Announcements

• Assignment 3 is posted and due on Nov 3

• Final exam: Dec 15 8:00-10:00 AM
Performance metrics

- Time to solution
- Time per step (iteration)
- Science progress (figure of merit per unit time)
- Floating point operations per second (flop/s)
- When comparing multiple data points:
  - Speedup, efficiency
What is the best performance we can get?

- Peak flop/s
- Peak memory bandwidth
- Peak network bandwidth
- Why do we not achieve peak performance?
What is happening in a program

- Integer operations
- Floating point operations
- Conditional instructions (branches)
- Loads/stores
- Data movement across the network (messages + I/O)
Performance issues

- Sequential performance issues
  - Inefficient memory access: data movement in the memory hierarchy
  - Inefficient floating point operations
- Load imbalance
  - Some processes doing more work than most
- Communication issues / parallel overhead
  - Spending increasing proportion of time on communication
- Insufficient parallelism
Performance issues

- Algorithmic overhead / replicated work
  - More computation when running in parallel (e.g. prefix sum)

- Speculative loss
  - Perform extra computation speculatively but not use all of it for the result

- Critical paths
  - Dependencies between computations spread across processes / threads

- Bottlenecks
  - Serial bottlenecks: one process doing some computation and holding everyone up
Sequential performance issues

- Identify issues using performance tools

- Solutions:
  - Minimize data movement
  - Data reuse
  - Optimize floating point calculations
Communication performance

- Overhead and grain size (Lots of tiny messages or a few very large messages)
- No overlap between communication and computation
- Increasing amounts of communication as we run on more processes
- Frequent global synchronization
Critical paths

- A long chain of dependencies across processes
- We want to identify and avoid having long critical paths
- Solutions:
  - Eliminate completely if possible
  - Shorten the critical path
  - Reduce time spent in a path by removing work on the critical path
Bottlenecks

• Detect bottlenecks
  • One process busy while all others wait

• Examples:
  • Reduce to one process and then broadcast
  • One process responsible for input/output
  • One process responsible for assigning work to others

• Solutions:
  • Parallelize as much as possible, use hierarchical schemes
Performance variability is a real concern

Performance of control jobs running the same executable and input varies as they are run from day-to-day on 128 nodes of Cori in 2018-2019

Bhatele et al. The case of performance variability on dragonfly-based systems, IPDPS 2020
Leads to several problems ...

- Individual jobs run slower:
  - More time to complete science simulations
  - Increased wait time in job queues
  - Inefficient use of machine time allocation

- Overall lower throughput

- Increased energy usage/costs
Affects software development cycle

- Debugging performance issues
- Quantifying the effect of various software changes on performance
  - Code changes
  - System software changes
- Estimating time for a batch job or simulation
Sources of performance variability

- Operating system (OS) noise/jitter
- Contention for shared resources
  - Network
  - Filesystem
Operating System

- Node on an HPC cluster may have:
  - A “full” linux kernel, or
  - A light-weight kernel
- Decides what services/daemons run
- Impacts performance predictability
Operating System (OS) Noise

- Also called “jitter”
- Impacts computation due to interrupts by OS
Measuring OS Noise

- **Fixed Work Quanta (FTW) and Fixed Time Quanta (FTQ)**

Measuring OS Noise

- Fixed Work Quanta (FTW) and Fixed Time Quanta (FTQ)

The Case of the Missing Supercomputer Performance

Performance

expected 1 ms run time, we summed the unexpected overhead. The idea to aggregate across processors within a node led to an important observation: Figure 11 clearly indicates that there is a regular pattern to the noise across QB's 1,024 nodes. Every cluster of 32 nodes contains some nodes that are consistently noisier than others.

Figure 12 zooms in on the data presented in Figure 11 in order to show more detail on one of the 32-node clusters. We can see that all nodes suffer from a moderate background noise and that node 0 (the cluster manager), node 1 (the quorum node), and node 31 (the RMS cluster monitor) are slower than the others. This pattern repeats for each cluster of 32 nodes.

In order to understand the nature of this noise we plot the actual time taken to perform the 1 million 1 ms computations in histogram format. Figure 13 shows one such histogram for each of the four groupings of nodes: nodes 0, 1, 2–30, and 31 of a 32-node cluster. Note that the scale of the $x$ axis varies from graph to graph. These graphs show that the noise in each grouping has a well-defined pattern with classes of events that happen regularly with well-defined frequencies and durations. For example, on any node of a cluster we can identify two events that happen regularly every 30 seconds and whose durations are 15 and 18 ms. This means that a slice of computation that should take 1 ms occasionally takes 16 ms or 19 ms. The process that experiences this type of interruption will freeze for the corresponding amount of time. Intuitively, these events can be traced back to some regular system activity as daemons or the kernel itself. Node 0 displays four different types of activities, all occurring at regular intervals, with a duration that can be up to 200 ms. Node 1 experiences a few heavyweight interrupts—one every 60 seconds—that freeze the process for about 335 ms. On node 31 we can identify another pattern of intrusion, with frequent interrupts (every second) and a duration of 7 ms.

Using a number of techniques on QB we were able to identify the source of most activities. As a general rule, these activities happen at regular intervals. The two events that take 15 and 18 ms on each node are generated by Quadrics's resource management system, RMS [18], which regularly spawns a daemon every thirty seconds. A distributed heartbeat that performs cluster management, generated at kernel level, is the cause of many lightweight interrupts (one every 125 ms) whose duration is a few hundred microseconds. Other daemons that implement the parallel file system and TruCluster, HP's cluster management software, are the source of the noise on nodes 0 and 1.

Table 2 summarizes the duration and location within each 32-node cluster of the various types of noise. Each of these events can be characterized by a tuple $h, L, E, P$ that describes the frequency of the event $F$, the average duration $L$, the distribution $E$, and the placement (the set of nodes where the event is generated) $P$. As will be discussed in Section 3.4, this characterization is accurate enough to closely model the noise in the system and is also able to provide
The Case of the Missing Supercomputer Performance

Figures 11 and 12 show the distribution of slowdown across nodes. Figure 11 clearly indicates that there is a regular pattern of slowdown that repeats for each cluster of 32 nodes. This pattern is not observed in the RMS cluster monitor, which is slower than the others.

Analyzing noise on a per-node basis instead of a per-processor basis reveals a regular structure across nodes. The idea to aggregate across processors as will be discussed in Section 3.4, this characterization is accurate enough to closely model the noise on nodes 0 and 1.

The noise on nodes 0 and 1 is generated by Quadrics's resource management system and TruCluster, HP's cluster management software, are the source of the noise on nodes 0 and 1. The two events that take 15 and 18 ms on each node can be up to 200 ms. Node 1 experiences a few heavy interrupts—one every 60 seconds—that freeze it. Node 0 displays four different types of activities, some regular system activity as daemons or the kernel itself. Node 0 shows one such histogram for each of the four groups of 32 nodes.

The RMS [18], which regularly spawns a daemon every second, and a duration of 7 ms, is noisier than others. The RMS cluster manager), node 1 (the quorum node), and node 31 (the parallel file system and TruCluster, HP's cluster management softwares, are generated at kernel level, every thirty seconds. A distributed heartbeat that periodically generates cluster management, generated at kernel level, every 125 ms, whose duration is a few hundred microseconds every 30 seconds and whose durations are of a cluster we can identify two events that happen regularly with well-defined frequencies and durations. For example, on any node of 32 nodes contains some nodes that are consistently noisier than others.

The events that cause the different types of noise can be identified using a number of techniques. Using techniques, we were able to identify the events that should take 1 ms occasionally takes 16 ms or 19 ms. The process that experiences this type of interference can be up to 200 ms. Node 1 experiences a few heavy interrupts—one every 60 seconds—that freeze it. Node 0 displays four different types of activities, some regular system activity as daemons or the kernel itself. Node 0 shows one such histogram for each of the four groups of 32 nodes.

Figure 12 zooms in on the data presented in Figure 11 in order to show more detail on one of the 32-node clusters. We can see that all nodes suffer from a moderate background noise and that node 0 (the cluster manager), node 1 (the quorum node), and node 31 (the parallel file system and TruCluster, HP's cluster management softwares, are generated at kernel level, every 125 ms, whose duration is a few hundred microseconds every 30 seconds and whose durations are of a cluster we can identify two events that happen regularly with well-defined frequencies and durations. For example, on any node of 32 nodes contains some nodes that are consistently noisier than others.

The two events that take 15 and 18 ms on each node can be up to 200 ms. Node 1 experiences a few heavy interrupts—one every 60 seconds—that freeze it. Node 0 displays four different types of activities, some regular system activity as daemons or the kernel itself. Node 0 shows one such histogram for each of the four groups of 32 nodes.
Impact on communication

![LogGP diagram of two barrier operations with process 4 delayed (an image)](https://htor.inf.ethz.ch/publications/img/hoefler-noise-sim.pdf)
Mitigating OS noise

- Running a light-weight OS
- Turn off unnecessary daemons
- Reduce the frequency of daemons
- Dedicated cores for OS daemons
- User programs can avoid using certain cores