CIUK challenge

Bristol Team A

November 2, 2022

We focus on the comparison between (i) two baremetal hardware and between (ii) baremetal and virtual machine on the same hardware. As expected baremetal offer better performance with respect to VM. However, we highlight that the correct choice of software is crucial to boost the performance.

1 Hardware Performance

1.1 baremetal.intel.25gb

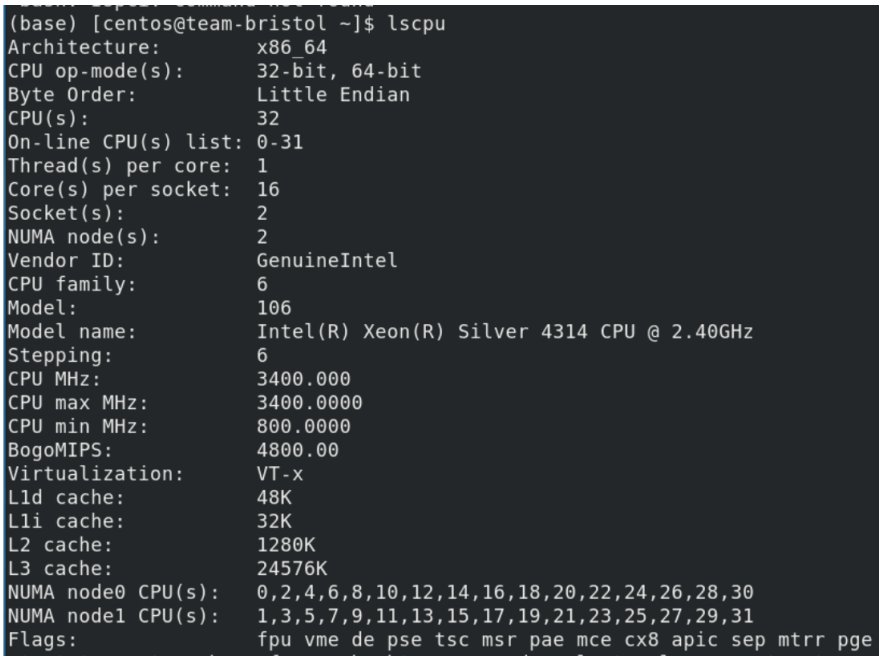

Figure 1: cpu baremetal

1.1.1 Memory Benchmark

Test of Memory using **STREAM**, compiled with *gcc 8.5 GNU* compiler. We run STREAM with two threads and compiled as : gcc -fopenmp -D_OPENMP -o3 -Ofast -mtune=native -DSTREAM_ARRAY_SIZE=60000000. Were -mtune and -mtune=native are required to have a

competitive memory performance. These are considered the default setting for stream benchmark and should be assumed if not stated otherwise.

Function	Best Rate MB/s	Avg time	Min time	Max time
Copy:	48290.2	0.030717	0.019880	0.042960
Scale:	30401.8	0.041976	0.031577	0.060825
Add:	32668.6	0.056252	0.044079	0.094283
Triad:	32490.2	0.052199	0.044321	0.059765

Results are consistent with DDR4 type of RAM.

1.1.2 Performance Benchmark

Test using **High Performance Computing Linpack Benchmark** (HPL) as linear algebra benchmark. We compiled hpl with *MKL* and *gcc 8.5 GNU* compiler. These are considered the default setting for stream benchmark and should be assumed if not stated otherwise. The HPL input data has been tuned to customize the hardware of the machine following the procedure at: HPL configure

T/V	N	NB	P	Q	Time	Gflop
WR11C2R4	40000	192	1	32	48.69	8.7637e+02
WR11C2R4	40000	192	2	16	42.98	9.9279e+02
WR11C2R4	40000	192	4	8	96.11	4.3872e+02

The blocking size is chose as default. $P \times Q$ are the number of MPI processes which is set equal to the total number of physical cores. N is the problem size, note that we have used a memory burden quite lower than the total memory available to the cluster.

$$Tot_{mem} \sim \sqrt{\frac{n_{nodes} \cdot RAM_{node}(bytes)}{8(bytes)}} \tag{1.1}$$

Note that HPL use double matrix (we have written and compile our c++ code to compare integer and float matrix-matrix multiplication), see below. We map into different grid.

1.1.3 Communication benchmark

Below are the results from the IMB-MPI1 benchmarks that measure the minimum latency, maximum bandwidth, and the Allgather benchmark for passing a 4kb message on the baremetal intel machine.

	Min latency	Max bandwidth	All gather
baremetal.intel.25gb	0.38	1232.03	210.05

1.2 baremetal.nvidia.a40

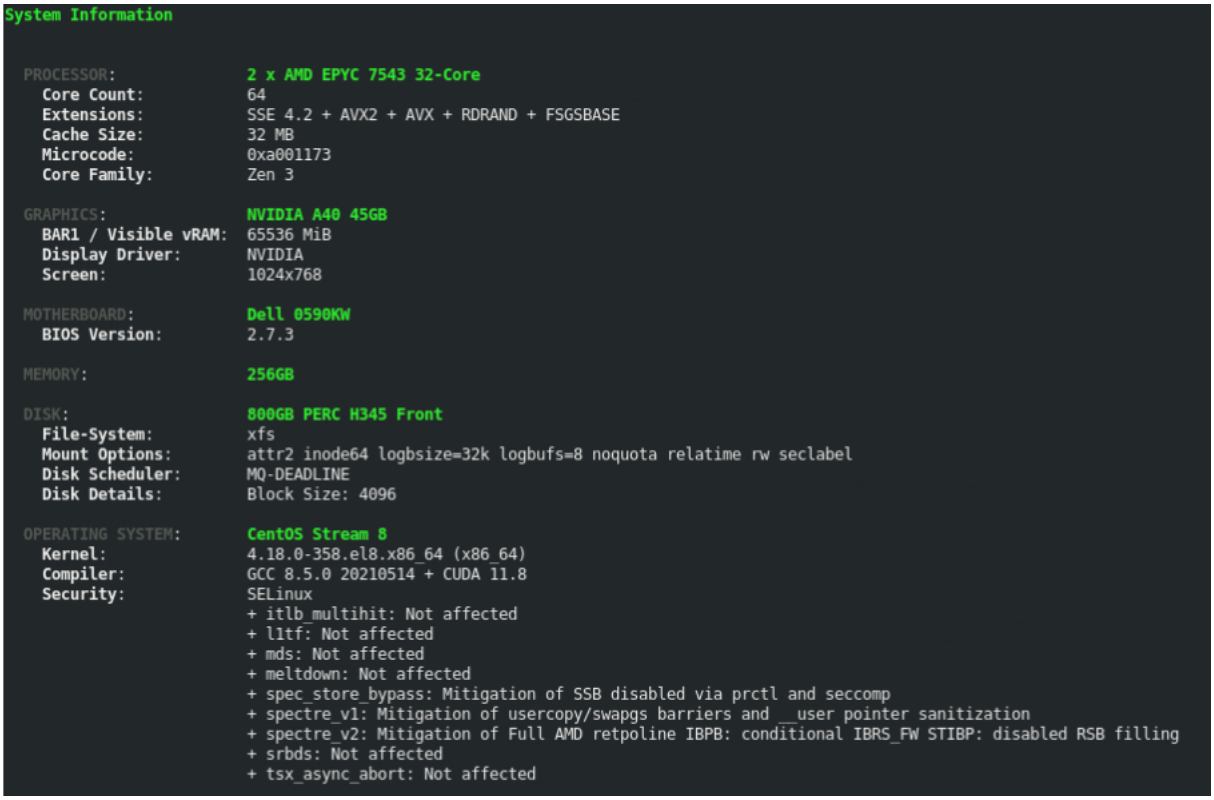

Figure 2: System Information for baremetal.nvidia.a40

1.2.1 Memory Benchmark

Test of Memory using **STREAM**. We observe an interesting increase of performance with respect to **baremetal.intel.25gb**. The increase of memory performance is extremely important whenever we run a program with large load of data that does not fits on CPU cache.

Function	Best Rate MB/s	Avg time	Min time	Max time
Copy:	91849.4	0.017485	0.017420	0.017548
Scale:	63283.4	0.025323	0.025283	0.025400
Add:	65361.1	0.036873	0.036719	0.037213
Triad:	65464.8	0.037276	0.036661	0.041412

Test of GPU Memory using **BabelStream** version 4.0. We use *CUDA* implementation with: Array size: 268.4 MB (=0.3 GB) and Total size: 805.3 MB (=0.8 GB). To compile we have used the *g++ 8.5 GNU* compiler. We observe an increase of almost one-order of magnitude with respect to the CPU memory performance.

Function	Best Rate MB/s	Avg time	Min time	Max time
Copy:	581119.061	0.00092	0.00094	0.00093
Scale:	577605.376	0.00093	0.00094	0.00094
Add:	582129.123	0.00138	0.00140	0.00139
Triad:	565224.851	0.00095	0.00096	0.00095

1.2.2 Performance Benchmark

Test using **High Performance Computing Linpack Benchmark** (HPL) as linear algebra benchmark. Compilation with *OpenBlas* and *crosstool-NG 7.3 GNU*.

T/V	N	NB	P	Q	Time s	Gflop
WR11C2R4	14208	192	1	64	76.78	2.4908e+01
WR11C2R4	14208	192	8	8	251.73	7.5970e+00

Same test as before but compiled with *gcc 8.5 GNU* and *OpenBlas*. Similar performance between the two compilers. It is quite peculiar that square grid a slower performance than highly asymmetric grid since HPL should prefer almost-square grid.

T/V	N	NB	P	Q	Time s	Gflop
WR11C2R4	14208	192	1	64	76.42	2.5025e+01
WR11C2R4	14208	192	2	32	141.50	1.3515e+01
WR11C2R4	14208	192	8	8	139.83	1.3676e+01

We are almost one order of magnitude slower than in **baremetal.intel.25gb**. To verify why such difference we run a comparison between *OpenBlas* and *MKL* using the same compiler for the **vm.nvidia.a40**, Sec 1.3

Test using **c++ matrix-matrix multiplication code for 1 core** test using integers, float and double square matrices. The result using *OpenBlas* library (cblas_dgemm) is also reported. Source code and make file at the repo: github repo. Note that we have intentionally compiled without "-Ofast -mtune=native -flto" flags and we make sure that *OpenBlas* run on a single core. Size of the matrix used is 1 million elements.

Integer	Float	Double	*OpenBlas*	
7.67256 s	3.40537 s	3.35236 s	1.59322 s	Wall Time
2.60539e+08	5.87015e+08	5.96296e+08	1.25469e+09	Operations/s

1.2.3 Communication benchmark

Below are the results from the IMB-MPI1 benchmarks that measure the minimum latency, maximum bandwidth, and the Allgather benchmark for passing a 4kb message on the baremetal nvidia a40 machine.

	Min latency	Max bandwidth	All gather
baremetal.nvidia	0.58	20088	167.97

1.3 vm.nvidia.a40

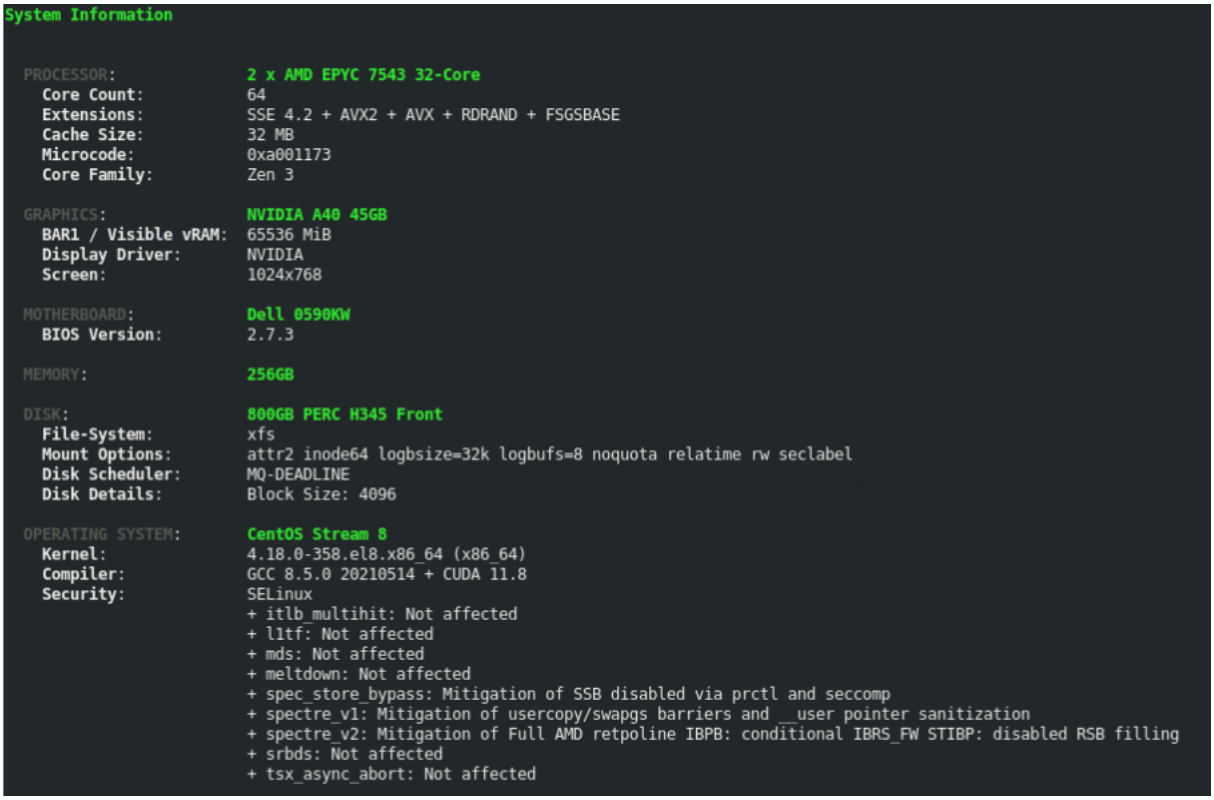

Figure 3: System Information for vm.nvidia.a40

1.3.1 Memory Benchmark

Test of Memory using **STREAM**. We observe an drastic decrease of performance with respect to **baremetal.nvidia.a40**

Function	Best Rate MB/s	Avg time	Min time	Max time
Copy:	51552.8	0.032197	0.031036	0.035072
Scale:	35472.1	0.045372	0.045106	0.046957
Add:	36421.0	0.067279	0.065896	0.074940
Triad:	36303.6	0.066218	0.066109	0.066311

1.3.2 Performance Benchmark

Test using **High Performance Computing Linpack Benchmark** (HPL) as linear algebra benchmark. Compilation with *OpenBlas* and *gcc 8.5 GNU*. Result for the performance benchmark are in extremely close agreement with the **baremetal.nvidia.a40**

T/V	N	NB	P	Q	Time s	Gflop
WR11C2R4	14208	192	1	64	158.90	1.2036e+01
WR11C2R4	14208	192	2	32	139.17	1.3742e+01
WR11C2R4	14208	192	8	8	250.00	7.6496e+00 14208 192 8 8

Here we want to compare *MKL* and *OpenBlas* performance for linear algebra. Test using **High Performance Computing Linpack Benchmark** (HPL) as linear algebra benchmark.

Compilation with *MKL* 2022 and *gcc 8.5 GNU*.

T/V	N	NB	P	Q	Time s	Gflop
WR11C2R4	40000	192	1	64	40.03	1.0660e+03
WR11C2R4	40000	192	2	32	32.31	1.3205e+03
WR11C2R4	40000	192	8	8	33.29	1.2815e+03

Interestingly the *MKL* package over-perform over *OpenBlas*. Probably a finer tuning on the compilation of *OpenBlas* would result in an increse of performance. Test using **c++ matrix-matrix multiplication code for 1 core** test using integers, float and double square matrices. ALL TESTs are extremely slower than **baremetal.nvidia.a40**

with a huge drop performance of OpenBlas.

Integer	Float	Double	*OpenBlas*	
10.0061 s	7.01154 s	7.0227 s	9.05995 s	Wall Time
1.99778e+08	2.85101e+08	2.84648e+08	2.20641e+08	Operations/s

2 Gromacs performance and compilation

2.1 Gromacs compilation

Gromacs is manually complied on **baremetal.nvidia.a40** using gcc and g++ compilers GNU 8.5 rather than clang compiler. Example of some flags for C compiler -mavx2 -mfma -fexcess-precision=fast -funroll-all-loops -pthread -O3. We allow for *CUDA* (11.8) and the FFTW is build-ed using the following optimization flags, -enable-sse2;−enable-avx. We ensure that the NVIDIA Kernel Mode Drive installed for the A40 is compatible with the *CUDA CUDA* Toolkit (v11.8). We have also used the OpenMPI 4.1.3 version.

2.2 Test Gromacs on different Hardware with phoronix-test-suite

2.2.1 baremetal.nvidia.a40

Using only the AMD EPYC CPU we reach 8.609 ns/day of molecular dynamics. It falls much above the median of the 579 OpenBenchmarking.org samples which is 1.47 ns/day. We are much more competitive that 2 x AMD EPYC 74F3 (5.96 ns/day) but rather slow compared to the 2 x AMD EPYC 7773X which has scored 10.989 ns/day.

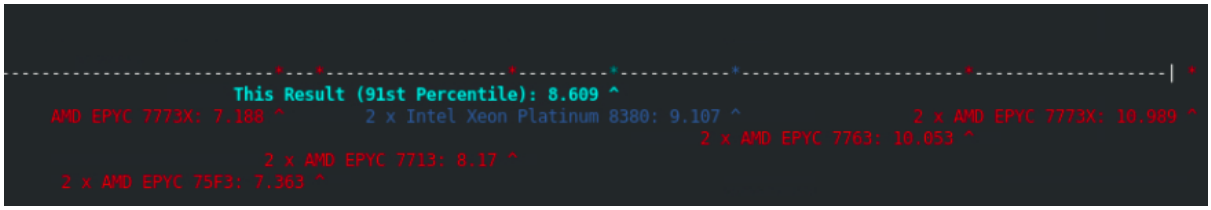

Figure 4: Test of Gromacs on full CPU hardware baremetal.nvidia.a40

Easy step forward is to burn the A40 to get a boost in the computational time. With the usage of the A40 we almost double the speed reaching 14.205 ns/day. It shows that the problem of computing and make operation with gradients makes the GPUs shine bright for such a task.

Interestingly the **Dell R6525 1U enterprise server** which also is equipped with 2 x AMD EPYC 7543 would result in a maximum performance of 8.6 ns/day at the price of 450W which is well below the 1280W of the **Dell R7525 2U visualisation server**. Therefore, even tough the overall speed up is dramatic, the usage of the GPU is not fully justifiable (at least in this particular application) due to the higher energy consumption. We note however that, any application for which the GPU allows a speed up of at least 3 times, make the virtualization server much more efficient.

Result uploaded at result gromacs baremetal A40

2.2.2 vm.nvidia.a40

Using only the 64 x AMD EPYC-Milan CPU the quite unsatified result of 1.865 ns/day of molecular dynamics. It falls close to the median of the 579 OpenBenchmarking.org samples which is 1.47 ns/day. With respect the baremetal result this is a really unsatisfactory result.

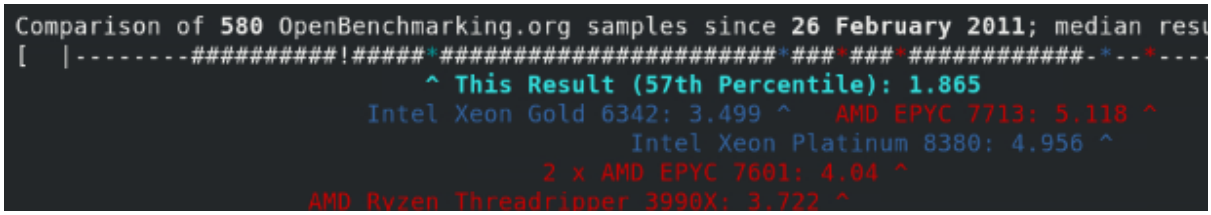

Figure 5: Test of Gromacs on full CPU hardware vm.nvidia.a40

With the usage of the A40 we reaches 11.403 ns/day. It shows that the problem of computing and make operation with gradients makes the GPUs shine bright for such a task. The CPU here has a huge impact on the speed up of the calculation. However the baremetal is still more than 25% faster than a virtual machine.

Result uploaded at https: result vm A40

3 Workflow

A full script to configure a fresh node and installing the required package to benchmark can be found at githup repo. This allow flexibility on which package install depending what is needed.

The script full workflow can be initialized as:
curl -o ‘pwd‘/tmp.sh https://raw.githubusercontent.com/emamars95/CIUK/main/setup.sh
bash tmp.sh

The script will follow four important process:

- setup:
  - The set up install and download important packages such as compilers, miniconda as environment manager, and a git repository where other scripts are located.
  - The set up take care of recognize the GPU present in the system and install the driver associated with it with little interaction from the user. It also download and compile MPI, MKL and CUDA.
  - The set up install gromacs.
- Gromcas job is lunched in the cluster. Not implemented
- Back up of the important data. Not Implemented
- Shut off the instance and cleaning the node. Not implemented.