ARCHER Tips and Tricks

A few notes from the CSE team

Reusing this material

Outline

- Using Intel MKL
- Impact of HyperThreads
- Showing process/thread placement
- Performance analysis: hardware counters on ARCHER
- Debugging: Disabling autotuning in Cray BLAS
- Enabling and using ATP

Intel MKL

- MKL can be used as an alterantive for LibSci
  - We have seen cases where either is better
  - Worth experimenting
- Not interfaced through modules
- Linking using GNU

```
-L$(MKLROOT)/lib/intel64/ -Wl,--start-group -lmkl_gnu_thread \
-lmkl_gf_lp64 -lmkl_core -Wl,--end-group –ldl
```

- Linking using Intel

```
-L$(MKLROOT)/lib/intel64/ -Wl,--start-group -lmkl_intel_lp64 \
-lmkl_core -lmkl_sequential -Wl,--end-group –ldl
```

Intel MKL (cont.)

- The link line is reasonably complicated.
- Use the MKL Link Line Advisor:

  http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor

- For ARCHER select:
  - Product: Intel Composer XE 2013 SP1
  - OS: Linux
  - Architecture: Intel(R) 64
  - Linking: Static
  - Interface Layer: LP64 (32-bit Integer)
  - (MPI: MPICH2 if required)

Impact of HyperThreads

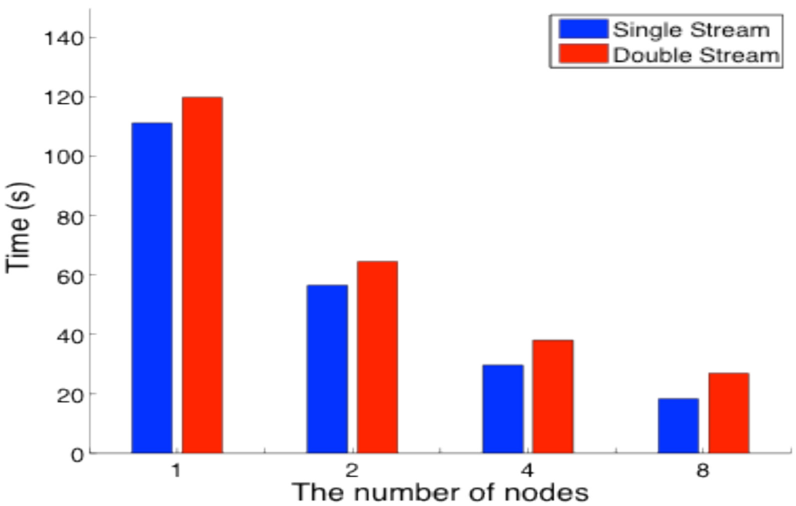
- HyperThreads allow up to 2 processes/threads to run concurrently on a single physical core
  - Managed in hardware so context switch is fast
  - Use CPU resource while one thread is stalled
- Very program dependent
  - Even a small improvement is worth it (as it is free)
  - Worth testing if it is useful for your program
- aprun syntax (2 nodes):

```
aprun -j 2 -n 96 -N 48 …
```

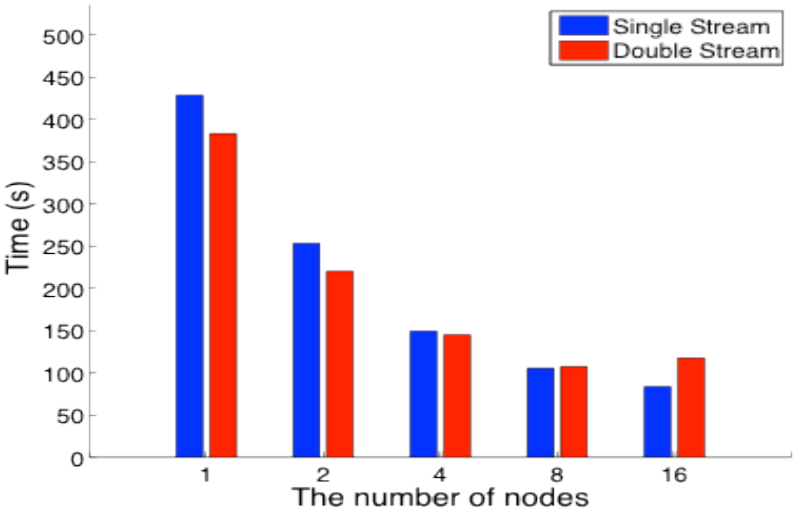
Hyperthreading example performance

- XC30: Sandy Bridge (8 cores), fully populated nodes

- VASP



- NAMD



Effects of Hyper-Threading on the NERSC workload on Edison
http://www.nersc.gov/assets/CUG13HTpaper.pdf

Show Process/Thread Placement

- Process/thread placement can have a large impact on performance
  - Particularly when underpopulating nodes or running mixed-mode (MPI/OpenMP) code.
- Add the following lines to your job submission script:

```
export MPICH_CPUMASK_DISPLAY=1
export MPICH_RANK_REORDER_DISPLAY=1
```

Placement (cont.)

```
[PE_0]: MPI rank order: Using default aprun rank ordering.
[PE_0]: rank 0 is on nid02421
[PE_0]: rank 1 is on nid02421
[PE_0]: rank 2 is on nid02421
…
[PE_0]: rank 24 is on nid02505
[PE_0]: rank 25 is on nid02505
[PE_0]: rank 26 is on nid02505
…
```

Placement (cont.)

```
[PE_0]: cpumask set to 1 cpu on nid02421, cpumask =
000000000000000000000000000000000000000000000001
[PE_34]: cpumask set to 1 cpu on nid02505, cpumask =
000000000000000000000000000000000000010000000000
[PE_33]: cpumask set to 1 cpu on nid02505, cpumask =
000000000000000000000000000000000000001000000000
[PE_35]: cpumask set to 1 cpu on nid02505, cpumask =
000000000000000000000000000000000000100000000000
[PE_47]: cpumask set to 1 cpu on nid02505, cpumask =
000000000000000000000000100000000000000000000000
…
```

Hardware Counters on ARCHER

- CrayPAT allows you to monitor performance at the hardware level
- Specify set of performance counters using the PAT_RT_PERFCTR environment variable in script that is running instrumented code:

```
PAT_RT_PERFCTR=1
```

(Group = 1 shows a summary with floating-point and cache metrics.)

```
========================================================================
  Total
------------------------------------------------------------------------
  PERF_COUNT_HW_CACHE_L1D:ACCESS                      458227922309
  PERF_COUNT_HW_CACHE_L1D:PREFETCH                      7837418131
  PERF_COUNT_HW_CACHE_L1D:MISS                         25703134212
  CPU_CLK_UNHALTED:THREAD_P                           884128952294
  CPU_CLK_UNHALTED:REF_P                               29852948968
  DTLB_LOAD_MISSES:MISS_CAUSES_A_WALK                    219955467
  DTLB_STORE_MISSES:MISS_CAUSES_A_WALK                    54655340
  L2_RQSTS:ALL_DEMAND_DATA_RD                          17968418083
  L2_RQSTS:DEMAND_DATA_RD_HIT                          14820163740
  User time (approx)              304.533 secs    822542437366 cycles
  CPU_CLK                           2.962GHz
  TLB utilization                 1790.78 refs/miss      3.498 avg uses
  D1 cache hit,miss ratios          94.8% hits            5.2% misses
  D1 cache utilization (misses)     19.13 refs/miss      2.392 avg hits
  D2 cache hit,miss ratio           87.8% hits           12.2% misses
  D1+D2 cache hit,miss ratio        99.4% hits            0.6% misses
  D1+D2 cache utilization          156.20 refs/miss     19.525 avg hits
  D2 to D1 bandwidth            3601.274MB/sec  1149978757281 bytes
```

Disable Cray BLAS autotuning

- If you are debugging and use the Cray LibSci library then you may want to disable autotuning.
  - Ensures autotuning is not causing the error.
- Add:

```
CRAYBLAS_AUTOTUNING_OFF=1
```

to your job scripts.

Using ATP

- ATP (Abnormal Termination Processing) catches dying applications and produces a merged stack backtrace
- Useful for getting more information on crashes
- Set:

  ```
  ATP_ENABLED=1
  ```

  in your job submission script.
- There is no need to recompile to use ATP

Using ATP (cont.)

- When your program crashes, ATP will:
  - Produce a stack trace of the first failing process
  - Produce a visualisation of every processes stack trace
  - Generate a selection of relevant core files
- Visualise the merged stack trace using *statview*:

```
module add stat
statview atpMergedBT.dot
```

- Very simple way to start the debugging process

statview (thanks to Cray)