LESSON 4

Multiple GPUs

TOPICS

- How to combine pthreads and CUDA
- How to use portable memory
- How to joint openMP and CUDA

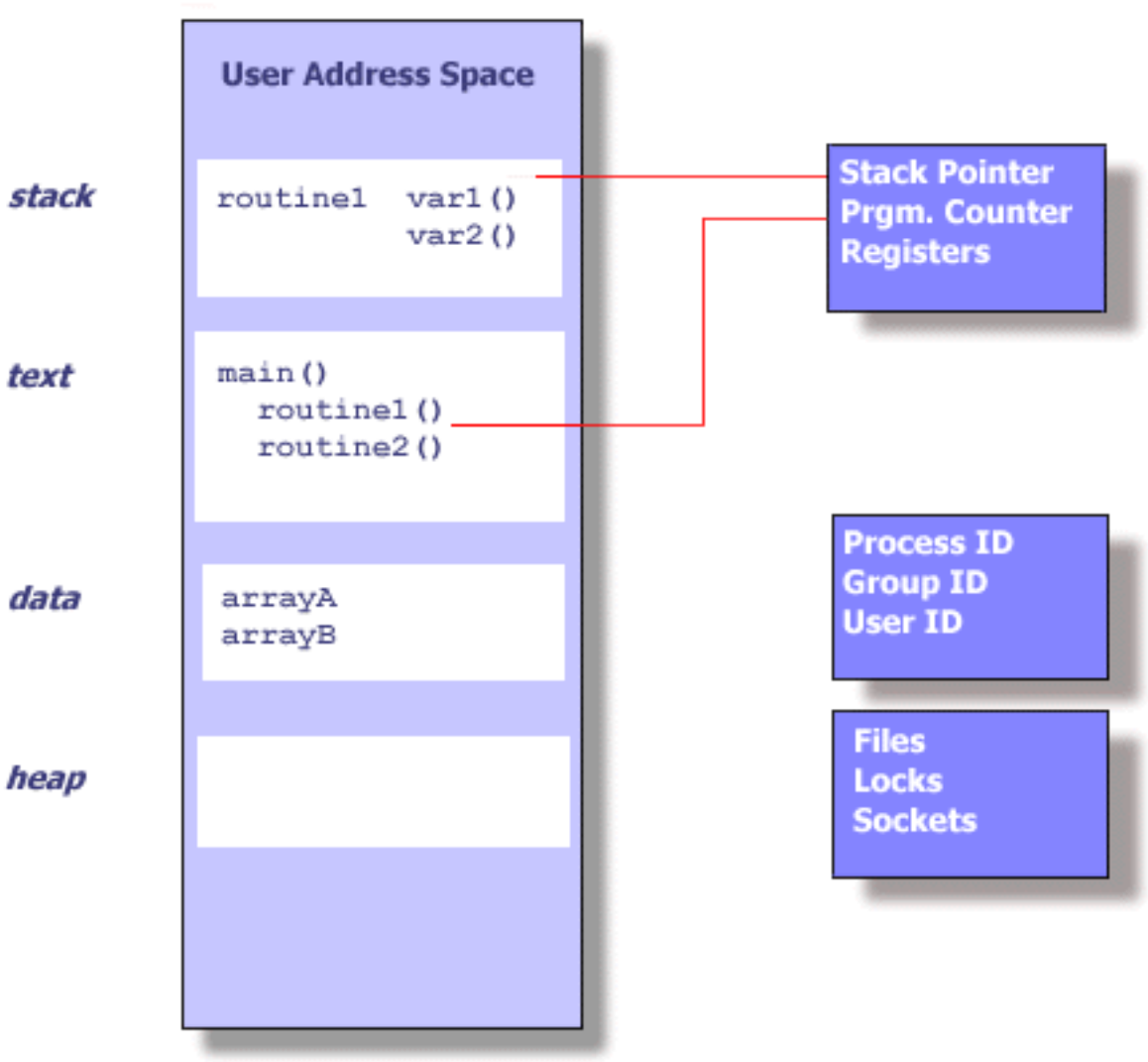
Key words: *pthread, pthread_t, pthread_create(), pthread_cancel(), pthread_join(), cudaSetDevice(), cudaSetDeviceFlags(), cudaHostAllocPortable.*

CPU Threads

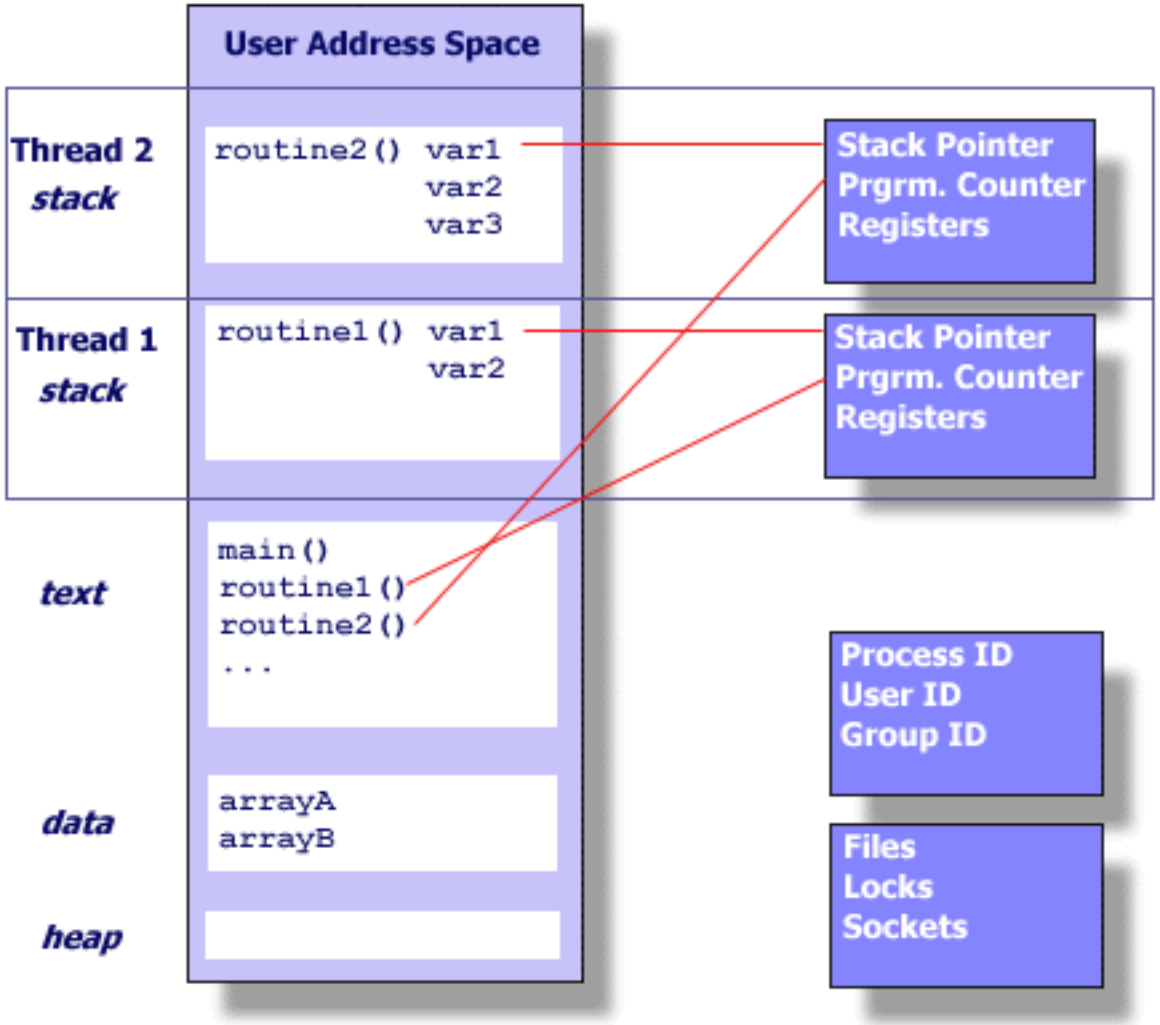
- A thread is defined as an independent stream of instructions that can be scheduled to run as such by the operating system.
- Threads share (heap) memory with other threads running in the same application.
- This is what makes threads useful: one thread can be fetching data in the background, while another thread is displaying the data as it arrives.

- The key difference between a thread and a processes is that the processes are fully isolated from each other.

- In summary a thead:
  - Exists within a process and uses the process resources
  - Has its own independent flow of control as long as its parent process exists and the OS supports it
  - Duplicates only the essential resources it needs to be independently schedulable
  - May share the process resources with other threads that act equally independently (and dependently)
  - Dies if the parent process dies - or something similar



UNIX PROCESS



THREADS WITHIN A UNIX PROCESS

Pthreads

- Pthreads are defined as a set of C language programming types and procedure calls.
- Implemented with a pthread.h header/include file.
- All threads within a process share the same address space.
- Reference: https://computing.llnl.gov/tutorials/pthreads/#Abstract

Examples

- 01hello_pthreads: This example shows the basic use of pthreads.

- 02add_pthreads: This example creates two pthreads in order to add two vectors.

Practice

- 03pi_threads: Implement the routine that calculates the PI value using four pthreads.

Pthreads + CUDA

- The trick to use multiple GPUs with CUDA is through the use of one CPU thread to control each GPU.

- It is necessary to know what is the routine that the thread must to run and to specify in which device it is going to be executed. Using the cuda function cudaSetDevice(), it is possible to declare the device to use in the operation.

Example

- 04multidevice: This example creates two threads in order to process the dot operation in two different GPUs.

- This example executes the same operation in the different devices, but with chunks of the same data; on other cases, each device could execute different routines using chunks or different data each one.

Portable Pinned Memory

- Pinned memory is host memory that has pages locked in physical memory to prevent it from being paged out or relocated.
- However, that pages can appear pinned just to a single CPU thread. The other threads will see the buffer as standard pageable data.
- To solve this inconvinient , it is possible to allocate pinned memory as portable, meaning that we well be allowed to migrate it between host threads and allow any thread to view it as a pinned buffer.
- To use portable memory, it is necessary to add the flag cudaHostAllocPortabe to the cudaHostAlloc() function.

- This means that you can allocate your host buffers as any combination of portable, zero-copy and write-combined.

- One of the requirements of allocating page-locked memory with cudaHostAlloc(), is that we have initialized the device first by calling cudaSetDevice().

- Then, we need to call cudaSetDevice() in the routine that we are going to execute, in order to ensure that each participating thread controls a different GPU.

- We only need to call cudaSetDevice() and cudaSetDeviceFlags() on devices where we have not made this calls.

- A common error is that once you have set the device on a particular thread, you cannot call cudaSetDevice() again, even if you pass the same device identifier.

Example

- 05multidevice_portable: Use the techniques already explained to improve the performance. It is up to you the way to implement it.

OpenMP + CUDA

- The OpenMP Application Program Interface (API) supports multi-platform shared-memory parallel programming in C/C++ and Fortran on all architectures, including Unix platforms and Windows NT platforms.

- OpenMP is a portable, scalable model that gives shared-memory parallel programmers a simple and flexible interface for developing parallel applications for platforms ranging from the desktop to the supercomputer.

- Reference: https://computing.llnl.gov/tutorials/openMP/#Introduction

Example

- 03multidevice_openMP: This example executes multiple GPUs using threads created with openMP.