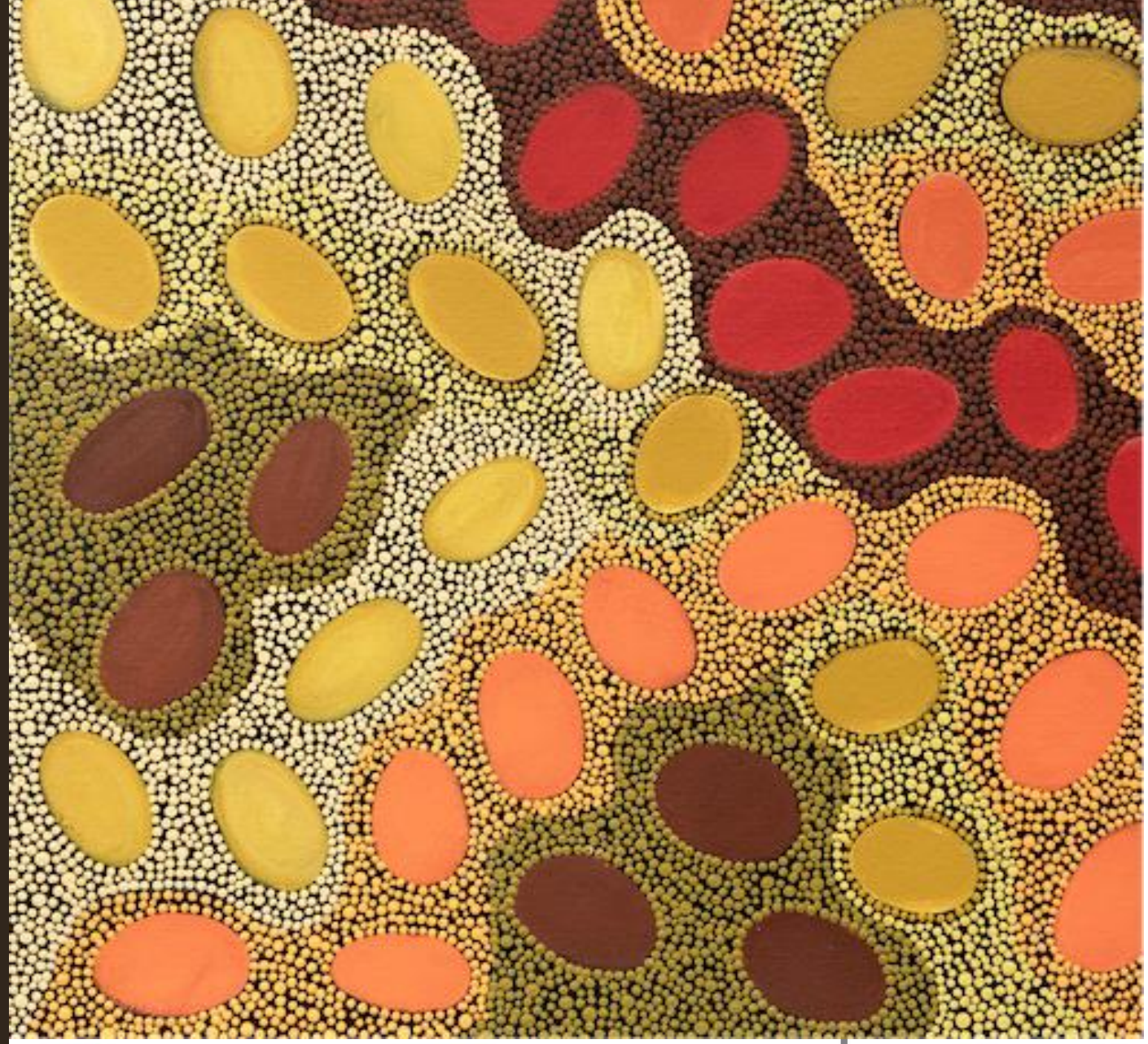
Pawsey

Profile Util library: A quick and easy way to get MPI, OpenMP and GPU runtime information

Dr. Pascal Jahan Elahi
Pawsey Supercomputing Research Centre
Perth, WA
EuroMPI 2024

Australian Government
NCRIS National Research Infrastructure for Australia
CSIRO
Curtin University
Murdoch University
Government of Western Australia
ECU
The University of Western Australia



Meteorites by Wajarri Yamatji artist Margaret Whitehurst

Acknowledgment of Country

Ngaala Kaaditji Noongar moort keyen kaadak nidja boodja

I acknowledge the traditional owners of this land, the Noongar Whadjuk People– their ancestors and elders, past, and present – as the original custodians of this land.

Origin: Understanding MPI Issues

- Phase-1 Setonix passed HPL tests but there were a number of issues encountered by users running production MPI-enabled codes. Not obvious what the underlying issues were and whether they were all related.
- Not all workflows impacted but some key stakeholders could not run.
- We did NOT have a simple set of diagnostic-oriented MPI tests.
  - MPI performance was measured using OSU Micro Benchmarks (OMB). However, these are not designed for debugging. OMB could pass when production codes would fail.
- Motivated by understanding these issues and realising there was a gap, we developed a suite of MPI stress tests focused on.

This was the crucible for the development of profile_util

Profile_util

Motivation

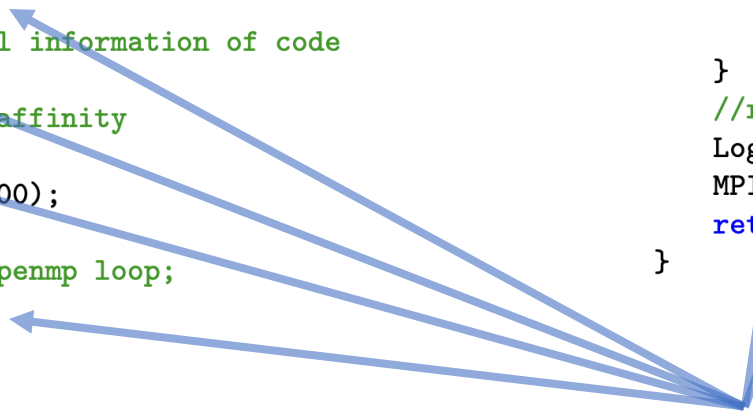
- Addressing performance bottlenecks critical for running efficiently at scale. Profiling tools essential for identifying performance issues by measuring various metrics such as CPU usage, GPU usage, memory consumption, and execution time.
- Wide variety available but many closed-source commerical products, often requiring instrumenting a code, or are tailored to a specific API (see for example Linaro Forge, Intel VTune, NVIDIA Nsight Compute, Omniperf).
- Profiling tools also not designed to provide a view into a code's performance in daily production-scale runs, where minimal impact and a higher-level view desirable

Profile_util

- Open-source library which can be simply integrated into codes for production-scale runs https://github.com/pelahi/profile_util
- C++20, CMAKE build system, and MPI, OpenMP, and GPU (CUDA and HIP) parallel APIs
- Simple API for integration into C++ code

Profile_util

```
// simple MPI+OpenMP code
#include <iostream>
#include <vector>
#include <mpi.h>
// include the profile_uti.h header
#include <profile_util.h>
int main() {
    // init MPI
    int ThisTask, NProcs;
    MPI_Init(&argc, &argv);
    MPI_Comm comm = MPI_COMM_WORLD;
    MPI_Comm_size(comm, &NProcs);
    MPI_Comm_rank(comm, &ThisTask);
    // Set the Logging communicator of the profile util library
    MPISetLoggingComm(comm);
    // a call to get parallel information of code
    LogParallelAPI();
    // a call to get thread affinity
    LogBinding();
    std::vector<float> vec(100);
    auto sum = 0.0;
    // construct timer for openmp loop;
    auto timer = NewTimer();
    // have parallel loop where to have logging information with
        std::cout
    // possible in loop, add LOGGING().
    #pragma omp parallel default(none) shared(vec) LOGGING()
        reduction(+:sum)
    {
        // now in the loop if desired, could get affinity for every
            thread spawned.
        #pragma omp critical
        LogThreadAffinity();
        #pragma omp parallel for
        for (auto &x:vec)
        {
            sum+=x;
        }
    }
    //report time taken
    LogTimeTaken(timer);
    MPI_Finalize();
    return 0;
}
```



Simple API for stdout but also similar API for logging to ostream

API Examples

- `LogParallelAPI()`: reports the parallel API's used.

```
Parallel API's
========
MPI Comm world size 2
OpenMP version 201811 with total number of threads = 2 with total number
    of allowed levels 1
Using GPUs: Running with HIP and found 4 devices
```

- `LogBinding()`: reports the overall binding of cores, GPU information (such as PCI address) for every MPI process

```
Core Binding
========
On node nid003012 : MPI Rank 0 : OMP Thread 0 : at nested level 1 : Core
    affinity = 0-7
On node nid003012 : MPI Rank 0 : OMP Thread 1 : at nested level 1 : Core
    affinity = 0-7
Current runtime environment gpu list is :0,1
On node nid003012 : MPI Rank 0 : GPU device 0 Device_Name=
    Bus_ID=0000:c1:00.0 Compute_Units=110 Max_Work_Group_Size=64
    Local_Mem_Size=65536 Global_Mem_Size=68702699520
On node nid003012 : MPI Rank 0 : GPU device 1 Device_Name=
    Bus_ID=0000:c6:00.0 Compute_Units=110 Max_Work_Group_Size=64
    Local_Mem_Size=65536 Global_Mem_Size=68702699520
```

API Examples

- `LogTimeTaken(timer)`: reports the time taken from creation of Timer to point at which this function is called. Also have GPU analogue `LogTimeTakenOnDevice(timer)`

```
@allocate_mem_host L132 (Wed Jul 24 13:41:03 2024) : Time taken between
    : @allocate_mem_host L132 - @allocate_mem_host L106 : 1.296 [s]
@allocate_mem_gpu L177 (Wed Jul 24 13:41:03 2024) : Time taken on device
    between : @allocate_mem_gpu L177 - @allocate_mem_gpu L158 : 33 [us]
```

- `LogCPUUsage(sampler)`: reports statistics of CPU usage over time taken. Also have GPU analogue, `LogGPUUsage(sampler)`

```
@main L386 (Wed Jul 24 13:41:19 2024) : CPU Usage (%) statistics taken
    between : @main L386 - @main L353 over 15.532 [s] :
    [ave,std,min,max] = [ 4458.294, 78.093, 95.200, 5536.000 ]
@main L387 (Wed Jul 24 13:41:19 2024) : GPU0 Usage (%) statistics taken
    between : @main L387 - @main L353 over 15.559 [s] :
    [ave,std,min,max] = [ 75.558, 3.619, 0.000, 100.000 ]
```

- `LogMemUsage()`: reports current & peak usage of process. Also have system analogue reporting memory state of Node, `LogSystemMem()`

```
[00000] @main L947 (Wed Jul 24 11:00:56 2024) : Node memory report @ main L947 :
  Node : nid002950^@ : VM current/peak/change : 33.097 [GiB] / 91.230
      [MiB] / 0 [B]; RSS current/peak/change : 183.777 [MiB] / 0 [B] / 0 [B]
[00000] @main L948 (Wed Jul 24 11:00:56 2024) : Node system memory report @ main L948 :
  Node : nid002950^@ : Total : 251.193 [GiB]; Used : 24.997 [GiB];
    Free : 229.620 [GiB]; Shared: 1.429 [GiB]; Cache : 7.451 [GiB]; Avail : 226.196 [GiB];
```

Use Case 1: Debugging MPI Issues on Setonix

- Despite passing acceptance tests, a number of researchers running more complex workflows encountered issues during Setonix Phase-1 (HPE Cray EX system, CPU-only). Main issues:

Memory Leaks	Multi-node jobs were crashing with a variety of reported errors: bus errors; generic SLURM kill errors; out-of- memory errors; xpmem or Open Fabrics Interface errors.
Poor Scaling	Multi-node scaling of software with significant pt2pt communication did not scale well past two nodes. Even a much older Cray XC system outperformed Setonix by factors of $\gtrsim$5 when using $\gtrsim$ 96 cores across multiple nodes.
Reduced Node Memory	Available memory on idle nodes slowly decreased.
Large-comm Instability	Crashes occurred when running jobs with pt2pt communication with large number of processes ($\gtrsim$ 700). Additionally, hangs were observed when using asynchronous pt2pt communication with high message counts.

- MPI performance was measured using OSU Micro Benchmarks (OMB). However, these are not designed for debugging. OMB could pass when production codes would fail. Profiling productions codes for regression testing not ideal.
- Develop simple MPI unit tests with profile_util logging to look at code AND node at runtime

Use Case 1: Debugging MPI Issues on Setonix

Memory Leaks (and Reduced Node Memory)

Crime Scene:

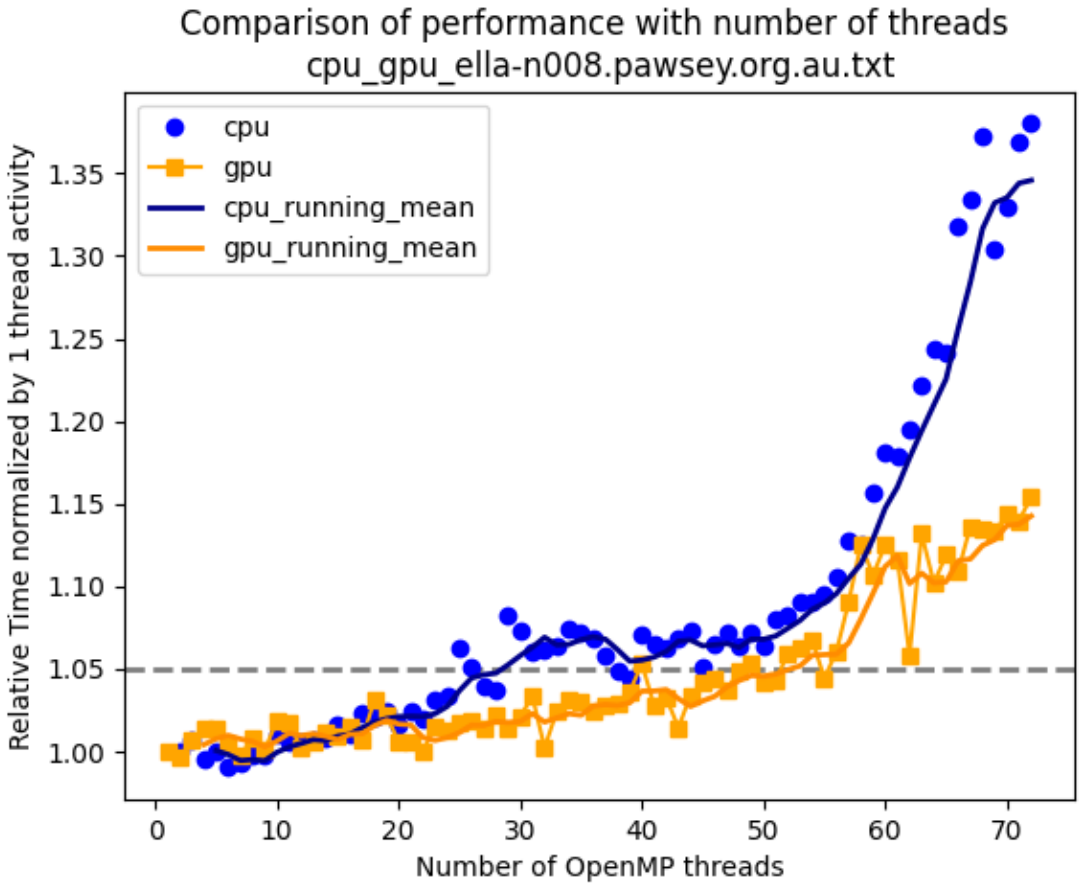
- Showed up in multi-node communication when communication included inter-node communication.
- Process memory reasonable but node memory showed clear reduction in available memory during communication.
- Amount of memory consumed dependent on comm size.
- Errors occurred when the amount of available node memory not enough given memory consumed by process or beyond total physical memory available on node.
- Jobs that completed would slowly reduce the amount of memory available on the node afterwards.
- Crashes not only left messages in kernel ring buffer that could involve the memory but could leave errors pertaining to Setonix Slingshot interconnet. In this case, node unusable as all subsequent MPI jobs would crash upon initialization.

Evidence found by using simple MPI unit tests with lots of memory logging (both process with `LogMemUsage` AND node `LogSystemMem`) around `MPI_Send/Isend/Ssend/MPI_Gather`, etc.

Lines of evidence pointed to libfabric since communication had to involve inter-node communication and memory consumed not visible in user space.

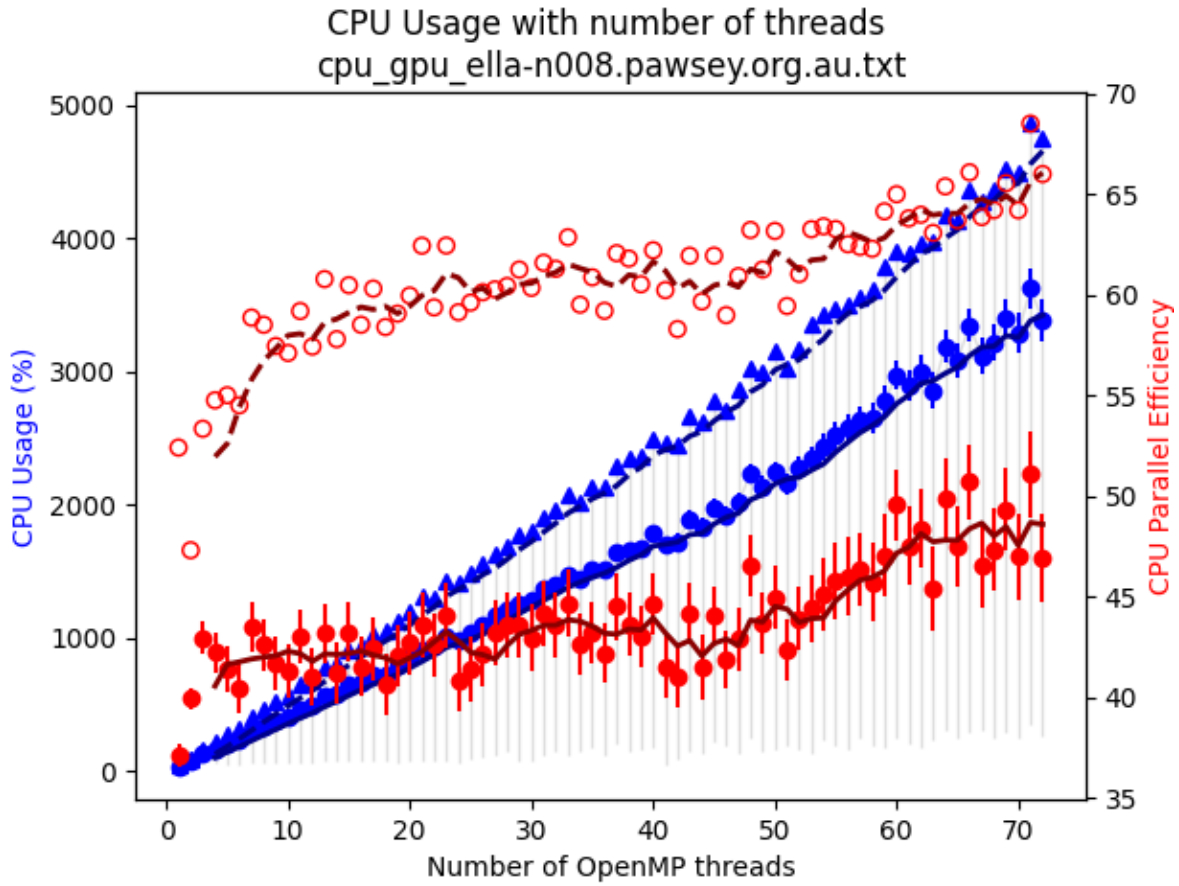
Use Case 2: GPU Performance

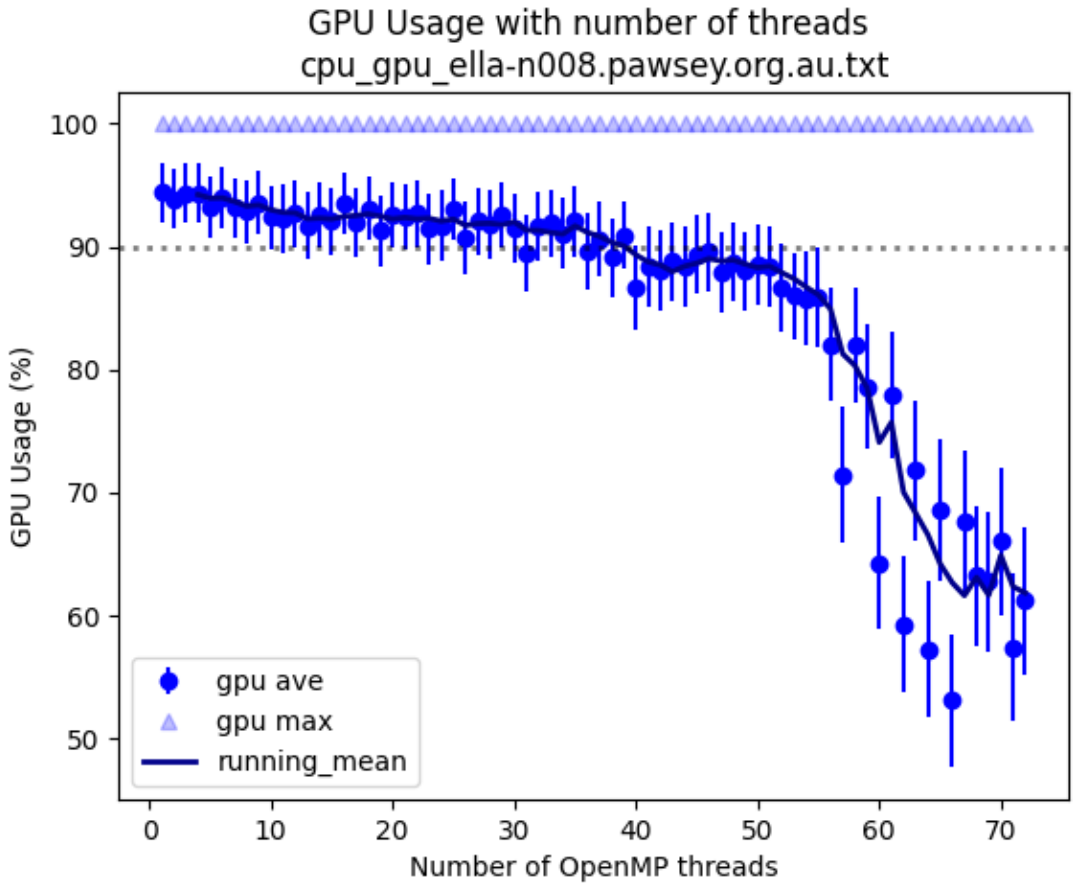
- New GraceHopper GPUs showed initially unusual performance (CPU and GPU codes slower than expected, variable). Again, difficult to diagnose -> Unit tests with profile_util sampling & timing



Use Case 2: GPU Performance

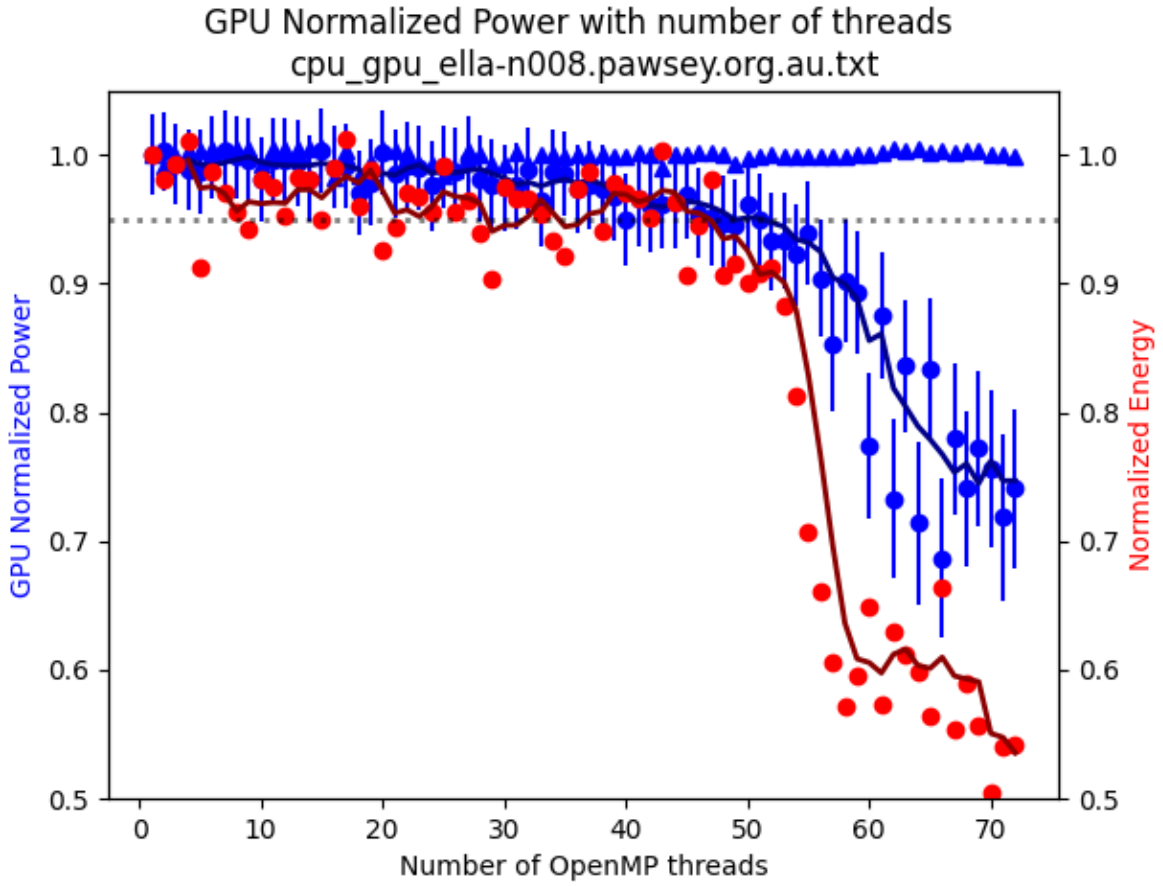
- Unit tests profiled showed CPU usage not drop as more threads add BUT GPU performance drops significantly after certain number of threads used.





Use Case 2: GPU Performance

- Performance impacted by power envelop of the GraceHopper chip. Hopper GPU power consumption heavily throttled as Grace CPU heavily used. However, drop in CPU performance indicates that CPU also throttled.



Profile_util

- Simple, easy to use library with C++ API. C/Fortran API in progress.
- Adds parallelism reporting, core affinity, MPI-aware logging, GPU (CUDA/HIP) logging, through simple API with minimal impact to code performance.
- So feel free to use this library to add simple profiling information to any MPI/OpenMP/CUDA/HIP codes

Any Questions?

pawsey