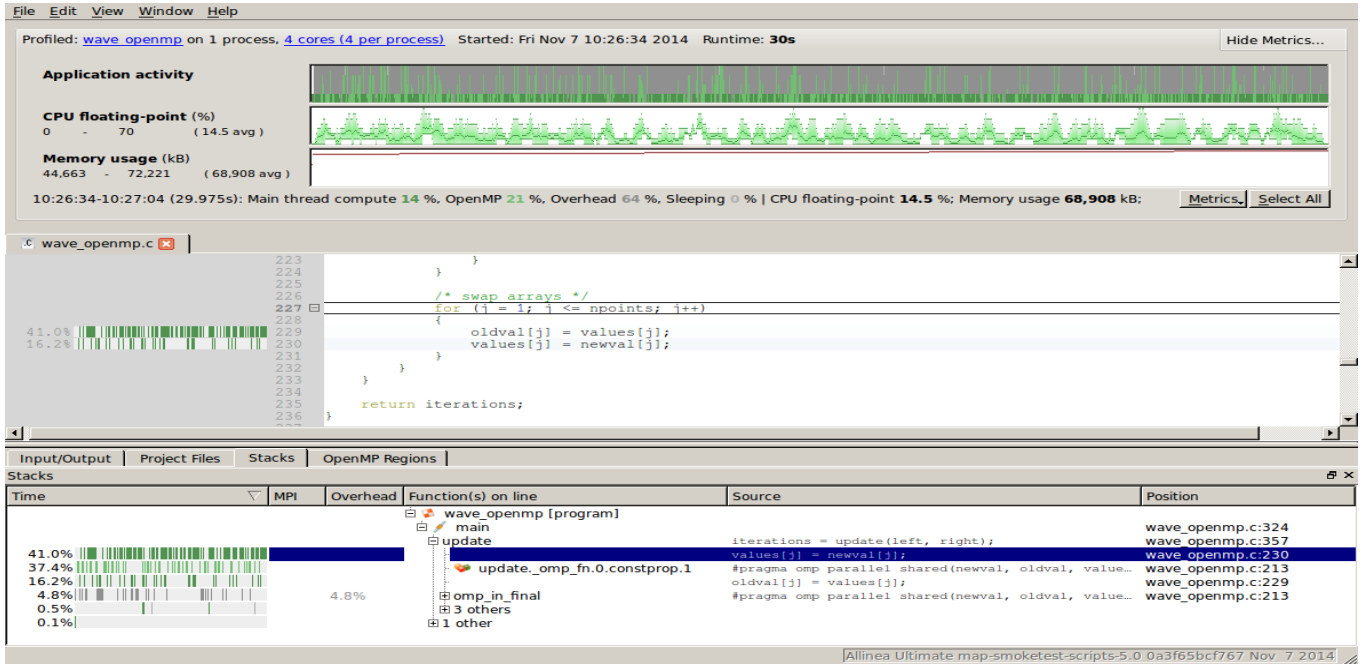
Spot MPI and OpenMP imbalance and overhead

Optimize CPU memory and vectorization in loops

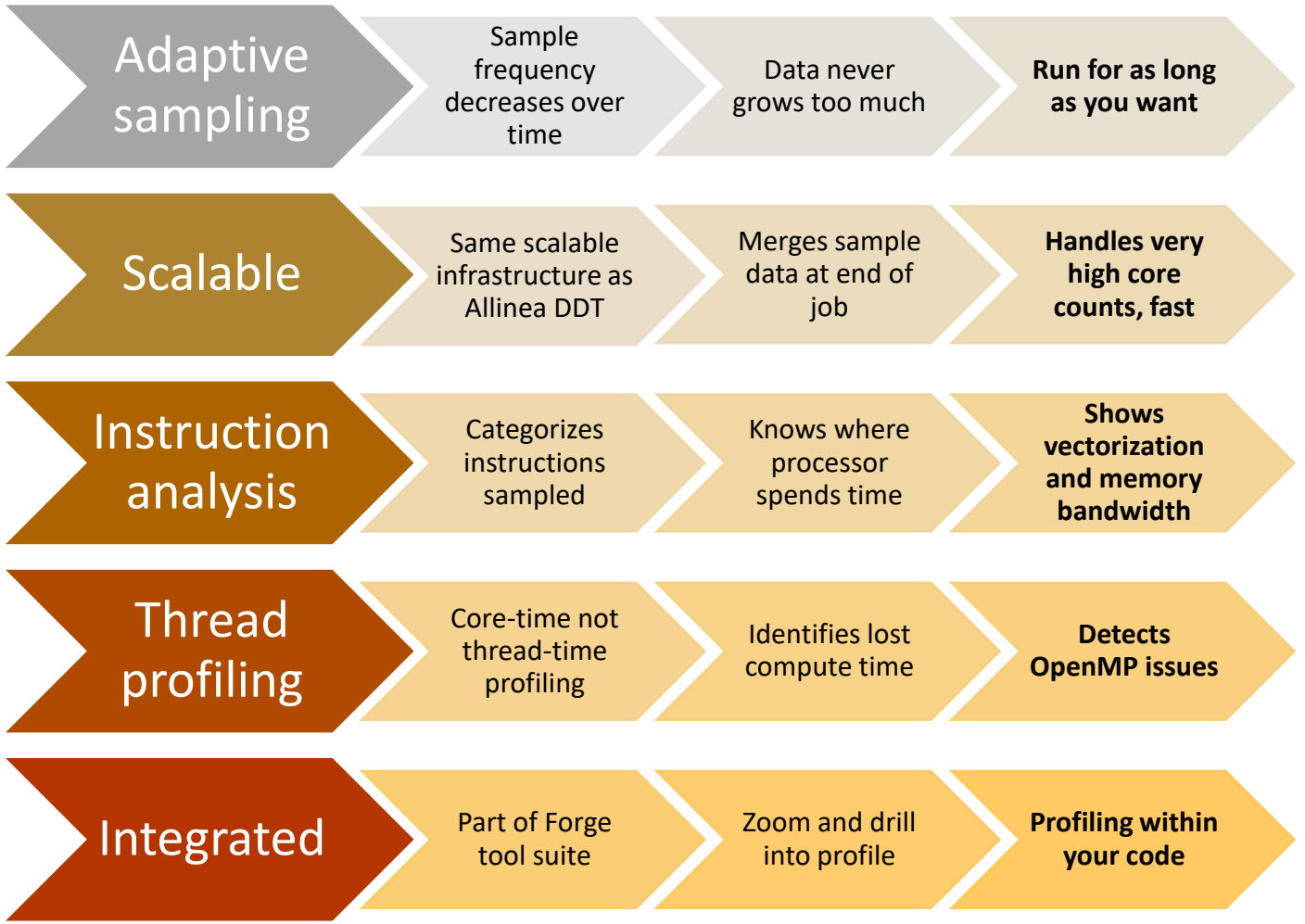
Detect and diagnose I/O bottlenecks at real scale

Allinea MAP – The Profiler

- Small data files
- <5% slowdown
- No instrumentation
- No recompilation



How Arm 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

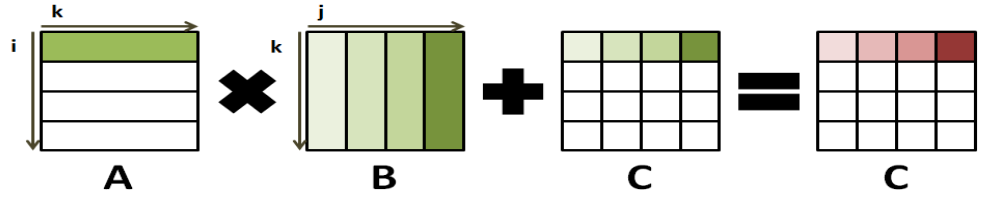
Basic Arm Map Profiling Example

Argonne NATIONAL LABORATORY

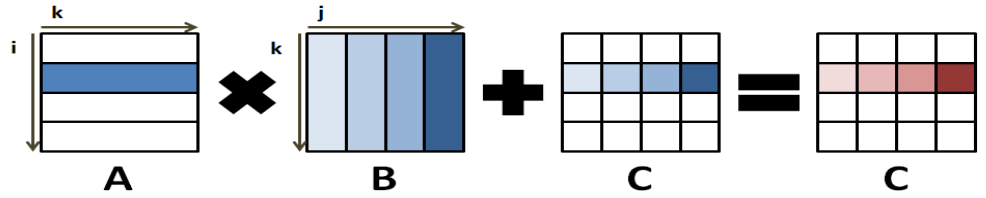
Matrix Multiplication Example

C = A x B + C

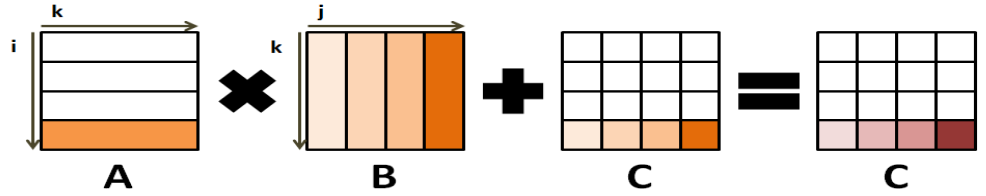
Master process



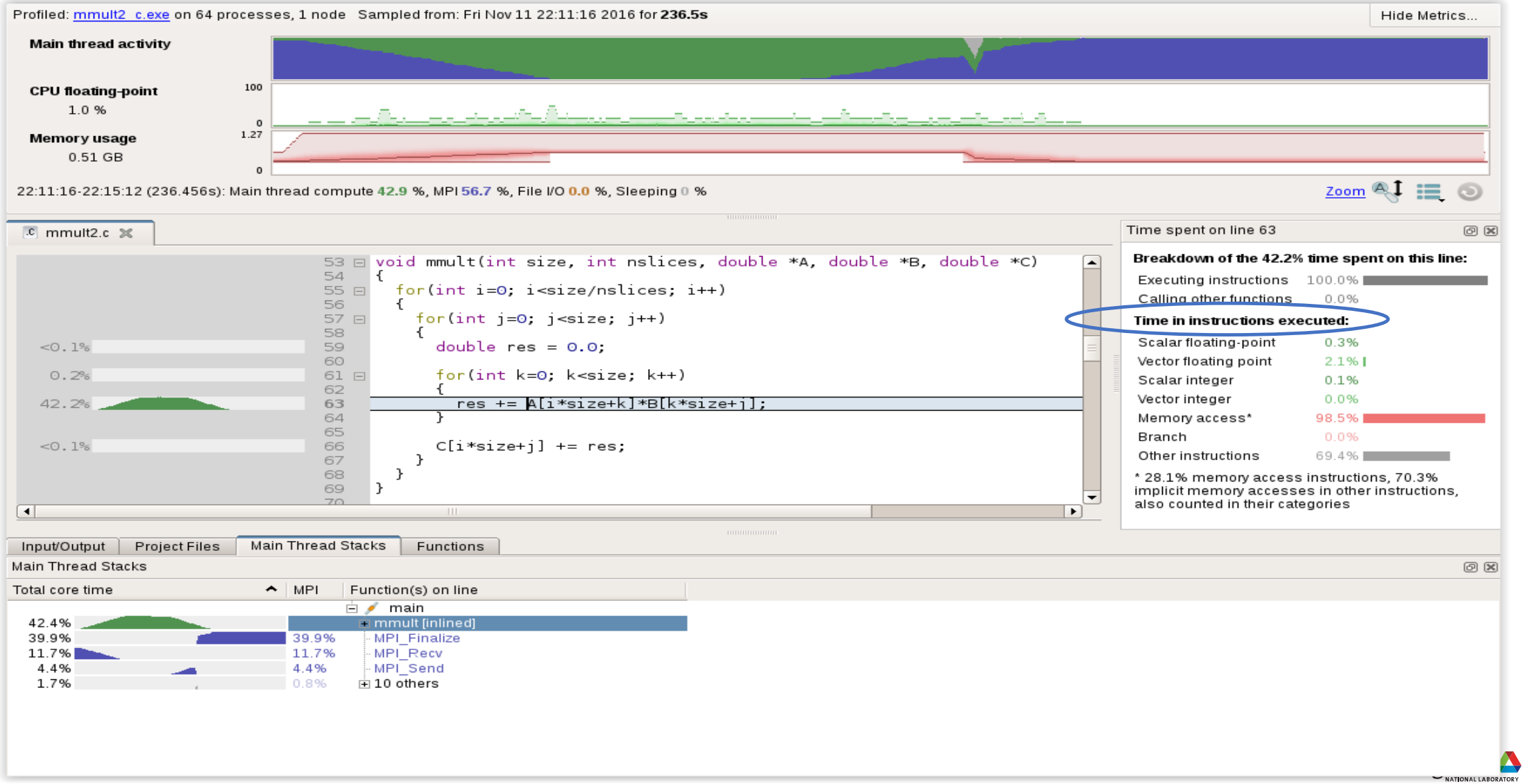
Slave process 1



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

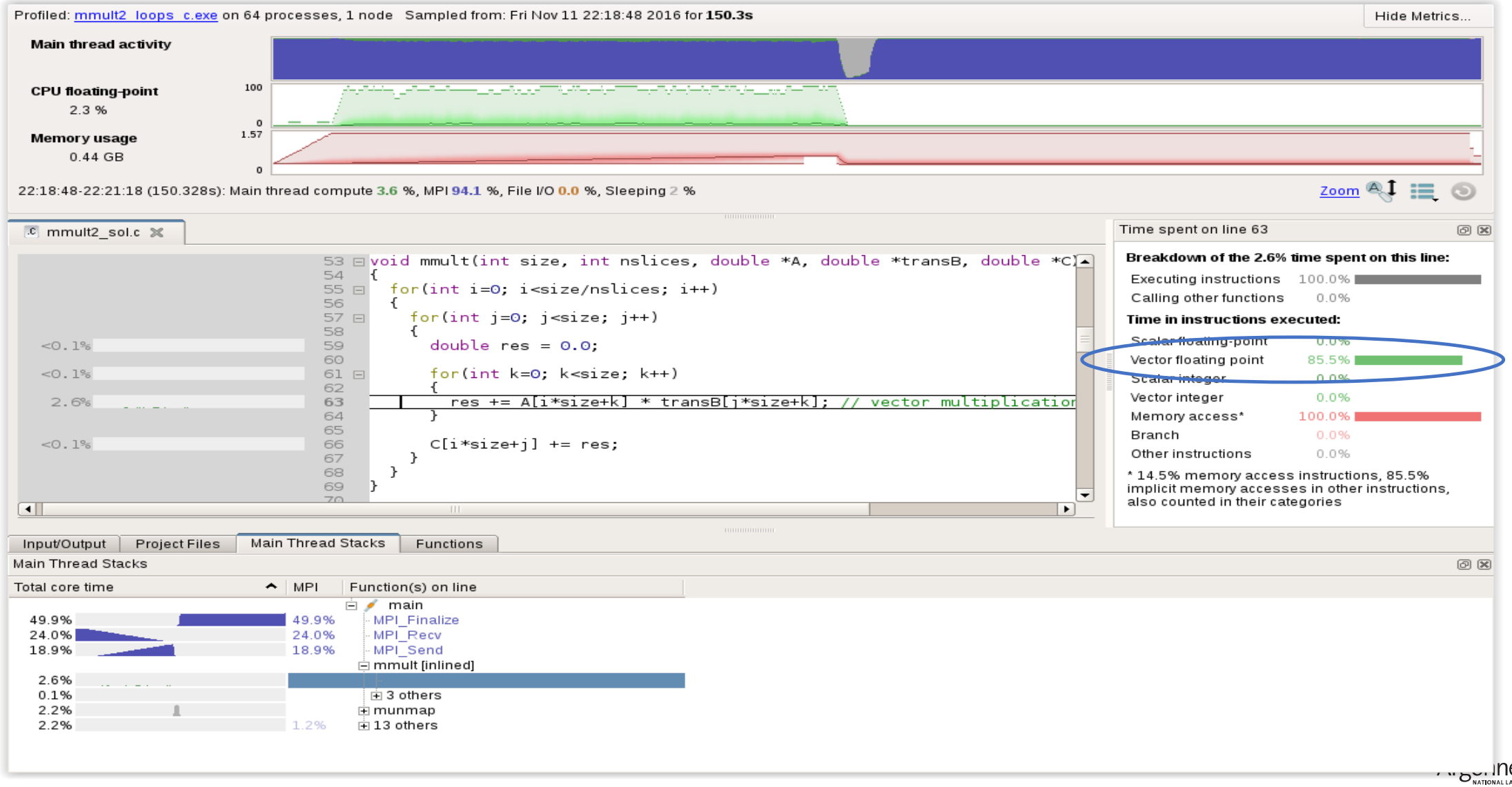
in C:

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

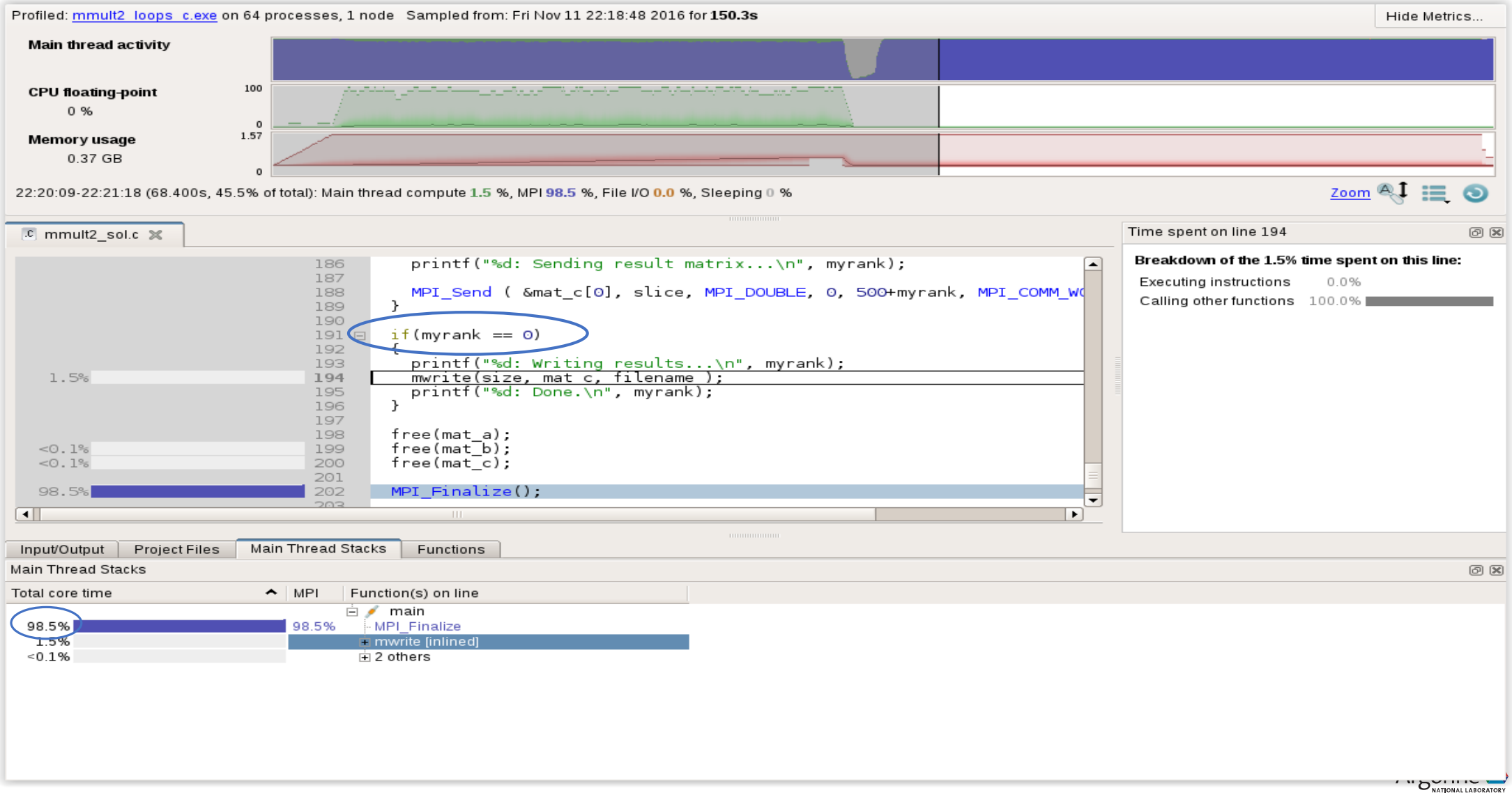
and in F90:

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

Improving Data Layout and Access Pattern



Serial Bottleneck



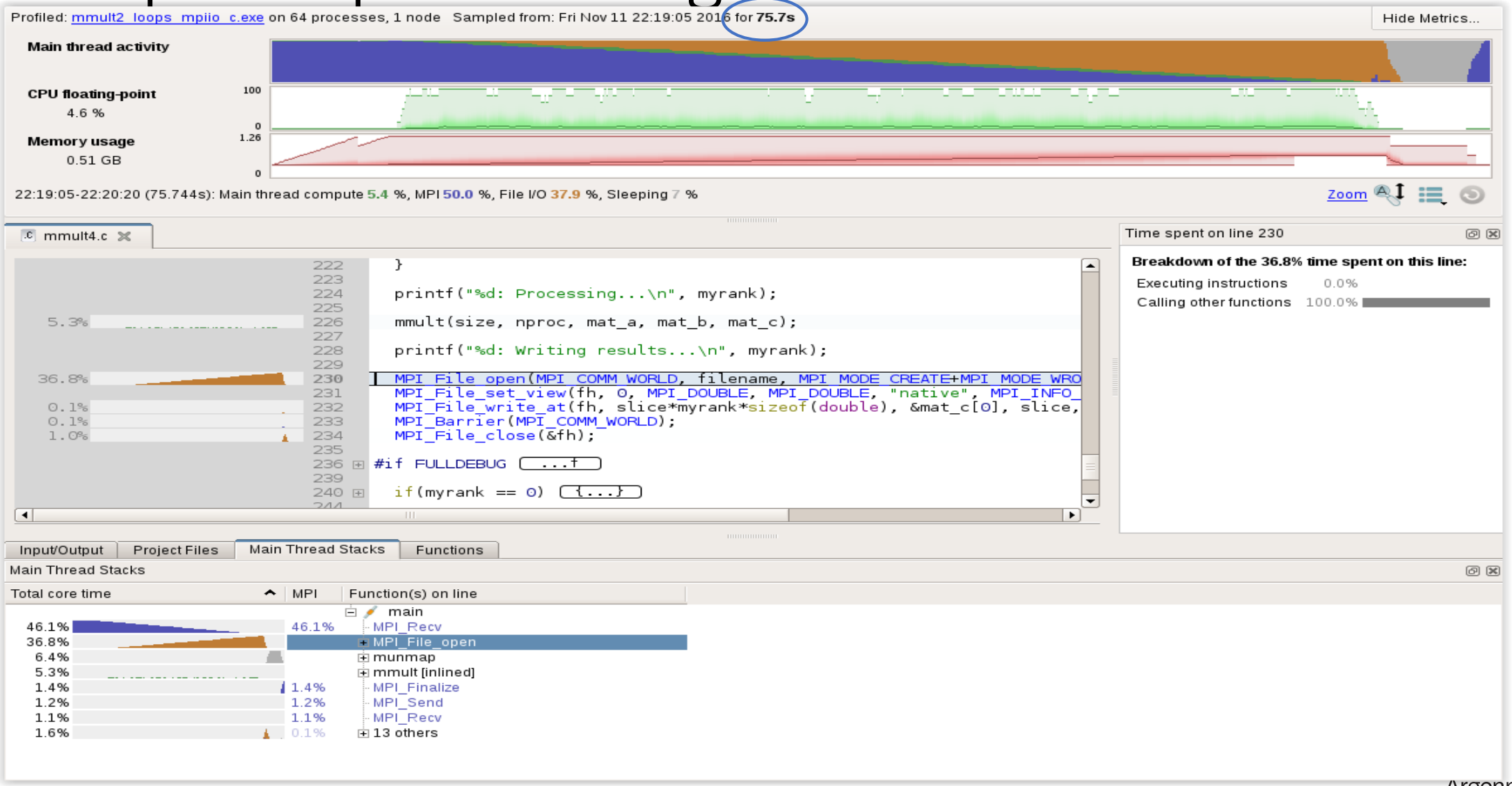
Inefficient I/O

```
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);
}
```

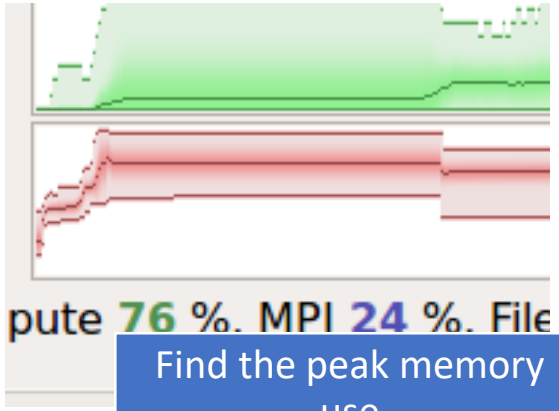
Improve Scalability of I/O Routines

- `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);`

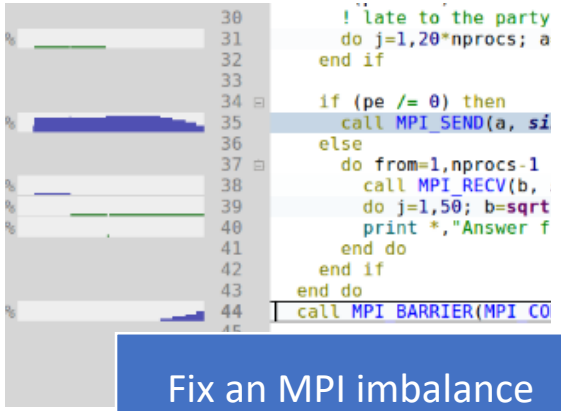
3x Speedup from Original Code



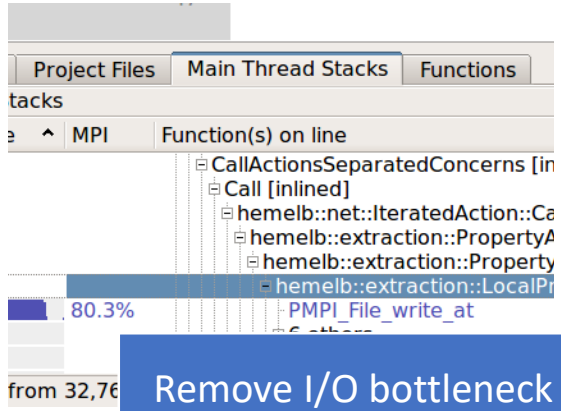
Six Great Things to Try with Arm MAP



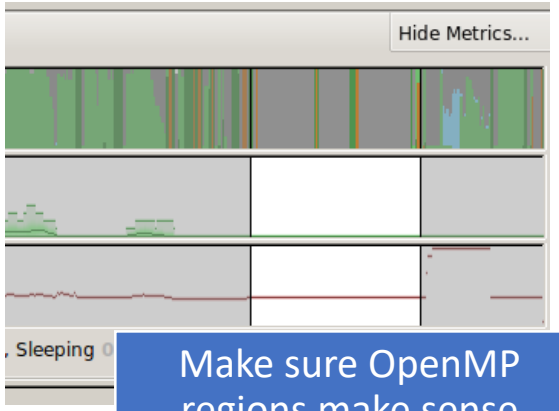
Find the peak memory use



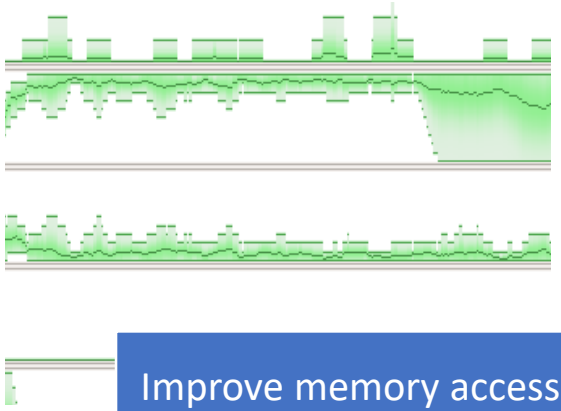
Fix an MPI imbalance



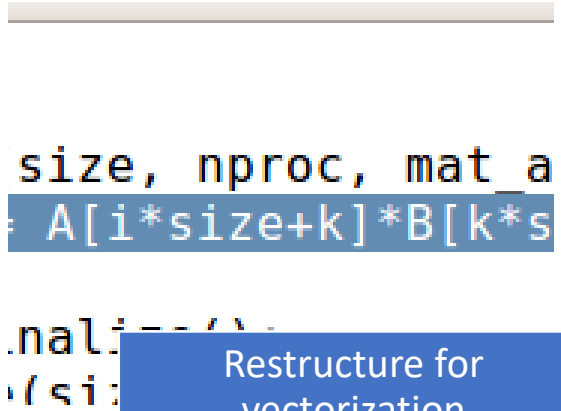
Remove I/O bottleneck



Make sure OpenMP regions make sense



Improve memory access



Restructure for vectorization

Questions?

Hands on Examples

Argonne NATIONAL LABORATORY

Arm Hands-on files

- The files for the examples that follow can be obtained on theta at the following location

- **`/projects/SDL_Workshop/arm/arm_handson.tgz`**

- This extracts 2 directories: demonstrations and arm_examples
- The demonstrations are there for you to play with and ask questions
- The examples are more like guided exercises

Launch Remote client

- Be sure to launch the remote client first
- Using a remote launch on your local machine is preferred
- Alternatively you can forward X11 when connecting to the login node of theta, and launch it there

```
module load forge/18.0.2
forge &
```

- If you accidentally close this window (easy to do), you will have to start it again

Hands-on Examples

- These examples are meant to be run on Theta in an interactive session

- **`qsub -I -q training -A SDL_Workshop -t 120 -n 1 --proccount 64`**

- Once a session has been allocated, load the Arm forge module

```
module load forge/18.0.2
```

Before Generating MAP profiles

- Static profiler libraries need to be created before MAP profiles can be generated
- Go to the arm_examples/wrapper directory
- Run

```
make-profiler-libraries --lib-type=static
```

- The Makefiles for the examples have already been modified to look for the profiler libraries in this directory

Go to exercise 1

- Exercise objectives
  - Familiarize with DDT user interface
  - Inspect values of u using multidimensional array viewer
  - Set watchpoint and tracepoint for diffnorm_global
  - Set breakpoint at line 89
- Typical run commands to use:

```
$> cd arm_examples/1_debug/
$> make
```

- Key DDT commands

```
On the login node:
$> forge &
In a submission file/interactive job:
$> ddt --connect aprun -n 4 ./jacobi.exe
```

Use Arm Forge to vectorize your codes

CPU

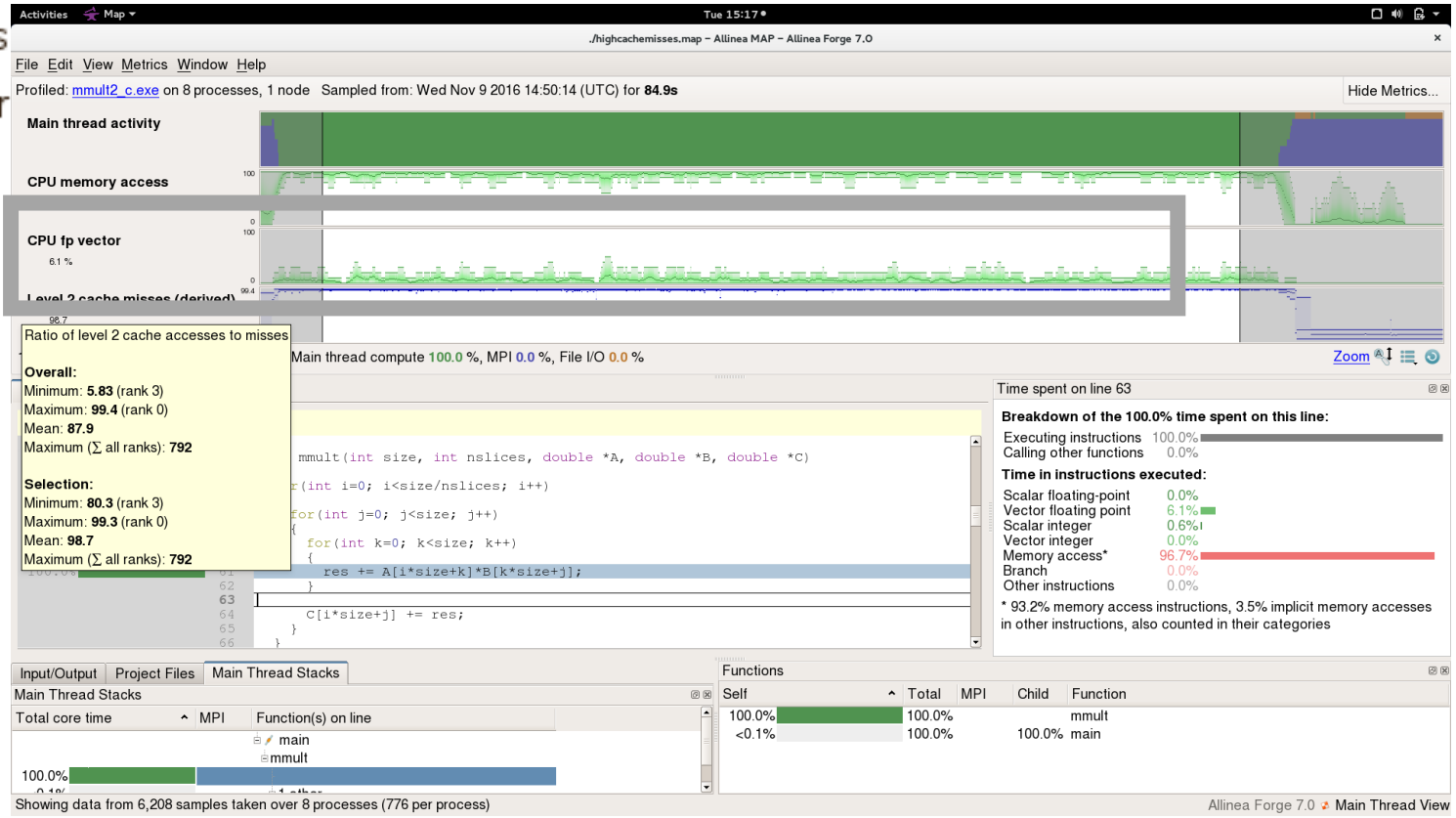
A breakdown of the 70.2% CPU time:

Scalar numeric ops 2.5%

Vector numeric ops 0.0%

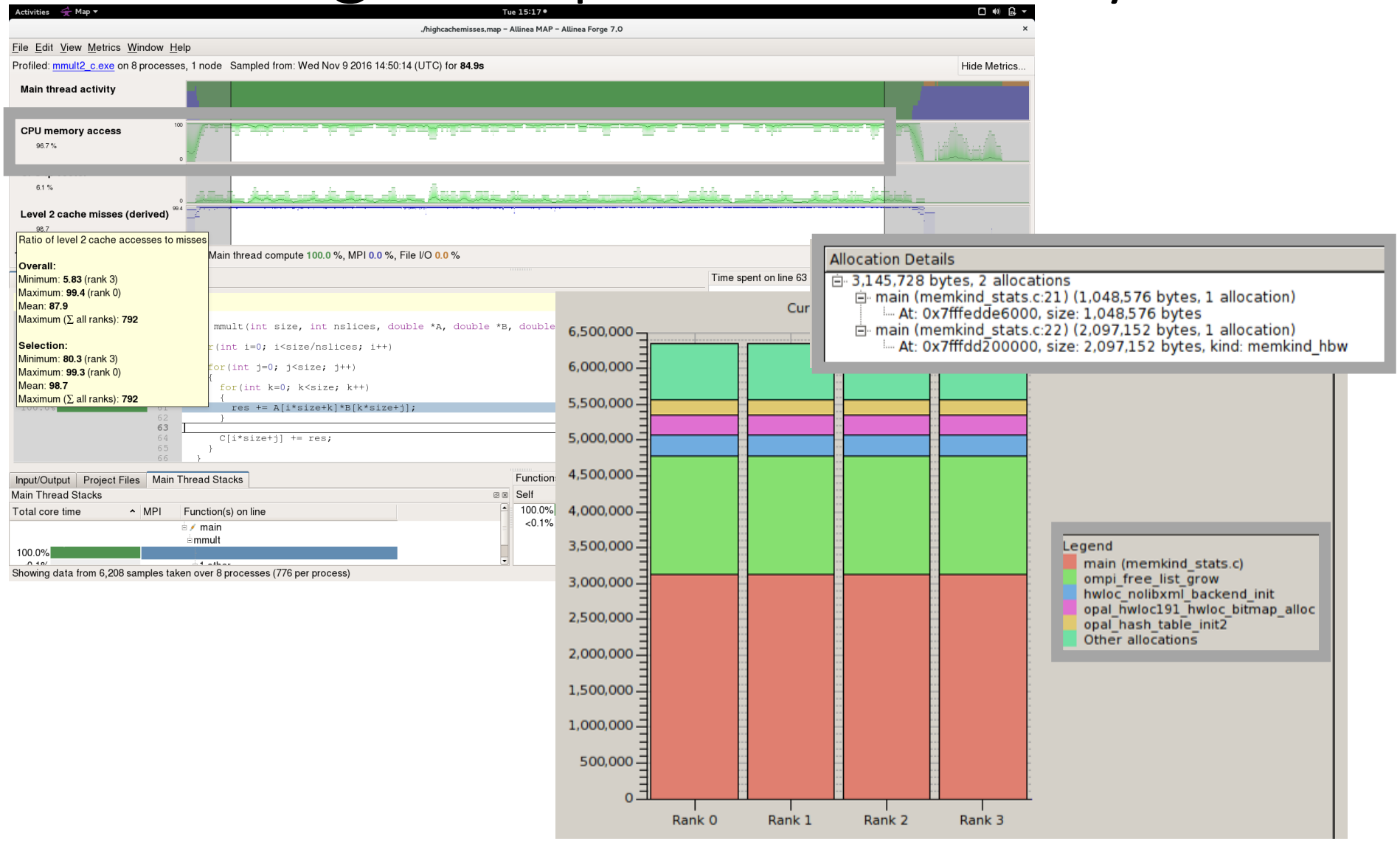
Memory accesses

Waiting for acceler



Argonne NATIONAL LABORATORY

Use Arm Forge to optimize memory access



Go to exercise 2

- Exercise objectives
  - Generate initial baseline profile
  - Ensure the matrices are stored in the MCDRAM (if applicable)
  - Fix the inefficient memory access issues
  - Further enable vectorization with Intel compiler flag **`-xMIC-AVX512`**
  - Generate profile with MAP after applying changes
- Typical run commands to use:

```
$> cd arm_examples/2_memory_accesses/
$> make
```

- Key Map commands

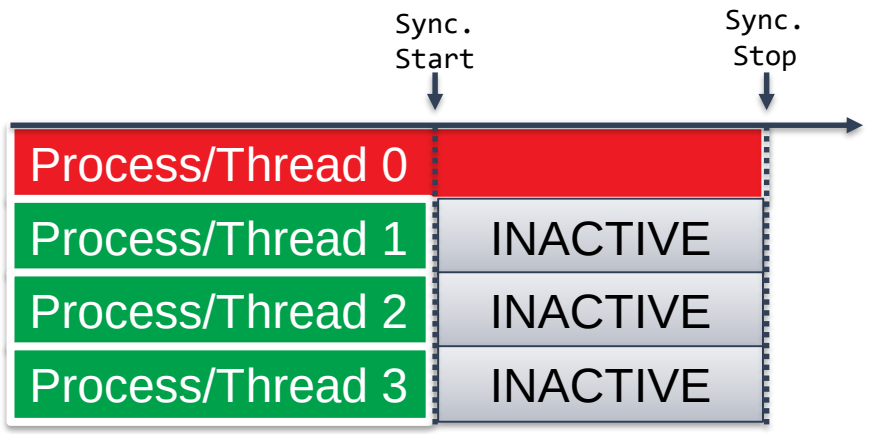
```
On the login node:
$> forge &
In a submission file/interactive job:
$> map --profile aprun -n 64 ./mmult2_c.exe
$> map --connect ./mmult2_c_*.map
```

How to identify load balancing issues?

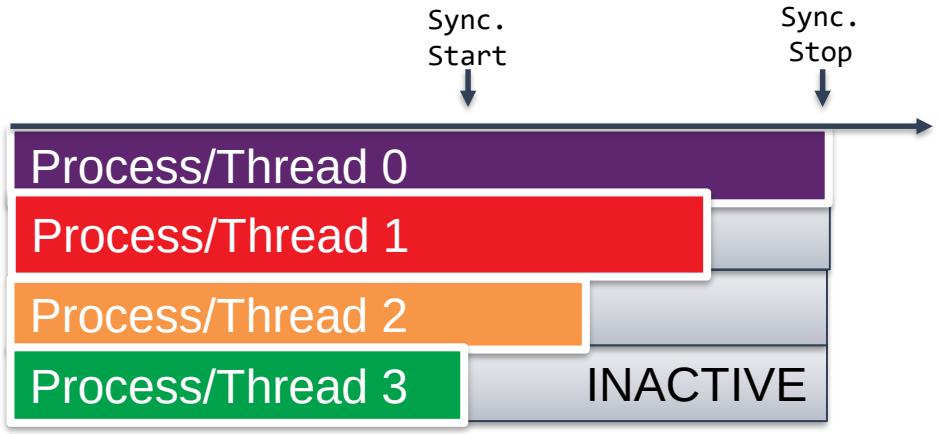
Problem: “one or some process(es) have too much work”

Clues visible in synchronization

- MPI Collective calls (MPI_Barrier, _Broadcast, etc.) with no actual data transfer
- Idle cores where threads are stuck in locks/mutexes

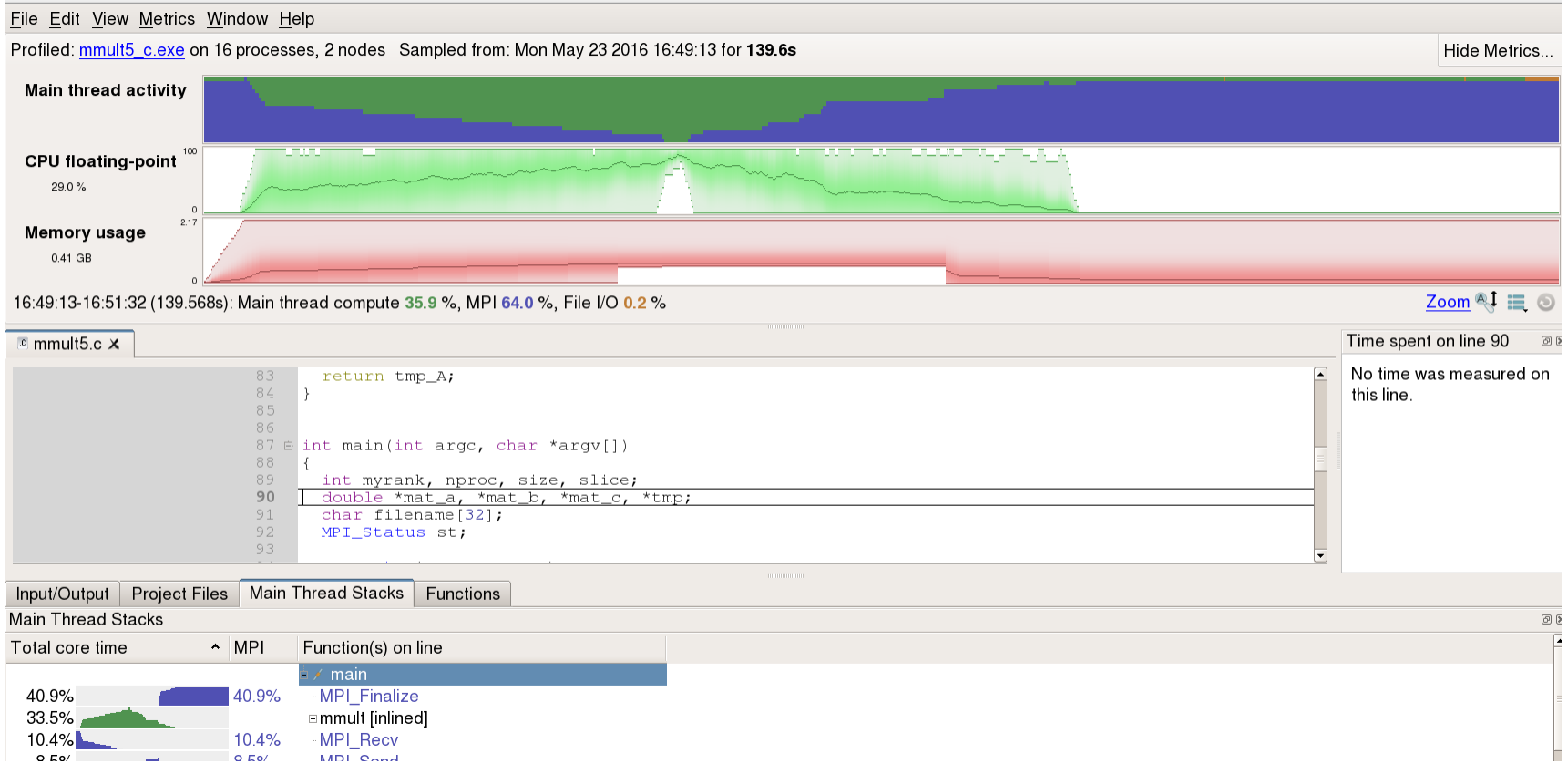



```
Total runtime: 100 sec
Total CPU time available: 400 sec
Total CPU time actually used: 250 sec
Efficiency: 62.5% of the machine time
```




```
Total runtime: 100 sec
Total CPU time available: 400 sec
Total CPU time actually used: 300 sec
Efficiency: 75% of the machine time
```

Use Arm MAP to balance your workloads



Go to exercise 3

- Exercise objectives
  - Expose workload imbalance issues in the code (preferably on 2 nodes)
  - Make suggestions to fix the problem
- Typical run commands to use:

```
$> cd arm_examples/3_imbalance/
$> make
```

- Key Map commands

```
$> map --profile aprun -n 64 ./mmult4_c.exe
$> map --connect mmult4_c_*.map
```

Go to exercise 4

Sometimes optimizations introduce bugs of their own

- Exercise objectives
  - Use ddt in offline mode to detect memory leaks
  - Examine the debug_report.txt file
  - Fix the leak
  - Generate new report
- Typical run commands to use:

```
$> cd arm_examples/4_memory_leak/
$> make
```

- Key ddt commands

```
$> ddt --offline --mem-debug --output=debug_report.txt aprun -n 4
   ./mmult6_c.exe
```

Solutions to Exercises

- Solutions to these exercises can be found in the **`.solution`** directory in each of the exercises