TDT4200 exam (example) autumn 2023

1 True/False (10%)

1. False: they synchronize all active threads within the program, not all the threads on the GPU

   Actual: false - CUDA __syncthreads synchronizes all threads within the same block

2. True: MPI_Allreduce will use more bandwidth as it reduces all, while MPI_Reduce only reduces an amount within the same interface?

   actual: False - MPI_Allreduce communicates to all processes, while MPI_Reduce only needs to communicate with the root. Even though they communicate to different numbers of processes, the message is still the same, thus the bandwidth are the same (???)

3. False - Context switching often have a negative impact on occupancy, rather than increasing it. As the context switching interrupts the "smooth" execution of threads.

   actual: ^

4. True

   actual: SMT allows for multiple threads to execute simultaneously on the same single processor core. It enables better utilization of the CPU's resources by allowing independent threads to execute concurrently

5. True - MPI is just a message passing interface, so it can be replaced with multiple point-to-point operations

   actual: ^ however it is not recommended to do so as collective operations are created specifically for this work.

6. True - When the CPU is faster, it can send instructions as to what to do, much faster. This increases the operational intensity.

   actual: ^

PROF

7. True - Worksharing directives with openMP first splits and then joins, this join acts as a barrier as it waits for all threads to join before continuing

   actual: ^

8. False

   actual: False - they can deadlock if not carefully synchronized. all operations can end up waiting for a recv that no one will send.

9. False - Strong scaling does not necessarily mean that an algorithm is suitable for use on highter processor counts, but rather means that a program is written to scale well with higher numbers

of threads or processes running at the same time. What i mean is that it scales well with regards to deadlocks, splits and joins, etc.

actual: True - An algorithm that works faster and more efficient with more processors or more threads is enforing strong scaling.

10. True

actual: True - it is common practice to use a mutex in conjunction with a codition variable to implement a mechanism where threads can wait for a certain condition to be satisfied before proceeding. The mutex protects the shared data associated witht he condition and the condition variable allows threads to efficiently wait for changes to that condition.

2 Message passing and MPI (15%)

2.1

They both send information between processes, however "ready" is used to send information between processes, where the sender can continue with what it needs to do whenever information is sent. This means that it continues before the receiving process has registered the incoming recv message. This may cause issues such as unpredictable results. Synchronized solves this problem by making the sender wait for the receiver to actually receive the information, making them "work together"

2.2

```
// Get process id
pid = get_process_id()
var barrier_counter = 0

// Check if process has reached barrier
fun reached_barrier(){

    // increment barrier counter when a process reached this barrier
    barrier_counter++

    send_message_to_other_processes("barrier reached")

    if(message == "barrier reached") barrier_counter++

    // Reset barrier counter when all messages have reached the barrier
    if(barrier_counter == number_of_processes) {
        barrier_counter = 0
    }


}

reached_barrier()
```

3 GPGPU programming

3.1

POSIX threads are based on cpus, while CUDA threads are based on threads on the GPU giving many more threads than cpu threads. POSIX threads are run often run out-of-order depending on the CPU schduler based on the system considerations, however CUDA threads are run in order with a single instruction on multiple threads, as they are so many that they do not need to run out of order

3.2