

Intro to Profiling

Linaro Forge on Baskerville
University of Birmingham

Why should you profile your code?

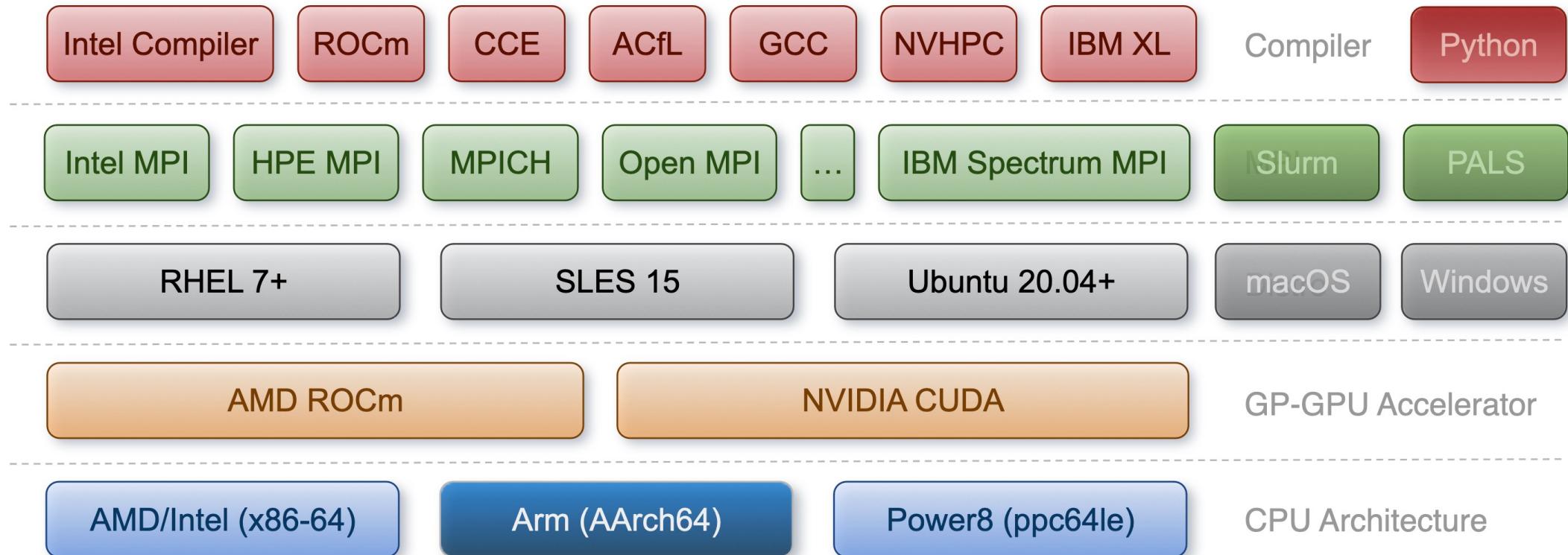
- Analyse the performance of your code
 - measure the time it takes to run
 - monitor the resources (CPU, GPU, memory, I/O) it consumes
- Discover bottlenecks and optimise your application
- Improve efficiency and consume less power!

Profiling tools on Baskerville

- Linaro Forge
 - Interoperable toolkit for debugging HPC applications
- Valgrind
 - Memory debugging and profiling
- Nsight compute
 - Nvidia CUDA applications

Linaro Forge

Supported Platforms



Linaro Forge

BASKERVILLE

Sign in to your account

Username or email

Password

[New User / Forgot Password?](#)

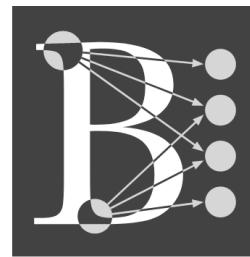
[Sign In](#)

Or sign in with (do not use for first time login)

[University of Birmingham](#) [CILogon Federated Identity](#)

[Alan Turing Institute](#) [ORCID](#)

<https://portal.baskerville.ac.uk/>



The Baskerville portal p

ccess to the Baskerville Tier 2 system

JupyterLab

GUIs

CST Studio Suite

Fiji

Linaro-Forge

RELION

This service is operated by Advanced Research Computing at the University of Birmingham and is funded by EPSRC
Grant EP/T022221/1

Linaro-Forge

This app will launch a [Linaro-Forge](#) within a VNC server on the [Baskerville cluster](#).

Forge version

23.0.1

This defines the version of Linaro-Forge you want to load.

Number of hours

1

Number of GPUs

0

Select 1 GPU for quick debugging and profiling and 0 GPUs for examining jobs requiring more resources using the "Submit to Queue" option

Baskerville Project

wongj-bham-training

Please select the Baskerville Project to which the job will be attached.

Queue

bham

Please select the Queue/QoS on which your job will run.

I would like to receive an email when the session starts

Launch

* The Linaro-Forge session data for this session can be accessed under the [data root directory](#).

- Select 1 GPU to debug a program quickly
- Select 0 GPUs to submit a job to the SLURM queue and debug with as many resources as required

Session was successfully created.



[Home](#) / My Interactive Sessions

Interactive Apps

JupyterLab

GUIs

CST Studio Suite

Fiji

Linaro-Forge

RELION

Linaro-Forge (529739)

1 node | 4 cores | Running

Host: [>_bask-pg0308u26a.cluster.baskerville.ac.uk](https://bask-pg0308u26a.cluster.baskerville.ac.uk)

Delete

Created at: 2023-11-24 12:01:25 GMT

Time Remaining: 59 minutes

Session ID: [c34755ab-32a0-4017-8222-301ab6d6cb6f](#)

Compression



0 (low) to 9 (high)

Image Quality



0 (low) to 9 (high)

[Launch Linaro-Forge](#)

[View Only \(Share-able Link\)](#)

Interactive Apps [Sandbox]

JupyterLab

JupyterLab

GUIs

Linaro-Forge

DDT



Linaro Forge

Linaro
DDTLinaro
MAP**RUN**

Run and debug a program.

ATTACH

Attach to an already running program.

OPEN CORE

Open a core file from a previous run.

MANUAL LAUNCH (ADVANCED)

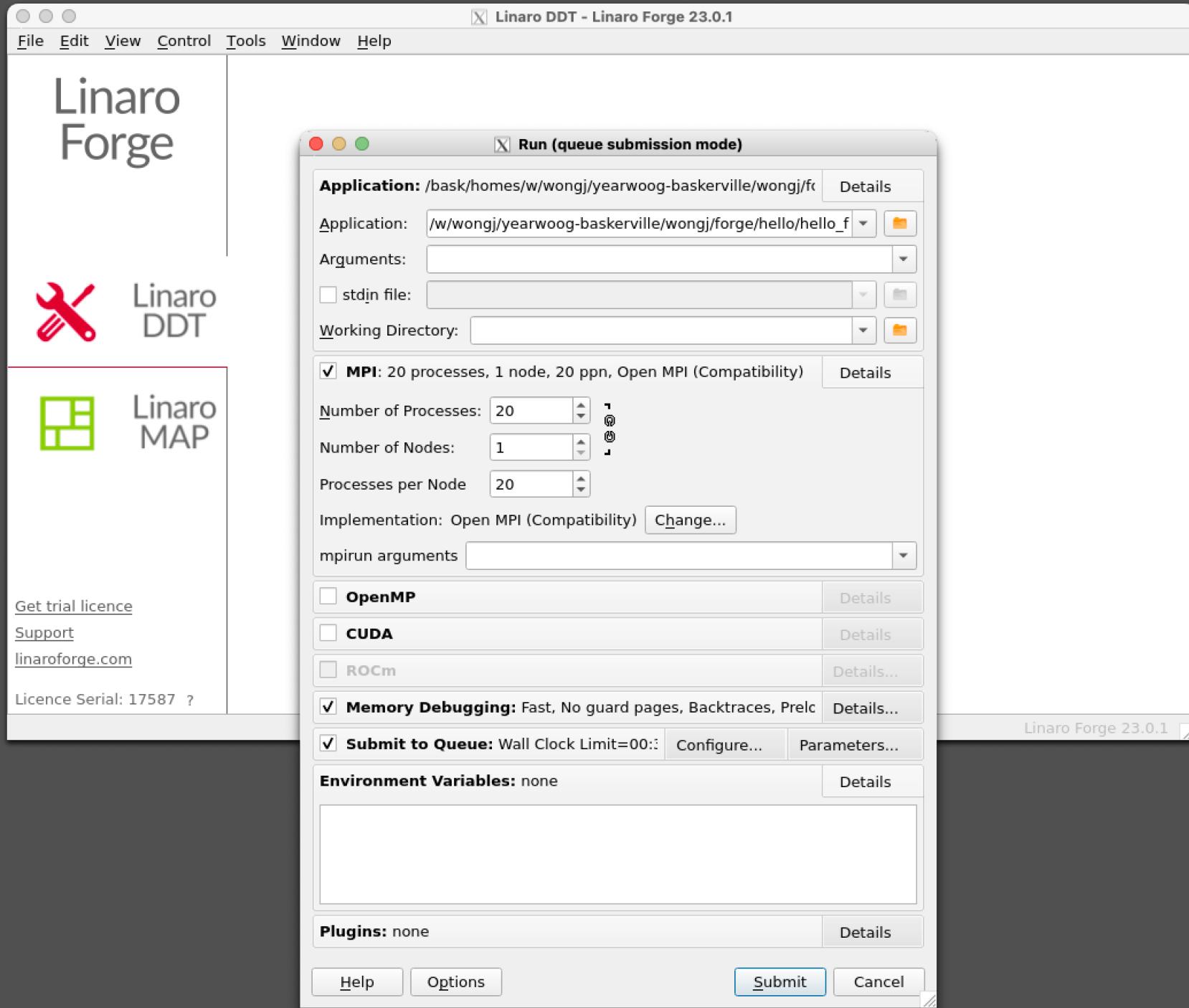
Manually launch the backend yourself.

OPTIONS

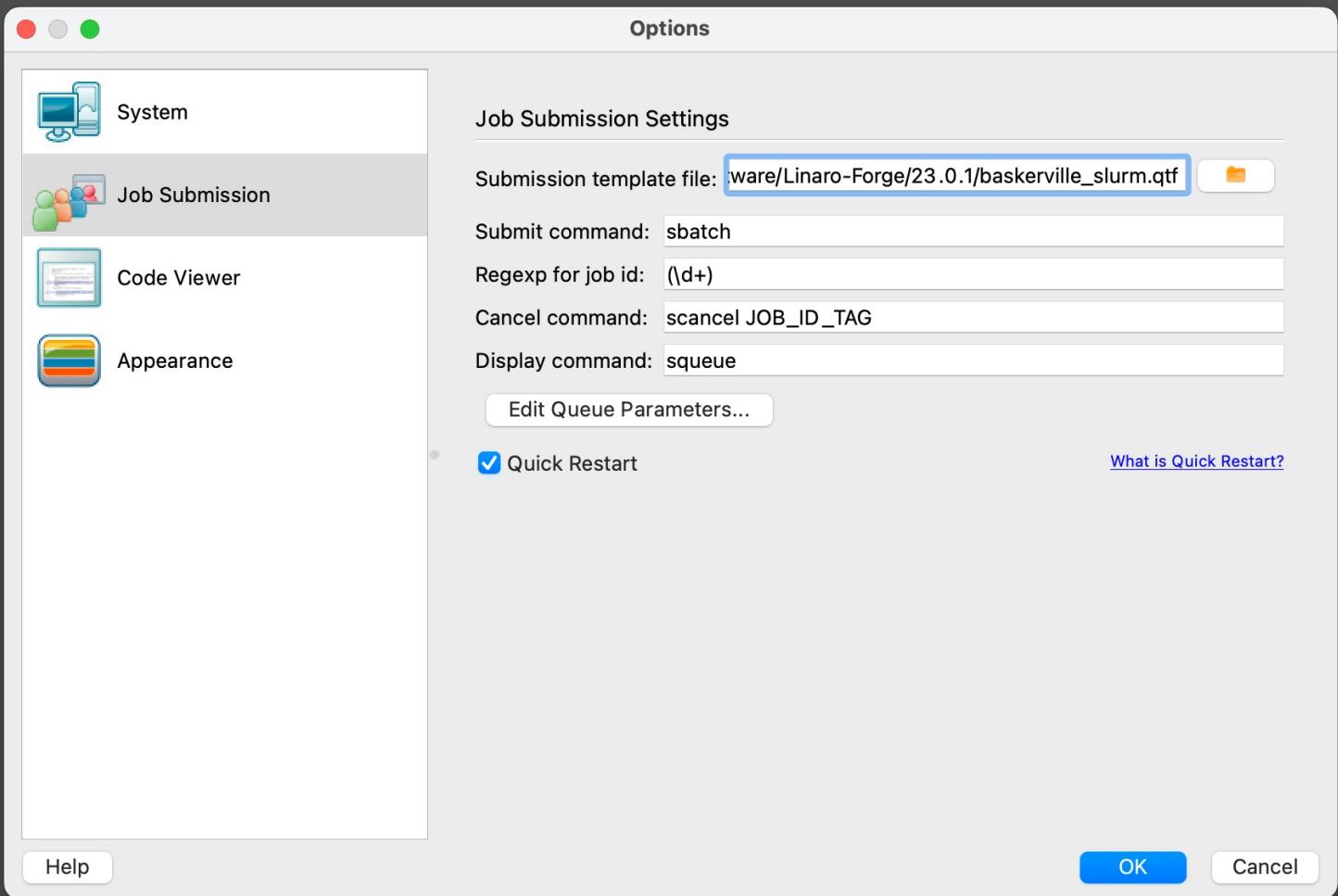
Remote Launch:

QUIT[Get trial licence](#)[Support](#)[linaroforge.com](#)

Licence Serial: 17587 ?

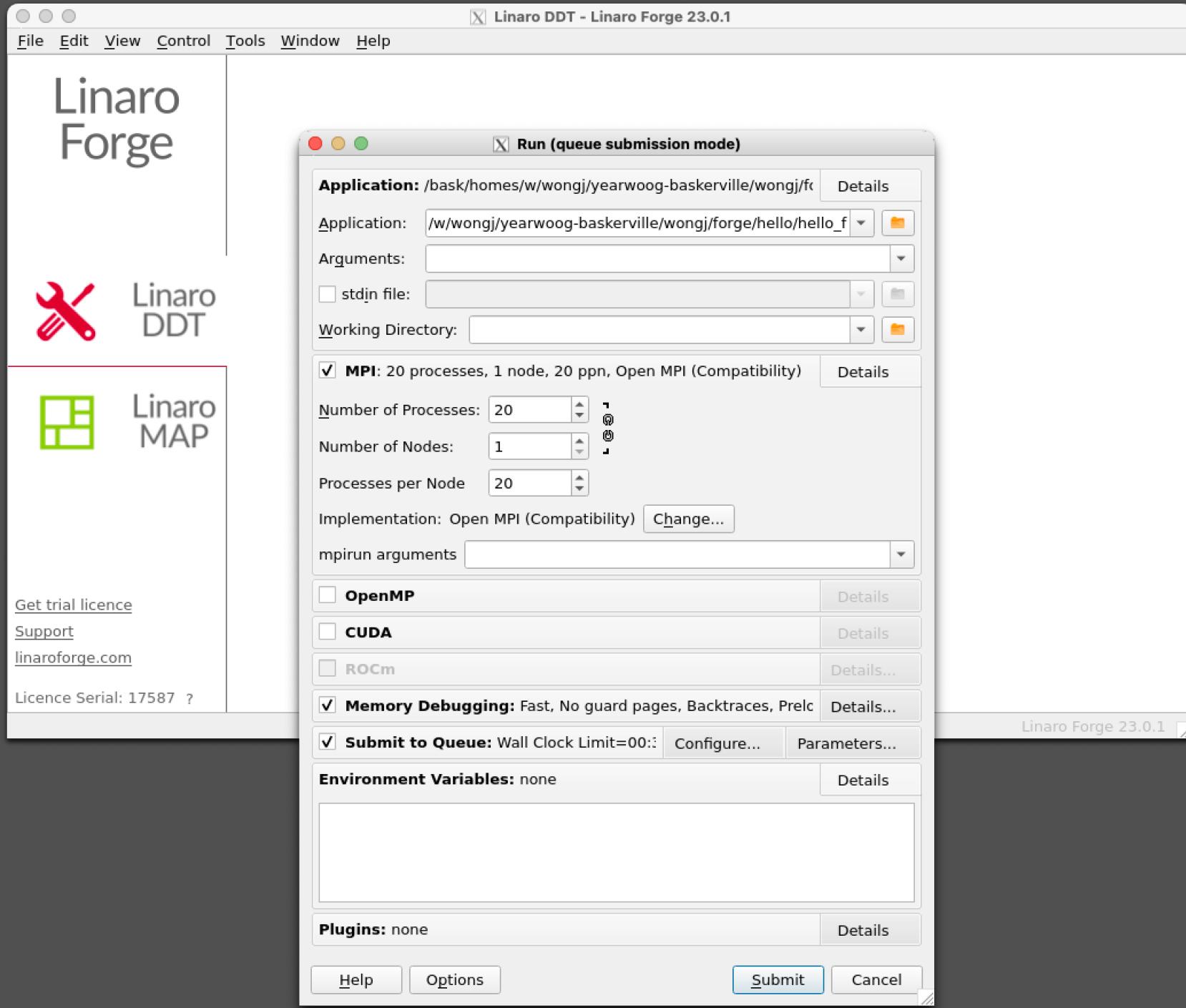


- Check the **MPI** box and set the options for **20 processes, 1 node and 20 processes per node**
- Ensure that the Implementation is **Open MPI (Compatibility)**, otherwise switch to this by clicking **Change...**
- Check the **Memory Debugging** box



- Submit the profiling job to SLURM by checking the option **Submit to Queue**
- (One-time only) Configure the job by clicking **Configure...** and then in the **Job Submission** tab on the left-hand side, provide a **Submission template file** by selecting

/bask/apps/live/EL8-ice/software/Linaro-Forge/23.0.1-foss-2022a/baskerville_slurm.qtf



- Configure the queue parameters by clicking the **Parameters...** button, which allows you to enter the **Wall Clock Limit**, **Queue**, **Account** and **GPUs** required for the job, and then click **OK**
- Click **Submit** to submit your job to SLURM

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

All 0 1 2 3 4 5 6 7 Create Group

Project Files Search (⌘K)

Application Code Sources mmult.c main(argc,c) minit(int sz,dou mmult(int sz,in mwriteln sz,d) External Code

mmu... 59 } 60 fclose(f); 61 } 62 63 64 65 void mmult(int sz, int nslices, double *A, double *B, double *C) 66 { 67 for(int i=0; i<sz/nslices; i++) 68 { 69 for(int j=0; j<sz; j++) 70 { 71 double res = 0.0; 72 73 for(int k=0; k<sz; k++) 74 { 75 res += A[i*sz+k]*B[k*sz*j]; 76 } 77 78 C[i*sz+j] += res; 79 } 80 } 81 82 83 84 int main(int argc, char *argv[]) 85 { 86 int mr, nproc, sz, slice; 87 double *mat_a, *mat_b, *mat_c; 88 char filename[32]; 89 int remainder; 90 MPI_Status st; 91 92 MPI_Init (&argc, &argv);

Locals Current Line(s) Current Stack

Name	Value
res	0
> A	0x7ffffd09db000
i	0
sz	1024
k	953
> B	0x7ffffd01db000
j	27

Processes 1-7:

Process stopped in mmult (mmult.c:75) with signal SIGSEGV (Segmentation fault).

Reason/Origin: invalid permissions for mapped object

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

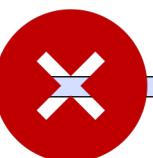
1/2 Continue Pause Pause All

Input/Output Breakpoints Watchpoints Stacks (All) Tracepoints Tracepoint Output Logbook Build Output

Input/Output

0: Size of the matrices: 1024x1024
0: Initializing matrices...
4: Receiving matrices...
0: Sending matrices...
1: Processing...
2: Processing...
3: Processing...
4: Processing...
5: Processing...
6: Processing...
0: Processing...
7: Processing...

Type here ('Enter' to send): More Ready



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

All 0 1 2 3 4 5 6 7 Create Group

Project Files

Search (%K)

Application Code

Sources mmult.c

External Code

.c mmult.c

```
59    }
60    fclose(f);
61 }
62
63
64
65 void mmult(int sz, int nslices, double *A, double *B, double *C)
66 {
67     for(int i=0; i<sz/nslices; i++)
68     {
69         for(int j=0; j<sz; j++)
70         {
71             double res = 0.0;
72
73             for(int k=0; k<sz; k++)
74             {
75                 res += A[i*sz+k]*B[k*sz*j];
76             }
77
78             C[i*sz+j] += res;
79         }
80     }
81
82
83
84 int main(int argc, char *argv[])
85 {
86     int mr, nproc, sz, slice;
87     double *mat_a, *mat_b, *mat_c;
88     char filename[32];
89     int remainder;
90     MPI_Status st;
91
92     MPI_Init (&argc, &argv);
```

Locals Current Line(s) Current Stack

Name	Value
res	0
> A	0x7fff0d09db000
i	0
sz	1024
k	953
> B	0x7fff0d01db000
j	27

Input/Output Breakpoints | Watchpoints | Stacks (All) | Tracepoints | Tracepoint Output | Logbook | Build Output

Input/Output

```
0: Size of the matrices: 1024x1024
0: Initializing matrices...
4: Receiving matrices...
0: Sending matrices...
1: Processing...
2: Processing...
3: Processing...
4: Processing...
5: Processing...
6: Processing...
0: Processing...
7: Processing...
```

Type here ('Enter' to send): More ▾

Evaluate

Name	Value
------	-------

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

All 0 1 2 3 4 5 6 7 Create Group

Project Files mmult.c

This file has been edited.

```
59 }
60 fclose(f);
61 }
62 }
63 }
64 }
65 void mmult(int sz, int nslices, double *A, double *B, double *C)
66 {
67     for(int i=0; i<sz/nslices; i++)
68     {
69         for(int j=0; j<sz; j++)
70         {
71             double res = 0.0;
72
73             for(int k=0; k<sz; k++)
74             {
75                 res += A[i*sz+k]*B[k*sz+j];
76             }
77
78             C[i*sz+j] += res;
79         }
80     }
81 }
82
83
84 int main(int argc, char *argv[])
85 {
86     int mr, nproc, sz, slice;
87     double *mat_a, *mat_b, *mat_c;
88     char filename[32];
89     int remainder;
90     MPI_Status st;
```

Locals Current Line(s) Current Stack

Name	Value
res	0
> A	0x7ffd09db000
i	0
sz	1024
k	953
> B	0x7ffd01db000
j	27

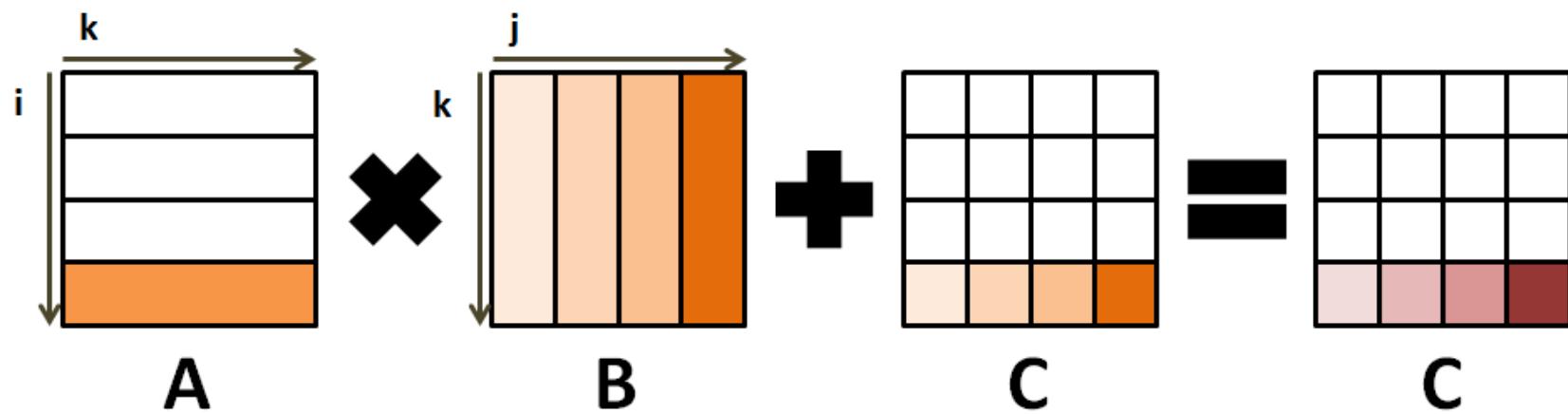
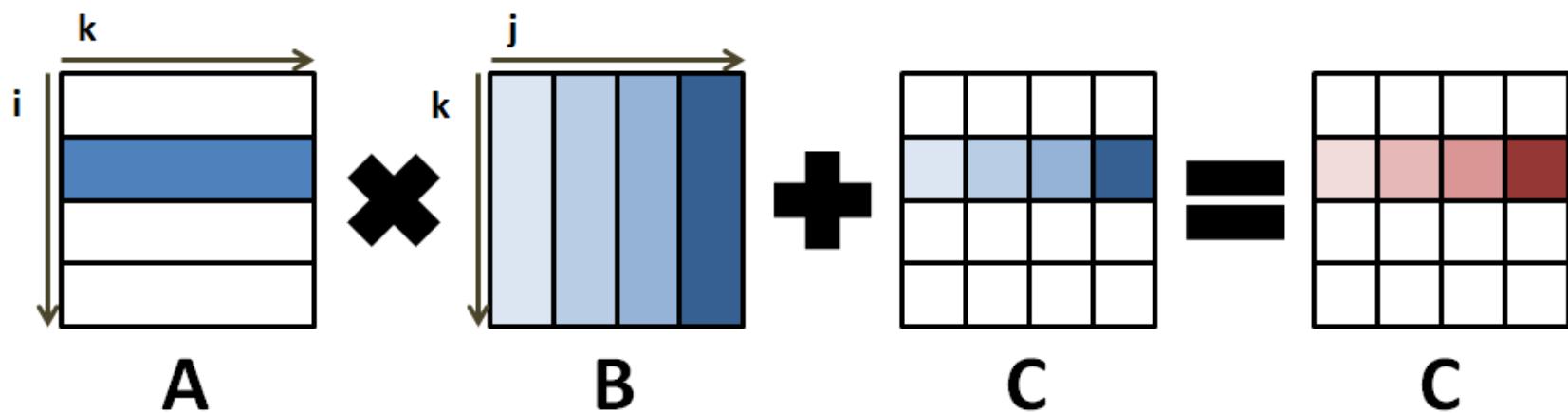
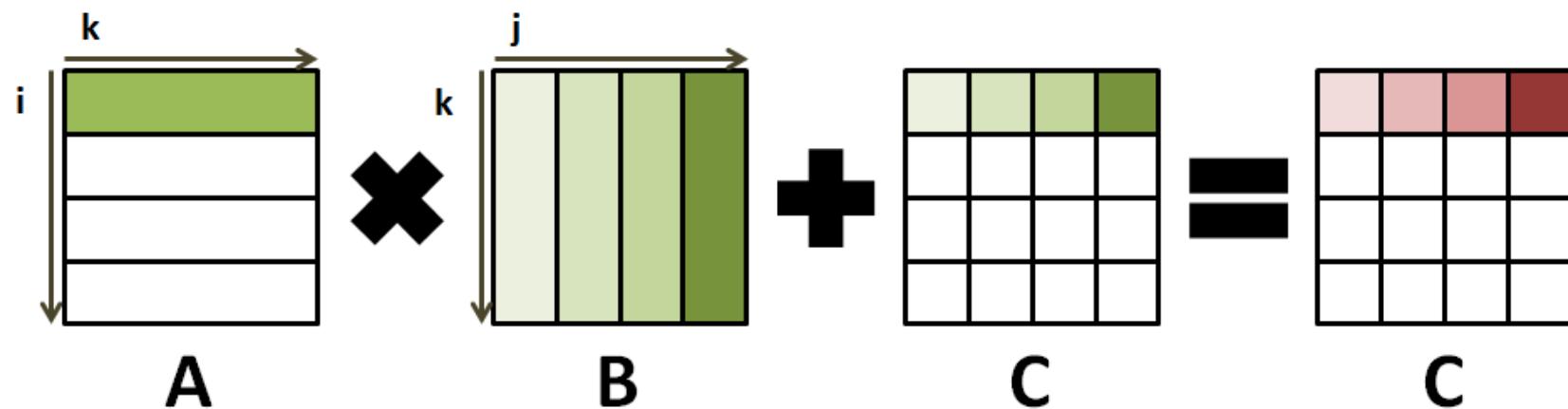
Input/Output Breakpoints Watchpoints Stacks (All) Tracepoints Tracepoint Output Logbook Build Output Evaluate

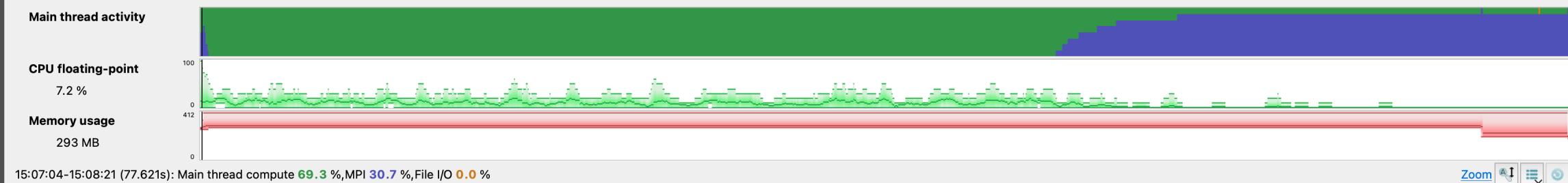
0: Size of the matrices: 1024x1024
0: Initializing matrices...
4: Receiving matrices...
0: Sending matrices...
1: Processing...
2: Processing...
3: Processing...
4: Processing...
5: Processing...
6: Processing...
0: Processing...
7: Processing...

Type here ('Enter' to send): More Ready

MAP

Example: Matrix Multiplication





mmult_corre...

```

63
64
65 v void mmult(int sz, int nslices, double *A, double *B, double *C)
66 {
67 v   for(int i=0; i<sz/nslices; i++)
68 {
69 v   for(int j=0; j<sz; j++)
70 {
71     double res = 0.0;
72
73 v     for(int k=0; k<sz; k++)
74 {
75       res += A[i*sz+k]*B[k*sz+j];
76     }
77
78     C[i*sz+j] += res;
79   }
80 }
81
82
83
84 v int main(int argc, char *argv[])
85 {
86   int mr, nproc, sz, slice;
87   double *mat_a, *mat_b, *mat_c;
88   char filename[32];

```

Time spent on line 75

Breakdown of the 66.0% time spent on this line:

- Executing instructions 100.0%
- Calling functions 0.0%

Time in instructions executed:

Instruction Type	Percentage
Scalar floating-point	10.9%
Vector floating point	0.0%
Scalar integer	3.4%
Vector integer	0.0%
Memory access*	85.5%
Branch	0.0%
Other instructions	3.6%

* 82.1% memory access instructions, 3.4% implicit memory accesses in other instructions, also counted in their categories

Total core time

MPI

Function(s) on line

Source

Main Thread Stacks

Position

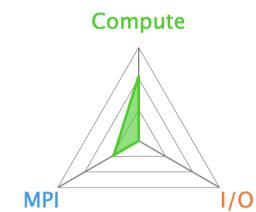
Library

Total core time	MPI	Function(s) on line	Source	Position	Library
66.0%		mmult_c_correct [program]			
2.5%		main			
<0.1%		mmult	{		
			mmult(sz, nproc, mat_a, mat_b, mat_c);	mmult_correct.c:85	mmult_c_correct
			res += A[i*sz+k]*B[k*sz+j];	mmult_correct.c:177	mmult_c_correct
			for(int k=0; k<sz; k++)	mmult_correct.c:75	mmult_c_correct
				mmult_correct.c:73	mmult_c_correct
24.9%		MPI_Send	MPI_Send (&mat_c[0], slice, MPI_DOUBLE, 0, 500+mr, MPI_COMM_WORLD);	mmult_correct.c:192	mmult_c_correct
5.5%		MPI_Finalize	MPI_Finalize();	mmult_correct.c:206	mmult_c_correct
1.1%		> 6 others			



Linaro Performance Reports

Command: /bask/homes/w/wongj/yearwoog-baskerville/wongj/forge/mmult/mmult_c_correct 3072
Resources: 1 node (72 physical, 144 logical cores per node)
Memory: 504 GiB per node
Tasks: 8 processes
Machine: bask-pg0308u23a.cluster.baskerville.ac.uk
Architecture: x86_64
CPU Family: icelake-x
Start time: Sat Nov 25 15:07:04 2023
Total time: 78 seconds (about 1 minutes)
Full path: /bask/projects/y/yearwoog-baskerville/wongj/forge/mmult



Summary: mmult_c_correct is **Compute-bound** in this configuration

Compute	69.3%	<div style="width: 69.3%; background-color: green;"></div>	Time spent running application code. High values are usually good. This is average ; check the CPU performance section for advice
MPI	30.7%	<div style="width: 30.7%; background-color: blue;"></div>	Time spent in MPI calls. High values are usually bad. This is average ; check the MPI breakdown for advice on reducing it
I/O	<0.1%	<div style="width: <0.1%; background-color: orange;"></div>	Time spent in filesystem I/O. High values are usually bad. This is very low ; however single-process I/O may cause MPI wait times

This application run was **Compute-bound** (based on main thread activity). A breakdown of this time and advice for investigating further is in the **CPU** section below.

CPU

A breakdown of the **69.3%** CPU time:

Scalar numeric ops	13.9%	<div style="width: 13.9%; background-color: green;"></div>
Vector numeric ops	0.0%	<div style="width: 0.0%; background-color: orange;"></div>
Memory accesses	85.9%	<div style="width: 85.9%; background-color: green;"></div>

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

No 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 **30.7%** MPI time:

Time in collective calls	18.1%	<div style="width: 18.1%; background-color: blue;"></div>
Time in point-to-point calls	81.9%	<div style="width: 81.9%; background-color: darkblue;"></div>
Effective process collective rate	0.00 bytes/s	<div style="width: 0.0%; background-color: orange;"></div>
Effective process point-to-point rate	38.3 MB/s	<div style="width: 38.3%; background-color: darkblue;"></div>

I/O

A breakdown of the **<0.1%** I/O time:

Threads

A breakdown of how multiple threads were used:

Example: MPI with CUDA



simpleMPI.... simpleMPI...

```

54 > __global__ void simpleMPIKernel(float *input, float *output) { ... }
55
56 // Initialize an array with random data (between 0 and 1)
57 void initData(float *data, int dataSize) {
58     for (int i = 0; i < dataSize; i++) {
59         data[i] = (float)rand() / RAND_MAX;
60     }
61
62 // CUDA computation on each node
63 // No MPI here, only CUDA
64 void computeGPU(float *hostData, int blockSize, int gridSize) {
65     int dataSize = blockSize * gridSize;
66
67     // Allocate data on GPU memory
68     float *deviceInputData = NULL;
69     CUDA_CHECK(cudaMalloc((void **)&deviceInputData, dataSize * sizeof(float)));
70
71     float *deviceOutputData = NULL;
72     CUDA_CHECK(cudaMalloc((void **)&deviceOutputData, dataSize * sizeof(float)));
73
74     // Copy to GPU memory
75     CUDA_CHECK(cudaMemcpy(deviceInputData, hostData, dataSize * sizeof(float),
76                           cudaMemcpyHostToDevice));
77
78     // Run kernel
79     simpleMPIKernel<<<gridSize, blockSize>>>(deviceInputData, deviceOutputData);
80
81     // Copy data back to CPU memory
82     CUDA_CHECK(cudaMemcpy(hostData, deviceOutputData, dataSize * sizeof(float),
83                           cudaMemcpyDeviceToHost));
84
85     // Free GPU memory
86     CUDA_CHECK(cudaFree(deviceInputData));
87
88     // Free GPU memory
89     CUDA_CHECK(cudaFree(deviceOutputData));

```

Time spent on line 73

Breakdown of the 68.6% time spent on this line:

- Executing instructions 0.0%
- Calling functions 100.0%

Input/Output | Project Files | **Main Thread Stacks** | Functions

Total core time | MPI | Function(s) on line | Source | Position | Library

Total core time	MPI	Function(s) on line	Source	Position	Library
68.6%		simpleMPI [program]			
		main	int main(int argc, char *argv[]) {	simpleMPI.cpp:61	
		computeGPU(float*,int,int)	computeGPU(dataNode, blockSize, gridSize);	simpleMPI.cpp:99	simpleMPI
0.4%		> cudaMalloc	CUDA_CHECK(cudaMalloc((void **)&deviceInputData, dataSize * sizeof(float)));	simpleMPI.cpp:73	simpleMPI
24.6%	24.6%	> 4 others			
3.2%		MPI_Scatter	MPI_CHECK(MPI_Scatter(dataRoot, dataSizePerNode, MPI_FLOAT, dataNode,	simpleMPI.cpp:90	simpleMPI
1.8%		> initData(float*,int)	initData(dataRoot, dataSizeTotal);	simpleMPI.cpp:83	simpleMPI
		MPI_Finalize	MPI_CHECK(MPI_Finalize());	simpleMPI.cpp:115	simpleMPI

Example: Python

Fibonacci Sequence

$$F_n = F_{n-1} + F_{n-2}, \quad F_0 = 0, \quad F_1 = 1.$$



.Py python-profiling...

```

4 To run the demo, from the examples directory, run:
5   $ make -f python-profiling.makefile
6   $ ../bin/map --start python python-profiling.py --index 30
7   ...
8 import os
9 import ctypes
10 import argparse
11
12 example_dir = os.path.dirname(os.path.realpath(__file__))
13 fibonacciLib = ctypes.CDLL(os.path.join(example_dir, 'libfibonacci.so'))
14
15 fibonacciLib.compute_fibonacci.argtypes = [ctypes.c_ulonglong]
16 fibonacciLib.compute_fibonacci.restype = ctypes.c_ulonglong
17 def fibonacci_c(index):
18     return fibonacciLib.compute_fibonacci(index)
19
20 def fibonacci_python(index):
21     return index if index in [0, 1] else fibonacci_python(index-1) + fibonacci_python(index-2)
22
23 def main():
24     parser = argparse.ArgumentParser(description='Compute a Fibonacci number.')
25     parser.add_argument("--index", dest="index", help="index in the Fibonacci sequence", type=int, default=20)
26     options = parser.parse_args()
27     print("The Fibonacci number at index %s is:\n%s (computed in C)\n%s (computed in Python)" % (options.index,
28         fibonacci_c(options.index),
29         fibonacci_python(options.index)))
30
31 if __name__ == "__main__":
32     main()
33

```

33.3% | 33.3% | 16.7% | 16.7%

Time spent on line 9

Breakdown of the 33.3% time spent on this line:

Executing instructions	0.0%
Calling functions	100.0%

Input/Output Project Files Main Thread Stacks Functions

Input/Output

```

Submitted batch job 530154
The Fibonacci number at index 20 is:
6765 (computed in C)
6765 (computed in Python)

```



Useful links:

- <https://docs.baskerville.ac.uk/>
- <https://admin.baskerville.ac.uk/>
- <https://apps.baskerville.ac.uk/>
- <https://portal.baskerville.ac.uk/>
- <https://github.com/baskerville-hpc/intro-to-baskerville>

Email:

- baskerville-tier2-support@contacts.bham.ac.uk