allinea

Now part of ARM

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

Developing, Debugging, and Optimizing GPU Codes for High Performance Computing with Allinea Forge

Ryan Hulguin
Applications Engineer
ryan.hulguin@arm.com

allinea FORGE

allinea DDT

allinea MAP

allinea PERFORMANCE REPORTS

Agenda

- Introduction
- Overview of Allinea Products
- GPU Demonstration Examples
- Q&A

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
Now part of ARM

Where to find Allinea's tools

Over 85% of Top 100 HPC systems

- From small to very large tools provision

8 of the Top 10 HPC systems

- Up to 700,000 core tools usage

Future leadership systems

- Millions of cores usage

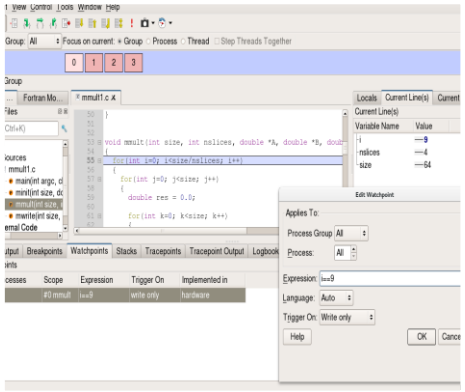
allinea
Now part of ARM

Allinea: Industry Standard Tools for HPC

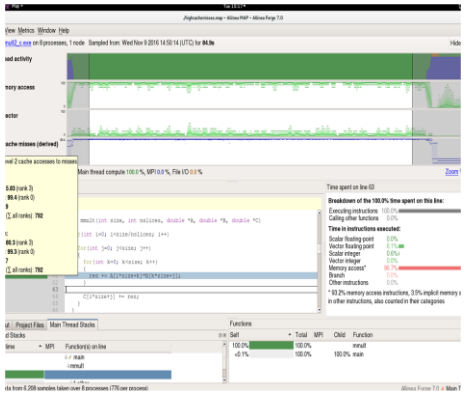
(and hundreds more)

allinea
Now part of ARM

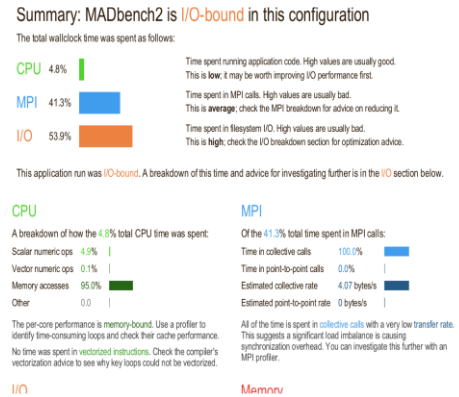
Allinea toolkits save users' and developers' time



Allinea DDT
(debugging)



Allinea MAP
(profiling)



allinea FORGE

allinea PERFORMANCE REPORTS

allinea
Now part of ARM

Analyze and tune application performance

A single-page report on application performance for users and administrators

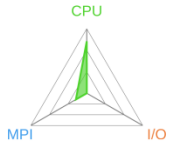
Identify configuration problems and resource bottlenecks immediately

Track mission-critical performance over time and after system upgrades

Ensure key applications run at full speed on a new cluster or architecture

allinea PERFORMANCE REPORTS

Command:	mpirun -n 8 CloverLeaf_ref/clover_leaf
Resources:	8 processes, 1 node (4 physical, 8 logical cores per node)
Machine:	kaze
Start time:	Fri Oct 31 15:42:41 2014
Total time:	24 seconds (0 minutes)
Full path:	/home/mark/Work/code/mantevo/CloverLeaf/CloverLeaf_ref
Input file:	
Notes:	2.1 Ghz CPU frequency



Summary: clover_leaf is CPU-bound in this configuration

CPU	80.6%	Time spent running application code. High values are usually good. This is **high**; check the CPU performance section for optimization advice.
MPI	19.4%	Time spent in MPI calls. High values are usually bad. This is **low**; this code may benefit from increasing the process count.
I/O	0.1%	Time spent in filesystem I/O. High values are usually bad. This is **very low**; however single-process I/O often causes large MPI wait times.

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

As little time is spent in MPI calls, this code may also benefit from running at larger scales.

CPU

A breakdown of the 80.6% CPU time:

Single-core code	0.4%
OpenMP regions	99.6%
Scalar numeric ops	42.4%
Vector numeric ops	4.0%
Memory accesses	53.6%

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

Little time is spent in vectorized instructions. Check the compiler's vectorization advice to see why key loops could not be vectorized.

MPI

A breakdown of the 19.4% MPI time:

Time in collective calls	41.7%
Time in point-to-point calls	58.3%
Effective process collective rate	1.68 kB/s
Effective process point-to-point rate	24.5 MB/s

Most of the time is spent in point-to-point calls with a low transfer rate. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait.

The collective transfer rate is very low. This suggests load imbalance is causing synchronization overhead; use an MPI profiler to investigate further.

I/O

A breakdown of the 0.1% I/O time:

Time in reads	0.0%
Time in writes	100.0%
Effective process read rate	0.00 bytes/s
Effective process write rate	611 kB/s

OpenMP

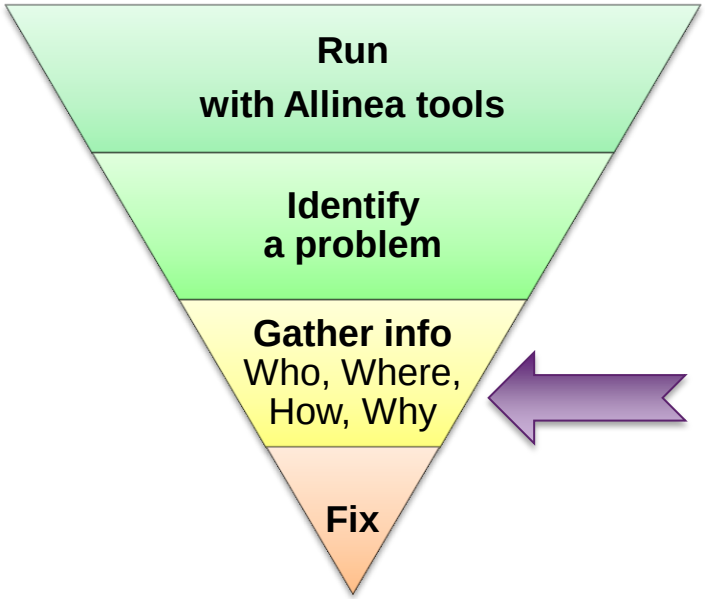
A breakdown of the 99.6% time in OpenMP regions:

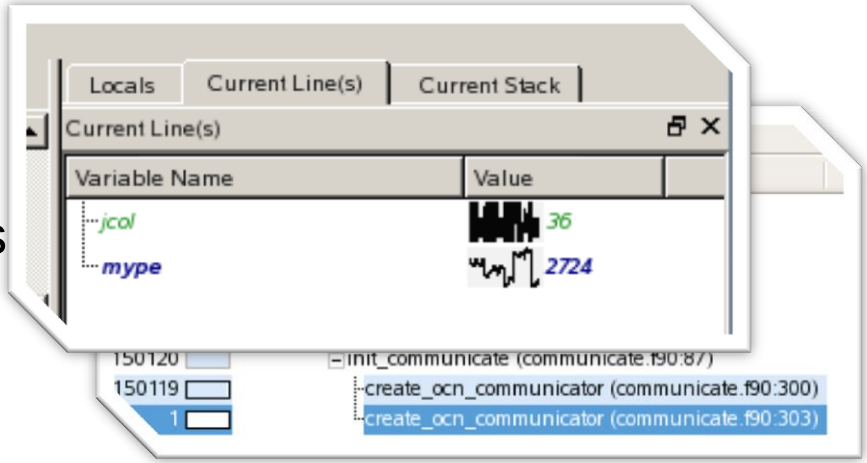
Computation	100.0%
Synchronization	0.0%
Physical core utilization	200.0%
Involuntary context switches per second	3.0

allinea
Now part of ARM

Allinea DDT – The Debugger

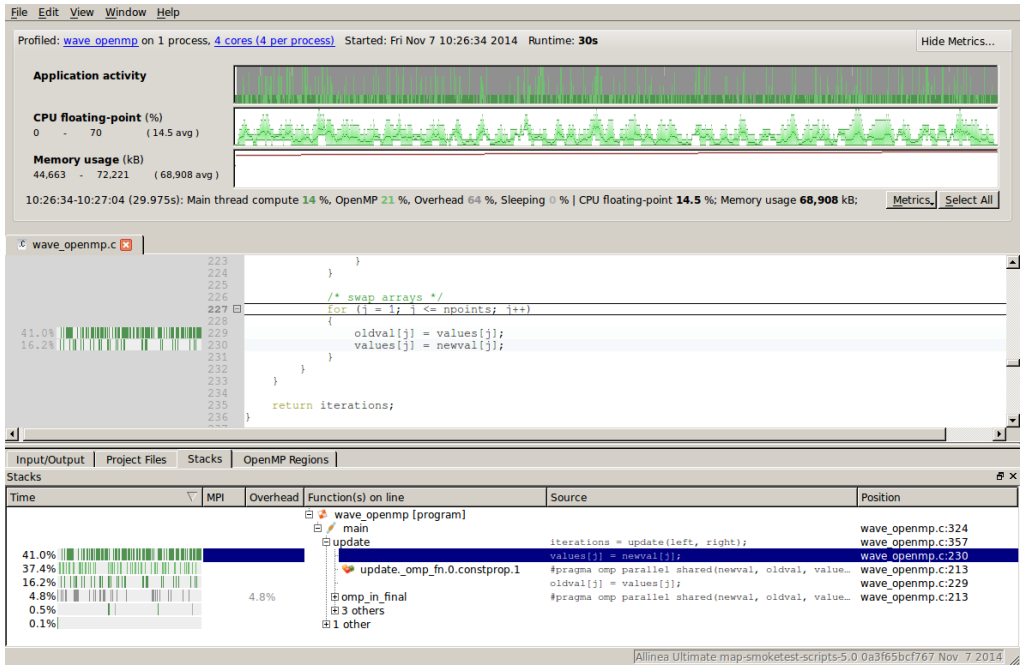
- Who had a rogue behavior ?
  - Merges stacks from processes and threads
- Where did it happen?
  - leaps to source
- How did it happen?
  - Diagnostic messages
  - Some faults evident instantly from source
- Why did it happen?
  - Unique “Smart Highlighting”
  - Sparklines comparing data across processes





Allinea MAP – The Profiler

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



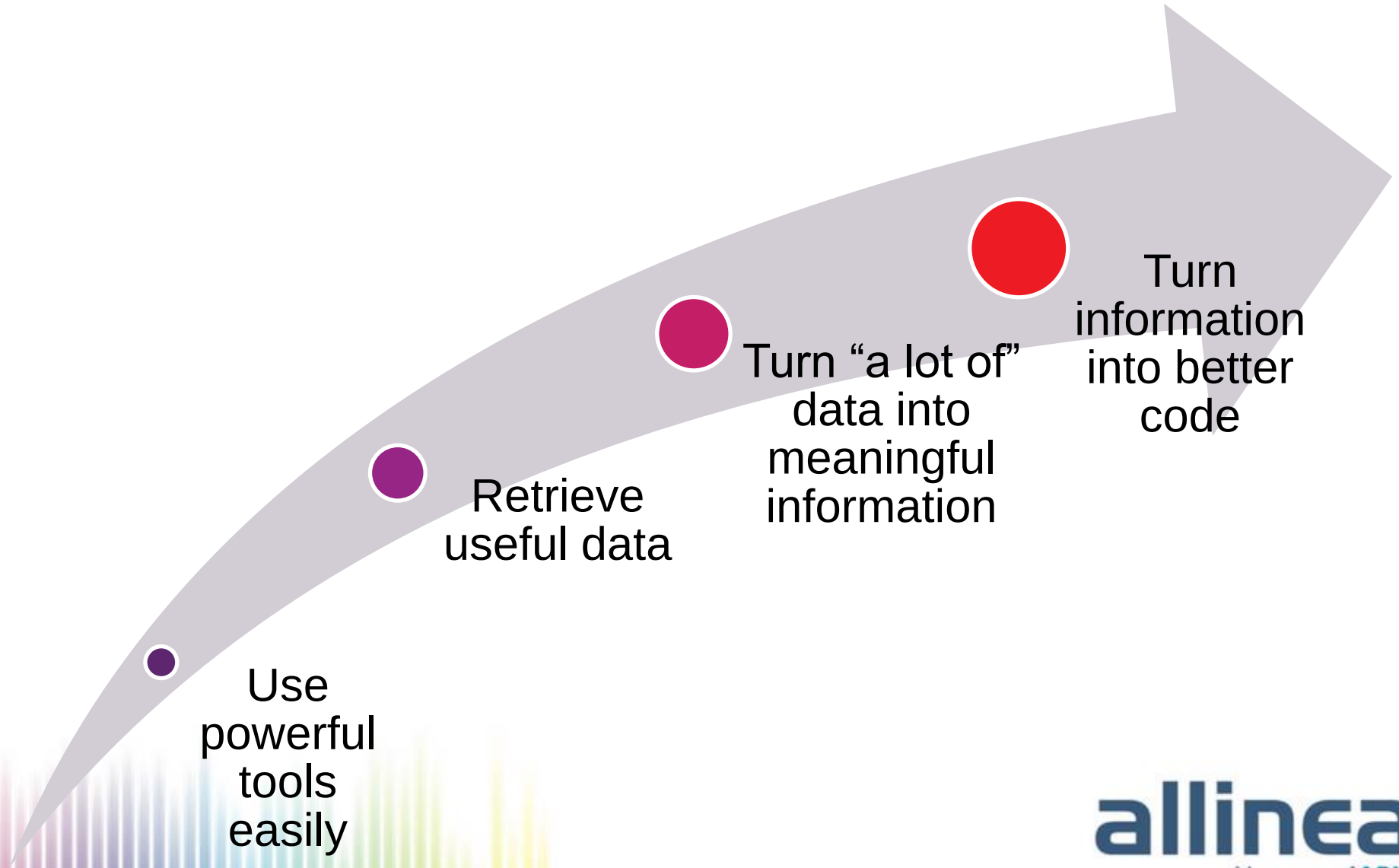
allinea
Now part of ARM

How Allinea MAP is different

Adaptive sampling	Sample frequency decreases over time	Data never grows too much	**Run for as long as you want**
Scalable	Same scalable infrastructure as Allinea DDT	Merges sample data at end of job	**Handles very high core counts, fast**
Instruction analysis	Categorizes instructions sampled	Knows where processor spends time	**Shows vectorization and memory bandwidth**
Thread profiling	Core-time not thread-time profiling	Identifies lost compute time	**Detects OpenMP issues**
Integrated	Part of Forge tool suite	Zoom and drill into profile	**Profiling within your code**

allinea
Now part of ARM

Enabling Performance Potential



allinea
Now part of ARM

Demonstration Examples

- The following examples are available through qwiklab

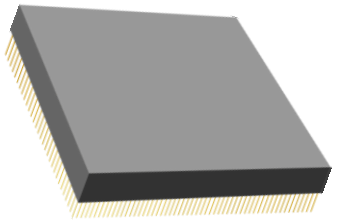
  https://spl-nvlabs.qwiklab.com/focuses/preview/261?locale=en

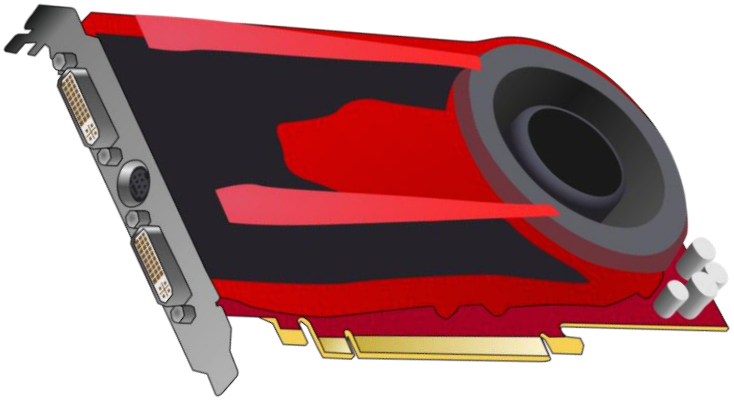
Goals

- Generate and analyze a performance profile of CPU code
- Use debugger to track down and fix fatal GPU bug
- Use debugger to track down and fix nonfatal GPU bug

Preparing to Migrate from CPU to GPU

- Identify bottlenecks that may prevent migration from CPU to GPU
- Identify areas that are suitable for use on GPU

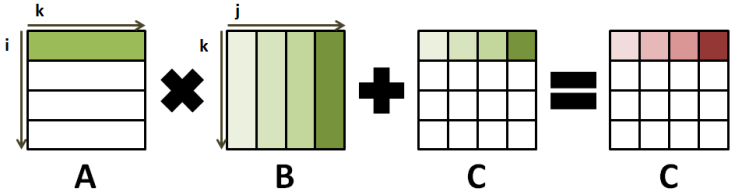




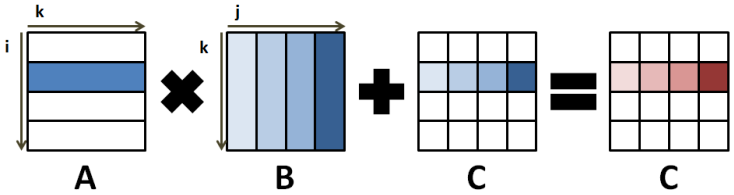
Matrix Multiplication Example

C = A x B + C

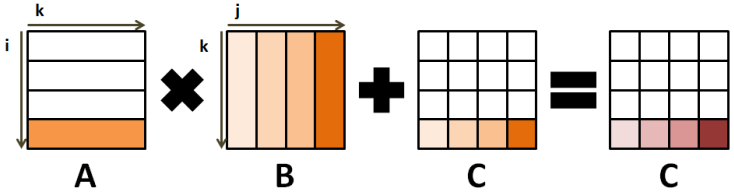
Master process



Slave process 1

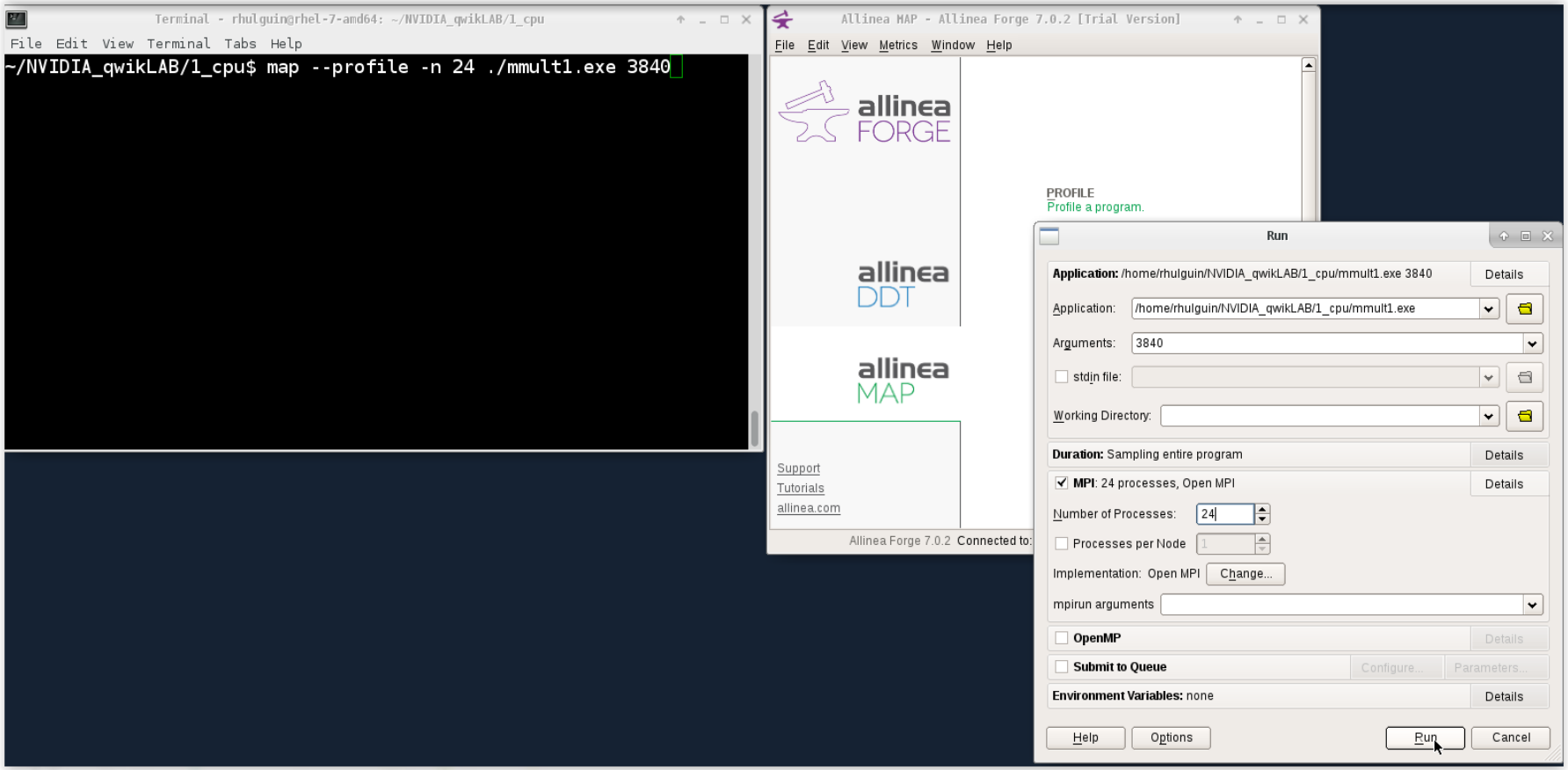


Slave process n-1



Generating a MAP profile

- Run MAP from command line or from the GUI



Compute Analysis

/home/rhulguin/NVIDIA_qwikLAB/1_cpu/mmult1_24p_1n_2017-05-06_03-07.map - Allinea MAP - Allinea Forge 7.0.2 [Trial Version]

File Edit View Metrics Window Help

Profiled: mmult1.exe on 24 processes, 1 node Sampled from: Sat May 6 2017 03:07:30 (UTC+01) for **105.4s** Hide Metrics...

Application activity

CPU memory access 64.6 %

CPU fp vector 4.5 %

22:07:30-22:08:45 (75.790s, 71.9% of total): Main thread compute 67.5 %, MPI 32.5 % Zoom

mmult1.c

```
53  void mmult(int size, int nslices, double *A, double *B, double *C)
54  {
55    for(int i=0; i<size/nslices; i++)
56    {
57      for(int j=0; j<size; j++)
58      {
59        double res = 0.0;
60
61        for(int k=0; k<size; k++)
62        {
63          res += A[i*size+k]*B[k*size+j];
64        }
65
66        C[i*size+j] += res;
67      }
68    }
69  }
```

67.4% <0.1%

Time spent on line 63

Breakdown of the 67.4% time spent on this line:

Executing instructions	100.0%
Calling other functions	0.0%

Time in instructions executed:

Scalar floating-point	0.0%
Vector floating point	6.7%
Scalar integer	1.8%
Vector integer	0.0%
Memory access*	95.8%
Branch	0.2%
Other instructions	0.0%

* 91.2% memory access instructions, 4.6% implicit memory accesses in other instructions, also counted in their categories

Showing data from 17,256 samples taken over 24 processes (719 per process) Allinea Forge 7.0.2 Connected to: (via tunnel) rhel-7-amd64:4201 -> rhel-7-amd64 Main Thread View

allinea Now part of ARM

MPI Analysis

/home/rhulguin/NVIDIA_qwikLAB/1_cpu/mmult1_24p_1n_2017-05-06_03-07.map - Allinea MAP - Allinea Forge 7.0.2 [Trial Version]

File Edit View Metrics Window Help

Profiled: mmult1.exe on 24 processes, 1 node Sampled from: Sat May 6 2017 03:07:30 (UTC+01) for **105.4s** Hide Metrics...

Main thread activity

MPI call duration 12.4 s 48.1 0 AUTO

22:07:56-22:09:15 (79.163s, 75.1% of total): Main thread compute 37.5 %, MPI 62.5 % Zoom

mmult1.c

```
158   if(myrank == 0)
159   {
160     printf("%d: Receiving result matrix...\n", myrank);
161
162     for(int i=1; i<nproc; i++)
163     {
164       MPI_Recv ( &mat_c[slice*i], slice, MPI_DOUBLE, i, 500+i, MPI_COMM_WORLD, &st );
165     }
166   }
167   else
168   {
169     printf("%d: Sending result matrix...\n", myrank);
170
171     MPI_Send ( &mat_c[0], slice, MPI_DOUBLE, 0, 500+myrank, MPI_COMM_WORLD );
172   }
173
174   if(myrank == 0) {...}
182
183   free(mat_a);
184   free(mat_b);
185   free(mat_c);
186
187   MPI_Finalize();
188
189   return 0;
190 }
```

2.5% 20.2% 39.7%

Showing data from 18,024 samples taken over 24 processes (751 per process) Allinea Forge 7.0.2 Connected to: (via tunnel) rhel-7-amd64:4201 -> rhel-7-amd64 Main Thread View

Next Steps

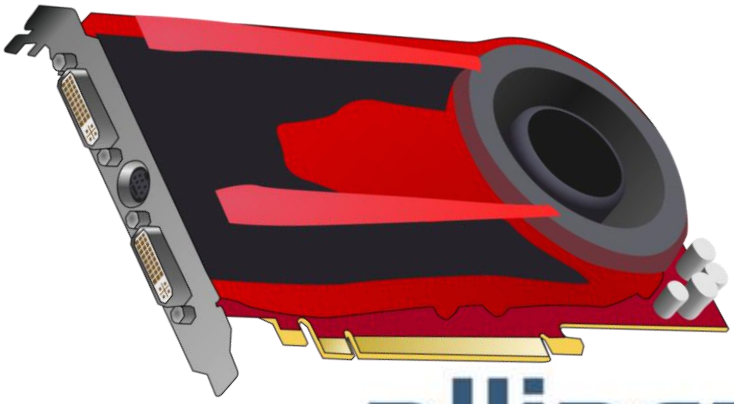
- The next example attempts to write a GPU kernel to perform the matrix multiplication, but introduces a fatal bug
- Allinea DDT can be used to track what is going wrong in this GPU kernel

Fatal Bug

```
Terminal - rhulguin@rhel-7-amd64: ~/NVIDIA_qwikLAB/1_cpu
File  Edit  View  Terminal  Tabs  Help
~/NVIDIA_qwikLAB/2_cuda$ ./mmult2.exe
0: Size of the matrices: 8192x8192
0: Initializing matrices...
0: Sending matrices...
0: Processing...
CUDA error
--------------------------------------------------------------------------
MPI_ABORT was invoked on rank 0 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.
--------------------------------------------------------------------------
~/NVIDIA_qwikLAB/2_cuda$
```

- Let’s smash this bug using Allinea DDT



allinea
Now part of ARM

A More Useful Error Message

Allinea DDT - Allinea Forge 7.0.2 [Trial Version]

File Edit View Control Tools Window Help

Focus on current: Process Thread Step Threads Together Step CUDA threads by: Warp (default)

Threads 1 2 3 K4

CUDA Threads (mmult_kernel) Block 0 25 0 Thread 0 0 0 Go Grid size: 1x64x1 Block size: 64x1x1

mmult2.c mmult2.cu

```
24    int j = blockDim.x*blockIdx.x+threadIdx.x;
25    double res = 0.0;
26
27    // Make sure we don't write any value out of C because the grid can be larger
28    if(i < size/nslices && j < size)
29    {
30      for(int k=0; k<size; k++)
31      {
32        res += A[i*pitch_A_nbelem+k] * B[k*pitch_B_nbelem+j];
33      }
34
35      C[i*pitch_C_nbelem+j] += res;
36    }
37  }
38
39
40  void mmult_cuda(int size, int nslices, double *A, double *B,
```

Program Stopped

Process 0:

Kernel 1 Thread <<<(0,25,0),(0,0,0)>>> stopped in mmult_kernel (mmult2.cu:32) with signal CUDA_EXCEPTION_1 (Lane Illegal 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

Continue Pause

Locals | Current Line(s) | Current Stack | GPU Devices

Current Line(s)

Variable Name	Value
A	0x703ee0000
B	0x703ee8000
i	25
j	0
k	0
pitch_A_nbel...	512
pitch_B_nbel...	512
res	0

Type: none selected

Input/Output | Breakpoints | Watchpoints | Stacks | Kernel Progress View | Tracepoints | Tracepoint Output | Logbook

Input/Output

```
0: Sending matrices...
0: Processing...
```

Note: Allinea DDT can only send input to the mpirun process with this MPI implementation

Type here ('Enter' to send): More

Evaluate

Expression	Value

Ready Connected to: (via tunnel) rhel-7-amd64:4201 -> rhel-7-amd64

allinea Now part of ARM

Where Did Array A (in GPU Kernel) Come From?

Input/Output | Breakpoints | Watchpoints | Stacks | Kernel Progress View | Tracepoints | Tracepoint Output | Logbook

Stacks

Threads	CUDA Threads	Function
1	0	main (mmult2.c:147)
1	0	mmult_cuda (mmult2.cu:80)
1	0	mmult_kernel (mmult2.cu:21)
1	0	__device_stub__Z12mmult_kernelIiPdmS_mS_m (tmpxft_00004854_00000000-4_mmult2.cudafe1.stub.c:10)
1	4096	mmult_kernel (mmult2.cu:20)
1	4096	mmult_kernel (mmult2.cu:32)
1	0	orte_progress_thread_engine

mmult2.c | mmult2.cu | tmpxft_00004854_00000000-4_mmult2.cudafe1.stub.c | cuda_runtime.h [read-only]

```
19  // CUDA version of mmult: size, pitch_A_nbelem, pitch_B_nbelem, pitch_C_nbelem are in number of elements
20  __global__ void mmult_kernel(int size, int nslices, double *A, size_t pitch_A_nbelem, double *B, size_t pitch_B_nbelem, double *C, size_t pitch_C_nbelem)
21  {
```

mmult2.c | mmult2.cu | tmpxft_00004854_00000000-4_mmult2.cudafe1.stub.c | cuda_runtime.h [read-only]

```
79  // Call kernel
80  mmult_kernel<<<grid,block>>>(size, nslices, d_A, pitch_A, d_B, pitch_B, d_C, pitch_C);
81
```

- Using the Stacks view, we can see that array **`A`** comes from the array **`d_A`** in the **`mmult_cuda`** function

How is d_A Allocated?

```
40  void mmult_cuda(int size, int nslices, double *A, double *B, double *C)
41  {
42    double *d_A, *d_B, *d_C;
43    size_t pitch_A, pitch_B, pitch_C;
44    int widthinbytes = size*sizeof(double);
45    int height = size/nslices;
46
47    // Allocate on device
48    CUDA_CHECK(cudaMallocPitch(&d_A, &pitch_A, widthinbytes, height));
49    CUDA_CHECK(cudaMallocPitch(&d_B, &pitch_B, widthinbytes, size));
50    CUDA_CHECK(cudaMallocPitch(&d_C, &pitch_C, widthinbytes, height));
51  #ifdef DEBUG ...f
57
58    // Initialize: host to device transfer
59    CUDA_CHECK(cudaMemcpy2D(
60        d_A, pitch_A,
61        A, widthinbytes,
62        widthinbytes, height, cudaMemcpyHostToDevice
63        ));
```

Stacks

Threads	CUDA Threads	Function
1	0	main (mmult2.c:147)
1	0	mmult_cuda (mmult2.cu:80)
1	0	mmult_kernel (mmult2.cu:21)
1	4096	mmult_kernel (mmult2.cu:20)
1	4096	mmult_kernel (mmult2.cu:32)
1	0	orte_progress_thread_engine
1	0	opal_libevent2021_event_base_loop (event.c:1633)

Evaluate

Expression	Value
i*pitch_A_nbelem+k	<No symbol "i" in current context.>

The **`mmult_cuda`** function is run on the host

`d_A` is allocated on the GPU using **`cudaMallocPitch`**

`d_A` gets values from host array **`A`** using **`cudaMemcpy2D`**

What Does cudaMallocPitch do?

- **`cudaMallocPitch`** is the preferred method for allocating 2D arrays as it pads the data and aligns it for better performance

mmult2.c | mmult2.cu | tmpxft_00004854_00000000-4_mmult2.cudafe1.stub.c | cuda_runtime.h [read-only]

```
40 void mmult_cuda(int size, int nslices, double *A, double *B, double *C)
41 {
42   double *d_A, *d_B, *d_C;
43   size_t pitch_A, pitch_B, pitch_C;
44   int widthinbytes = size*sizeof(double);
45   int height = size/nslices;
46
47   // Allocate on device
48   CUDA_CHECK(cudaMallocPitch(&d_A, &pitch_A, widthinbytes, height));
```

Locals | Current Line(s)

Current Line(s)

Variable Name	Value
d_A	0x703ee0000
height	64
nslices	1
pitch_A	512
size	64
widthinbytes	512

- From the NVIDIA documentation, **`pitch_A`** is the length (in bytes) of the padded row for **`d_A`**
- The allocation looks fine, we must be indexing it improperly

Improper Indexing

mmult2.c | mmult2.cu | tmpxft_00004854_00000000-4_mmult2.cudafe1.stub.c | cuda_runtime.h [read-only]

```
79   // Call kernel
80   mmult_kernel<<<grid,block>>>(size, nslices, d_A, pitch_A, d_B, pitch_B, d_C, pitch_C);
81
```

mmult2.c | mmult2.cu | tmpxft_00004854_00000000-4_mmult2.cudafe1.stub.c | cuda_runtime.h [read-only]

```
19 // CUDA version of mmult: size, pitch_A_nbelem, pitch_B_nbelem, pitch_C_nbelem are in number of elements
20 __global__ void mmult_kernel(int size, int nslices, double *A, size_t pitch_A_nbelem, double *B, size_t pitch_B_nbelem, double *C, size_t pitch_C_nbelem)
21 {
```

```
22   // Set thread indexes
23   int i = blockDim.y*blockIdx.y+threadIdx.y;
24   int j = blockDim.x*blockIdx.x+threadIdx.x;
25   double res = 0.0;
26
27   // Make sure we don't write any value out of C because the grid can be larger
28   if(i < size/nslices && j < size)
29   {
30     for(int k=0; k<size; k++)
31     {
32       res += A[i*pitch_A_nbelem+k] * B[k*pitch_B_nbelem+j];
33     }
```

- We learned from the previous slide that **`pitch_A`** and **`pitch_B`** are length in bytes
- If we want the number elements for indexing purposes, we need to divide by the **`sizeof(double)`**

Edit Within DDT

Allinea DDT - Allinea Forge 7.0.2 [Trial Version]

File Edit View Control Tools Window Help

New Session...
Load Session...
Save Session...
Restart Session...
Change Attached Processes...
End Session
Open Source File... Ctrl+O
Save Source File Ctrl+S
Save Source File As.... Ctrl+Shift+S
Revert to Saved Source File
Close Source File Ctrl+W
Close All Source Files
Build Ctrl+B
Configure Build...
Commit...
Options...
Import Configuration File...
Quit

Step Threads Together | Step CUDA threads by: Warp (default)

Thread | Go

tmpxft_00004854_00000000-4_mmult2.cudafe1.stub.c | cuda_runtime.h [read-only]

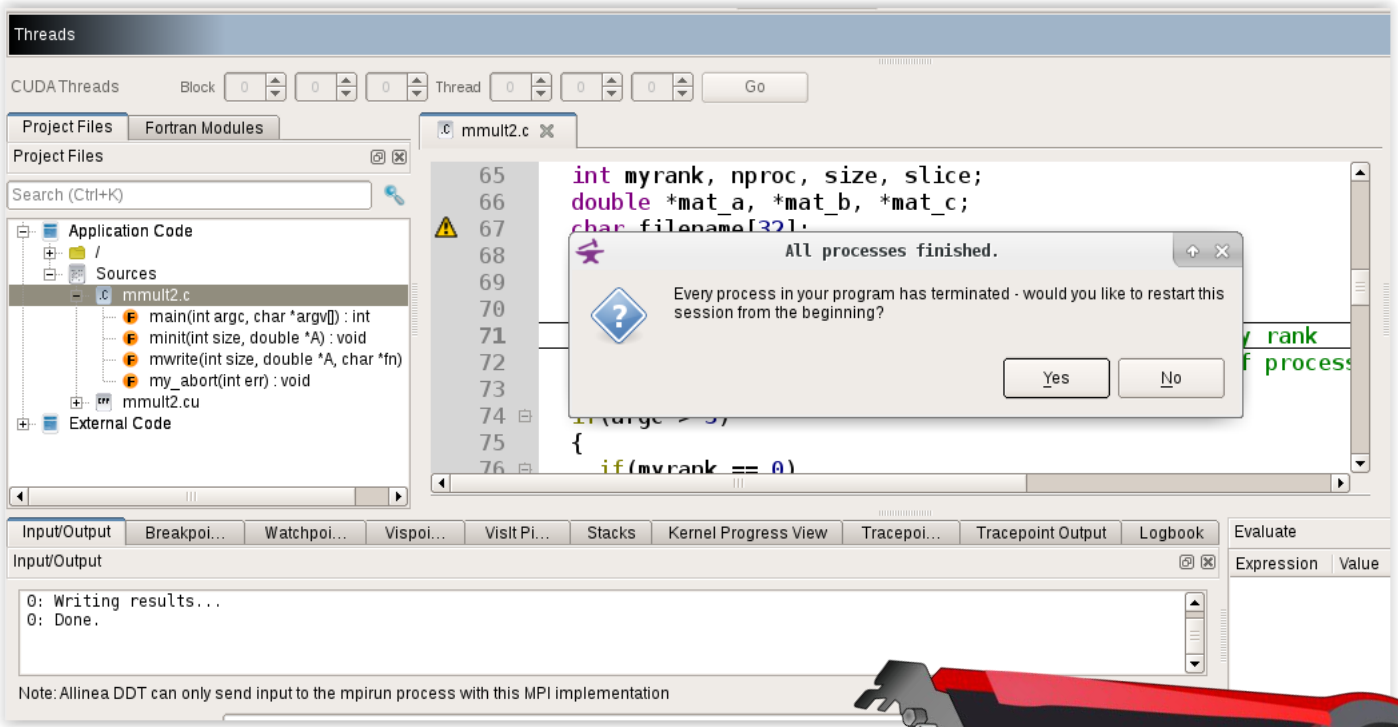
```
mcpy2D(
es,
 size, cudaMemcpyHostToDevice
mcpy2D(
es,
 height, cudaMemcpyHostToDevice
```

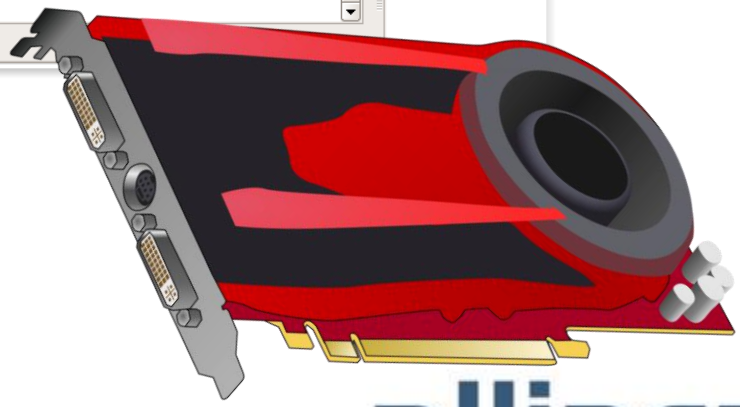
```
75  // Set CUDA grid dimensions
76  dim3 block(BLOCK_X, 1);
77  dim3 grid((size+block.x-1)/block.x,(height+block.y-1)/block.y); // allocate one more block on x or y if size%BLOCK_X!=0
78
79  // Call kernel
80  mmult_kernel<<<grid,block>>>(size, nslices, d_A, pitch_A/sizeof(double),
81                                              d_B, pitch_B/sizeof(double),
82                                              d_C, pitch_C/sizeof(double));
```

Ready Connected to: (via tunnel) rhel-7-amd64:4201 -> rhel-7-amd64

allinea
Now part of ARM

Smash that Bug





allinea
Now part of ARM

Further Optimization

- The next example attempts to improve performance further by moving data into shared GPU memory

- This time a nonfatal bug is introduced where the solution is incorrect
- Allinea DDT can help track this bug down

Track Data Before and After Calculation Loop

- Click **`Run to here`** on the line right before the calculation is stored

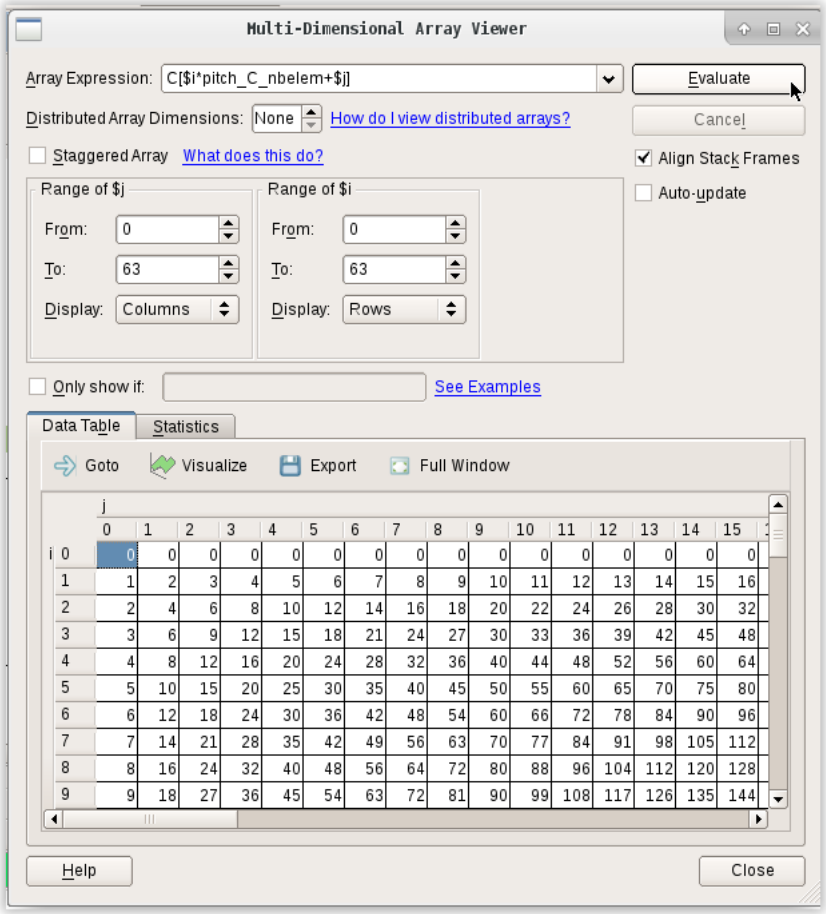
mmult3.c | mmult3.cu

```
19  // CUDA version of mmult: size, pitch_A_nbelem, pitch_B_nbelem, pitch_C_nbelem are in number of elements
20  __global__ void mmult_kernel(int size, int nslices, double *A, size_t pitch_A_nbelem, double *B, size_t pitch_B_nbelem,
21  {
22    // Set thread indexes
23    int i = blockDim.y*blockIdx.y*2+threadIdx.y; // each thread computes 2 values on y
24    int j = blockDim.x*blockIdx.x+threadIdx.x;
25    int tx = threadIdx.x;
26    __shared__ double sh_A1[BLOCK_X], sh_A2[BLOCK_X]; // declare shared memory to store values of A
27    double res1 = 0.0f, res2 = 0.0f;
28
29    // Make sure we don't write any value out of C because the grid can be larger
30    if(i < size/nslices && j < size)
31    {
32      // In each CUDA block
33      for(int ks=0; ks<size; ks=ks+BLOCK_X)
34      {
35        // Each thread stores 2 values of A in shared memory
36        sh_A1[tx] = A[i*pitch_A_nbelem+ks+tx];
37        sh_A2[tx] = A[(i+1)*pitch_A_nbelem+ks+tx];
38        // Make sure A is stored in shared memory
39        __syncthreads();
40        for(int k=ks; k<ks+BLOCK_X; k++) {...
46        }    // Make sure all threads have computed res1 and res2
47        // Make sure all threads have computed res1 and res2
48        __syncthreads();
49      }
50
51      // Each thread stores 2 result in C
52      C[i*pitch_C_nbelem+j] += res1;
53      C[i+1*pitch_C_nbelem+j] += res2;
54    }
55  }
56
```

Add breakpoint
Add tracepoint (C, i, pitch_C_nbelem, j, res1)
Run to here
Undo Ctrl+Z
Redo Ctrl+Shift+Z
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete
Select All
Open in external editor
Close Ctrl+W

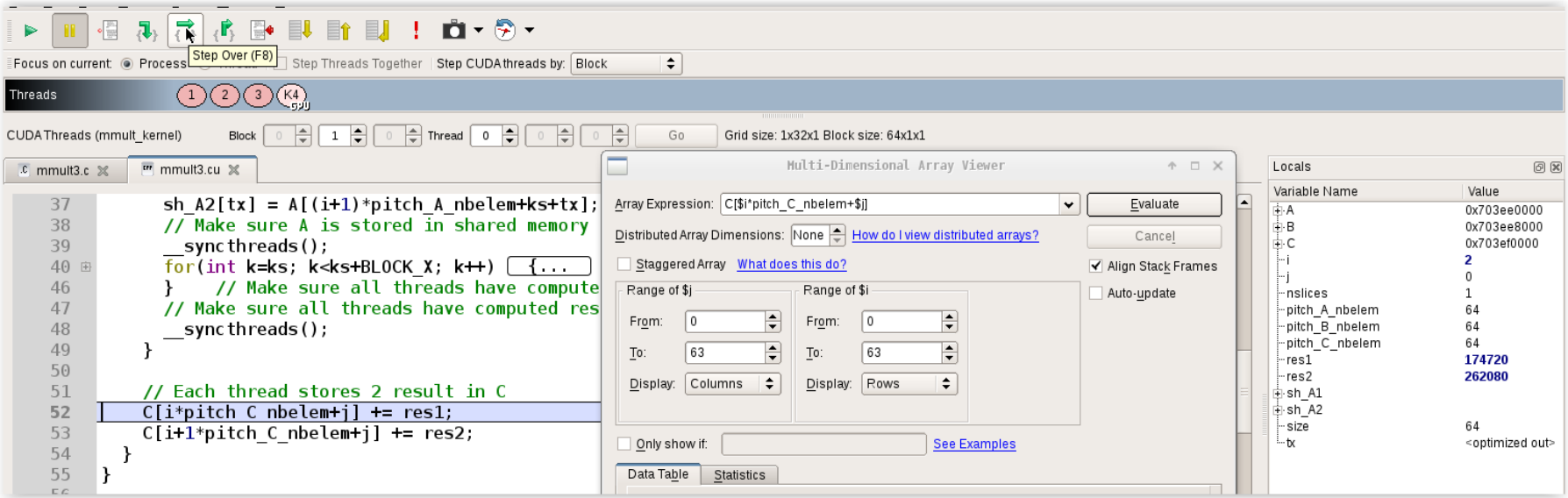
Ready Connected to: (via tunnel) rhel-7-amd64:4201 -> rhel-7-amd64

Set Parameters for Multi-Dimensional Array Viewer



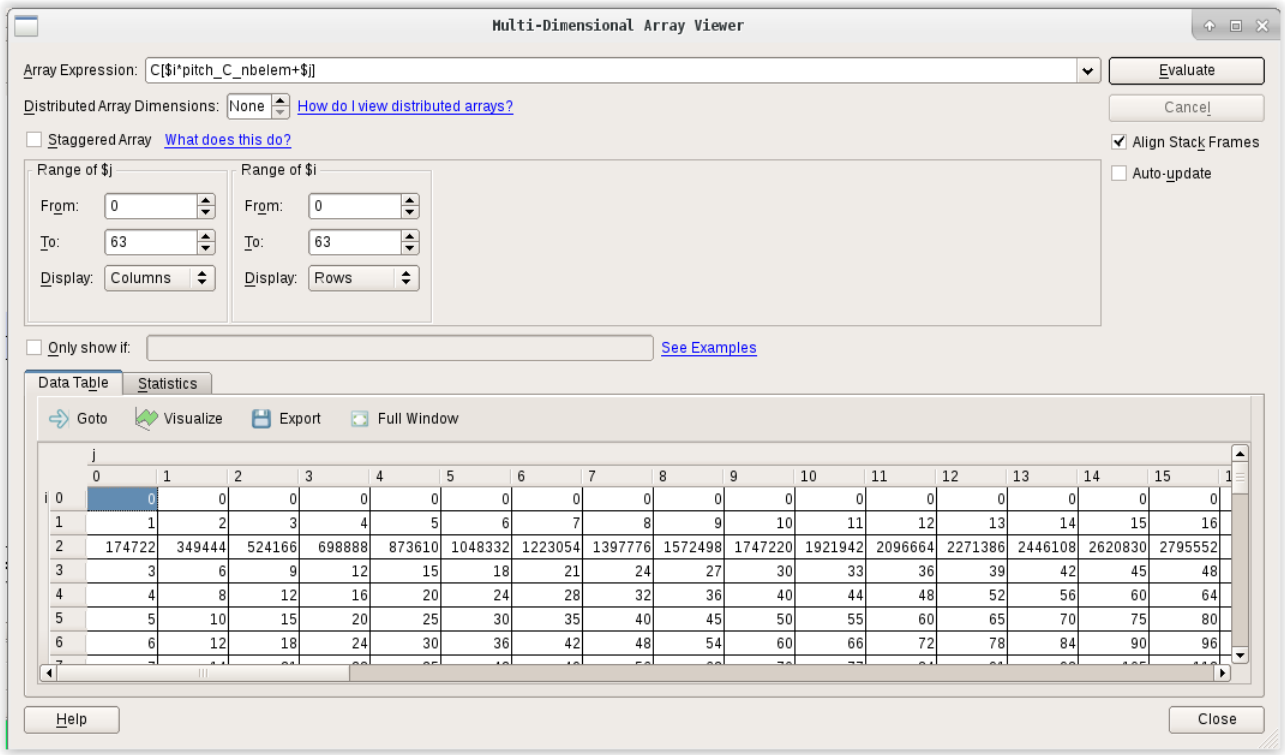
- Modify subscripts **`i`** and **`j`** and place **`$`** in front of them
- Set the range from 0 to 63
- Click Evaluate

Select Block 1



- Select Thread 0 of Block 1 and Click Go
- Since **`i=2`**, we expect row 2 of the array to be updated
- Click Step Over to execute line 52

Multidimensional Array Viewer Shows Exact Changes



- Click Evaluate to update the array viewer
- Row 2 updated as expected
- Click Step Over again and update the array viewer

allinea
Now part of ARM

Wrong Row Updated

CUDA Threads (mmult_kernel) Block 0 1 0 Thread 0 0 0 Go Grid size: 1x32x1 Block size: 64x1x1

mmult3.c | mmult3.cu

```
51      // Each thread stores 2 result in C
52      C[i*pitch_C_nbelem+j] += res1;
53      C[i+1*pitch_C_nbelem+j] += res2;
54    }
55  }
56
57
58  voi
59  {
60    d
61    s
62    i
63    i
64
65    /
66    C
67    C
68    C
69  #if
70    /
71    C
72    C
```

Multi-Dimensional Array Viewer

Array Expression: C[$i*pitch_C_nbelem+$j]

Evaluate

Distributed Array Dimensions: None How do I view distributed arrays?

Cancel

Staggered Array What does this do?

Align Stack Frames

Auto-update

Range of $j: From: 0 To: 63 Display: Columns

Range of $i: From: 0 To: 63 Display: Rows

Only show if: See Examples

Data Table | Statistics

Goto Visualize Export Full Window

i \ j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	1	2	262083	524164	786245	1048326	1310407	1572488	1834569	2096650	2358731	2620812	2882893	3144974	34
2	16685762	17122564	524166	698888	873610	1048332	1223054	1397776	1572498	1747220	1921942	2096664	2271386	2446108	26
3	3	6	9	12	15	18	21	24	27	30	33	36	39	42	
4	4	8	12	16	20	24	28	32	36	40	44	48	52	56	

Kernel Progress View

Kernel
mmult_kernel

- It appears that we forgot a pair of parentheses at line 53

Correct the Instruction Used to Update the Array

Threads 1 2 3 K4

CUDA Threads (mmult_kernel) Block 0 1 0 Thread 0 0 0 Go Grid size: 1x32x1 Block size: 64x1x1

mmult3_sol.c | mmult3_sol.cu

```
48        __syncthreads();
49      }
50
51      // Each thread stores 2 result in C
52      C[i*pitch_C_nbelem+j] += res1;
53      C[(i+1)*pitch_C_nbelem+j] += res2;
54    }
55  }
56
57
58  void mmult_cuda(int size, int nslices, doul
59  {
60    double *d_A, *d_B, *d_C;
61    size_t pitch_A, pitch_B, pitch_C;
62    int widthinbytes = size*sizeof(double);
63    int height = size/nslices;
```

Input/Output | Breakpoints | Watchpoints | Stacks | Kernel Progress View | Trace

Input/Output

```
0: Size of the matrices: 64x64
0: Initializing matrices...
0: Sending matrices...
0: Processing...
```

Multi-Dimensional Array Viewer

Array Expression: C[$i*pitch_C_nbelem+$j] Evaluate

Distributed Array Dimensions: None How do I view distributed arrays? Cancel

Staggered Array What does this do? Align Stack Frames

Auto-update

Range of $j: From: 0 To: 63 Display: Columns

Range of $i: From: 0 To: 63 Display: Rows

Only show if: See Examples

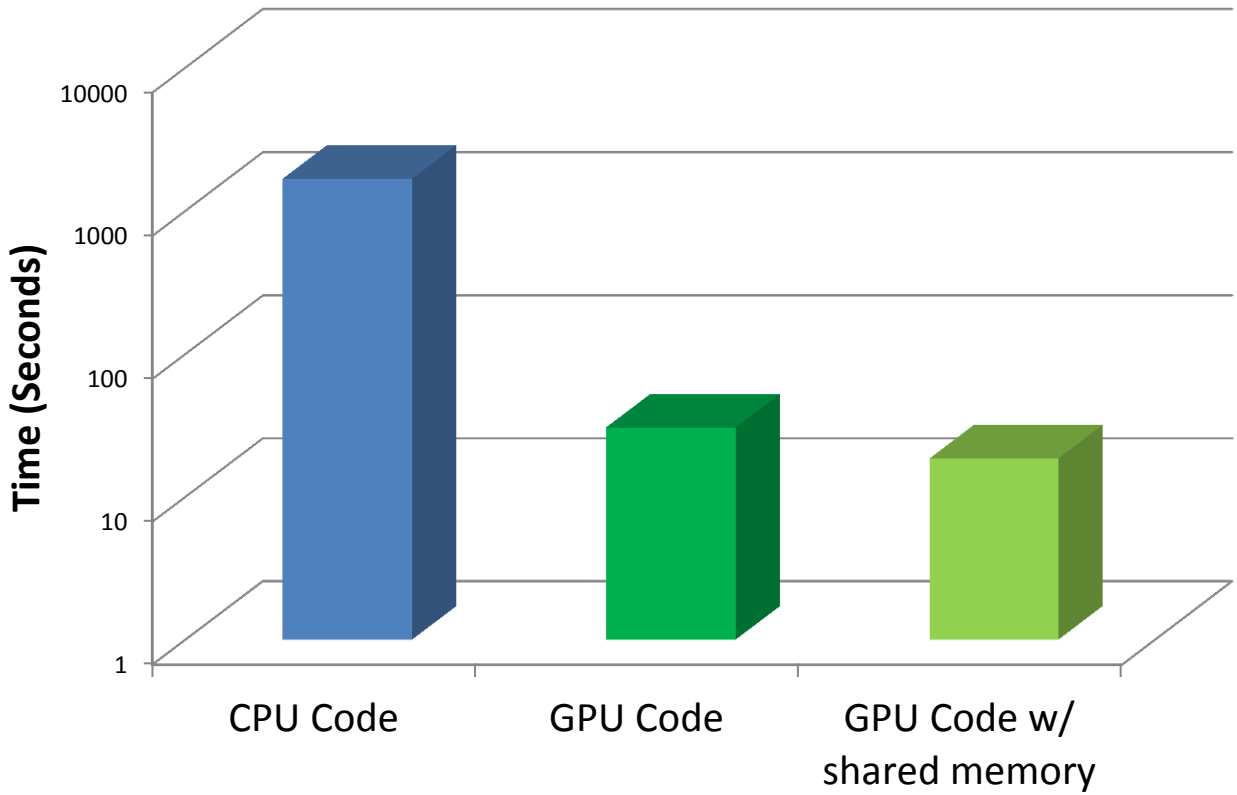
Data Table | Statistics

Goto Visualize Export Full Window

i \ j	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	
1	1	2	3	4	5	6	7	8	
2	174722	349444	524166	698888	873610	1048332	1223054	1397776	157
3	262083	524166	786249	1048332	1310415	1572498	1834581	2096664	235
4	4	8	12	16	20	24	28	32	
5	5	10	15	20	25	30	35	40	
6	6	12	18	24	30	36	42	48	

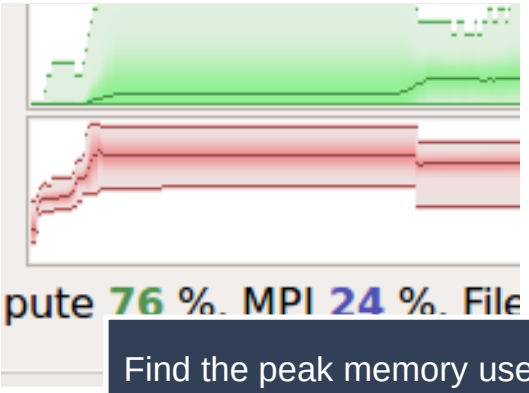
- The behavior is now correct
- Let's compare the performance of the optimized versions

Differences in Runtime

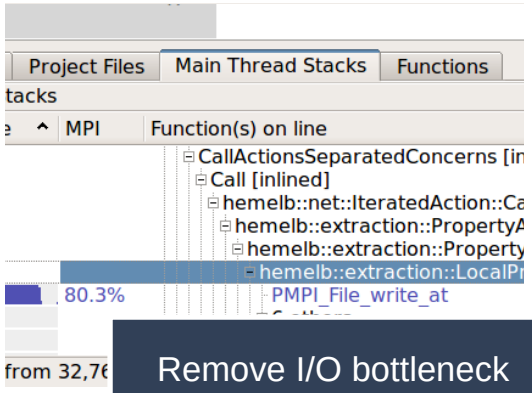


Timings were generated on a problem size of 7680 on
Dual Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz
Single Tesla K80

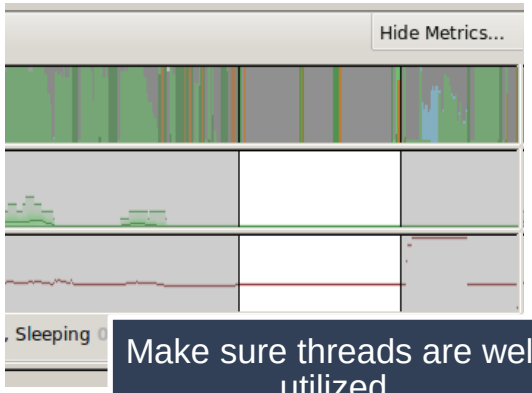
Great Things to Try with Allinea MAP



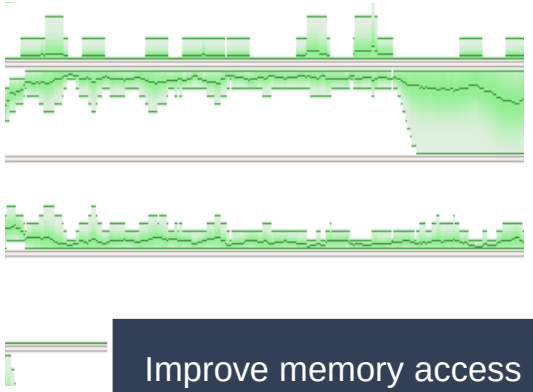
Find the peak memory use



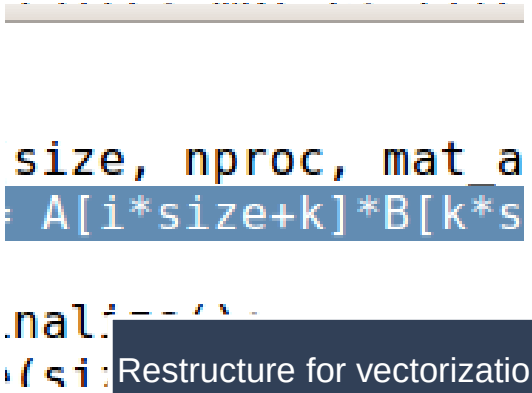
Remove I/O bottleneck



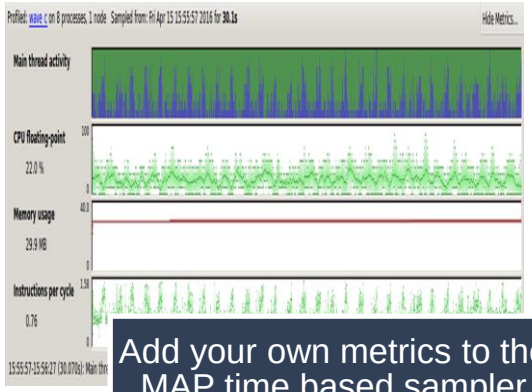
Make sure threads are well utilized



Improve memory access



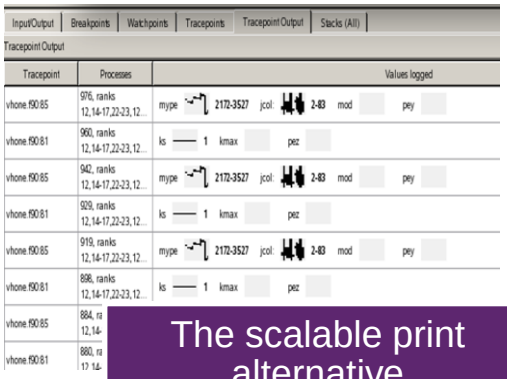
Restructure for vectorization



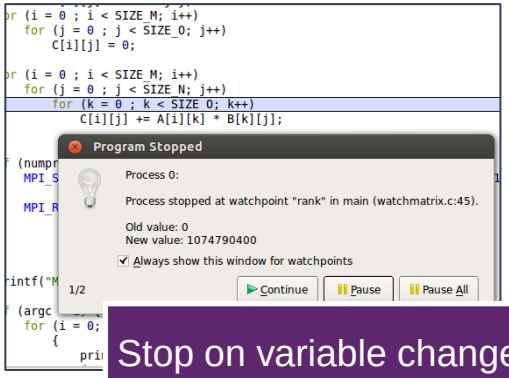
Add your own metrics to the MAP time based sampler

allinea
Now part of ARM

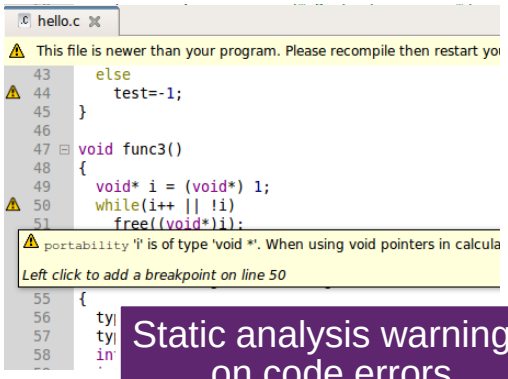
Great things to try with Allinea DDT



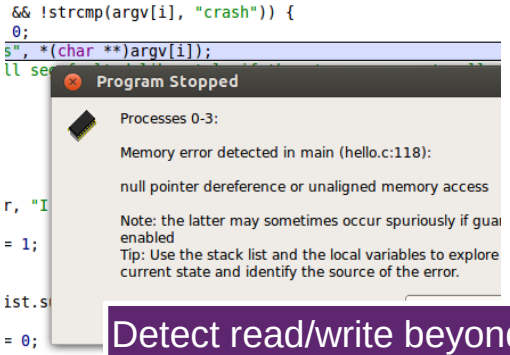
The scalable print alternative



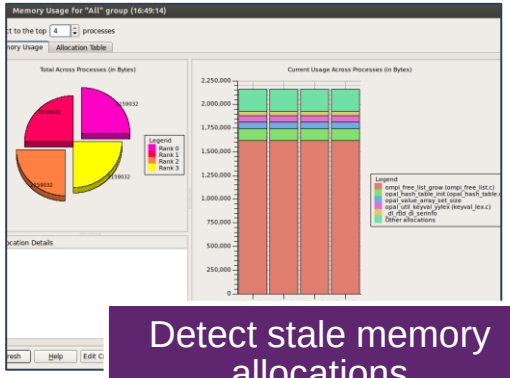
Stop on variable change



Static analysis warnings on code errors



Detect read/write beyond array bounds



Detect stale memory allocations

allinea
Now part of ARM

This session will be gathering major CUDA Developer Tools vendors, including NVIDIA and PGI to share their latest feature development.

David Lecomber - Senior Director, HPC Tools, ARM – will be taking part in this event

Tuesday, May 9, 2:00 PM - 4:00 PM – Hilton Market

allinea
Now part of ARM

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

Q&A and Wrap-up

allinea FORGE
allinea DDT
allinea MAP
allinea PERFORMANCE REPORTS

allinea
Now part of ARM

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

Thank you!

Any questions, feel free to ask.

allinea FORGE
allinea DDT
allinea MAP
allinea PERFORMANCE REPORTS