PD_10_Kushagra Suryawanshi

Assignment No:6

Title: Write a CUDA program to find the maximum/minimum element in the N-element vector. (Where N is large number)

Aim: Design and implement a a CUDA program to find the maximum/minimum element in the N-element vector. (where N is large number)

Objective:
1. Write a CUDA program for finding maximum and minimum element in N element vector.
2. To understand and implement the CUDA-Device memory management functionalities
3. To understand and implement parallelism using CUDA.

Theory:
Give Introduction to CUDA and GPUs in brief.

ANS:

CUDA:

- CUDA is the acronym for Compute Unified Device Architecture.

1. A parallel computing architecture developed by NVIDIA.
2. The computing engine in GPU.
3. CUDA can be accessible to software developers through industry standard programming languages.

- CUDA gives developers access to the instruction set and memory of the parallel computation elements in GPUs.

GPU:

- GPUs are massively multithreaded many-core chips

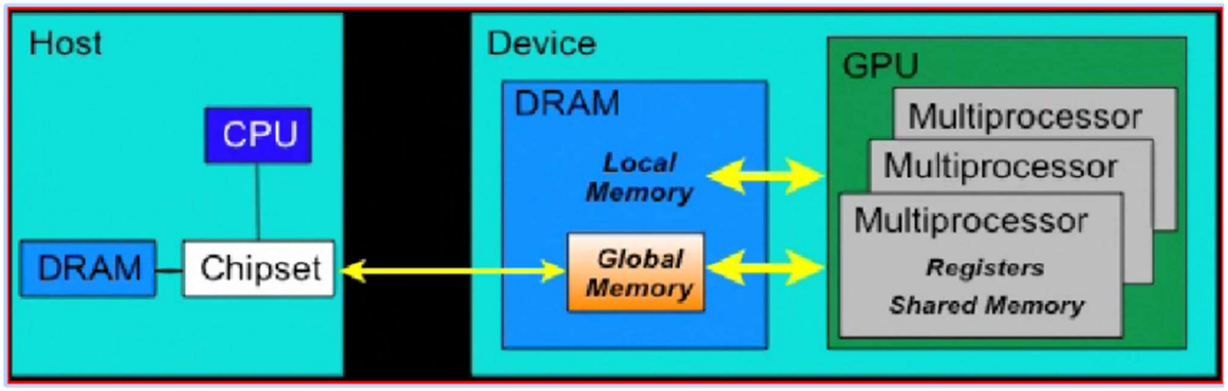
1. Hundreds of scalar processors
2. Tens of thousands of concurrent threads
3. 1 TFLOP peak performance
4. Fine-grained data-parallel computation

- Users across science & engineering disciplines are achieving tenfold and higher speedups on GPU

Write about the CUDA-Device memory management functionalities.

ANS:

Memory management refers to how to allocate memory space and transfer data between host and device. Memory management on a CUDA device is similar to how it is done in CPU programming. You need to allocate memory space on the host, transfer the data to the device using the built-in API, retrieve the data (transfer the data back to the host), and finally free the allocated memory. All of these tasks are done on the host



CUDA Device Memory Allocation

1. cuda. Malloc()
   – Allocates object in the device. Grid Block (0, 0) Global Memory
   – Requires two parameters Block (1, 0) Shared Memory:

   • Address of a pointer to the allocated object

   • Size of allocated object

2. cuda. Free()
   – Frees object from device Global Memory
   • Pointer to freed object

Input: N element vector

Output: Finding out the Maximum/minimum element in the N-element vector
Platform: Windows 11 [installation of CUDA toolkit and presence of CUDA enabled platform]

Conclusion: Thus, successfully studied, analysed and implemented CUDA program.

FAQs:
1. How do GPUs bring about massive parallelism?

GPU computing, and specifically CUDA, is designed to process massive data parallelism with huge data sets efficiently. It's exactly the embarrassingly parallel problems that exhibit massive data parallelism, and so they are the problems that show the best speed-ups by shifting the processing from the CPU to the GPU The CUDA architecture layer makes the GPU look more processor-like, and more appropriate for general parallel problems. Another thing that the CUDA layer does is to hide the underlying processing hardware. GPU as parallel co-processor- GPUs are fully programmable processors, but unlike CPUs they are not used to run stand-alone programs. Instead, a CPU application manages the GPU and uses it to offload specific computations. GPU code is encapsulated in parallel routines called kernels. CPU executes the main program, which prepares the input data for GPU processing, invokes the kernel on the GPU, and then obtains the results after the kernel terminates. A GPU kernel maintains its own application state, which, for example, may include GPU-optimized data structures not shared with the CPU. A GPU kernel looks like an ordinary sequential function, but it is executed in parallel by thousands of GPU threads. The hardware supplies each thread with a unique identifier, allowing different threads to select different data and control paths. Developers may write GPU programs in plain C++/C or Fortran with only few restrictions and minor language extensions. Note that vector sum is a purely data-parallel application with identical, completely independent subtasks. A majority of real applications are not as easily parallelizable, however, and have a more complex structure. Consider, for example, computing a dot product of two vectors. This task includes parallel reduction, which requires coordination among threads. To understand how GPUs can support this kind of workloads we delve one level deeper into the GPU hardware and software model, and then revisit the dot product example. We note the hardware model described here deliberately simplifies various technical details for clarity, while highlighting the most important concepts.

2. Give any 2 applications of CUDA and explain them

1**] Fast Video Transcoding**

Transcoding is a very common, and highly complex procedure which easily involves trillions of parallel computations, many of which are floating point operations. Applications such as Badaboom have been created which harness the raw computing power of GPUs in order to transcode video much faster than ever before. For example, if you want to transcode a DVD so it will play on your iPod, it may take several hours to fully transcode. However, with Badaboom, it is possible to transcode the movie or any video file faster than real time.

2 **Video Enhancement**

Complicated video enhancement techniques often require an enormous amount of computations. For example, there are algorithms that can upscale a movie by using information from frames surrounding the current frame. This involves too many computations for a CPU to handle in real time. ArcSoft was able to create a plugin for it’s movie player which uses CUDA in order to perform DVD upscaling in real time! This is an amazing feat, and greatly enhances any movie watching experience if you have a high definition monitor. This is fine example of how mainstream programs are harnessing the computational power of CUDA in order to delight their customers. Another fine example would be vReveal, which is able to perform a variety of enhancements to motion video, and then save the resulting video.