Course Review

Mark Greenstreet

CpSc 418 – November 30, 2018

- The Final Exam
- The Big Picture
- Parallel Algorithms
- Parallel Performance
- Parallel Architectures
- Parallel Paradigms

The Final Exam

- *"I looked at several prior terms' midterms, and this term's exam was **nothing** like those."*
  - The **concepts** are the same each term, and the **topics** are pretty much the same.
    - ★ We can't cover every topic on the exam.
    - ★ The depth and pace varies a little from term to term.
  - But, this term's midterm isn't just plugging different numbers into the questions from last term's midterm.
- Likewise for the final: **focus on the topics sand concepts**.

The big picture

- Everything that “matters” in computing is parallel.
  - There is no other way to get higher performance, and push into new application areas.
- Parallelism is fungible – it’s the currency of computing. You can use parallelism to:
  - Make your program faster.
  - Hide latency.
  - Reduce energy consumption.
- Computing is limited by power and/or energy
  - True for mobile devices, desktop computers, data centers, and everything else.
- Communication is the main concern in parallel software design.
  - How do processes communicate and synchronize?
  - Parallel computations are very often communication bound.
  - λ

Parallel Algorithms

- Reduce & Scan:
  - divide-and-conquer for communication
  - Makes associative computations parallel
- Map – often “embarassingly parallel”
  - Monte Carlo simulation (e.g. “The Life of Brian”, HW2)
  - The recurrence problems from HW5.
- Sorting:
  - sorting networks
  - the 0-1 principle
  - bitonic sort
- Matrix multiplication & Convolution
  - beautifully data parallel
  - extensive applications in machine learning, signal processing (e.g. multimedia), and graphics.

Parallel Performance: Performance Loss

- communication and synchronization overhead
- idle processors: especially and beginning and end of computation
- extra computation
- memory overhead
- inherently sequential code

Dependencies matter.

Parallel Performance: Speed-Up

- Definition:

$$SpeedUp = \frac{T_{seq}}{T_{par}},$$

$$\text{equivalently}: \quad SpeedUp\% = \left(\frac{T_{seq}}{T_{par}} - 1\right) 100\%$$

- Parallel Efficiency $SpeedUp/P$
  - Assumes sequential and parallel processors are somehow "equivalent".
  - Be very cautious if the processors operate at different clock rates, have different architectures (e.g. x86 core vs. GPU SP), etc.
- Super-linear speed-up: when $SpeedUp > P$.
  - Usually because the parallel machine has more (fast) memory.
- Embarassingly parallel problems
  - Almost no dependencies.
  - Easy to get $SpeedUp \approx 1$.
  - The most widely used form of speed-up in the real world.
    - ★ Look for embarrasingly parallel problems,
    - ★ and don't be embarrassed to make them parallel.

Parallel Performance: Amdahl's Law

$$SpeedUp \leq \frac{P}{Ps + 1 - s}$$

- $P =$ number of processors.
- $s =$ fraction of computation that is inherently sequential
- Exercise: simplify for cases when $P \ll 1/s$ and $P \gg 1/s$.
- Amdahl's law says a bit of sequential computation can ruin your parallel program.
- More generally, **any** bottleneck becomes a major problem when P is large enough.

Gustafson's Law: many real-world problems have s that decreases with problem size.

Parallel Performance: Dependencies and Work-Span

- Dependencies matter
  - Dependencies determine which tasks need to be performed in a specific order.
  - Dependencies show the communication and synchronization requirements of a computation.
  - Common dependencies: RAW, WAR, WAW, and Control.
    - ★ WAR and WAW can be removed by using more memory (e.g. register renaming.
    - ★ Control dependencies can be mitigated by speculation – but speculation comes at an energy cost of discarding misspeculated computations.
- Convenient to represent dependencies as a graph.
  - Work: number of vertices in the graph.
  - Span: height of graph (in vertices).

Parallel Performance: Brent's Formula

- Let $T_1 = \text{Work}$, the time to perform a computation on one processor.
- Let $T_\infty = \text{Span}$, the time to perform a computation on an unbounded number of processors.
- Brent's Formula:

$$SpeedUp \geq \frac{T_1}{(T_1 - T_\infty)/P + T_\infty}$$

- Brent's Formula ignores the high cost of communication, λ.
- But it still can be useful to identify potential speed-up of a computation.
- Know how work/span scales with problem size for the algorithms listed on slide 4.

Parallel Performance: What is Energy?

- Let's say that I can bicycle from UBC to SFU (25km) at 15km/hour with the calories from two bananas.
  - A banana is a unit of **energy**.
  - If I rode to UBC-O (400km) at 15km/hour how many bananas would I need to eat?
- A banana/hour is a unit of **power**.
  - How many banana/hours of energy do I expend when bicycling at 15km/hour?
- By Rayleigh's drag formula, air-drag (force) grows as the square of velocity.
  - If I ride to SFU at 25km/hour, I will need to eat

$$2\text{bananas} * \left(\frac{25\text{km/hour}}{15\text{km}\mathit{hour}}\right)^2 = 5.\overline{5}\,\text{bananas}$$

  - What is my power consumption (in bananas/hour) riding at 25km/hour?

Parallel Performance: Energy vs. Time

- A simple, empirical model for energy and time in hardware is that the energy to perform an operation (e.g. an `add` instruction) is inversely proportional to the time allowed for the operation.
- E.g. if we run the processor at half the speed:
  - It uses half as much energy (bananas) per instruction.
  - Because it is performing half as many instructions per unit time, it uses one fourth the power (bananas/hour).
- *ET* is a constant.
- As computer scientists, we often ignore the units.
  - The accepted unit for energy is a "Joule" – one newton of force applied through a distance of 1 meter. Roughly lifting a one-litre bottle of water 10cm.
  - The accepted unit for power is a "watt" – one Joule per second.
  - Energy is how much charge is used from your cell-phone battery and thus how long the phone can run on a single charge. Energy is also what BC Hydro charges you and/or the data center. One kilowatt-hour is

$$1000\text{watts} * (1\textit{hour}) * 3600\frac{\text{seconds}}{\text{watt}} \quad = \quad 3.6 * 10^6\text{joules}.$$

  - Power is the rate at which the processor produces heat, and thus defines how fancy the cooling hardware needs to be.

Parallel Performance: Energy vs. Time Example

- Let's say I have huge arrays to sort.
  - Using a single processor, I can perform the sort in $N \log_2 N$ time.
  - Using P processors with bitonic sort, I can sort in roughly

$$\frac{1}{2} \frac{N}{P} \log_2 N \log_2 P$$

    time.
  - If P parallel processors run at the same speed as one sequential processor:
    - ★ What is the speed-up?
      Note: my "roughly" will give silly numbers for $P < 10$.
    - ★ What is the efficiency?
- If I have 64 processors, the speed-up is about 20, but it would use about 3× the energy of the sequential version.
- Or, I could run each parallel processor at $\frac{1}{4}$ the speed of the sequential machine.
  - Now, the parallel version is $\sim 5\times$ faster than the sequential one.
  - And it uses about 75% of the energy as the sequential one.
  - Parallelism gives me a solution that is faster **and** uses less energy.

Parallel Architecture: Shared Memory Machines

- Be able to describe the states of the MESI protocol and identify the state transitions for a particular sequences of loads and stores.
- Know what “sequential consistency” means.
- Know that MESI can be implemented by “snooping” for machines with a few processors and by dedicated hardware for message passing for larger machines.
- Know that real computers don’t guarantee sequential consistency
  - They do make memory consistency guarantees, but they are more complicated than what we cover in this class.
  - If you want to use shared memory: use a thread library, or use Erlang.

Parallel Architecture: Message Passing Machines

- Be able to describe ring, mesh (2D and 3D), torus (2D and 3D), tree, and hypercube topologies.
- Know what "network bisection" is and how it limits performance.
  - Find the partition of nodes into two sets, A, and B such that each has (roughly) half of the processors, and the number of edges from A to B is minimised.
  - Do we worry about the edges from B to A?
    - ★ Don't worry, be happy.
    - ★ In most networks, links between processors are bi-directional.
    - ★ Thus, if there is an edge from p_i to p_j, there is also and edge from p_j to p_i.
    - ★ Furthermore, because A and B minimize the number of edges from A to B, there must be **at least** as many edges from B to A.
  - Bisection width is a pure number (no units).
- $BisectionBandwith = BisectionWidth * LinkBandwidth$
  - Bisection bandwidth has units: bytes/second or words/second
  - We use bisection bandwidth to compute the time for sending data in a network.

Network example: “magic sort” on a 2D mesh.

- $N \times N$ mesh. $P = N^2$. M values to sort.
- Initially: each processor holds M/P data values.
- The “magic” part is we will pretend that each processor knows where each value that it holds must go at the end of the sort.
- What is the worst-case initial placement of the data?
  - Hint: pound on the cross-section
- What is a lower bound for the time to sort (in the worst case) based on bisection bandwidth.
- Cool fact: the bound can be reached to within a constant factor (i.e. 3). Schnorr and Shamir’s algorithm (not on the exam).

Parallel Architecture: SIMD Machines

- SIMD = Single-Instruction, Multiple Data
  - Each instruction that is fetched and decoded is executed for multiple data values.
  - Typically, multiple execution pipelines, each with its own register file.
  - By loading thread-ids into dedicated registers, the pipelines can calculate different array indices and work on array elements in parallel.
- Example, a GPU
  - We focused on nVidia's CUDA capable GPUs, but others are very similar.
  - A GPU has multiple SIMD processors (SMs in nVidia jargon).
  - Each SIMD processor has many execution pipelines (SPs in nVidia jargon).
  - The memory hierarchy is exposed to the programmer:
    - ★ Global memory: off-chip DRAM
    - ★ Register: on-chip, per-thread
    - ★ Shared memory: on-chip, per-block
    - ★ There are other on-chip memories, but we didn't cover them.

Parallel Paradigms

Parallel paradigms reflect the diversity of parallel architectures

- Message Passing
  - Programming abstraction for distributed machines connected by a network.
    - ★ High-performance network (e.g. Infiniband) for scientific super-computers.
    - ★ Commodity networks (e.g. ethernet) for data centers.
    - ★ Message passing is a surprisingly good model on shared-memory machines – it gives a safe abstraction so that the programmer doesn't have to worry about the details of the memory model.
  - Example: Erlang
- Data Parallel
  - Programming abstraction for SIMD architectures
    - ★ Naturally fits many array-centric computations.
    - ★ The model that threads execute in lockstep can simplify synchronization and communication.
    - ★ However, lockstep execution doesn't scale well to 100s or 1000s of threads.
    - ★ Thus, very limited communication for large-scale parallelism.
  - Example: CUDA
- Others: we covered message passing and data parallel to let you see that there's more than one parallel paradigm. There are many others, e.g. "futures" (e.g. Scala), shared memory (e.g. Cilk), and more. There won't be questions about these other paradigms on the final.

What to Expect on the Final Exam

- Each question will come from some cross-product of algorithms, performance, architectures, and paradigms.
- The ideal question asks: “Do you know what the 0-1 principle is?” or “Do you know what SIMD computation is?” Questions like this are hard to answer – and some students would just say “yes” even if they weren’t sure.
- So, I ask questions that probe these concepts.
- When you read a question ask yourself:
  - What yes-or-know “Do you understand” question is it asking.
  - Identify the key, relevant concepts from algorithms, performance, architecture, and paradigms.
    - ★ Most questions won’t hit all four.
    - ★ But I’ll try to hit combinations of two or three on several questions.
  - Give an answer that makes it clear that you understand the key concepts.
- **Good Luck!**

Thank You!