CYBER-INSIGHT[*] Evaluating Cyberinfrastructure Total Cost of Ownership Benchmark Results

Wim R. M. Cardoen
Center for High-Performance Computing
University of Utah
wim.cardoen@utah.edu

Principal Investigator: Daniel Reed
University of Utah
cyberinsight-feedback@lists.utah.edu[†]

October 11, 2021

[*]**Funded in part by the National Science Foundation (NSF)** - Award Number 1812786.
[†]Email to which questions, feedback or bug reports should be reported.

Overview

The following applications were used to perform benchmarks:

1. High-Performance Linpack (HPL) [32] (Sec. I)
2. High-Performance Conjugate Gradients (HPCG) [19] (Sec. II)
3. Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) [34] (Sec. III)
4. Vienna Ab Initio Simulation Package (VASP) [23, 24] (Sec. IV)
5. Amber
6. Machine Learning (ML) Applications/Benchmarks (Sec. VI)
   (a) AI-Benchmark (based on Tensorflow) [20]
   (b) PyTorch-Benchmark [36]
   (c) The use of Convolutional Neural Networks (CNN) to discern different lymphoma types. [26]
7. Containers (Sec. VII)
8. IO (Sec. VIII)

I High-Performance Linpack Benchmark

The High-Performance Linpack Benchmark (HPL) executable was generated from source (HPL version 2.3) [32].

I.1 HPL runs on CHPC

The executable for the CHPC was generated using Intel Parallel Studio XE Developer Suite (2018.1.163) [21]). The HPL Benchmarks were run on nodes `notch`139 and `notch`192.

The node `notch`139 has the following characteristics:

- `Model Name CPU`: Intel(R) Xeon(R) Gold 6230 CPU @ 2.10 GHz - Dual socket - 40 physical cores
- `Memory`: 192 GB

The node `notch`192 has the following characteristics:

- `Model Name CPU`: AMD EPYC 7702P - 64 physical cores
- `Memory`: 256 GB

A series of preliminary HPL iterations were performed on `notch`139 to optimize the HPL simulation parameters. The energy used during each HPL run was recorded from the outlet feeding solely `notch`139, as reported by the Power Distribution Unit (PDU). In total 10 identical HPL runs (using 40 MPI Tasks) were performed.

The HPL run-time parameters were also optimized for `notch`192. The final run with the optimized parameters was repeated six times (using 64 MPI Tasks). The use of the MKL AVX2 kernels on the AMD CPU was enforced through `MKL_DEBUG_CPU_TYPE=5` [14].

I.2 HPL Runs on AWS

The HPL executable was generated using Intel Parallel Studio XE Developer Suite (2020.4.304). The HPL Benchmarks were run on a node of the type `c5n.18xlarge` (AWS Region: US West (Oregon)). Its CPU was of the following type: Intel(R) Xeon(R) Platinum 8124M CPU 3.00GHz - 36 physical cores.

We again optimized the HPL input parameters. Subsequently, we performed 6 identical HPL runs.

I.3 HPL Runs on AZ

The HPL benchmarks were run on 2 different types of nodes: `HC44rs` and `HB120rs_v2`. The former has an Intel Xeon Platinum 8168 CPU @ 2.7 GHz (containing 44 cores); the latter an AMD EPYC 7V12 CPU (120 physical cores). None of these Azure nodes supported hyperthreading.

Both HPL executables were compiled and executed using the Intel Parallel Studio XE Developer Suite (2020.4.304). However, in the latter OpenBlas was linked into the executable instead of Intel's MKL. All the HPL parameters were optimized. The final runs (with its optimized parameters) were executed six times.

I.4 HPL Runs on GCP

The HPL benchmarks were run on 5 different types of nodes: `c2-60`, `n2h-64`, `n2h-80`, `n2d-128`, `n2d-224`. The first three belong to the Intel family, but their hardware specifics were obfuscated by Google. The latter two nodes contained AMD processors of the type `EPYC 7B12`. The number of hyperthreaded cores for each of the aforementioned processors can be retrieved from the suffix in its resp. name: e. g. `c2-60` possesses 60 hyperthreaded cores or 30 physical cores.

All the HPL executables were compiled and executed using the Intel Parallel Studio XE Developer Suite (2020.1.217). In order to force the use of the AVX2 kernels on the AMD CPUs, we used Daniël de Kok's LD_PRELOAD suggestion [14]. [1]. All the HPL parameters were optimized. The final runs (with its optimized parameters) were executed six times.

[1] Neither the `MKL_DEBUG_CPU_TYPE=5` fix nor the `LD_PRELOAD` trick worked when I used the Intel 2020.4 series. This is why we opted for the OpenBlas library

I.5 Results

The results (on-premises) as well as for the three cloud-provides AZ, AWS and GCP are displayed in Table 1. The mean $\overline{x}$ (Eq. 1) and the sample standard deviation s (Eq. 2) were calculated in the following way [12]:

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{1}$$

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2} \tag{2}$$

where x_i stands observed value and n for the total number of measurements. In Table 1 as well as other tables the values for the aforementioned statistics are displayed in the format $\mathbf{x} \pm \mathbf{s}$.

Platform	Node	Time (s)	Tflops	Energy Used (kWh)	Cost per run ($)
CHPC	notch139[2]	1350.59 ± 4.74	1.6660 ± 0.0058	0.1445	n/a
	notch192	2156.52 ± 0.37	1.5188 ± 0.0003	0.1599	n/a
AWS	c5n.18xlarge	857.41 ± 6.97	2.3706 ± 0.0191	n/a	0.926
AZ	HC44rs	1812.35 ± 4.27	2.5231 ± 0.0059	n/a	1.619
	HB120rs_v2	2544.84 ± 0.65	3.1874 ± 0.0008	n/a	2.589
GCP	c2-60	960.49 ± 3.83	1.9046 ± 0.0076	n/a	0.837
	n2h-64	187.24 ± 0.73	1.8230 ± 0.0071	n/a	0.120
	n2h-80	235.93 ± 1.30	2.0600 ± 0.0113	n/a	0.188
	n2d-128	516.83 ± 3.17	1.9619 ± 0.0119	n/a	0.574
	n2d-224	855.22 ± 3.46	2.6310 ± 0.0107	n/a	1.661

Table 1: HPL Benchmark: Results

I.6 Discussion

The cost per run varies quite widely due to the time it took to finish the benchmark.

[2]Sampled 10 times instead of 6 times

II High-Performance Conjugate Gradient (HPCG) benchmarks

The High-Performance Conjugate Gradients benchmark (HPCG) [18], [16] was intended as a complement to the HPL software. All our benchmarks used the HPCG 3.1 Binary [19] (the CUDA source is not publicly available) which had been compiled with GCC 4.8.5 on a Centos7 OS and required the presence of CUDA 10.0.130 and OpenMPI 3.1.x.

II.1 General

On the CHPC clusters we installed OpenMPI 3.1.6. The required version of CUDA was already installed.

On AWS we started with an AWS Parallel Cluster Image that we created to perform calculations on GPU devices and was Centos 7 based. We downgraded its existing CUDA version 10.2 to CUDA 10.0.130. We also installed OpenMPI 3.1.6 (libfabrics and CUDA support) on the image to run our simulations.

On AZ we started with a Centos 7.6 image (Cloud Maven Solutions) in which we installed the latest NVIDIA GPU Driver extension. We subsequently downgraded CUDA to 10.0.130 and installed OpenMPI 3.1.6.

On GCP we first created a base GPU image. We started from a Centos 7 image, installed the Intel 2020.1 compiler suite and the latest CUDA driver (at the time of the image build) i. e. 460.27.04. The same base GPU image was also used to bake other GPU based images as described in Sections V, VI. We then used this GPU base image to install the additional requirements for HPCG (i. e. OpenMPI 3.1.6, CUDA 10.0.130, the HPCG binary).

II.2 Results

On the CHPC cluster we ran the HPCG code 6 times on 2 types of GPU devices: `Tesla P100-PCIE-16GB` and `Tesla V100-PCIE-16GB`.

On AWS the tests on a `p3.2xlarge` node were performed 6 times as well. This type of node contains 8 logical CPUs and 1 GPU device (`Tesla V100-SXM2-16GB`). Its enlarged version i. e. `p3.8xlarge` has 32 logical CPUs and 4 GPU devices (`Tesla V100-SXM2-16GB`). The tests on the `p3.8xlarge` nodes were repeated 3 times.

On AZ we ran the tests on nodes of the `NCv3` series [30]. This type of node contains x GPU devices (`Tesla V100-PCIE-16GB`) (where $x \in \{1, 2, 4\}$). The corresponding CPU hardware is: $6\times$ x vCPUs and $112\times$ x GB memory. All simulations on AZ were run 6 times.

On GCP we ran the tests on nodes of the `N1` type. Each GPU device (be it a $K80/V100$) was matched on the CPU side with 4 virtual cores and 15 GB Ram. To discern the GCP nodes we used the following nomenclature: `n1-$(4x)cpu-$(y)-$(x)d` where `$(x)` stands for the number of GPU devices; `$(4x)` is the number of virtual CPU cores. The string `$(y)` represents the type of the used GPU device.

In order to run valid simulations (as stipulated in the HPCG manual) we had to run for at least 1 h. For all the runs a domain size of $256 \times 256 \times 256$ was chosen. The number of MPI tasks equals the number of used GPU devices.

The results of the benchmarks (on-premises & cloud) are to be found in Table 2. For the observed values of the random variables time and Gflops the sample mean (Eq. 1) as well as the standard sample deviation (Eq. 2) are displayed.

II.3 Pricing

The `p3.2xlarge` is priced at \$3.06/h. The use of a node of the type `p3.8xlarge` is billed at \$12.24/h. Both prices were recorded at 05/14/2020.

The cost for the use of `Standard_NC6s_v3` was \$3.06/h. The use of a node of the type `Standard_NC24s_v3` was \$12.24/h. On top of that we also paid \$0.2340/h for the use of the Centos 7 (Maven Solutions) license.

On GCP the compute cost per hour[3] (On-Demand) was calculated as follows:

- `n1-4cpu-k80-1d` → \$0.651643/h
  - `n1-standard-4` - 15 GB RAM (\$0.189999/h) Ref. [9]
  - 50 GB `SSD` - (\$0.17/GB/month) Ref. [4]
  - `NVIDIA Tesla K80` - (\$0.45/h) Ref. [5]

[3] GCP counts 730 hours/month

# Devices used	Provider	Node (Type/Id)	Time (s)	Gflops	Cost per run
1	CHPC	notch001	3664.47 ± 1.88	137.81 ± 0.07	0.25 kWh
	AWS	p3.2xlarge	3661.14 ± 0.33	138.25 ± 0.04	$3.11
	AZ	Standard_NC6s_v3	3663.04 ± 2.81	138.48 ± 0.08	$3.35
	GCP	n1-4cpu-k80-1d	3663.34 ± 0.28	25.82 ± 0.01	$0.66
	GCP	n1-4cpu-v100-1d	3661.49 ± 0.72	138.62 ± 0.04	$2.73
2	CHPC	notch003	3673.90 ± 2.04	291.48 ± 0.23	0.50 kWh
	GCP	n1-8cpu-v100-2d	3660.58 ± 0.65	291.54 ± 0.17	$5.25
3	CHPC	notch002	3680.34 ± 2.78	432.07 ± 0.16	0.75 kWh
4	AWS	p3.8xlarge	3660.0 ± 0.13	575.28 ± 0.34	$12.44
	AZ	Standard_NC24s_v3	3657.14 ± 12.73	570.78 ± 0.42	$12.67
	GCP	n1-16cpu-v100-4d	3660.77 ± 7.27	562.66 ± 3.67	$10.29

Table 2: HPCG: Benchmark results

- n1-4cpu-v100-1d → $2.681643/h
  - n1-standard-4 - 15 GB RAM ($0.189999/h) Ref. [9]
  - 50 GB SSD - ($0.17/GB/month) Ref. [4]
  - NVIDIA Tesla V100 - ($2.48/h) Ref. [5]
- n1-8cpu-v100-2d → $5.161643/h
  - n1-standard-8 - 30 GB RAM ($0.379998/h) Ref. [9]
  - 50 GB SSD - ($0.17/GB/month) Ref. [4]
  - 2 NVIDIA Tesla V100 - ($4.96/h) Ref. [5]
- n1-16cpu-v100-4d → $10.121643/h
  - n1-standard-16 - 60 GB RAM ($0.759996/h) Ref. [9]
  - 50 GB SSD - ($0.17/GB/month) Ref. [4]
  - 4 NVIDIA Tesla V100 - ($9.92/h) Ref. [5]

II.4 Discussion

The #GFlops achieved per V100 device is very similar for the on-premises case as for the cloud providers. The computational output scales linearly with the number of GPU devices. The Tesla K80 has significantly lower performance in GFlops. Among the cloud providers GCP provides the lowest price for the use of similar devices. Although the K80 device is significantly cheaper per unit of time than its V100 counterpart, it is not the best choice to perform the HPCG benchmarks: the cost to achieve the same output (GFlops) as the V100 devices surpasses the cost of its V100 counterparts.

III LAMMPS Benchmarks

III.1 General

The LAMMPS package [34] was designed to perform large scale classic molecular dynamics (MD) simulations. Due to the parallel nature of the LAMMPS code it has acquired a significant following among the computational chemistry and computational material science communities.

We used the same LAMMPS source code (version 7Aug2019 [22]) for all simulations unless stated otherwise.

The calculations were (unless stated otherwise) either run in a hyperthreaded mode (HT) i. e. running 2 MPI processes per physical core, or non-hyperthreaded mode (NOHT) i. e. running 1 MPI process per physical core.

For the simulations the following input files/systems (based on benchmarks published at the LAMMPS website [33]) were used:

- polymer: bead-spring polymer melt of 100-mer chains, FENE bonds and LJ pairwise interactions with a $\sqrt[6]{2}\sigma$ cutoff (5 neighbors per atom), NVE integration.
- metallic: metallic solid, Cu EAM potential with 4.95 Å cutoff, NVE integration.
- lj: atomic fluid, Lennard-Jones potential with 2.5σ cutoff (55 neighbors per atom), NVE integration.

III.1.1 CHPC

The on-premises calculations were run on the nodes containing the AMD EPYC 7702P processors belonging to the notchpeak cluster. The LAMMPS code on the CHPC cluster was compiled using the Intel Compiler Suite (version 2019.5.281).

III.1.2 AWS

On the AWS cloud, we started with a Centos 7 image. Due to poor performance we switched to the Amazon Linux 2 OS [4] The codes which ran (with the Elastic Fabric Adapter (EFA) [15] support) on the c5n.18xlarge nodes were compiled using the Intel Compiler Suite (version 2020.1.217) and Amazon's libfabric module.

III.1.3 AZ

The LAMMPS calculations on AZ were performed on the HC44rs nodes (as described in Section I). We started with OpenLogic Centos 8.1 image. In a subsequent step we installed the Intel Compiler Suite and other dependencies. We then used the Intel Compiler Suite (version 2020.4.304) to compile the LAMMPS and VASP codes. In a subsequent step we generalized the VM to create a new computational image. In order to run the multinode calculations we used Azure's CylceCloud service/framework [29] with our newly created computational image.

III.1.4 GCP

The image used to perform the GCP calculations was built on top of a Centos 7 base image. In a first step, the Intel Compiler Suite (version 2020.1.217) and its dependencies were installed in the image. In a subsequent step, the LAMMPS and VASP codes were built from source. This newly created image was then stored within the Google Cloud and used as base image to setup a GCP Slurm Cluster [8] using Hashicorp's Terraform framework [17]. The LAMMPS simulations were run on a variety of GCP nodes: c2-60, n2h-64, n2h-80, n2d-128, n2d-224 where the last number stands for the number of virtual cores. For each type of aforemntioned GCP nodes we created two different Slurm partitions (NOHT, HT) in the Slurm/Terraform framework.

[4] We discussed the poor performance of the scaling of the LAMMPS simulations with the Amazon engineers. The lack of performance was due to 2 bugs i. e.:

1. Kernel bug in RHEL7/Centos7.
2. Bug in Amazon Linux/RHEL/CentOS with EFA - we used this workaround:
   `sudo bash -c 'echo 5128 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages'`

III.1.5 Rescale

Some of the `LAMMPS` calculations (as well as some of the VASP calculations (See Section IV) were run on the Rescale cloud. The Rescale environment allows users to easily perform HPC calculations using a graphical user interface (GUI). To automate we wrote a publicly available package to fulfill this task (i. e. submit, check, delete, modify jobs and upload the inputs and download the results). The Python package and corresponding examples can be found at:

- https://github.com/wcardoen/reschpc
- https://github.com/wcardoen/reschpc-examples

On the Rescale cloud our `LAMMPS` jobs were submitted to the AZ nodes `carbon` (we were only granted (free) access to the AZ nodes).

III.2 Results of the `LAMMPS` runs

Table 3 contains the results for the Polymer simulation. Table 4 presents the results for the Metal case. Table 5 presents the results for the Lennard-Jones system. Each MD simulation was run for $5,000$ time steps. We also ran the same molecular systems for $50,000$ time steps. The results are not presented in this document because they show essentially the same pattern.

III.3 Discussion of the `LAMMPS` results

From the timings displayed in Tables 3, 4 and 5 we can infer that the strong parallel scaling on the CHPC, AZ and Rescale instances is performing better than on the AWS and GCP counterparts due to the presence of Infiniband interfaces. The strong scaling on AWS is not performing as well as on the former clusters but it clearly outperforms its GCP counterpart where an advanced network interface is lacking. On the GCP clusters the simulations on the instances of the type `c2-60` and `n2h-64` seem to scale better. From a pecuniary perspective (`LAMMPS` case-study) we can conclude that:

1. Among the three cloud providers AZ makes it worthwhile to do HPC computations. We experienced a rather small increase of the cost per job when running on multiple nodes (combined with a decrease in walltime).
2. The return of using a larger number of nodes on GCP is low.
3. The behavior of the AWS cluster was closer to the AZ case than its GCP counterpart but still not as good as in the AZ case.

Platform	Node Type	Av. RunTime (s)/Cost($)					
		1 Node	2 Nodes	4 Nodes	6 Nodes	8 Nodes	10 Nodes
AWS	c5n.18xlarge (NOHT)	44.22 ± 0.36 s $0.0478	23.08 ± 0.2 s $0.0499	11.88 ± 0.15 s $0.0513	8.98 ± 0.08 s $0.0582	7.54 ± 0.06 s $0.0652	8.81 ± 0.25 s $0.0951
	c5n.18xlarge (HT)	39.59 ± 0.2 s **$0.0428**	29.51 ± 0.24 s $0.0637	24.62 ± 0.25 s $0.1063	17.72 ± 0.28 s $0.1148	16.55 ± 0.31 s $0.1429	17.76 ± 0.16 s $0.1918
AZ	hc44rs (NOHT)	42.14 ± 0.41 s $0.0376	19.27 ± 0.43 s **$0.0344**	10.02 ± 0.49 s $0.0358	6.74 s $0.0361	5.2 ± 0.04 s $0.0371	
CHPC	AMD (NOHT)	26.98 ± 0.06 s 0.001686 **kWh**	13.14 ± 0.03 s 0.000742 **kWh**	7.22 ± 0.04 s 0.000366 **kWh**	5.42 ± 0.01 s 0.000267 **kWh**	4.48 ± 0.05 s 0.000220 **kWh**	3.84 ± 0.02 s 0.000192 **kWh**
	AMD (HT)	23.05 ± 0.29 s 0.001565 **kWh**	10.67 ± 0.14 s 0.000694 **kWh**	6.37 ± 0.22 s 0.000381 **kWh**	5.25 ± 0.27 s 0.000310 **kWh**	4.44 ± 0.54 s 0.000255 **kWh**	4.17 ± 0.27 s 0.000245 **kWh**
GCP	c2-60 (NOHT)	53.17 ± 0.25 s $0.0466	35.21 ± 0.58 s $0.0617	22.71 ± 0.23 s $0.0796	23.69 ± 0.74 s $0.1245	21.39 ± 0.67 s $0.1499	20.24 ± 0.73 s $0.1773
	c2-60 (HT)	47.23 ± 0.10 s $0.0414	32.84 ± 0.31 s $0.0575	30.19 ± 0.48 s $0.1058	26.95 ± 0.49 s $0.1416	26.33 ± 0.65 s $0.1845	25.07 ± 1.06 s $0.2196
	n2h-64 (NOHT)	54.83 ± 0.57 s $0.0351	33.93 ± 0.94 s $0.0435	22.25 ± 0.42 s $0.057	21.58 ± 0.76 s $0.083	17.17 ± 0.29 s $0.088	20.10 ± 0.86 s $0.1288
	n2h-64 (HT)	46.62 ± 0.27 s $0.0299	32.30 ± 0.86 s $0.0414	23.34 ± 0.81 s $0.0598	24.51 ± 0.49 s $0.0942	23.27 ± 0.77 s $0.1192	25.37 ± 1.89 s $0.1625
	n2h-80 (NOHT)	46.50 ± 0.67 s $0.0372	70.69 ± 11.61 s $0.1131	67.68 ± 16.07 s $0.2165	106.45 ± 22.55 s $0.5111	75.41 ± 6.73 s $0.4826	57.65 ± 6.87 s $0.4611
	n2h-80 (HT)	40.16 ± 0.58 s $0.0321	38.37 ± 2.57 s $0.0614	29.81 ± 0.84 s $0.0954	34.61 ± 1.48 s $0.1661	29.74 ± 3.20 s $0.1903	29.23 ± 1.95 s $0.2338
	n2d-128 (NOHT)	22.54 ± 0.10 s $0.0251	45.77 ± 6.27 s $0.1018	58.23 ± 4.53 s $0.2591	44.37 ± 2.02 s $0.2961	97.75 ± 4.50 s $0.8697	
	n2d-128 (HT)	21.58 ± 0.24 s $0.024	53.36 ± 3.93 s $0.1187	50.07 ± 4.93 s $ 0.2228			
	n2d-224 (NOHT)	14.08 ± 0.4 s 0.0274	33.63 ± 3.66 s 0.1307	47.94 ± 1.95 s 0.3728			
	n2d-224 (HT)	11.08 ± 0.13 s **$0.0215**	28.45 ± 2.29 s $0.1106	62.30 ± 1.41 s $0.7266			
Rescale		38.57 s $0.52	17.45 s $2.21	8.78 s $4.17	6.24 s $7.48	4.77 s $9.06	4.00 s $9.30

Table 3: LAMMPS: Benchmark results for the Polymer system (5,000 Time steps)

Platform	Node Type	Av. RunTime (s)/Cost($)					
		1 Node	2 Nodes	4 Nodes	6 Nodes	8 Nodes	10 Nodes
AWS	c5n.18xlarge (NOHT)	64.03 ± 0.28 s $0.0692	35.98 ± 0.09 s $0.0777	20.28 ± 0.11 s $0.0876	15.22 ± 0.08 s $0.0986	17.03 ± 0.14 s $0.1472	13.28 ± 0.3 s $0.1435
	c5n.18xlarge (HT)	60.83 ± 0.21 s **$0.0657**	44.37 ± 0.07 s $0.0958	36.41 ± 0.13 s $0.1573	28.18 ± 0.25 s $0.1826	27.8 ± 0.42 s $0.2402	28.62 ± 0.32 s $0.3091
AZ	hc44rs (NOHT)	70.46 ± 0.33 s $0.0629	33.92 ± 0.15 s **$0.0606**	17.89 ± 0.2 s $0.0639	13.22 ± 0.48 s $0.0708	10.23 ± 0.01 s $0.0731	8.32 ± 0.15 s $0.0743
CHPC	AMD (NOHT)	119.36 ± 0.06 s 0.007460 **kWh**	58.83 ± 0.67 s 0.003320 **kWh**	30.07 ± 0.03 s 0.001526 **kWh**	20.93 ± 0.02 s 0.001032 **kWh**	16.37 ± 0.01 s 0.000804 **kWh**	13.7 ± 0.02 s 0.000685 **kWh**
	AMD (HT)	89.96 ± 0.11 s 0.006107 **kWh**	44.29 ± 0.74 s 0.002881 **kWh**	22.72 ± 0.3 s 0.001360 **kWh**	16.95 ± 0.19 0.001002 **kWh**	13.43 ± 0.49 s 0.000770 **kWh**	11.62 ± 0.5 s 0.000682 **kWh**
GCP	c2-60 (NOHT)	92.49 ± 0.18 s $0.081	61.72 ± 0.29 s $0.1081	39.26 ± 0.31 s $0.1376	36.52 ± 0.39 s $0.1919	33.08 ± 0.46 s $0.2318	27.4 ± 0.26 s $0.24
	c2-60 (HT)	86.42 ± 0.20 s $0.0757	57.62 ± 0.13 s $0.1009	46.22 ± 0.22 s $0.162	37.22 ± 0.44 s $0.1956	38.73 ± 0.54 s $0.2715	32.71 ± 0.49 s $0.2866
	n2h-64 (NOHT)	92.86 ± 0.38 s $0.0595	58.20 ± 0.48 s $0.0746	38.93 ± 0.29 s $0.0997	34.89 ± 0.26 s $0.1341	29.67 ± 0.46 s $0.152	27.95 ± 0.54 s $0.179
	n2h-64 (HT)	86.70 ± 0.24 s **$0.0555**	57.37 ± 0.33 s $0.0735	41.88 ± 0.17 s $0.1073	36.36 ± 0.32 s $0.1397	33.30 ± 0.35 s $0.1706	36.26 ± 0.59 s $0.2322
	n2h-80 (NOHT)	78.35 ± 0.73 s $0.0627	85.85 ± 16.82 s $0.1373	65.53 ± 4.10 s $0.2097	74.01 ± 11.91 s $0.3552	79.97 ± 10.49 s $0.5117	63.81 ± 10.03 s $0.5104
	n2h-80 (HT)	73.60 ± 0.64 s $0.0589	64.15 ± 3.62 s $0.1026	45.66 ± 1.41 s $0.1461	49.44 ± 2.05 s $0.2372	41.60 ± 5.02 s $0.2662	38.10 ± 0.83 s $0.3048
	n2d-128 (NOHT)	95.98 ± 0.10 s $0.1067	69.43 ± 1.20 s $0.1544	50.52 ± 1.40 s $0.2247	44.59 ± 0.96 s $0.2976	49.59 ± 1.70 s $0.4412	
	n2d-128 (HT)	78.24 ± 0.62 s $0.087	56.49 ± 1.70 s $0.1257	46.77 ± 1.74 s $0.208			
	n2d-224 (NOHT)	61.28 ± 1.47 s $0.1191	48.02 ± 0.73 s $0.1867	45.49 ± 0.75 s $0.3537			
	n2d-224 (HT)	43.30 ± 0.12 s $0.0842	42.02 ± 1.05 $0.1634	76.11 ± 3.33 s $0.8876			
Rescale		38.57 s $0.52	17.45 s $2.21	8.78 s $4.17	6.24 s $7.48	4.77 s $9.06	4.00 s $9.30

Table 4: LAMMPS: Benchmark results for the Metal system (5,000 Time steps)

Platform	Node Type	Av. RunTime (s)/Cost($)					
		1 Node	2 Nodes	4 Nodes	6 Nodes	8 Nodes	10 Nodes
AWS	c5n.18xlarge (NOHT)	31.76 ± 0.09 s	18.11 ± 0.07 s	9.56 ± 0.04 s	7.46 ± 0.1 s	8.95 ± 0.06	5.88 ± 0.31
		$0.0343	$0.0391	$0.0413	$0.0484	$0.0773	$0.0635
	c5n.18xlarge (HT)	32.05 ± 0.05 s	21.73 ± 0.17 s	17.76 ± 0.13 s	9.34 ± 0.1 s	9.04 ± 0.18	13.4 ± 0.16
		$0.0346	$0.0469	$0.0767	$0.0605	$0.0781	$0.1447
AZ	hc44rs (NOHT)	39.41 ± 0.19 s	16.28 ± 0.12 s	8.13 ± 0.1 s	6.06 ± 0.05 s	4.67 ± 0.03	4.06 ± 0.28
		$0.0352	$0.0291	**$0.0291**	$0.0325	$0.0333	$0.0363
CHPC	AMD (NOHT)	36.14 ± 0.06 s	18.12 ± 0.05 s	9.54 ± 0.02 s	6.81 ± 0.01 s	5.55 ± 0.01 s	4.81 ± 0.02 s
		0.002259 **kWh**	0.001023 **kWh**	0.000484 **kWh**	0.000336 **kWh**	0.000273 **kWh**	0.000241 **kWh**
	AMD (HT)	36.44 ± 0.1 s	17.58 ± 0.29 s	9.26 ± 0.23 s	7.22 ± 0.16 s	6.02 ± 0.37 s	5.02 ± 0.16 s
		0.002474 **kWh**	0.001143 **kWh**	0.000554 **kWh**	0.000427 **kWh**	0.000345 **kWh**	0.000295 **kWh**
GCP	c2-60 (NOHT)	47.49 ± 1.42 s	29.81 ± 0.1 s	18.54 ± 0.08 s	15.43 ± 0.1 s	15.82 ± 0.23 s	13.71 ± 0.12 s
		$0.0416	$0.0522	$0.065	$0.0811	$0.1108	$0.1201
	c2-60 (HT)	44.75 ± 0.41 s	28.57 ± 0.17 s	21.52 ± 0.21 s	20.32 ± 0.19 s	18.05 ± 0.26 s	16.64 ± 0.33 s
		$0.0392	$0.0501	$0.0754	$0.1068	$0.1265	$0.1458
	n2h-64 (NOHT)	45.34 ± 0.3 s	27.25 ± 0.38 s	17.91 ± 0.15 s	15.95 ± 0.1 s	13.6 ± 0.17 s	12.82 ± 0.16 s
		$0.029	$0.0349	$0.0459	$0.0613	$0.0697	$0.0821
	n2h-64 (HT)	43.03 ± 0.25 s	27.72 ± 0.16 s	19.93 ± 0.19 s	17.31 ± 0.27 s	15.98 ± 0.11 s	15.63 ± 0.34 s
		$0.0276	$0.0355	$0.0511	$0.0665	$0.0819	$0.1001
	n2h-80 (NOHT)	39.44 ± 0.43 s	40.95 ± 7.32 s	41.52 ± 6.16 s	42.74 ± 7.02 s	54.53 ± 17.5 s	33.73 ± 6.9 s
		$0.0315	$0.0655	$0.1328	$0.2051	$0.3489	$0.2698
	n2h-80 (HT)	37.82 ± 0.3 s	32.05 ± 2.44 s	22.7 ± 1.23 s	22.25 ± 1.82 s	19.39 ± 3.29 s	17.15 ± 0.86 s
		$0.0303	$0.0513	$0.0726	$0.1068	$0.1241	$0.1372
	n2d-128 (NOHT)	29.16 ± 0.05 s	22.57 ± 0.21 s	18.4 ± 0.2 s	16.33 ± 0.44 s	15.36 ± 0.94 s	
		$0.0324	$0.0502	$0.0819	$0.1089	$0.1366	
	n2d-128 (HT)	32.52 ± 0.55 s	22.22 ± 0.35 s	15.92 ± 0.34 s			
		$0.0362	$0.0494	$0.0708			
	n2d-224 (NOHT)	18.71 ± 0.72 s	19.07 ± 0.36 s	15.91 ± 0.64 s			
		$0.0364	$0.0741	$0.1237			
	n2d-224 (HT)	18.14 ± 0.06 s	17.49 ± 0.09	23.32 ± 2.00 s			
		$0.0353	$0.068	$0.2719			

Table 5: LAMMPS: Benchmark results for the LJ system (5,000 Time steps)

IV VASP Benchmarks

IV.1 General

The Vienna Ab initio Simulation Package (VASP) [23,24] has been designed to model materials based on quantum mechanical calculations (pseudo-potentials, projector augmented wave method and plane wave basis sets). Although the density functional method (DFT) forms the corner stone of the code, VASP offers some more advanced methods to treat electronic correlation.

All the VASP simulations were performed using VASP (version 5.4.4). We performed calculations on the following systems:

1. Medium benchmark: HfO_2 with 96 atoms (32 Hf atoms and 64 O atoms) per unit cell.
2. Large benchmark: TiO_2 with 540 atoms (180 Ti atoms and 360 O atoms) per unit cell.

The input files for the aforementioned benchmarks were previously published on the TeraGrid website.

IV.1.1 CHPC

The on-premises calculations were run on the nodes containing the AMD EPYC 7702P processors belonging to the notchpeak cluster. The VASP code installed on the CHPC cluster was compiled using the Intel Compiler Suite (version 2017.4.196).

IV.1.2 AWS

On the AWS cloud, we started with the Amazon Linux 2 image (for identical cause as described in III.1.2). The VASP code which ran (with the Elastic Fabric Adapter (EFA) [15] support) on the c5n.18xlarge nodes were compiled using the Intel Compiler Suite (version 2020.4.304) and Amazon's libfabric module.

IV.1.3 AZ

The VASP calculations on AZ were performed on the HC44rs nodes (as described in Section I). The images to perform the simualtions was described previously in III.1.3. We used Azure's CylceCloud service/framework [29] with our newly created computational image to run the simulations.

IV.1.4 GCP

The image used to perform the VASP calculations on GCP was described earlier in Section III.1.4. The VASP simulations were run on a variety of GCP nodes: c2-60, n2h-64 where the last number stands for the number of virtual cores. The Slurm Cluster was again set up using Terraform (as in the LAMMPS case).

IV.2 Benchmark results & Discussion

The VASP results for the Medium molecular system are to be found in Table 6. The results for the Large molecular system/benchmark are to be found in Table 7. Quite a few data points are missing due to a variety of reasons: crashes, mismatch number k-points and MPI tasks, numerical instabilities (e. g, numerical loss of the hermiticity of the Hamiltonian, linear dependencies, etc.)

In the medium benchmark the wall time of the simulation increases when using 4 or 6 nodes. Single run nodes appear to be the most cost efficient. The situation is similar for the large benchmark.

Platform	Node Type	Av. RunTime (s)/Cost($)					
		1 Node	2 Nodes	4 Nodes	6 Nodes	8 Nodes	10 Nodes
CHPC	`AMD (NOHT)`	176.01 ± 0.44 s 0.013079 **kWh**	98.60 ± 0.61 s 0.006407 **kWh**	69.94 ± 0.54 s 0.004316 **kWh**	75.85 ± 2.55 s 0.004566 **kWh**	75.09 ± 1.31 s 0.004324 **kWh**	89.55 ± 2.59 s 0.005217 **kWh**
AWS	`c5n.18xlarge (NOHT)`		110.90 ± 0.96 s $0.240		103.47 ± 6.45 s $0.671	86.54 ± 5.63 s $0.748	215.83 ± 100.37 s $2.331
	`c5n.18xlarge (HT)`	218.64 ± 4.94 s $0.236	146.72 ± 3.92 s $0.317	111.12 ± 5.38 s $0.480	130.81 ± 2.02 s $0.848	133.62 ± 4.25 s $1.155	
AZ	`hc44rs (NOHT)`		173.73 ± 16.25 s $0.310	145.79 ± 16.05 s $0.521			
GCP	`c2-60 (NOHT)`			156.80 ± 4.25 s $0.549			
	`c2-60 (HT)`		272.70 ± 19.73 s $0.478	286.87 ± 11.78 s $1.005	491.42 ± 9.24 s $2.583		
	`n2h-64 (NOHT)`		223.66 ± 29.63 s $0.287	192.93 ± 17.76 s $0.494	208.57 ± 5.19 s $0.802		
	`n2h-64 (HT)`	353.96 ± 35.21 s $0.227	271.85 ± 14.77 s $0.348	272.63 ± 4.66 s $0.699	464.71 ± 5.23 s		

Table 6: VASP: Results for the `Medium` Benchmark

Platform	Node Type	Av. RunTime (s)/Cost ($)			
		1 Node	2 Nodes	4 Nodes	6 Nodes
CHPC	`AMD (NOHT)`	457.03 ± 0.85 s 0.033550 **kWh**	262.62 ± 0.78 s 0.018643 **kWh**	172.15 ± 1.59 s 0.011856 **kWh**	163.27 ± 1.28 s 0.010781 **kWh**
AWS	`c5n.18xlarge (NOHT)`			112.20 ± 0.30 s $0.485	
	`c5n.18xlarge (HT)`		224.18 ± 1.27 s $0.484		
AZ	`hc44rs (NOHT)`	294.61 ± 0.76 s $0.263	170.08 ± 1.22 s $0.304		
GCP	`c2-60 (NOHT)`			175.26 ± 0.86 s $0.614	221.22 ± 4.87 s $1.163
	`c2-60 (HT)`		293.17 ± 3.92 s $0.514	266.52 ± 6.91 s $0.934	
	`n2h-64 (NOHT)`			177.01 ± 3.71 s $0.454	
	`n2h-64 (HT)`		287.91 ± 11.99 s $0.369		

Table 7: VASP: Results for the `Large` Benchmark

V AMBER Benchmarks

V.1 General

We used the AMBER 18 [11] and AmberTools19 codes to perform all AMBER simulations. We used the Amber18 RCW Benchmark Suite [35] to perform the benchmarks (1 GPU device).

V.1.1 CHPC

On the CHPC clusters the AMBER source codes were compiled using the Intel Compiler Suite (v. 2017.4.196) and CUDA (v. 10.1.168). The simulations were run on the nodes notch086 and notch003.

V.1.2 AWS

On the AWS platform we started with a Centos7 based image. In order to compile the AMBER 18 and AmberTools19 codes, we used gcc (v. 4.8.5), CUDA (v. 10.2) and OpenMPI (v. 4.0.2). The cost to run a p3.2xlarge instance was \$3.06/h.

V.1.3 AZ

On AZ we started with an OpenLogic image (cost \$0.234/h) loaded with a Centos 7.9 OS and the NVIDIA Device Driver (v. 465.19.01). The CUDA version had to be downgraded to (v. 10.2) (as required by the AMBER 18 installation). To install Amber as such we used: gcc (v. 4.8.5), CUDA (v. 10.2) and OpenMPI (v. 4.1.0) (compiled from source). The hourly rate for a Standard NC6s_v3 instance was \$3.06/h.

V.1.4 GCP

On GCP we started with a Centos7 based image, and first installed CUDA 10.2 (highest version allowed by AMBER 18) and OpenMPI 4.1.0. On top of it we built the AMBER binaries.

V.2 Results & Discussion

The results for the different benchmarks studies are represented in Tables 8, 9, 10, 11, 12. They were all based on 6 measurements except for AWS where we used 3. For the time measurements the average time $\overline{x}$ (Eq. 1) and the standard sample deviation s (Eq. 2)are displayed.

From the results in the aformentioned Tables, we can deduce that the calculations on the $V100$ devices were the fastest on the GCP and the AWS instances followed by the $V100$ devices on the AZ and the CHPC instances. In the commercial cloud the best price per run was obtained on GCP, followed by AWS and AZ. For Amber we also observed several cases of excellent performance (walltime based) on the GeForce RTX 2080 Ti device. The Tesla T4 card performed quite well (in a financial sense) although its corresponding walltimes were rather long.

Provider	Node	Device type	Time (s)	Cost
CHPC	notch086	GeForce RTX 2080 Ti	$8,429.42 \pm 18.49$	0.782061 kWh
	notch003	Tesla V100-PCIE-16GB	$7,585.96 \pm 53.38$	0.783332 kWh
AWS	p3.d2xlarge	Tesla V100-SXM2-16GB	$7,112.87 \pm 26.09$	\$6.046
AZ	Standard NC6s_v3	Tesla V100-PCIE-16GB	$7,556.03 \pm 78.86$	\$6.914
GCP	n1-standard-8	Tesla P4	$21,462.99 \pm 31.88$	\$4.456
		Tesla T4	$16,315.84 \pm 193.42$	\$4.521
		Tesla V100-SXM2-16GB	$7,188.10 \pm 42.80$	\$5.745

Table 8: Benchmark results (100 ns) for the Jac_Production_NVE - 23,558 atoms PME (4 fs).

Provider	Node	Device type	Time (s)	Cost
CHPC	notch086	GeForce RTX 2080 Ti	$8,338.90 \pm 205.50$	0.773663 kWh
	notch003	Tesla V100-PCIE-16GB	$7,208.36 \pm 23.03$	0.744341 kWh
AWS	p3.d2xlarge	Tesla V100-SXM2-16GB	$6,811.55 \pm 151.93$	$5.790
AZ	Standard NC6s_v3	Tesla V100-PCIE-16GB	$7,145.71 \pm 48.18$	$6.538
GCP	n1-standard-8	Tesla P4	$32,056.40 \pm 49.43$	$6.656
		Tesla T4	$19,827.88 \pm 292.12$	$5.494
		Tesla V100-SXM2-16GB	$6,885.24 \pm 42.15$	$5.503

Table 9: Benchmark results (20 ns) for the Factor_IX_Production_NVE - 90,906 atoms PME.

Provider	Node	Device type	Time (s)	Cost
CHPC	notch086	GeForce RTX 2080 Ti	$9,698.06 \pm 17.24$	0.899763 kWh
	notch003	Tesla V100-PCIE-16GB	$7,608.76 \pm 10.09$	0.785687 kWh
AWS	p3.d2xlarge	Tesla V100-SXM2-16GB	$7,262.54 \pm 3.07$	$6.173
AZ	Standard NC6s_v3	Tesla V100-PCIE-16GB	$7,518.06 \pm 69.13$	$6.879
GCP	n1-standard-8	Tesla P4	$35,405.00 \pm 38.57$	$7.351
		Tesla T4	$25,645.59 \pm 219.73$	$7.106
		Tesla V100-SXM2-16GB	$7,226.09 \pm 12.01$	$5.776

Table 10: Benchmark results (5 ns)f or the Cellulose_Production_NVE - 408,609 atoms PME.

Provider	Node	Device type	Time (s)	Cost
CHPC	notch086	GeForce RTX 2080 Ti	$7,880.96 \pm 200.65$	0.731176 kWh
	notch003	Tesla V100-PCIE-16GB	$4,736.91 \pm 73.93$	0.489138 kWh
AWS	p3.d2xlarge	Tesla V100-SXM2-16GB	$4,534.31 \pm 56.07$	$3.854
AZ	Standard NC6s_v3	Tesla V100-PCIE-16GB	$4,682.85 \pm 45.52$	$4.285
GCP	n1-standard-8	Tesla P4	$20,032.77 \pm 157.00$	$4.159
		Tesla T4	$15,333.29 \pm 229.78$	$4.248
		Tesla V100-SXM2-16GB	$4,395.46 \pm 122.05$	$3.513

Table 11: Benchmark results (50 ns) for the Myoglobin_Production - 2,492 atoms GB.

Provider	Node	Device type	Time (s)	Cost
CHPC	notch086	GeForce RTX 2080 Ti	$28,275.34 \pm 144.03$	2.623318 kWh
	notch003	Tesla V100-PCIE-16GB	$15,649.34 \pm 60.08$	1.615964 kWh
AWS	p3.d2xlarge	Tesla V100-SXM2-16GB	$14,819.90 \pm 10.17$	$12.597
AZ	Standard NC6s_v3	Tesla V100-PCIE-16GB	$15,746.31 \pm 90.15$	$14.408
GCP	n1-standard-8	Tesla P4	$102,410.11 \pm 182.75$	$21.263
		Tesla T4	$69,009.58 \pm 1,172.89$	$19.121
		Tesla V100-SXM2-16GB	$14,973.14 \pm 87.18$	$11.968

Table 12: Benchmark results (5 ns) for Nucleosome_Production - 25,095 atoms GB.

VI Machine Learning Applications

We have tested several ML benchmarks.

VI.1 AI Benchmark

VI.1.1 General

The AI Benchmark [20] is an open source python library to evaluate AI Performance on various hardware platforms. Under the hood it relies on the TensorFlow machine learning library. It contains a wide variety of tests (Recurrent Neural Nets (RNN), Convolutional Neural Nets (CNN), etc.). The package gauges the performance of the code by returning a Device AI Score (i. e. the sum of Training and Inference Scores).

The AI Benchmark package requires the installation of the following entities: the NVIDIA Device Driver, CUDA Toolkit, the CuDNN library, Tensorflow and the AI Benchmark library. All the tests have been run on 1 GPU device.

The On-Prem results were obtained on an array of different GPU devices. On the Azure Cloud the results were obtained on a VM of the type `Standard_NC6s_v3`: 6 vCPUs, `Intel Xeon E5-2690 v4 Broadwell` and 112 GB Memory. The GPU Device was a `Tesla V100-PCIE-16GB` card. The cost/h on AZ had 2 factors: 3.06$/h for the VM and 0.234$/h for the Centos7.6 license (Maven Solutions).

On AWS all results were computed on an instance of the type `p3.2xlarge` which contains a 1 GPU Device: `Tesla V100-PCIE-16GB`. Its cost was: 3.06$/h.

On GCP, all packages were installed on top of a previously created GPU base image (see Section II). The new image had the following modifications w. r. t. the base image:

- Latest version of miniconda
- CUDA 11.0.2
- CUDNN 8.0.3
- Tensorflow-GPU 2.24
- AI-Benchmark

In order to run GPU tests we again used nodes of the `N1` type. To discern the GCP nodes we use the following nomenclature: `n1-$(x)vcpu-$(y)` where `$(x)` stands for the number of virtual CPU cores. The string `$(y)` represents the type of the GPU device which has been used.

VI.1.2 Results & discussion

The results of the runs are shown in Table 13. A series of 10 independent runs was done for each device. The walltime on instances containing a `Tesla V100` device was the lowest on an on-premises instance, followed by the AZ, AWS and GCP instances. The cost to perform these run on a `Tesla V100` device was the lowest on the GCP instance, followed by AZ and AWS.

The cost to run the benchmark on GCP's `Tesla K80, P4, T4` cards was substantially lower than on its `Tesla V100` counterpart, but its AI score was decreased almost proportionally. When we calculate the cost per unit of AI-Score the result obtained on a `Tesla T4` device outperforms `Tesla V100` device.

VI.2 PyTorch Benchmark

VI.2.1 General

The PyTorch benchmark [36] tests the learning and inference speed of various CNN models using the PyTorch framework [31]. In order to do the tests/comparisons we use the same version of Python (3.8.3), CUDA Version (10.2) and CuDNN (7605) The timings for each run are an aggregate (several models as well half-precision, single precision and double precision formats).

Provider	Device	Node Type/Id	Device AI Score	Time (s)	Cost per run
CHPC	GeForce GTX 1080 Ti	notch060	20995	636.97 ± 15.86	0.142 kWh
	GeForce RTX 2080 Ti	notch088	27908	512.02 ± 15.05	0.058 kWh
	Tesla V100-PCIE-16GB	notch002	31830	523.79 ± 19.91	0.059 kWh
	A100-PCIE-40GB	notch293	51512	452.89 ± 16.36	0.074 kWh
AWS	Tesla V100-SXM2-16GB	p3.2xlarge	33658	598.92 ± 4.79	$0.51
AZ	Tesla V100-PCIE-16GB	Standard_NC6s_v3	33545	527.70 ± 3.34	$0.48
GCP	Tesla K80	n1-8vcpu-k80	6035	1094.22 ± 20.46	$0.26
	Tesla P4	n1-8vcpu-p4	10797	797.33 ± 20.56	$0.22
	Tesla T4	n1-8vcpu-t4	13437	786.04 ± 13.59	$0.16
	Tesla V100-SXM2-16GB	n1-4vcpu-v100	32492	596.02 ± 12.27	$0.44
	Tesla V100-SXM2-16GB	n1-8vcpu-v100	33370	555.00 ± 0.94	$0.44

Table 13: AI-Benchmark: Results

VI.2.2 Results & discussion

The benchmark results are displayed Table 14. The on-Prem results were sampled through 10 independent runs; the cloud results were sampled through 6 independent runs.

The cost to run the simulation on a Tesla V100 was the cheapest on GCP and the most expensive on AZ. The cost was the cheapest on the Tesla K80.

Provider	Device	Node Type/Id	Time (s)	Cost per run
CHPC	GeForce GTX 1080 Ti	notch060	5640.67 ± 73.61	1.368 kWh
	GeForce RTX 2080 Ti	notch088	4891.52 ± 32.00	0.605 kWh
	Tesla V100-PCIE-16GB	notch003	1847.47 ± 14.78	0.225 kWh
AWS	Tesla V100-SXM2-16GB	p3.2xlarge	1804.59 ± 10.28	$1.53
AZ	Tesla V100-PCIE-16GB	Standard_NC6s_v3	1800.43 ± 17.27	$1.65
GCP	Tesla K80	n1-8vcpu-k80	4119.77 ± 16.47	$ 0.96
	Tesla T4	n1-8vcpu-t4	8363.76 ± 56.73	$ 1.72
	Tesla V100-SXM2-16GB	n1-8vcpu-v100	1730.15 ± 21.62	$ 1.38

Table 14: PyTorch-Benchmark: Results

VI.3 Lymphoma Benchmark

VI.3.1 General

We also ran a software package (CNN using Tensorflow) [26] to discern different Lymphoma types (Burkitt and Diffuse Large B-cell). The code ran on different types of GPU devices (CHPC's notchpeak cluster) as well as on the Azure GPU devices and the GPU devices. In order to run the Lymphoma package we installed on each of the platforms the following libraries/packages:

- CUDA 10.0.130_410.48
- CUDNN 7.6.2 for CUDA 10.0
- Anaconda3-2018.12

VI.3.2 Results & discussion

The Lymphoma code used only 1 GPU Device but was not very efficient in its use of the GPU device. Based on the timing of 1 run on the Standard_NC6s_v3 card, it was computationally and monetarily more advantageous to run

on a `Standard_NC24s_v3` node (and use 1 of the 4 GPU devices) instead of using the `Standard_NC6s_v3` node with its 1 GPU device. We only did 1 run on the `Standard_NC6s_v3` node. All runs on the `Standard_NC24s_v3` node were executed 3 times. The runs on CHPC's notchpeak cluster were executed 6 times. All results are to be found in Table 15.

Provider	Device	Node (Type/Id)	Dataset	Calc. type	Time (s)	Cost per run
CHPC	GeForce RTX 2080 Ti	notch088	I	1	5087.79 ± 11.17	0.670 **kWh**
		notch088		2	5165.52 ± 68.78	0.667 **kWh**
		notch088	II	1	8735.70 ± 36.26	1.157 **kWh**
		notch088		2	8994.32 ± 68.05	1.154 **kWh**
	Tesla V100-PCIE-16GB	notch002	I	1	7090.12 ± 63.64	0.828 **kWh**
		notch003		2	7574.27 ± 51.30	0.877 **kWh**
		notch002	II	1	12258.02 ± 420.89	0.877 **kWh**
		notch003		2	13292.88 ± 416.82	1.537 **kWh**
	A100-PCIE-40GB	notch293	I	1	1267.93 ± 21.86	0.181 **kWh**
		notch293		2	1267.80 ± 5.86	0.180 **kWh**
		notch293	II	1	1265.47 ± 8.20	0.180 **kWh**
AWS	Tesla V100-SXM2-16GB	p3.8xlarge	I	1	7661.52 ± 15.11	$26.04
				2	7078.41 ± 259.99	$24.07
			II	1	12376.48 ± 1077.69	$42.08
				2	12843.91 ± 397.18	$43.67
AZ	Tesla V100-PCIE-16GB	Standard_NC6s_v3	I	1	108776.07	$99.53
	Tesla V100-PCIE-16GB	Standard_NC24s_v3	I	1	4161.35 ± 14.66	$14.42
				2	4144.68 ± 10.83	$14.36
			II	1	7298.47 ± 19.37	$25.29
				2	7124.85 ± 17.16	$24.69
GCP	Tesla V100-PCIE-16GB	n1-standard16-2v100	I	1	8631.38 ± 46.27	$13.74
		n1-highmem16-2v100	I	1	8145.06 ± 179.97	$13.39
				2	8437.17 ± 247.48	$13.87
			II	1	14506.39 ± 760.88	$23.85
				2	15195.17 ± 139.06	$24.98
		n1-highcpu16-2v100	I	1	8756.90 ± 102.81	$13.47

Table 15: Results for the Lymphoma Benchmark

VII Container

Containers have found a popular audience in the IT world. In the cloud, they can be invoked through a variety of services (for AWS see e. g. [1], for AZ see e. g. [27], for GCP see e. g. [13]).

VII.1 General

In the past few years NVIDIA started to offer GPU-optimized containers for AI, Machine Learning and HPC (NVIDIA GPU Cloud - `NGC`). The `NGC` containers can run in either a Docker [28] or Singularity [25] mode.

Besides containers NVIDIA also provided OS images to the main cloud vendors to run its optimized `NGC` containers.

We used again the LAMMPS version `10Feb2021` software (downloadd from `NGC` [6]) to generate benchmark timings. The input file `in.lj.txt` which has been used is identical to the one on `NGC` web site[5] with one major exception: the initial number of time steps was increased from 100 to 50,000 in order to get more sensible timings.

VII.1.1 CHPC

On CHPC we ran the `LAMMPS` benchmark (`NGC` container) using `Singularity` 3.6.4 and `CUDA` 11.3.58. We ran the `NGC` container on different types and numbers of GPU devices: `Ti2080`, `V100`, `A100`.

VII.1.2 AWS

We used the `NVIDIA HPC SDK GPU-Optimized Image` (based on `Ubuntu` 18.04.5 `LTS`) to run the `LAMMPS NGC` docker container. The following AWS instances were used to perform the simulations: `p3.2xlarge` (1 V100 device), `p3.4xlarge` (2 V100 devices), and `p3.16xlarge` (8 V100 devices)

VII.1.3 AZ

We used `NVIDIA GPU-Optimized Image for AI & HPC` (based on `Ubuntu` 18.04.5 `LTS`) to run the `LAMMPS NGC` docker container. The following AZ instances were used to perform the simulations: `nc6s_v3` (1 V100 device), `nc12s_v3` (2 V100 devices), `nc24s_v3` (4 V100 devices).

VII.1.4 GCP

We used `NVIDIA HPC SDK GPU-Optimized Image` (based on `Ubuntu` 18.04.5 `LTS`) to run the `LAMMPS NGC` docker container. The following GCP instances were used to perform the simulations: `n1-standard-8`, `n1-standard-16`, `n1-standard-32`, `n1-standard-64` (with 1, 2, 4 and 8 V100 devices) as well as `a2-highgpu-1g`, `a2-highgpu-2g` and `a2-highgpu-4g` (containing 1, 2 and 4 A100 devices).

VII.2 Results & discussion

Table 16 contains the results for the `LAMMPS NGC` benchmark. For the time measurements the average time $\overline{x}$ (Eq. 1) and the standard sample deviation s (Eq. 2) are displayed.

In Table 16 we observe that the walltime per run using 1 `V100` device is the lowest at AWS, followed by GCP, AZ and then the on-premises environment. The scaling when using multiple `V100` devices is behaving the best at AWS, followed by GCP and really disappointing in the Azure cloud. From a pecuniary perspective GCP was the cheapest (to use 1 `V100` devices). If we consider also the use of the newest `A100` devices we see the excellent scaling on GCP and on-premises. For the cloud the use of the `A100` on GCP is the most cost-effective.

[5] https://lammps.sandia.gov/inputs/in.lj.txt (`NVE` ensemble, containing 8,192,000 atoms and the use of a Lennard-Jones potential)
[6] Standard sample deviation

Provider	Node	Device type	#Devices	Time (s)[6]	Cost per Run
CHPC	notch271	Ti2080	1	4430.69 ± 12.98	2.534545 **kWh**
			2	2323.68 ± 1.32	1.335266 **kWh**
			4	1589.66 ± 4.17	0.915479 **kWh**
			8	756.68 ± 4.11	0.412538 **kWh**
	notch003 notch001 notch002	V100	1	1231.02 ± 1.58	0.195453 **kWh**
			2	679.35 ± 0.29	0.137960 **kWh**
			3	598.84 ± 0.79	0.130869 **kWh**
	notch293	A100	1	766.61 ± 0.63	0.210526 **kWh**
			2	427.83 ± 1.93	0.173041 **kWh**
			4	251.36 ± 0.79	0.111326 **kWh**
AWS	p3.2xlarge p3.8xlarge p3.16xlarge	V100-SXM2	1	1142.28 ± 1.27	$0.971
			4	340.08 ± 6.19	$1.156
			8	183.13 ± 1.76	$1.245
AZ	nc6s_v3 nc12s_v3 nc24s_v3	V100-SXM2	1	1211.12 ± 0.42	$1.029
			2	1226.74 ± 4.50	$2.085
			4	923.25 ± 1.59	$3.139
GCP	n1-standard-8 n1-standard-16 n1-standard-32 n1-standard-64	V100-SXM2	1	1153.53 ± 0.69	$0.917
			2	643.85 ± 0.65	$1.024
			4	366.06 ± 0.96	$1.164
			8	379.67 ± 1.77	$2.413
	a2-highgpu-1g a2-highgpu-2g a2-highgpu-4g	A100-SXM4-40GB	1	738.43 ± 0.48	$0.754
			2	398.25 ± 0.51	$0.813
			4	210.92 ± 0.07	$0.861

Table 16: Results for LAMMPS MD Simulation using an NGC container.

VIII IO

VIII.1 FIO Benchmark

The open-source FIO [10] code was used to test the IO Performance within each of the environments (On-premises as well as cloud). On each node we tested out the same random read and random write performance (size of 4 kB) after the initial creation of a 4 GB file. Each test was run 6 times on each of the nodes.

On-premises, we used the node notch176 and FIO compiled from source (version 3.27) using gcc 10.2.0. In the Cloud each node ran on the same OS and FIO version i. e. Ubuntu 18.0.4 and FIO 3.10.

AWS and Azure provide instances which are storage optimized for low latency and high random IO performance: the I3-series [2] and Lsv2-series [7], respectively.

GCP does not provide the same functionality. Therefore, we opted for the instances of the type c2-standard [3] with a 50 GB local persistent SSD Disk.

The on-premises results were obtained through the use of /scratch/local space.

VIII.2 Results & discussion

Table 17 contains the results of the benchmark described above (Subsection VIII.1).

Provider	Node	Time (s)[7]	Cost per Run
CHPC	notch176	2830.65 ± 6.29	0.084340 **kWh**
AWS	i3.2xlarge	600.76 ± 200.21	$0.104
	i3.4xlarge	682.65 ± 0.59	$0.237
	i3.8xlarge	683.25 ± 1.87	$0.474
AZ	Standard_L8s_v2	557.92 ± 1.32	$0.097
	Standard_L16s_v2	433.08 ± 1.17	$0.150
	Standard_L32s_v2	386.93 ± 13.54	$0.268
GCP	c2-standard-8	1522.50 ± 153.26	$0.181
	c2-standard-16	1401.26 ± 22.15	$0.330
	c2-standard-32	1405.85 ± 17.70	$0.616

Table 17: FIO Benchmark: Results

The run per walltime was by far the shortest on AZ, followed by AWS. The on-premises environment performed the worst (test performed on /scratch/local). The AZ and AWS results reflect the presence of high-performance IO instances. From a monetary perspective AZ offers the best result.

[7] Standard sample deviation

Conclusion/Final Remarks

All the benchmarks that were run in the Cloud used On-demand type instances. Each of the Cloud providers who services we used, offers alternatives like Spot instances, Reserved instances, dedicated instances (using AWS linguo). Among these type of instances the On-Demand option is by far the most expensive, but at the same time the least unambigous (qua instance cost).

Depending on the Cloud provider the calculation of the cost of a run can become non-trivial (the presence of a hidden costs which are hard to factor in). At the end, the monetary cost becomes clear when the final bill arrives.

In the calculation of the cost of the benchmark runs in the Cloud, the personnel cost was not included. Running simulations in the Cloud requires expertise in different domains: system administration, software installation, networking, etc.The development of this expertise requires a significant investment. Or with the great freedom of the Cloud come great responsibilities.

Acknowledgement

First and foremost, I would like to express my gratitude to the team members of the TCO project: Thomas E. Cheatham, III, David Chen (Oracle), Paul Fischer, Julia Harrison, David Richardson, Dao White, Chonghuan Xia. Outside the TCO circle I also would like to thank the following CHPC members: Martin Čuma, Brian Haymore, David Heidorn, Sam Liston, and Anita Orendt.

Among the Cloud Providers I want to thank:

- AWS : Jianjun Xu and Claudiu Farkas
- AZ: Clayton Barlow, and the Cyclecloud development team
- GCP: Jess Masciarelli, Jeff Nessen and Dan Speck (Burwood)
- Rescale: Kevin Kelly and Madhu Vellakal

I also want to thank Samuel Leventhal (CS Graduate Student - University of Utah) to provide and help us with code to discern the lymphoma types.

We are grateful for the original University of Iowa team (Steve Fleagle, Kang-Pyo Lee, Ben Rogers, and Spencer Kuhl) who developed a preliminary prototype of the CYBER-INSIGHT tool.

This material is based upon work supported by the National Science Foundation under grant no. OAC-1812786

References

[1] The 17 ways to run containers on AWS. https://www.lastweekinaws.com/blog/the-17-ways-to-run-containers-on-aws/. Accessed: 2021-08-06.

[2] Amazon EC2 I3 instances. https://aws.amazon.com/ec2/instance-types/i3/. Accessed: 2021-08-03.

[3] Compute-optimized machine family: C2 Standard VM. https://cloud.google.com/compute/docs/compute-optimized-machines#c2_vms. Accessed: 2021-08-03.

[4] Disks and images pricing (us-central1). https://cloud.google.com/compute/disks-image-pricing. Accessed: 20201-01-19.

[5] Gpus pricing (us-central1). https://cloud.google.com/compute/disks-image-pricing. Accessed: 20201-01-19.

[6] LAMMPS on NGC. https://ngc.nvidia.com/catalog/containers/hpc:lammps. Accessed: 2021-08-06.

[7] Lsv2-series. https://docs.microsoft.com/en-us/azure/virtual-machines/lsv2-series. Accessed: 2021-08-03.

[8] Slurm on the google cloud platform. https://github.com/SchedMD/slurm-gcp. Accessed: 2021-03-01.

[9] Vm instances pricing (us-central1). https://cloud.google.com/compute/vm-instance-pricing. Accessed: 20201-01-19.

[10] J. Axboe. FIO - Flexible I/O Tester. git://git.kernel.dk/fio.git. Accessed: 2021-08-01.

[11] D. Case, I. Ben-Shalom, S. Brozell, D. Cerutti, T. Cheatham III, V. Cruzeiro, T. Darden, R. Duke, D. Ghoreishi, M. Gilson, H. Gohlke, A. Goetz, D. Greene, R. Harris, N. Homeyer, Y. Huang, S. Izadi, A. Kovalenko, T. Kurtzman, T. Lee, S. LeGrand, P. Li, C. Lin, J. Liu, T. Luchko, R. Luo, D. Mermelstein, K. Merz, Y. Miao, G. Monard, C. Nguyen, H. Nguyen, I. Omelyan, A. Onufriev, F. Pan, R. Qi, D. Roe, A. Roitberg, C. Sagui, S. Schott-Verdugo, J. Shen, C. Simmerling, J. Smith, R. SalomonFerrer, J. Swails, R. Walker, J. Wang, H. Wei, R. Wolf, X. Wu, L. Xiao, D. York, and P. Kollman. Amber2018.

[12] G. Casella and R. L. Berger. *Statistal Inference*. Cengage Learning, 2nd edition, 2001.

[13] Chabane R. Top 5 ways to run your legacy containers on Google Cloud. https://dev.to/stack-labs/top-5-ways-to-run-your-legacy-containers-on-google-cloud-3k9b. Accessed: 2021-07-22.

[14] Daniël de Kok. Intel MKL on AMD Zen. https://danieldk.eu/Posts/2020-08-31-MKL-Zen.html. Accessed: 2021-06-10.

[15] Elastic Fabric Adapter. https://aws.amazon.com/hpc/efa/. Accessed: 2020-06-04.

[16] M. Fatica. Optimizing the High Performance Conjugate Gradient Benchmarks for GPUs. https://devblogs.nvidia.com/optimizing-high-performance-conjugate-gradient-benchmark-gpus/. Accessd: 2020-06-04.

[17] HashiCorp. Terraform. https://www.terraform.io/downloads.html. Accessed: 2021-03-01.

[18] M. A. Heroux and J. Dongarra. Toward a new metric for ranking high performance computing systems. 6 2013.

[19] HPCG 3.1 Binary for NVIDIA GPUs including Volta based on CUDA 10. https://www.hpcg-benchmark.org/software/index.html. Accessed: 2020-06-04.

[20] A. Ignatov. AI Benchmark: open source Python library for evaluating AI performance of various hardware platforms, including CPUs, GPUs and TPUs. https://pypi.org/project/ai-benchmark/. Accessed: 2020-12-20.

[21] Intel Parallel Studio XE. https://software.intel.com/en-us/parallel-studio-xe. Accessed: 2020-01-03.

[22] A. Kohlmeyer. Lammps: Stable release 7 August 2019. https://github.com/lammps/lammps/archive/refs/tags/stable_7Aug2019.tar.gz. Accessed: 2021-09-21.

[23] G. Kresse and J. Furthmüller. Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set. *Phys. Rev. B*, 54:11169–11186, Oct 1996.

[24] G. Kresse and D. Joubert. From ultrasoft pseudopotentials to the projector augmented-wave method. *Phys. Rev. B*, 59:1758–1775, Jan 1999.

[25] G. M. Kurtzer, V. Sochat, and M. W. Bauer. Singularity: Scientific containers for mobility of compute. 12(5):e0177459, May 2017.

[26] S. Levental. Deep dense convolutional neural net to discern between different Lymphoma types (Burkitt and Diffuse Large B-Cell) from images of patient tissue samples. https://github.com/sam-lev/Convolutional_Neural_Net_Lymphoma. Accessd: 2020-12-20.

[27] M. Melcher. Running Containers on Azure – All Options Explained. https://techcommunity.microsoft.com/t5/azure-architecture-blog/running-containers-on-azure-all-options-explained/ba-p/1795938. Accessed: 2020-10-19.

[28] D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[29] microsoft.com. Azure :: Cyclecloud. https://docs.microsoft.com/en-us/azure/cyclecloud/. Accessed: 2021-09-21.

[30] microsoft.com. Azure:: NCv3-series . https://docs.microsoft.com/en-us/azure/virtual-machines/ncv3-series. Published: 2020-02-3.

[31] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[32] A. Petitet, C. Whaley, J. Dongarra, A. Cleary, and P. Luszczek. HPL 2.3 - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, December 2018. [Online; updated December 2, 2018].

[33] S. Plimpton. Lammps Benchmarks. https://lammps.org/bench.html. Accessed: 2020-05-01.

[34] S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J. Comp. Phys.*, pages 1–19, 1995.

[35] Original Benchmarks (from Ross Walker). https://ambermd.org/Amber18_Benchmark_Suite_RCW.tar.bz2. Accessed: 2020-06-0.

[36] Ryu, Jaehun . Learning and inference speed of different gpu with various CNN models in pytorch. https://github.com/ryujaehun/pytorch-gpu-benchmark. Accessed: 2012-12-20.