CSCI-UA.0480-003

Parallel Computing

Answers

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com

Questions

- **Suppose that you have 8 threads that are computation intensive and another 8 memory bound… how will you assign them to cores on T4?**
  - Assign two threads of different types to the same core
- **What if all threads are computation bound?**
  - With 8 cores available, you need to make a decision. If each thread needs a lot of resources, then assign 8 threads to 8 cores. Once a core is done with its thread, another thread can be assigned. If, on the other hand, each thread requires modest resources, then you can assign two of them to each core.
- **What if they are all memory bound?**
  - Then assign one thread per core. When a core is done, assign another thread to it. Ensure, as much as you can, that threads accessing the memory at the same time are accessing neighboring locations.
- **T4 gives the software the ability to *pause* a thread for few cycles. When will you use this feature?**
  - When there is contention on a resource. Contention involves more than one thread, and causes those contending threads to slow down. So better do them one at a time, if needed.

Important!!

What is the relationship between cache coherence and nondeterminism? Isn't cache coherence enough to ensure determinism?

Cache coherence is not enough, for the following reasons:

- Critical section can span several cache blocks
- Two threads can execute on the same core, hence using the same cache.

Busy-waiting

```
my_val = Compute_val ( my_rank ) ;
if ( my_rank == 1)
    while ( ! ok_for_1 ) ;  /* Busy-wait loop */
x += my_val ;  /* Critical section */
if ( my_rank == 0)
    ok_for_1 = true ;  /* Let thread 1 update x */
```

What is wrong with that?

The thread with rank 1 is doing useless work waiting for the other thread to set ok_for_1 = true.
This is wasting resources.

Does your knowledge of the underlying hardware change your task dependency graph? If yes, how?

Yes, it does. If you know that two tasks communicate a lot, then better put them on neighboring cores, if we know the interconnection.
Also, if we know how many cores, and we know whether it has hyperthreading technology, we can do better scheduling of our tasks.

Example

A program runs in 10 seconds on computer A, which has a 4 GHz. clock. We are trying to help a computer designer build a new machine B, that will run this program in 6 seconds. The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate, but has informed us that this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program. What clock rate should we tell the designer to target?“

ETa = 10 = IC*CPI*CT = (IC*CPI)/4GHz
ETb = 6 = 1.2(IC*CPI)/x

Let’s divide both

10/6 = x/(4*1.2) → x = 48/6 = 8 GHz

CPI Example

- Suppose we have two implementations of the same instruction set architecture (ISA).

  For some program,
  Machine A has a clock cycle time of 250 ps and a CPI of 2.0
  Machine B has a clock cycle time of 500 ps and a CPI of 1.2
  What machine is faster for this program, and by how much?

[10^{-3} = milli, 10^{-6} = micro, 10^{-9} = nano, 10^{-12} = pico, 10^{-15} = femto]

ETa = IC* CPI * CT = IC * 2.0 * 250 = 500 * IC

ETb = IC * 1.2 * 500 = 600 * IC

IC is the same, as we are testing both machines with the same program.

Despite that machine A has higher CPI, but it actually has lower execution time, hence, higher performance.

#Instructions Example

- **A compiler designer is trying to decide between two code sequences for a particular machine. Based on the hardware implementation, there are three different classes of instructions: Class A, Class B, and Class C, and they require one, two, and three cycles (respectively).**

  The first code sequence has 5 instructions:
  2 of A, 1 of B, and 2 of C

  The second sequence has 6 instructions:
  4 of A, 1 of B, and 1 of C.

  Which sequence will be faster? How much?
  What is the CPI for each sequence?

ET = IC * CPI * CT = total_num_cycles * CT

Sequence 1: total_num_cycles = (2*1)+(1*2)+(2*3) = 10
ET for sequence 1 = 10CT

Sequence 2: total_num_cycles = (4*1)+(1*2)+(1*3) = 9
ET for sequence 2 = 9CT
Sequence 2 gives better performance. CT is fixed as we are testing on the same machine.

CPI = total number of cycles / # instructions
CPI for sequence 1 = 10/5 = 2
CPI for sequence 2= 9/6 = 1.5

MIPS Example

- Two different compilers are being tested for a 4 GHz. machine with three different classes of instructions: Class A, Class B, and Class C, which require one, two, and three cycles (respectively). Both compilers are used to produce code for a large piece of software.

  The first compiler's code uses 5 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

  The second compiler's code uses 10 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

- Which sequence will be faster according to MIPS?
- Which sequence will be faster according to execution time?

MIPS is calculate as =
total # instructions in millions / total time in seconds

Total time in seconds is the usual equation of
ET = IC * CPI * CT = (total # cycles) * CT

CT = 1 / frequency

Plug the numbers in the above equations!

Questions

Suppose we have p processes, and we need to compute a vector sum. If we ignore the I/O time, can we get more than p speedup over sequential version?

Yes we can. Because with more processors we get more cache memories. This leads to less cache misses than the sequential version.

Questions

Assume we have p processes and we need to implement a binary tree search. Can we get more than p speedup, also ignoring I/O delay?

Tree search in parallel is faster than depth first where the required item is in the right part of the tree near root.

Questions

What is the best way to implement matrix multiplication using MPI?

It depends on how big the matrix is, whether it is a square matrix, and how many processes do we have.

- Each process is responsible for subset of elements.
- However, as a best practice and to reduce communication, the elements assigned to each process are in a block (e.g. 4x4) no full row or column.

Questions

How about Traveling Salesman Problem in MPI?

One way of doing it is to do it in 3 steps:

1. Calculate all the permutations (can be done sequentially by the master process or in parallel although not easy)
2. Assign each group of permutations to a process to get the value of the path (in parallel by all processes)
3. Pick the shortest (by master process)

Question

We have seen that in reduction, OpenMP uses local variables and initializes them to identity value (e.g. 1 for multiplication and 0 for addition). What is the identity value for:

- && TRUE
- || FALSE
- & 1
- | 0
- ^ 0

Question

Do we have to worry about the following:

```
#pragma omp parallel for num_threads(2)
  for( i =0 ; i < n; i++) {
      x[i] = a + i*h;
      y[i] = exp(x[i]);
  }
```

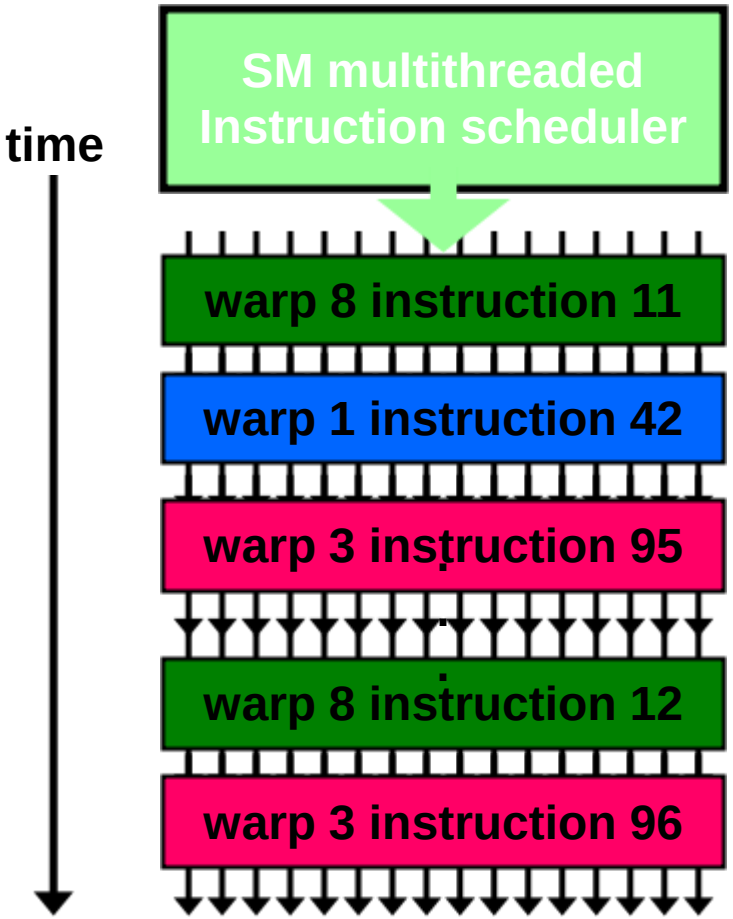
No we don't. There is no inter-loop dependence.
If loop k finishes before look k+1, the result will still be correct.

Question

Can we parallelize the following loop? If yes, do it. If not, why not?

```
a[0] = 0;
for( i = 1; i < n; i++)
    a[i] = a[i-1] + i;
```

Yes, we can, if we realize that:
a[0] = 0 a[1] = a[0]+i = i
a[2] = a[1] + i = 2i
a[3] = 3i
and so on



Exercise: Suppose 4 clock cycles are needed to dispatch the same instruction for all threads in a Warp in G80. If there is one global memory access every 4 instructions, how many warps are needed to fully tolerate 200-cycle memory latency?

Simply ceiling(200/(4*4))

Exercise

The GT200 has the following specs (maximum numbers):

- 512 threads/block
- 1024 threads/SM
- 8 blocks/SM
- 32 threads/warp

What is the best configuration for thread blocks to implement matrix multiplications 8x8, 16x16, or 32x32?

Solution

8x8 blocks → 64 threads/block → we need 1024/64 = 16 blocks/SM to keep SM busy → but we are limited to 8 blks/SM
→we end up with 8x64 = 512 threads/SM → underuntilization

16x16 blocks → 256 threads/block → 1024/256 = 4 blocks/SM → within the limit

32x32 blocks → 1024 threads/ block → beyond the limit

Quick Exercises

If a CUDA device's SM can take up to 1,536 threads and up to 4 blocks, which of the following block configs would result in the most number of threads in the SM?

- 128 threads/blk
- 256 threads/blk
- 512 threads/blk
- 1,024 threads/blk

Solution:

- 4 blks * 128 → 512 threads per SM
- 4 blks * 256 → 1024 threads per SM
- Can't have 4 blocks with 512 threads each but 3 blks * 512 threads → 1536 threads ← **The best one**
- We can only have 1 block of 1024 threads

Quick Exercises

- For a vector addition, assume that the vector length is 2,000, each thread calculates one output element, and the thread block size 512 threads. How many threads will be in the grid?
  - Solution: ceil (2000/512) = 4 → 4*512 = 2048

- Given the above, how many warps do you expect to have divergence due to the boundary check on the vector length?
  - Solution: Only one warp because for the extra 48 threads, 32 will be totally out of bound. The remaining 16 will be part of a warp where half the threads will be in-bound.

Quick Exercises

A CUDA programmer says that if they launch a kernel with only 32 threads in each block, they can leave out the __syncthreads() instruction wherever barrier synchronization is needed. Do you think this is a good idea? Explain.

No it is not a good idea because there is no guarantee that the warp size will continue to be 32. Making assumptions about the warp size in the code may cause the code to break if the warp size changes in future architectures.

Some Refreshing Exercises!

- Suppose registers and shared memory capacities were not an issue. When is it still beneficial to put values fetched from memory into the shared memory?

Solution:

To share common input values between threads in a block by only loading them once from global memory, and then read-sharing the values out of the shared memory. If each thread directly loaded the value from global memory into a register in this case, significantly more global memory bandwidth is consumed for the same computation.

Some Refreshing Exercises!

- Assume a kernel is launched with 1000 blocks. Each block has 512 threads.
  - If a variable is declared as local in the kernel, how many versions will be created throughout the lifetime of the kernel?
  - How about if the variable is created as shared?

Solution:

- For the first question the answer is 512,000.
- For the second the answer is 1,000.

Some Refreshing Exercises!

- A kernel contains 36 floating point operations and 7 32-bit word global memory accesses per thread. For each of the following device properties, indicate whether this kernel is compute- or memory-bound.
  - Peak FLOPS = 200 GFLOPS, Peak Memory Bandwidth = 100 GB/s
    - (36 ops)/(200e9 ops/s) = 0.18e-9 s
    - (7*4 Bytes)/(100GB/3) = 0.28e-9 s > 0.18e-9 s
    - Therefore the kernel is memory bound.
  - Peak FLOPS = 300 GFLOPS, Peak Memory Bandwidth = 250 GB/s
    - (36 ops)/(300e9 ops/s) = 0.12e-9 s
    - (7*4 Bytes)/(250GB/3) = 0.112e-9 s < 0.12e-9 s
    - Therefore the kernel is compute bound.