allinea

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

Accelerating Real Applications

Best Practices for Profiling and Debugging Complex Code

Beau Paisley

Senior Solutions Architect US

allinea FORGE allinea DDT allinea MAP allinea PERFORMANCE REPORTS

Allinea: The industry standard tools for HPC

Oil – X
Confidential

(and hundreds more)

We have enjoyed a long and productive relationship with Allinea to scale and deploy DDT on Titan and previous systems. We now see MAP as a performance tool that will help our users with the transition from Titan to Summit by providing a portable performance analysis solution.

— Buddy Bland, Project Director for the Oak Ridge Leadership Computing Facility

Best Practices for Profiling and Debugging Complex Code

In the beginning

- Offloading a simple kernel

Real-world complexity

- Understanding and analysing real application performance

Science: it works

- Profiling and debugging in extreme conditions

Best Practices for Profiling and Debugging Complex Code

In the beginning

- Offloading a simple kernel

Real-world complexity

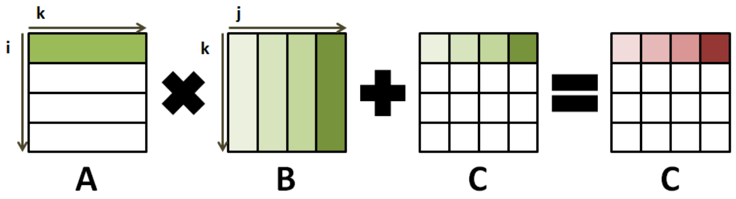
- Understanding and analysing real application performance

Science: it works

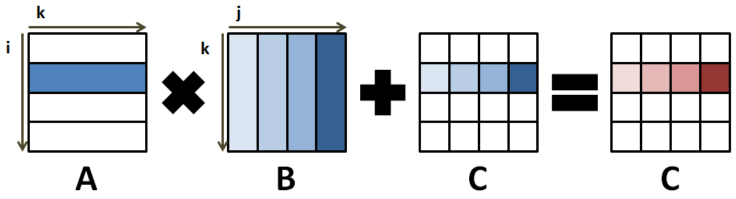
- Profiling and debugging in extreme conditions

In the beginning: offloading a simple multiplication kernel

Process master:

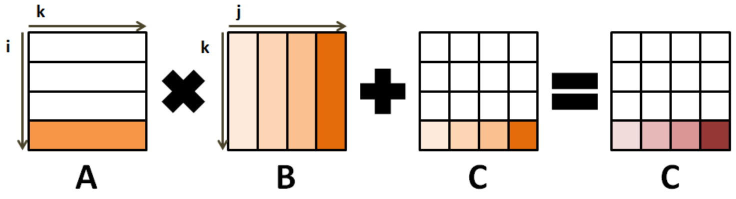


Process slave 1:



…   …

Process slave n:



In the beginning: offloading a simple multiplication kernel

```
void mmult(int size, int nslices, double *A, double *B, double *C)
{
  for(int i=0; i<size/nslices; i++)
  {
    for(int j=0; j<size; j++)
    {
      double res = 0.0;

      for(int k=0; k<size; k++)
      {
        res += A[i*size+k]*B[k*size+j];
      }

      C[i*size+j] += res;
    }
  }
}
```

Phase 1: Profile our simple matrix multiplication kernel

Running the example program:

```
$ mpiexec -n 8 ./mmult1.exe
```

Profiling the example program:

```
$ map mpiexec -n 8 ./mmult1.exe
```

Phase 3: A correctly-implemented matrix multiplication kernel!

That little demo is *nothing like the real world* at all

In the beginning

- Offloading a simple kernel

Real-world complexity

- Understanding and analysing real application performance

Science: it works

- Profiling and debugging in extreme conditions

Introducing a real application: Discovar DeNovo

Matrix multiply example:

Language	files	blank	comment	code
C	**1**	39	0	**151**

Discovar DeNovo, a genome assembly code:

Language	files	blank	comment	code
C++	312	15898	14797	99857
C/C++ Header	405	15219	15718	47118
Bourne Shell	9	5107	5878	32283
m4	12	971	100	8456
make	4	651	1600	3580
SUM:	**742**	37846	38093	**191294**

Introducing a real application: Discovar DeNovo

Understand the run → Check hot code → Investigate oddities → Experiment

Phases

- **Stacks** and **OpenMP regions**
- What application **intends** and **does**

Low-level

- **Functions**: low-level **time**
- **Memory** or **FPU** bound? **Vectorized**?

Metrics

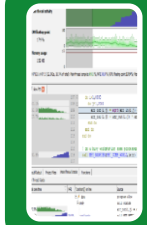
- Look for **slopes, spread** and **trends**

Which lines of code **are** hot?

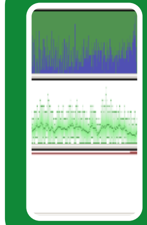
+

Should they be?

Spread implies task imbalance



Slope implies workload imbalance



Trends over time are often leaks or algorithmic oversights

Observation

Hypothesis

Experiment

On the subject of making mistakes, what about "Phase 2…"?

Demo output from our matrix multiplication example:

```
2: Receiving matrices...
3: Receiving matrices...
…
6: Processing...
7: Processing...
0: Processing...
…
0: Receiving result matrix...
7: Sending result matrix...
0: Done.

real    0m2.675s
user    0m7.490s
sys     0m2.561s
```

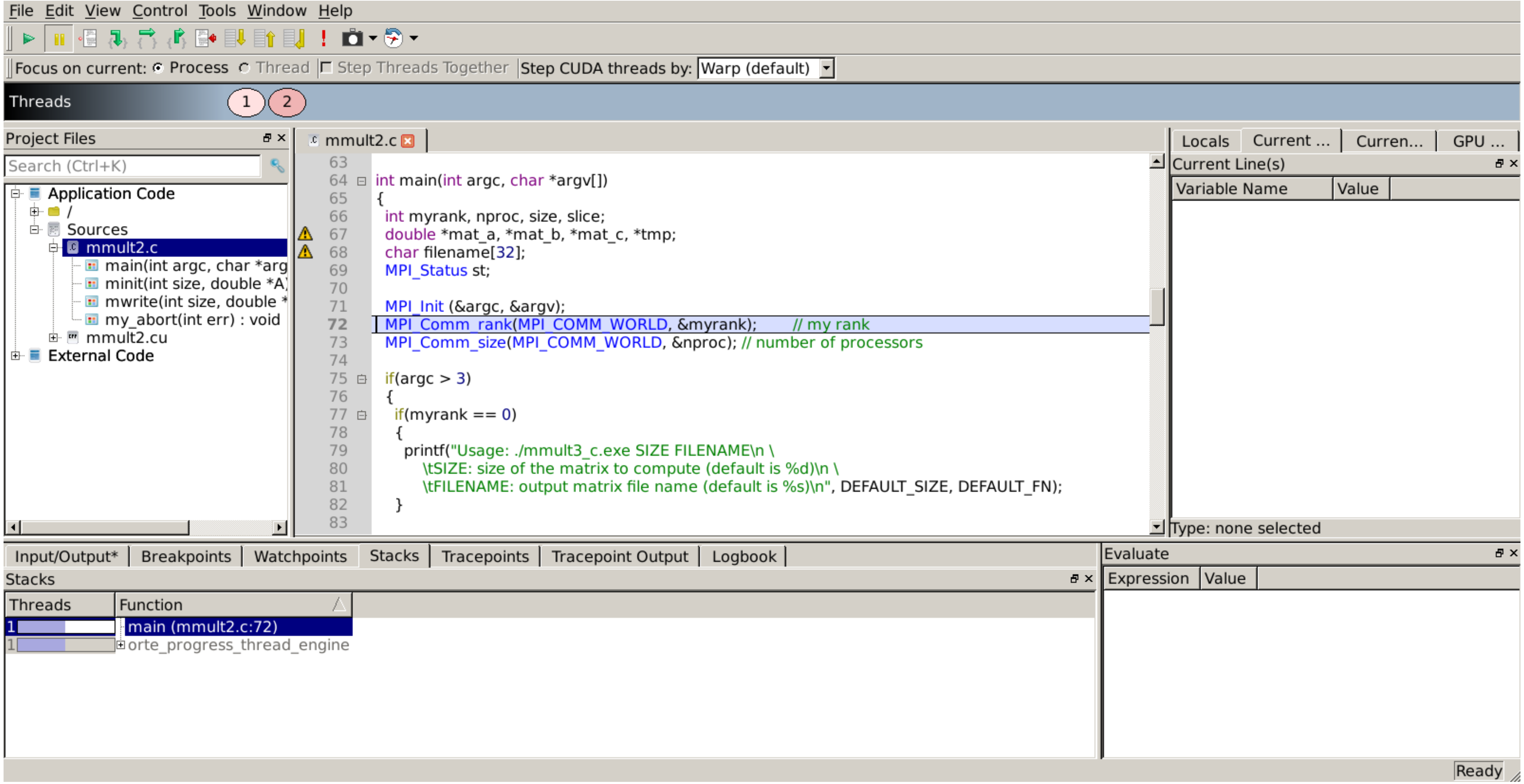
On the subject of making mistakes, what about “Phase 2…”?

More typical output after offloading a real-world kernel:

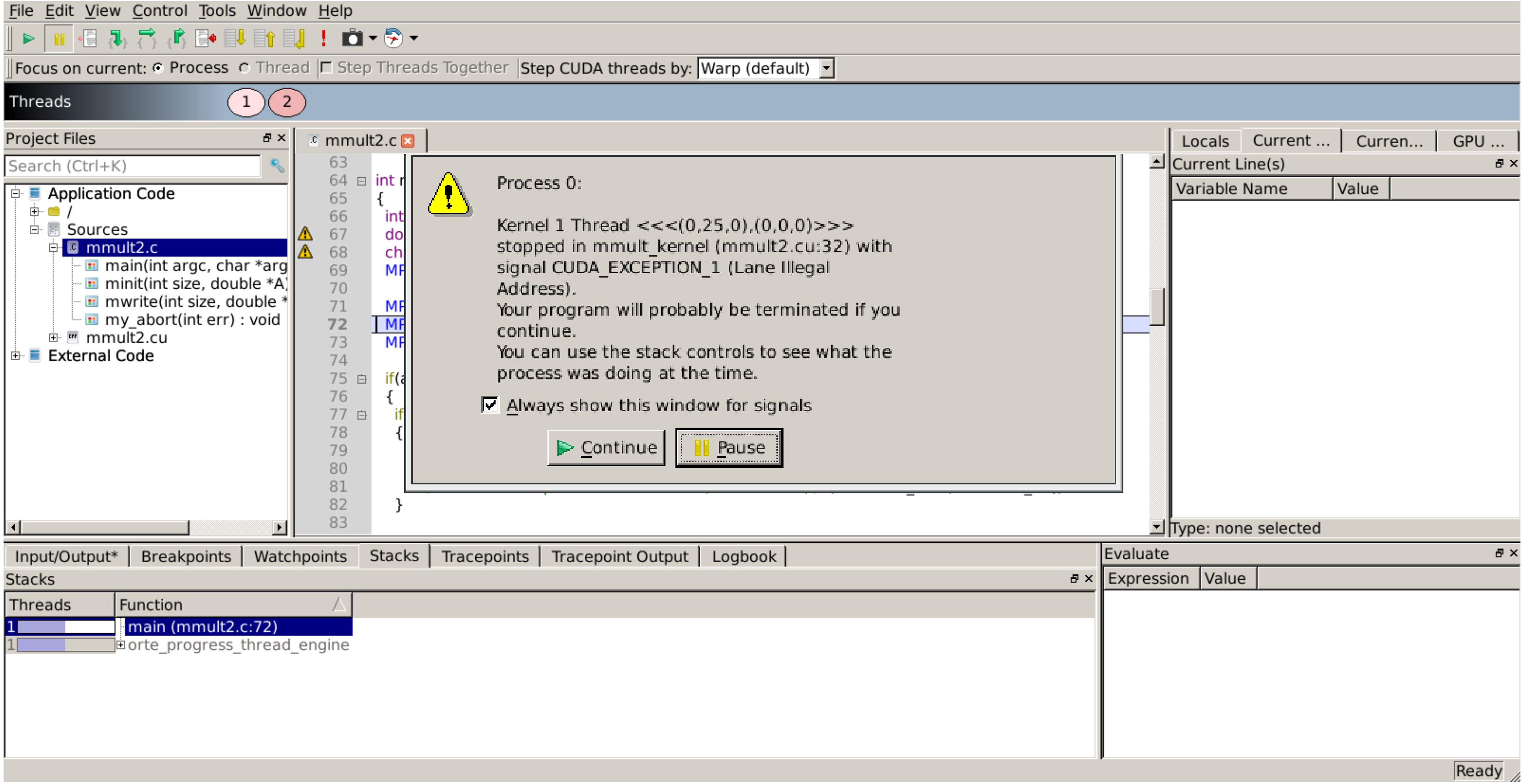
```
1: Receiving matrices...
7: Receiving matrices...
0: Sending matrices...
…
7: Processing...
0: Processing...
CUDA error
--------------------------------------------------------------------------
MPI_ABORT was invoked on rank 1 in communicator MPI_COMM_WORLD
with errorcode 77.

NOTE: invoking MPI_ABORT causes Open MPI to kill all MPI processes.
You may or may not see output from other processes, depending on
exactly when Open MPI kills them.
--------------------------------------------------------------------------
```

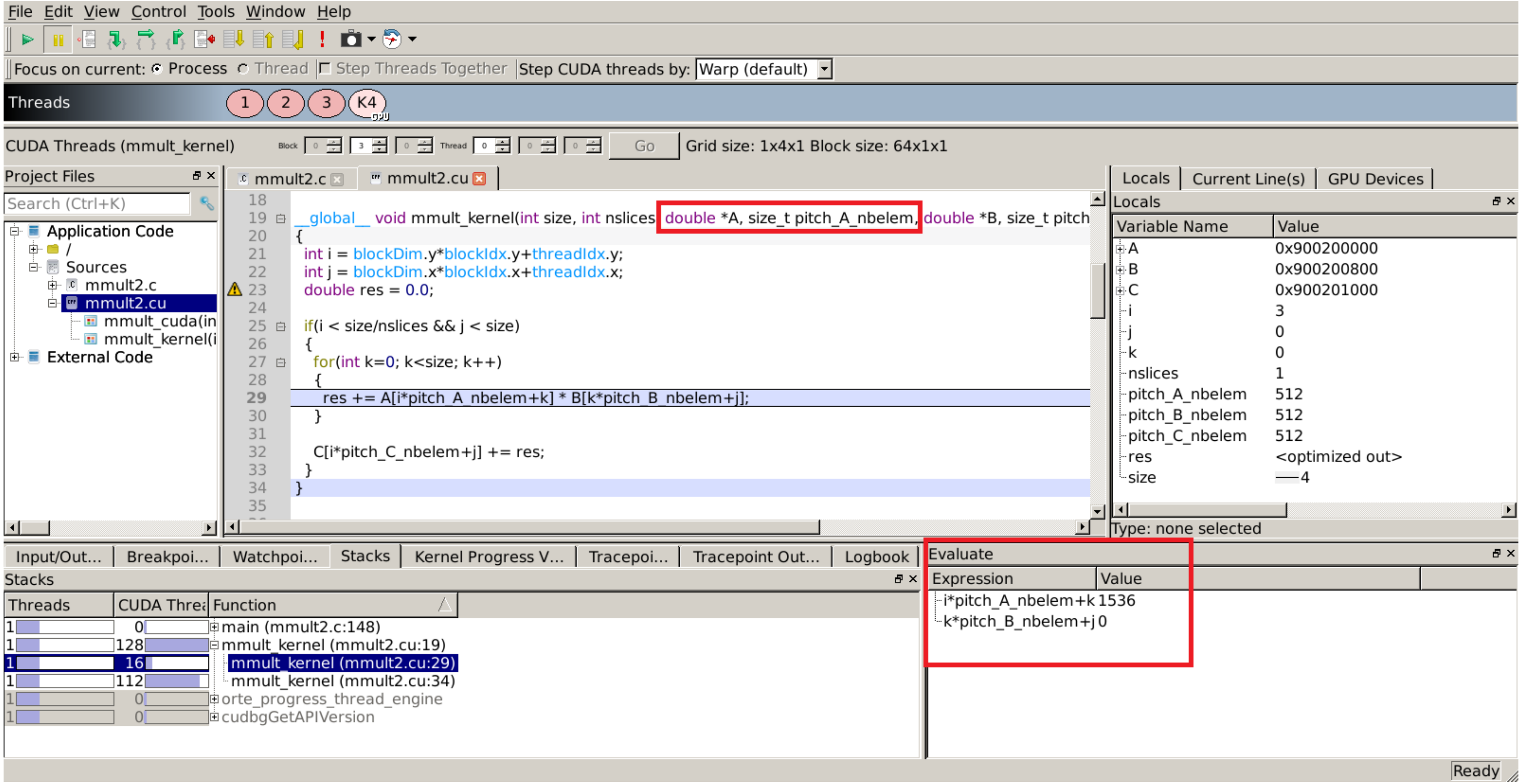
Shared interface with integrated GPU + CPU memory debugging



Just hit Play!



This is the exact line the program crashed on – now look at GPU variables to see why



Real-world debugging requires a systematic approach

In the beginning

- Offloading a simple kernel

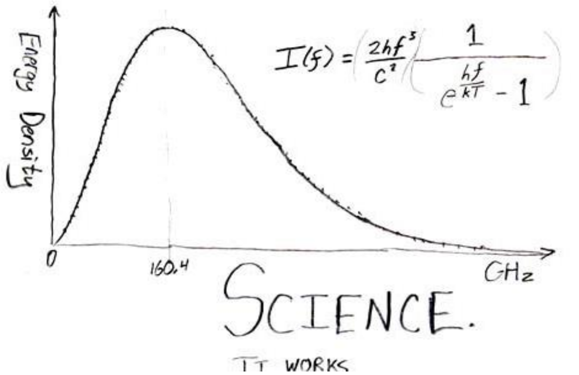
Real-world complexity

- Understanding and analysing real application performance

Science: it works

- Profiling and debugging in extreme conditions

Real-world debugging requires a systematic approach



Discipline Magic Inspiration Science

Debugging by Discipline

Simple techniques, rigorously applied, will dramatically improve your life.

(At least when it's time to debug)

Discipline #3: Continuous Integration and Regression Testing

Simple

- Sanity and performance checks
- Reliability is crucial – no false positives allowed

Regular

- Run on every code commit
- Speed is important – don't run entire cases

Auto

- Use source control hooks to submit test jobs
- OSS to view and manage runs (http://jenkins-ci.org)

Discipline #3: Continuous Integration and Regression Testing

DDT

- Prefix sanity tests with ddt --offline $REV.html …
- Integrate debug reports into Jenkins/CI system

MAP

- Prefix performance tests with: map --profile …
- MAP's editor highlights source lines changed

PR

- Generate HTML reports directly or from MAP files
- Integrate into Jenkins/CI & graph metrics over time

Debugging by Magic

Any technology sufficiently advanced is indistinguishable from magic.

Unpredictable, dangerous, irresistible.

Debugging by Magic

Some problems are perfect for investigating with a debugging tool:

Crashes

Deadlock

Memory problems

Learn to use the bisect command with a test script to isolate the revision that failed:

```
$ hg bisect --bad
$ hg bisect --good 4
$ hg bisect -c logs/my-test.sh
$ hg log -pr <changeset id>
```

Bonus - static analysis
(integrated into DDT)

Debugging by Inspiration

Look at the problem, see the solution.

Trust your instincts.

Test whether they're right.

Debugging by Inspiration

When you have a sense for what the problem is:

Test it: `$ ddt -offline log.html -trace-at mmult.c:412,rx,ry,rz`

Log it: `$ cat >> logs/short-problem-name`

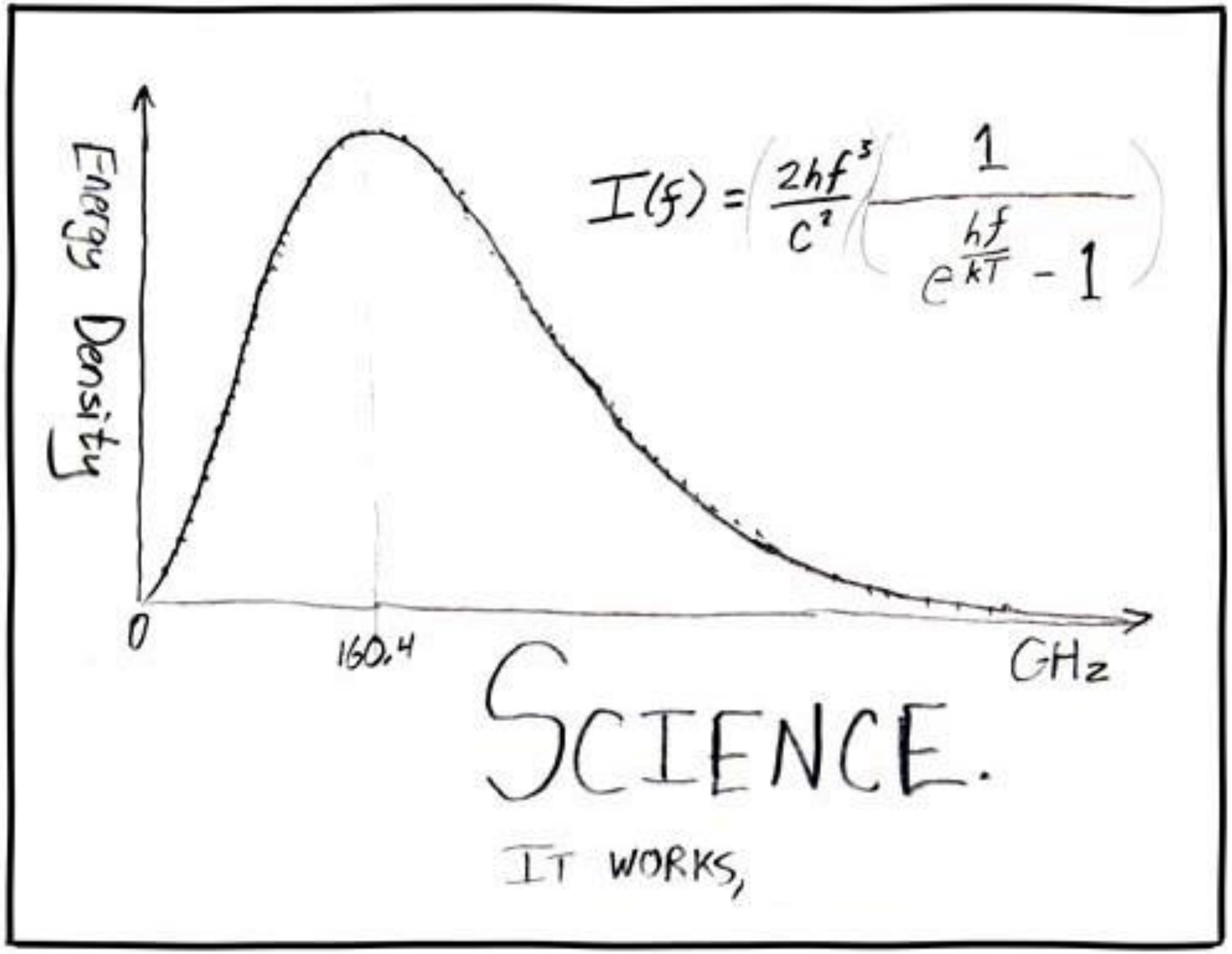
```
Suspect rx is out of bounds in mmult.c:412.
Testing with -trace-at mmult.c:412,rx,ry,rz showed...
```

Search your logbooks: `$ grep -ri "out of bounds" logs/*`

If in doubt: explain it to a rubber duck.

Tip - set a **time limit** for debugging by inspiration.
After 15 minutes, try **science**.

Debugging by Science



1. Hypothesis
2. Prediction
3. Experiment
4. Observation
5. Conclusion

There is a **reason** for the bug and you **will** find it!

A logbook is at the heart of debugging by science:

```
hypothesis: cause is in shell_sort()
prediction: At sort.c:6, expect a[] = [11, 4] and size = 2
experiment: -trace-at sort.c:6,a[0],a[1],size
observation: a[] = [11, 14, ?] and size = 3
conclusion: rejected

hypothesis: calling shell_sort with size=3 causes failure
prediction: setting size=2 should make program work
experiment: Set size=2 before call using debugger
observation: As predicted
conclusion: confirmed
```

Real-world performance optimization is also a process:

Understand the run → Check hot code → Investigate oddities → Experiment

Phases

- **Stacks** and **OpenMP regions**
- What application **intends** and **does**

Low-level

- **Functions**: low-level **time**
- **Memory** or **FPU** bound? **Vectorized**?

Metrics

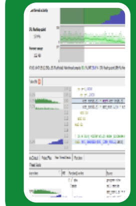
- Look for **slopes, spread** and **trends**

Which lines of code **are** hot?

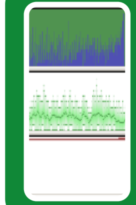
+

Should they be?

Spread implies task imbalance



Slope implies workload imbalance



Trends over time are often leaks or algorithmic oversights

Observation

Hypothesis

Experiment

Best Practices for Profiling and Debugging Complex Code

In the beginning

- Offloading a simple kernel

Real-world complexity

- Understanding and analysing real application performance

Science: it works

- Profiling and debugging in extreme conditions

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

Accelerating Real Applications

Best Practices for Profiling and Debugging Complex Code

Beau Paisley

Senior Solutions Architect US

allinea FORGE

allinea DDT

allinea MAP

allinea PERFORMANCE REPORTS