CMSC416: Introduction to Parallel Computing

Topic: performance analysis
Date: April 4th, 2024

Review:
CUDA only can use in NVIDIA

1. Performance matrics
HOW FAST YOUR PROGRAM CAN RUN?
E.g.: simple program may 4-5 hrs, complex one can take days.
- time to solution
- Time per step(iteration)
- Science progress(figure of merit per unit time) e.g.: simulate covid-19 in 180 days, separate into 5days more matrix
- Floating point predations per second(flop/s)
- when comparing multiple data points

2. Best performance
- Peak flop/s(Rpeak: advertise, something never achieved) (Rmax: realistic): 20/40%
- Peak memory bandwidth
- Peak network bandwidthine
- WHY not achieve peak performance?
    - Integer operations
    - Floating point operations
    - Conditional instructions(branches)
    - Loads/stores (e.g.: take data from memory)
    - Data movement across the network(messages + I/O)(I/O: file read etc.)

  NOTE: sequential code has the same movements

3. Performance issues
- serial code performance issues
    - Inefficient memory access
    - Inefficient floating point operations

    - Performance tools
    - Solutions:
        - minimize data movement in the memory hierarchy
        - Maximize data reuse
        - Optimize floating point calculations(e.g.: approximation of square root)
- Load imbalance

            ‣ The fast process need to wait the slower ones
    ◦ communication issues/ parallel overhead
            ‣ Communication overhead/ I/O overhead(over head and grainsize: lots of tiny messages or a fewer larger messages)
            ‣ Spending increasing proportion of time on communication(in reading amounts of communication ass we run with more processes)
            ‣ No overlap between communication and computation
            ‣ Frequent global synchronization
    ◦ algorithmic overhead/replicated work
            ‣ Speculative loss: perform extra computation speculatively buy not use all the results
            ‣ Critical path: dependencies during communication(long communication chain of operations with consecutive dependencies across processes)
                • Solutions:
                    ◦ Eliminate completely
                    ◦ shorten the critical path
            ‣ Insufficient parallelism
            ‣ Bottlenecks: same to serial bottlenecks(have load imbalance): one process ask others to wait
                • Examples:
                    ◦ Reduce to one process and then broadcast
                    ◦ One process responsible for input/output, or assign work to others
                • Solutions:
                    ◦ Parallelize as much as possible, use hierarchical schemes.

4. Performance variability is a real concern
    ◦ Individual jobs run slower
    ◦ Overall lower system throughput
    ◦ Increased energy usage/cost
    ◦ Affects software development cycle
            ‣ Debugging performance issues
            ‣ Quantifying the effect of various software changes on performance
                • Code changes
                • System software changes

5. Source of performance variability
    ◦ OS noise/jitter
            ‣ Node on an HPC cluster may have: full/light-wight kernel
            ‣ Determines what services/daemons(e.g.: checking WI-FI work correctly) run
            ‣ Measuring OS noise:

    - Fixed work quanta(FWQ) & fixed time quanta(FTQ)
- Impacts computation due to interrupts by OS