2. Task in *Scientific Computing*

Deadline: April 27, 2021, 11:59pm

GPU computing using CUDA

---

1. Start with CUDA Install CUDA on your computer

- manjaro-Linux: `sudo pacman -S nvidia nvidia-utils cuda` (hints[1])
- ubuntu-Linux: `sudo apt install nvidia-cuda-toolkit cuda` (hints[2])
- Running Ubuntu in Windows via WSL2 requires appropriate drivers which is described in general by Microsoft[3] and in detail by NVIDIA[4].

and check

- Compiler: `nvcc --version`
- GPU/CUDA: `nvidia-smi`
- example Code from our git repository:

```
cd scicomp_21/examples/CUDA/firstSteps
nvcc data_mv_GH.cu
./a.out
```

2. Your first CUDA Code Copy the *CUDA* directory into your directory and download[5] the Code from [HaSh19] and use `float` as data type for all tasks.
Start with your copy of file *data_mv_GH.cu*: [5 pts]

(i) The given code realizes on GPU $\underline{b} := \underline{a}$ followed by `++`$\underline{b}$.
Extend the code with a kernel function for $\underline{c} := \underline{a} + \underline{b}$. Check the result on the host.

(ii) Compare your code with code *vector_addition_gpu_thread_block.cu*[6] from [HaSh19, §1].

(iii) Extend your code with two kernel functions for $\underline{b} := \ln(\underline{a})$ and $\underline{c} := \exp(\underline{b})$. Check the result $\underline{c}$ on the host.

(iv) Write a second main function that uses unified memory[7] (intro[8]) instead of the malloc-cudaMalloc-cudaMemcpy framework. Have a look at code *unified_memory.cu*[9] from [HaSh19, p.70f] on how to use `cudaMallocManaged`.

(v) Add timing for the kernel calls, see general performance metrics[10] on GPU.

---

[1] https://miloserdov.org/?p=4181
[2] https://linuxconfig.org/how-to-install-cuda-on-ubuntu-20-04-focal-fossa-linux
[3] https://docs.microsoft.com/en-us/windows/win32/direct3d12/gpu-cuda-in-wsl
[4] https://docs.nvidia.com/cuda/wsl-user-guide/index.html
[5] https://github.com/PacktPublishing/Learn-CUDA-Programming
[6] https://github.com/PacktPublishing/Learn-CUDA-Programming/tree/master/Chapter02/02_memory_overview/02_vector_addition/vector_addition_gpu_thread_block.cu
[7] https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#um-unified-memory-programming-hd
[8] https://devblogs.nvidia.com/unified-memory-cuda-beginners/
[9] https://github.com/PacktPublishing/Learn-CUDA-Programming/blob/master/Chapter02/02_memory_overview/06_unified_memory/unified_memory.cu
[10] https://devblogs.nvidia.com/how-implement-performance-metrics-cuda-cc/

- By using CPU-timing (`clock()` or `std::chrono::system_clock`). Don't forget the call `cudaDeviceSynchronize()` to wait until the kernel functions finish on GPU.
- By using `cudaEvent_t`, see example.

3. Reduction We need frequently a reduction function, i.e., to compare two vectors on the GPU in the previous tasks.
Have a look at codes in *CUDA/skalar* computing the inner product of two vectors.

- *skalar_3_fast.cu*: The data management is similar to the previous task. The inner product calculation follows Mark Harris' presentation in *CUDA_intro/reduction_Mark_Harris.pdf* with its own kernel functions.
  `nvcc --ptxas-options=-v -restrict skalar_3_fast.cu`
- *skalar_4.cu*: Uses cuBLAS for inner product calculation.
  `nvcc -restrict skalar_4.cu -lcublas`
- *skalar_5.cu*: Uses the Thrust[11] library for vector management in combination with STL-algorithms. See the documentation[12].
  `nvcc --ptxas-options=-v -restrict skalar_5.cu`

Your tasks for realizing $\underline{c} == \underline{a}$ on GPU: [8 pts]

(i) Write a kernel function (e.g.: `equal`) for comparing $\underline{c} == \underline{a}$. You need a reduction operation similar to the inner product in *CUDA/skalar/skalar_3_fast.cu* .

(ii) Write new kernel functions according to the improved reduction[13] by Justin Luitjens. Try it first with the inner product, check for correctness and compare the run time regarding the old version.
See also the reduction kernel in [HaSh19, §3, p.117-126] and the code versions[14] of it.

(iii) Copy and Rewrite your code from Task 2 by using Thrust. You might use only `thrust::reduce()`, see [NSLS14] for combining unified memory with Thrust. You might also rewrite all vector operations regarding Thrust.

(iv) ?? Can we call cuBLAS routines as `cublasDdot` with unified memory vectors or/and Thrust vector ??

4. Using cuBLAS 1-3 The basic idea of these tasks consist in applying cuBLAS[15] (BLAS) calls for all vector and/or matrix operations. Check also the runtime for reasonable matrix sizes.
Start with columnwise stored dense matrices (example *CUDA/densmatrices_libs*. Use `float` as data type. See also [HaSh19, §8, p.320ff] and its code[16]. [8 pts]

(i) BLAS1[17]: Realize $\underline{y} := \alpha\underline{x} + \underline{y}$; $\underline{x} := \alpha\underline{x} + \underline{y}$; $\underline{z} := \alpha\underline{x} + \beta\underline{y}$; $< \underline{x}, \underline{y} >$; $\| \underline{x} \|$;

[11] `https://developer.nvidia.com/thrust`
[12] `https://docs.nvidia.com/cuda/thrust/index.html`
[13] `https://devblogs.nvidia.com/faster-parallel-reductions-kepler/`
[14] `https://github.com/PacktPublishing/Learn-CUDA-Programming/tree/master/Chapter03/03_cuda_thread_programming`
[15] `https://developer.nvidia.com/cublas`
[16] `https://github.com/PacktPublishing/Learn-CUDA-Programming/tree/master/Chapter08/08_cuda_libs_and_other_languages/01_sgemm`
[17] `http://www.netlib.org/blas/#_level_1`

(ii) BLAS2: Realize $\underline{r} = M * \underline{x}$; $\underline{r} = M^T * \underline{x}$ with a dense non-symmetric real matrix M. Realize $\underline{r} = T \cdot \underline{x}$ with a tridiagonal matrix $T = [-1, 2, -1]$ (assuming non-constant sub-/diagonal entries).

(iii) BLAS3: Realize $A = M * M^T$ and compare the result of $\underline{z} := A * \underline{x}$ with $\underline{y} := M * (M^T * \underline{x})$.

(iv) Extract the diagonal from a dense matrix M with a BLAS1-function.

References

[HaSh19] Jaegeun Han and Bharatkumar Sharma. *Learn CUDA Programming.* Packt> (2019); buy[18]; download code[19], download figures[20]

[NSLS14] Dan Negrut, Radu Serban, Ang Li and Andrew Seidl. *Unified Memory in CUDA 6: A Brief Overview and Related Data Access/Transfer Issues.* TR-2014-09[21] (2014)

[ChAn17] Andrzej Chrzeszczyk and Jacob Anders *Matrix computationson the GPUCUBLAS, CUSOLVER and MAGMA by examples.* NVIDIA (2017); download book[22].

[18] https://www.packtpub.com/eu/application-development/cuda-cookbook
[19] https://github.com/PacktPublishing/Learn-CUDA-Programming
[20] https://static.packt-cdn.com/downloads/9781788996242_ColorImages.pdf
[21] https://www.researchgate.net/publication/326920953_Unified_Memory_in_CUDA_6_A_Brief_Overview_and_Related_Data_AccessTransfer_Issues
[22] https://developer.nvidia.com/sites/default/files/akamai/cuda/files/Misc/mygpu.pdf