DD2360HT22 HW2 GPU Architecture

Yeqi Wang

November 29, 2022

1 Reflection on GPU-accelerated Computing

1.1 Difference between GPUs and CPUs in architecture

In term of architecture, CPU is kind of latency-oriented architecture, which means that this kind of architecture minimize the running time of a single sequential program by avoiding task-level latency whenever possible. In order to meet this goal, the feature of CPU includes larger memory caches, out-of-order execution, speculative execution and pipelining. In contrast, GPU is kind of throughput-oriented architecture, which means that this kind of architecture tackle problems and workloads in which parallelism is abundant, yielding design decisions that are different from more traditional latency-oriented processors. The feature of this kind of architecture includes simple cores, small caches, lots of ALUs.

1.2 Report the name of the supercomputers in top 10 and their GPU vendor and model

There are 8 in top 10 supercomputers that is using GPU as accelerator.Names, vendor and model are listed as table 1

1.3 Calculate the power efficiency for the top 10 supercomputers

(Using Rpeak as throughput as in FLOPS)
Frontier: 7.99×10^{10}
Fugaku: 1.80×10^{10}
LUMI: 7.27×10^{10}
Summit: 1.98×10^{10}
Sierra: 1.68×10^{10}
Sunway TaihuLight: 7.95×10^{9}
Perlmutter: 3.59×10^{10}
Selene: 2.99×10^{10}
Tianhe-2A: 5.41×10^{9}
Adastra: 6.62×10^{10}

Name	Vendor	Model
Frontier	AMD	HPE Cray EX235a
Lumi	AMD	HPE Cray EX235a
Summit	NVIDIA	IBM Power System AC922
Sierra	NVIDIA	IBM Power System S922LC
Perlmutter	NVIDIA	HPE Cray EX235n
Selene	NVIDIA	Nvidia
Tianhe-2	-	TH-IVB-FEP
Adastra	AMD	HPE Cray EX235a

Table 1: TOP 10 Supercomputers using GPU.

```
 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "Tesla T4"
  CUDA Driver Version / Runtime Version          11.2 / 11.2
  CUDA Capability Major/Minor version number:    7.5
  Total amount of global memory:                 15110 MBytes (15843721216 bytes)
  (40) Multiprocessors, ( 64) CUDA Cores/MP:     2560 CUDA Cores
  GPU Max Clock rate:                            1590 MHz (1.59 GHz)
  Memory Clock rate:                             5001 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 4194304 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total shared memory per multiprocessor:        65536 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  1024
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 3 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Enabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Managed Memory:                Yes
  Device supports Compute Preemption:            Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 0 / 4
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.2, CUDA Runtime Version = 11.2, NumDevs = 1
Result = PASS
```

Figure 1: The result of deviceQuery test

2 Device Query

2.1 Result of deviceQuery test

The result of deviceQuery shows as figure 1.

2.2 Architectural Specifications

From the beginning to the end of the result, I find things interesting includes:
1. Number of CUDA cores = 2560
GPU is always consist of a great number of cores, which accounts for highly parallel execution of GPUs.
2. L2 Cache Size = 4194304 Bytes
In general, the size of L2 Cache is bigger than L1 Cache while the speed of L2 Cache is lower than L1 Cache. L2 Cache is usually shared across Simple cores. When data is required to flow from DRAM to GPU, the data flows from the DRAM to the L2. With the help of interconnection network, it then flows to L1 Cache.
3. Warp Size = 32
Warps are the basic unit of thread scheduling. Each thread resident on a single SM are executed in groups (warps) of 32.
4.Maximum number of threads per multiprocessor = 1024
Number of threads represents the performance of the GPU dealing with parallel execution.

Name	L2 Cache Size /MB	Memory Bandwidth /GB/s	TDP /Watt
GeForce RTX 4090	72	1008	450
GeForce MX570	2	96	15-45
Tesla H100	50	3072	700
Tesla T4	41	320	70

Table 2: Main changes in different architecture.

Name	Number of SM	Cores per SM	Clock Frequency(Boosted) / MHz	Peak Throughput
GeForce RTX 4090	128	128	2235(2520)	82.6
GeForce MX570	16	128	1155(1477)	6
Tesla H100	132	128	1065(1650)	60
Tesla T4	40	64	585(1590)	8.1

Table 3: Details of SM and clock frequency

2.3 Calculate the GPU memory bandwidth (in GB/s)

$$Bandwidth = Memorybuswidth * Memoryclock * 2 = \frac{256}{8 \times 10^9} * 5001 \times 10^6 * 2 = 320.064 \quad (1)$$

2.4 Compare calculated bandwidth with published bandwidth

The calculated bandwidth is really close to the published bandwidth of NVIDIA Tesla T4, which is 320 GB/s.

3 Compare GPU Architecture

3.1 List 3 main changes in architecture

Listed as Table 2.

3.2 Details of SM and clock frequency

Listed as Table 3.

3.3 Compare with GPU using now

Also included in Table 2 and 3.

4 Rodinia CUDA benchmarks and Profiling

4.1 Benchmark 1 - lavaMD

Results are listed as Figure 2 and 3.

```
kernel  lavaMD  main.c  main.h  main.o  makefile  README  result.txt  run  util
thread block size of kernel = 128
Configuration used: boxesld = 10
Time spent in different stages of GPU_CUDA KERNEL:
 0.231867998838 s, 49.309906005859 % : GPU: SET DEVICE / DRIVER INIT
 0.000335999997 s,  0.071455001831 % : GPU MEM: ALO
 0.001882000011 s,  0.400233089924 % : GPU MEM: COPY IN
 0.234638005495 s, 49.898986816406 % : GPU: KERNEL
 0.000839999993 s,  0.178637504578 % : GPU MEM: COPY OUT
 0.000661999977 s,  0.140783369541 % : GPU MEM: FRE
Total time:
0.470225989819 s
```

Figure 2: LavaMD cuda version

```
kernel  lavaMD  main.c  main.h  main.o  makefile  README  run  util
Configuration used: cores = 4, boxes1d = 10
Time spent in different stages of CPU/MCPU KERNEL:
 0.000000000000 s,  0.000000000000 % : CPU/MCPU: VARIABLES
 0.000012000000 s,  0.000169981766 % : MCPU: SET DEVICE
 0.000000000000 s,  0.000000000000 % : CPU/MCPU: INPUTS
 7.059568881989 s, 99.999824523926 % : CPU/MCPU: KERNEL
Total time:
7.059580802917 s
```

Figure 3: LavaMD omp version

4.2 Benchmark 2 - SRAD

Results are listed as Figure 4 and 5.

```
a.out               graphics.c     makefile          run
compress_kernel.cu  image_out.pgm  prepare_kernel.cu  srad
define.c            include.h      README            srad2_kernel.cu
device.c            main.cu        reduce_kernel.cu  srad_kernel.cu
extract_kernel.cu   main.o         resize.c          timer.c
Time spent in different stages of the application:
 0.000001000000 s,  0.000067688903 % : SETUP VARIABLES
 0.000010000000 s,  0.000676889089 % : READ COMMAND LINE PARAMETERS
 0.693980991840 s, 46.974815368652 % : READ IMAGE FROM FILE
 0.001720999950 s,  0.116492606699 % : RESIZE IMAGE
 0.288300007582 s, 19.514711380005 % : GPU DRIVER INIT, CPU/GPU SETUP, MEMORY ALLOCATION
 0.000214999993 s,  0.014553114772 % : COPY DATA TO CPU->GPU
 0.000028000000 s,  0.001895289286 % : EXTRACT IMAGE
 0.025613000616 s,  1.733715772629 % : COMPUTE
 0.000004000000 s,  0.000270755612 % : COMPRESS IMAGE
 0.000423999998 s,  0.028700096533 % : COPY DATA TO GPU->CPU
 0.466298997402 s, 31.563266754150 % : SAVE IMAGE INTO FILE
 0.000751000014 s,  0.050834368914 % : FREE MEMORY
Total time:
1.477347016335 s
```

Figure 4: SRAD cuda version

```
define.c    include.h  main.o    README    run   timer.c
graphics.c  main.c     makefile  resize.c  srad
Time spent in different stages of the application:
0.000000000000 s, 0.000000000000 % : SETUP VARIABLES
0.000015000000 s, 0.002941228449 % : READ COMMAND LINE PARAMETERS
0.019394999370 s, 3.803008317947 % : READ IMAGE FROM FILE
0.000650000002 s, 0.127453222871 % : RESIZE IMAGE
0.000059999998 s, 0.011764913797 % : SETUP, MEMORY ALLOCATION
0.003523000050 s, 0.690796494484 % : EXTRACT IMAGE
0.449773013592 s, 88.192337036133 % : COMPUTE
0.004571999889 s, 0.896486461163 % : COMPRESS IMAGE
0.031052999198 s, 6.088931083679 % : SAVE IMAGE INTO FILE
0.000950000016 s, 0.186277791858 % : FREE MEMORY
Total time:
0.509990990162 s
```

Figure 5: SRAD omp version

4.3 Benchmark 3 - lud

Results are listed as Figure 6 and 7.

4.4 Changes in makefile

Necessary changes in the makefile include:
1. "-arch" must be set to sm_75 in all makefile of each benchmark since it represents an unique feature of the GPU which is used. Programming via colab and the GPU used here is Tesla T4, which has the architecture of Turing, we thus set it to sm_75.
2. "CC := icc " needs to be changed to "CC := gcc " and "CXX := icc" needs to be changed to

```
/content/drive/MyDrive/rodinia_3.1/cuda/lud/cuda
lud.cu    lud_kernel.cu      lud_kernel.o  Makefile
lud_cuda  lud_kernel.cu.old  lud.o         README
WG size of kernel = 16 X 16
Generate input matrix internally, size =1024
Creating matrix internally size=1024
Time consumed(ms): 9.220000
```

Figure 6: lud cuda version

```
/content/drive/MyDrive/rodinia_3.1/openmp/lud/omp
lud.c  lud_omp    lud_omp.o        Makefile
lud.o  lud_omp.c  lud_omp_offload  Makefile.offload
Generate input matrix internally, size =1024
Creating matrix internally size=1024
running OMP on host
Time consumed(ms): 144.837000
```

Figure 7: lud omp version

"CXX := gcc" in the makefile of lud in omp vesion.

4.5 Problem of expected acceleration of GPU

According to the result of benchmarks, both lud and LavaMD enjoy a great acceleration using GPU as accelearator thanks to the ability of dealing with parallel workloads. However, SRAD in openmp version performs better than the one in CUDA version. The reason can be attributed to the essential feature of SRAD. In CUDA version, each stage is a separate kernel (due to synchronization requirements) that operates on data already residing in GPU memory. The code features efficient GPU reduction of sums. Since the result will be sent back to CPU, which can be time-consuming, the one in CUDA version performs worse in term of speed.

5 GPU Architecture Limitations and New Development

Buddy compression: Enabling larger memory for deep learning and HPC workloads on GPUs

5.1 What limitations this paper proposes to address?

The article points out the weakness of the capacity of high-bandwidth memory which is highly required in high-throughput application of GPUs. In general, memory compression used in CPUs can be inefficient and inapplicable when using in GPUs due to architecture difference and frequent compressibility changes in GPU data.

5.2 What workloads/applications does it target?

The paper mentions GPUs in high-memory-footprint applications requires high capacity of high-bandwidth memory. The high-memory-footprint applications includes HPC(High Performance Computation) and DL(Deep Learning). As for HPC workloads, a subset of SpecACCEL OpenACC v1.2 and CUDA versions of the DOE benchmarks HPGMG and LULESH is used here. As for DL Workloads, 5 convolutional neural networks name AlexNet, Inception v2, SqueezeNetv1.1, VGG16 and ResNet50 with the ImageNet running on the Caffe framework are used here.

5.3 What new architectural changes does it propose? Why it can address the targeted limitation?

Buddy Compression is introduced to improve the capacity of GPU memory. Buddy Compression divides compressed memory allocations between the GPU device memory and a larger-but-slower buddy-memory connected with a high-bandwidth interconnect.If a cacheline-sized (128B) memory-entry is sufficiently compressed, it is sourced completely from device memory; if not, it is sourced

from both device and buddy-memory.The design requires no re-allocations or page movement if the compressibility of the data changes over time.The design maintains a good compression ratio and high performance while avoiding the complexity and performance concerns of using CPU memory compression approaches on GPUs.

5.4 What applications are evaluated and how did they setup the evolution environment?

10 HPC and 6 DL network training workloads.As for evaluation environment, dependency-driven GPU performance simulator is uesd and real hardware used mentioned in the paper is NVIDIA's P100 Pascal GPU and the interconnect characteristics of recent Volta GPUs. Multi-level cache hierarchy with private L1 caches and a shared sectored L2 cache with 128B lines and 32B sectors were modeled and caches are banked to provide the necessary parallelism to saturate DRAM bandwidth. In addition, software-based cache coherence in the private caches, similar to state-of-the-art GPUs, is also modeled. Decompression latency modeled as 11 DRAM cycles. Besides the metadata cache is 4KB and 4-way set associative per MC.