HPSC Lab 5

Luna McBride

October 5 2023

1 Alpine Architecture

The alpine infrastructure is a beast that is hard to grasp for those not strong in general hardware knowledge like me. It is made up of various components, most of which deal with memory and general electrical concepts. The information is out there to find, it just needs to be pieced together and laid out in order to effectively use this supercomputer architecture.

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                64
On-line CPU(s) list:   0-63
Thread(s) per core:    1
Core(s) per socket:    64
Socket(s):             1
NUMA node(s):          1
Vendor ID:             AuthenticAMD
CPU family:            25
Model:                 1
Model name:            AMD EPYC 7713 64-Core Processor
Stepping:              1
CPU MHz:               3063.821
BogoMIPS:              3992.57
Virtualization:        AMD-V
L1d cache:             32K
L1i cache:             32K
L2 cache:              512K
L3 cache:              32768K
NUMA node0 CPU(s):     0-63
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
     pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb
    rdtscp lm constant_tsc rep_good nopl nonstop_tsc cpuid extd_apicid aperfmperf
    pni pclmulqdq monitor ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes
     xsave avx f16c rdrand lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a
    misalignsse 3dnowprefetch osvw ibs skinit wdt tce topoext perfctr_core
    perfctr_nb bpext perfctr_llc mwaitx cpb cat_l3 cdp_l3 invpcid_single hw_pstate
    sme ssbd mba sev ibrs ibpb stibp vmmcall sev_es fsgsbase bmi1 avx2 smep bmi2
    invpcid cqm rdt_a rdseed adx smap clflushopt clwb sha_ni xsaveopt xsavec
    xgetbv1 xsaves cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local clzero irperf
    xsaveerptr wbnoinvd amd_ppin arat npt lbrv svm_lock nrip_save tsc_scale
    vmcb_clean flushbyasid decodeassists pausefilter pfthreshold v_vmsave_vmload
```

```
vgif umip pku ospke vaes vpclmulqdq rdpid overflow_recov succor smca
```

Listing 1: CPU Info

This is the information generated by sending the lscpu command to atesting via sbatch (as to pull specifically from the alpine unit). Atesting was used instead of the common amilan due to the long wait times associated with amilan. This partition still uses milan hardware from further research. It is shown on the https://www.amd.com/en/product/10916 that it is part of the 7703 line, which means it is also a milan system. This partition also shows to have a single socket with 64 cores. The https://curc.readthedocs.io/en/latest/clusters/alpine/alpine-hardware.html shows that these milan systems can have anywhere from 1 to 2 sockets with anywhere from 32 to 64 cores depending on the individual partition used. This includes some higher memory and GPU enabled partitions, which vary in GPU usage and RAM/core depending on which is used.

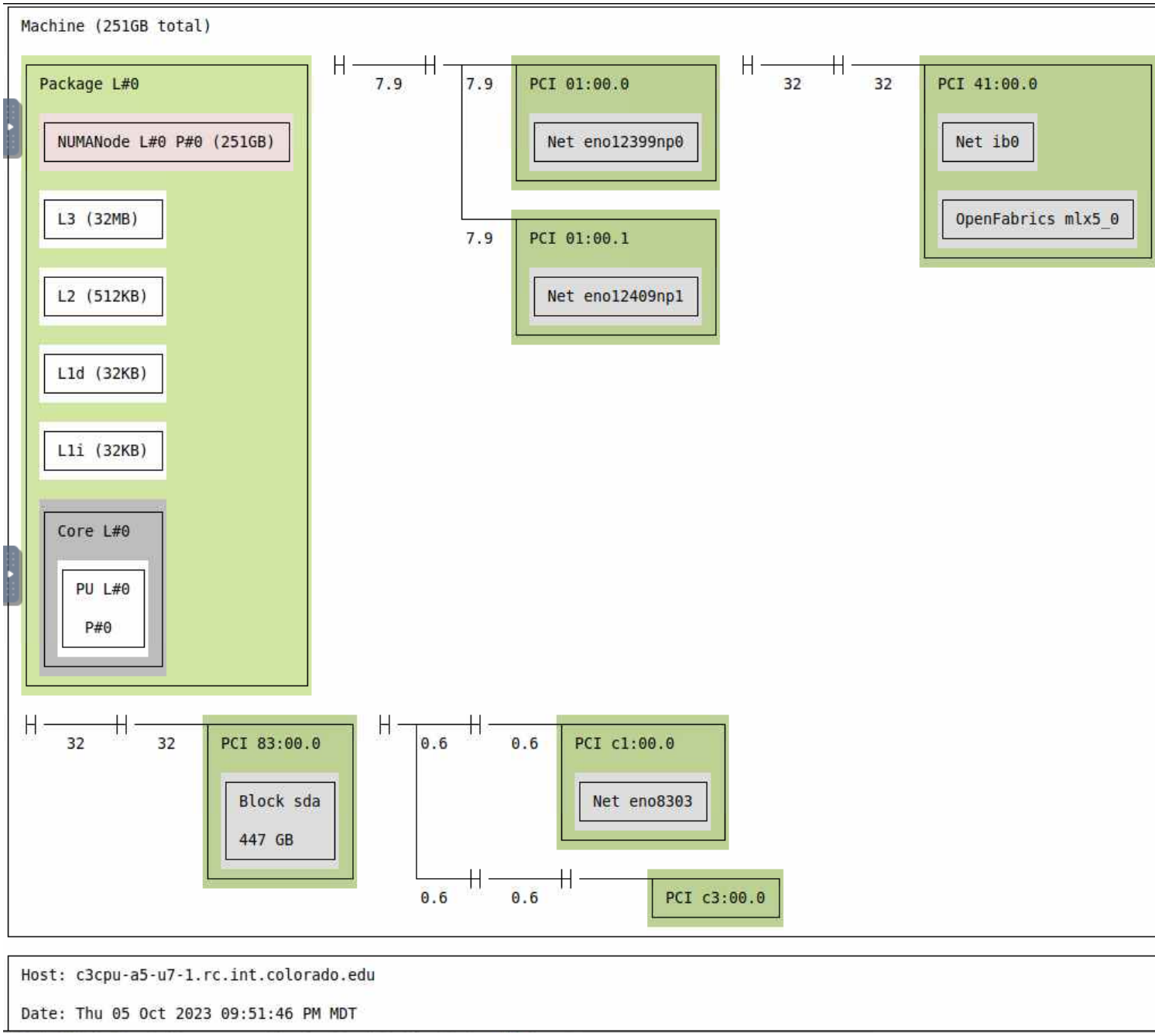

Figure 1: Topographical Information

Please ignore the double sidebars on this image. I spent over an hour trying to get the ASCII to work here to no avail, so I ended up taking multiple screenshots and stitching them together with paint.

As for the topographical information, the image shows the CPU socket, the major layers of cache, and the NUMA node. Memory in this infrastructure starts at the NUMA node, which is used in

this architecture to access memory that may be remote given the data warehouse infrastructure. This is then brought down through cache, which get smaller as the cache goes from L3 to L1. This is organized by level of usage, as the most used items should be stored the closest distance to the actual cores for ease of access. Speaking of the cores, that is the next key section in the infrastructure. The socket is labeled the core in this instance, but they are pretty interchangeable at this level. They do the main computing, as is probably self explanatory at this point.

2 CPU and Bandwidth Calculations

Flop Rate:
Cores = 4 (limit created upon loading on-demand)
Threads per Core = 1 (given in the cpuInfo, not hyperthreaded)
Clock = 3063.821 -> 3.063821 (MHz -> GHz)
Instructions/cycle = 256 bits / 64 double precision bits = 4
Fusing Factor = 2 (best case scenario, efficiency at fusing add and multiply operations pulled from lecture)
Flops = (cores*threads) * Instructions/cycle * Clock * fusing factor
Flops = (4*1) * 4 * 3.063821 * 2
Flops = 98.04 GFlops/Sec
—
Memory Bandwidth:
I could not find how memory channels and other similar aspects split in a shared system like this, so I will be using the numbers pulled directly from EPYC 7713's website where not otherwise listed.
Channels = 8 (pulled from the EPYC 7713 website)
Sockets = 1
Memory Transfer Rate = 3200 MT/sec
Bytes per Transfer = 64 bits / 8 bits per byte = 8
Bandwidth = Channels * Sockets * Memory Transfer Rate * Bytes per Transfer
Bandwidth = 8 * 1 * 3200 * 8
Bandwidth = 204800 MB/sec -> 204.8 GB/sec
This fits with the hypothetical value listed on EPYC 7713's website, so there is a quick sanity check.

3 Empirical Roof-line Toolkit (ERT)

```
# Linux workstation, MPI and OpenMP (8-core Intel Xeon CPU E5530, 2.40 GHz)

ERT_RESULTS Results.alpine

ERT_DRIVER  driver1
ERT_KERNEL  kernel1

ERT_MPI         True
ERT_MPI_CFLAGS  -I/usr/include/mpi
ERT_MPI_LDFLAGS -lmpi
```

```

ERT_OPENMP          True
ERT_OPENMP_CFLAGS   -qopenmp
ERT_OPENMP_LDFLAGS -qopenmp

ERT_FLOPS    1,2,4,8,16
ERT_ALIGN    32

ERT_CC       mpicc
ERT_CFLAGS   -O3 -msse3 -I/usr/local/boost_1_52_0  -std=c++11 -qopenmp -xAVX

ERT_LD       mpicc
ERT_LDFLAGS
ERT_LDLIBS


ERT_PRECISION FP64

ERT_RUN      export OMP_NUM_THREADS=ERT_OPENMP_THREADS; mpirun -np ERT_MPI_PROCS
    ERT_CODE

ERT_PROCS_THREADS  1
ERT_MPI_PROCS      1
ERT_OPENMP_THREADS 1

ERT_NUM_EXPERIMENTS 1

ERT_MEMORY_MAX 1073741824

ERT_WORKING_SET_MIN 1

ERT_TRIALS_MIN 1

ERT_GNUPLOT gnuplot
```

Listing 2: Alpine Config File

```
#!/bin/bash

# -
# |
# | This is a batch script for running a MPI parallel job on Summit
# |
# | (o) To submit this job, enter:  sbatch --export=CODE='/home/trmc7708/Lab5/cs-
    roofline-toolkit/Empirical_Roofline_tool-1.1.0' ex01.bat
# |
# | (o) To check the status of this job, enter: squeue -u <username>
# |
# -

# -
# |
# | Part 1: Directives
# |
# -

```

```
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --time=00:01:00
#SBATCH --partition=atesting
#SBATCH --output=ex01-%j.out

# -
# |
# | Part 2: Loading software
# |
# -

module purge
module load intel
module load impi
module use /projects/scru5660/public/software/module

# -
# |
# | Part 3: User scripting
# |
# -

echo "=="
echo "||"
echo "|| Begin Execution of Roofline Toolkit in slurm batch script."
echo "||"
echo "=="

lstopo --of ascii > topo
lscpu > cpuInfo
./ert Config/alpine  > tty.out

echo "=="
echo "||"
echo "|| Execution of Roofline Toolkit in slurm batch script complete."
echo "||"
echo "=="
```

Listing 3: Slurm File

In the end, I could not get this to work, even at 1. The 1 would get stuck loading locally, but throw this error when using sbatch:

```
==
||
|| Begin Execution of Roofline Toolkit in slurm batch script.
||
==
Traceback (most recent call last):
  File "./Scripts/summary.py", line 56, in <module>
    x      = [float(line.split()[0].strip(',').strip('(')) for line in lines]
  File "./Scripts/summary.py", line 56, in <listcomp>
    x      = [float(line.split()[0].strip(',').strip('(')) for line in lines]
IndexError: list index out of range
  Failure...
```

```
Unable to complete Results.alpine/Run.030/FLOPS.001/MPI.0001/OpenMP.0001,
    experiment 1
  Failure...
Unable to process Results.alpine/Run.030/FLOPS.001/MPI.0001/OpenMP.0001

--- Processing ERT results failed ---

==
||
|| Execution of Roofline Toolkit in slurm batch script complete.
||
==
```

Listing 4: Error 1

As for all elements set to 2+, both local and sbatch runs gave this error:

```
==
||
|| Begin Execution of Roofline Toolkit in slurm batch script.
||
==
cat: 'Results.alpine/Run.002/*/*/*/sum': No such file or directory
Traceback (most recent call last):
  File "./ert", line 80, in <module>
    if ert.roofline() != 0:
  File "/home/trmc7708/Lab5/cs-roofline-toolkit/Empirical_Roofline_Tool-1.1.0/
    Python/ert_core.py", line 744, in roofline
    database = self.build_database(gflop_lines,gbyte_lines)
  File "/home/trmc7708/Lab5/cs-roofline-toolkit/Empirical_Roofline_Tool-1.1.0/
    Python/ert_core.py", line 613, in build_database
    for metadata in gflop[h:]:
UnboundLocalError: local variable 'h' referenced before assignment
==
||
|| Execution of Roofline Toolkit in slurm batch script complete.
||
==
```

Listing 5: Error 1

This has been really hard for me to debug, even when updating my other files based on the professor's. Even looking back on previous labs, I was never able to get intel/impi modules to work. I originally got around this in previous labs using packages gcc and openmpi instead, but that was not going to work here. There was one point where it also threw up an error regarding Intel commands, which I have not been able to recreate to show here. I actually can pull it up from my past Google searches:

Please verify that both the operating system and the processor support Intel(R) X87, CMOV, MMX, FXSAVE, SSE, SSE2 and SSE3 instructions.

From researching this error that I have not been able to recreate, it has something to do with AMD CPUs versus Intel ones. I am out of time to keep going down that rabbit hole (I have not had the most time to work on this and it is 11:35PM Thursday as of typing this), but that seems to be the main lead I would continue down. I had actually intended to go to office hours today for help since he was not here to ask Tuesday, but I had a migraine, and it is not like I was going to learn anything on a topic I already have trouble with in that state.