Taking Stock

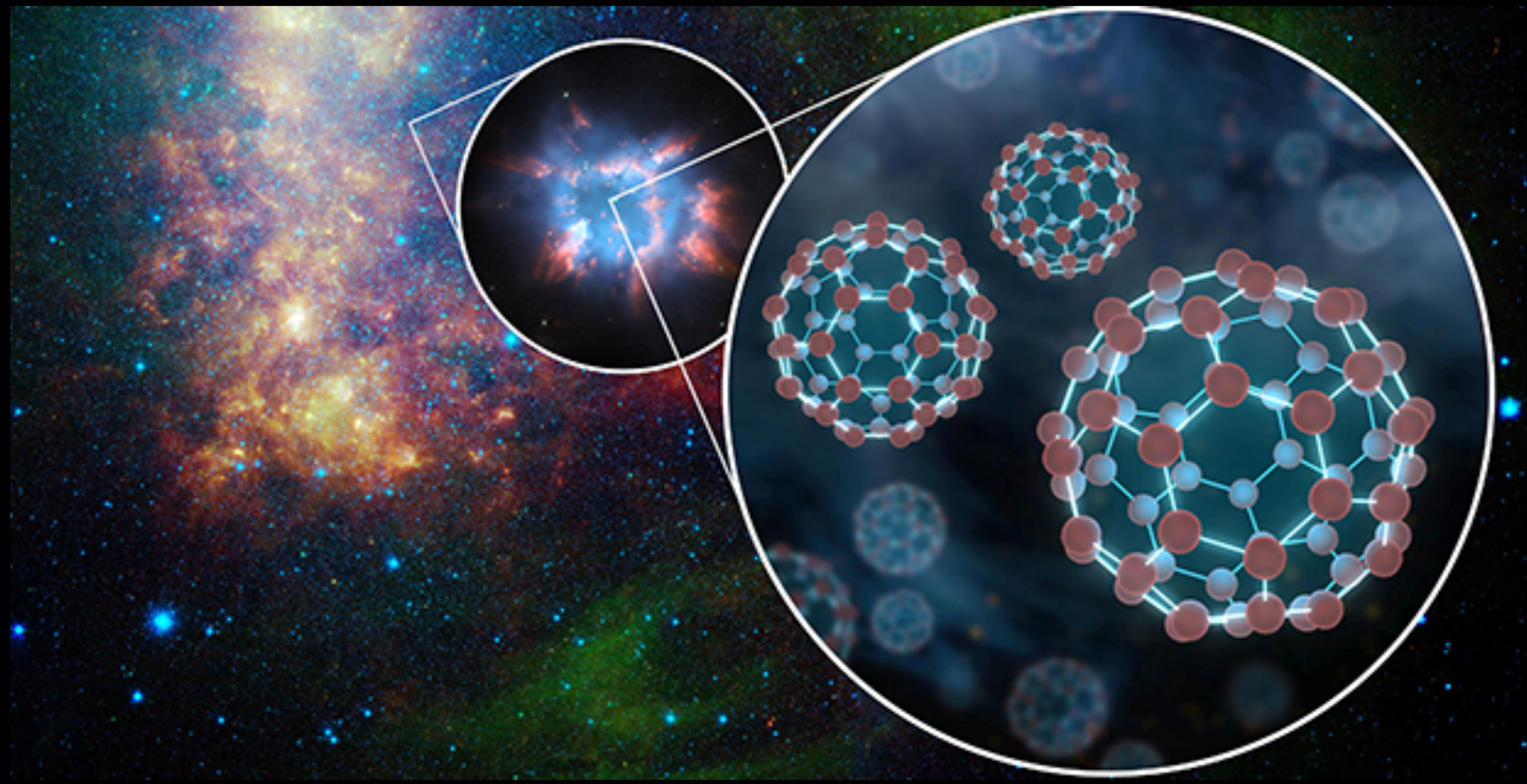
Week 13

Amittai Aviram — 8 December 2020 — Boston University

CS 591 A1 – Parallel Computing and Programming

Microscopic and Macroscopic Scales

A Metaphor for Kinds of Parallelism



- Microscopic Scale — Parallelism on a Single Node
  - Multiprocessing
  - Kernel threads
  - User threads
  - GPU threads, warps, blocks, and grids
- Macroscopic Scale — Parallelism across a Network
  - Distributed computing
  - (CPU-GPU complex)

Making Data Available

- Shared memory — exposing data on a single node
  - Kernel threads
    - C++ threads
    - OpenMP
    - Vectorized CPU instructions
  - Interprocess communication
    - Python processes
  - Distributed memory — moving data among processors or nodes
    - CUDA — between the CPU and the GPU
    - MPI — across a network

Advantages and Disadvantages

High Level

- Data need to be made available to parallel processes or execution threads BUT
- Data per process or thread need to be segregated to protect their integrity.

Shared Memory
Advantages and Disadvantages

- Advantages
  - Convenient—data are immediately available to all processes or threads executing the same program.
  - Efficient (sometimes)—data do not need to be copied.
- Disadvantages
  - Unsafe—data are exposed to race conditions.
  - Inefficient (sometimes)—false sharing may require extra data copying.

Distributed Memory
Advantages and Disadvantages

- Advantages
  - Safe — data are segregated per process
  - Deterministic — you always get the same results (because there are no data races)
  - Easy to reason about
- Disadvantages
  - Slow and inefficient
  - Inconvenient — you have to know in advance which data to move and move the data explicitly

Applications
The General Computing Context

- Moore’s Law has reached physical limits
  - Quantum effects between conducting elements
  - Excess heat
- The only way forward with conventional hardware (not quantum computing) is to parallelize.
- Modern commodity hardware has multiple cores and shared memory support.

Applications
Areas with a Natural Fit

- Simulation
  The real world consists of many concurrent processes.
  - Simulating the objects and processes themselves in parallel
  - Simulating multiple “possible worlds” by running the same (stochastic) simulation many times in parallel.
- Machine learning
  AI is designed to imitate the brain, which is *massively parallel*.
  The core math is linear algebra—operations on vectors.
  - Finding the best parameter settings to result in predictions as close as possible to training data—each parameter computation is independent.
  - Computing component partial derivatives of gradients to minimize error in multidimensional problems.
- Massive-scale online services—social networking, etc.
  Parallel processes over distributed systems support individual experiences for massive user bases.

Outlook

Immediate Future Trends

- Intel and other manufacturers are competing to support massive parallelism on the CPU
  - CUDA's days are probably limited
- Deep learning (multilayered machine learning) is expanding.
  - In depth—more layers will require piplelines of parallel processes
  - In applications—ML and DL are used in more and more areas.
  - ML and DL depend heavily on parallel computing support.
- We are running into limitations in shared memory architectures.
  - False sharing will get worse as more threads share the same data.
- The difficulty of reasoning about parallel computing makes *simpler* generally *better*—
  - We need relatively stable libraries and tools that automate parallelism, such as OpenMP.
  - CUDA-style massive parallelism and vector SIMD instructions will lead as AI applications expand.

Parting Tips and Reminders

Final Project
Tips and Reminders

- Choose the *simplest* solution that fulfills your requirements.
  - Do not templatize unless you really need more than one data type for the same function or class!
- Make your logic clear and evident.
- Choose meaningful symbol names.
- Avoid code duplication—try to parameterize your code to support re-use.
- Document, document, document! And make sure that your documentation is in *formal Standard English* (no “wanna,” “gonna,” etc.)
- Write documentation for people who do *not* know what you know.

Final Project
Coding Style

- Do not use **using namespace std** !
- Use STL containers instead of lower-level heap allocation.
- If you must allocate explicitly on the heap, use smart pointers.
- Never use **malloc** or its relatives! (Use **cudaMalloc** for the GPU if necessary.)
- Follow all the guidelines posted in the Assignments area.

Final Project
Horizontal Spacing

- Use horizontal spaces around operators but *not* immediately inside delimeters or between a collection name and “[“ a function name and “(“, or a template name and “<“:
  - A[i*num_cols+j] Wrong!
  - A [i * num_cols + j] Wrong!
  - A[i * num_cols + j] Right.
  - std::vector < int > Wrong!
  - std::vector<int> Right.
  - a= b * 3; Wrong!
  - a = b * 3; Right.
  - a+=42; YUCK! Wrong!
- Always leave a horizontal space before an opening brace:
  - ```
    int f(int input) {
        return input * MULTIPLIER + OFFSET;
    }
    ```
    ```

Keep Your Purpose in Sight

Final Project and Elsewhere

- Make sure that your output makes immediate sense to the user
  - Remember: assume that your user *does not know* all about your project's subject matter! Your user is a *non-expert!*
- Your output should include graphs of relevant projections over time—new infections, illnesses, recoveries, deaths, etc.
- Make sure that your output graphs make sense—i.e., that they have the expected shapes.

Explalin Everything
Final Project Write-Up

- Explain the choices that you made in designing your simulation.
  - What did you consider?
  - Why did you choose your particular design?
- Explain the technical decisions you made in your implementation.
  - What were your options?
  - Why did you choose a particular technical approach?
- Give details of who did what.

Thank You!