Parallelism Basics

Week 2 – Part 1

Amittai Aviram – 15 September 2020

CS 591 A1 – Parallel Computing and Programming

Speedup

Let p be the number of processing units, $T(p)$ be the amount of time taken by p processors to accomplish a constant unit of work, and $S(p)$ be the *speedup* provided by parallel processing on p processors as compared with use of a single processor. Then the speedup is given by

$$S = \frac{T(1)}{T(p)}$$

The maximal speedup is usually *linear*.

Efficiency and Cost

The *efficiency* of a parallel computing system is how much speedup you get per processing unit—given as a percent.

$$E = \frac{S}{p} = \frac{T(1)}{T(p) \times p}$$

The ideal is 100%, which is linear speedup.

The *cost* of the system is measured in *time*—the total time taken by all processors.

$$C = T(p) \times p$$

Computation and Communication

- The cost of communicating data can significantly affect the efficiency of the system.
- The higher the ratio of communication to computation time, the less efficient the system.

Scalability

The Maintenance of Efficiency as p Increases

- Strong Scalability
  The algorithm or system remains efficient as we add processors while keeping the data input size constant.

- Weak Scalability
  The algorithm or system remains efficient as both the number of processors *and* the size of the input grow.

- An algorithm or system is *not scalable* if its performance degrades significantly as the size of the system (and input) grows.

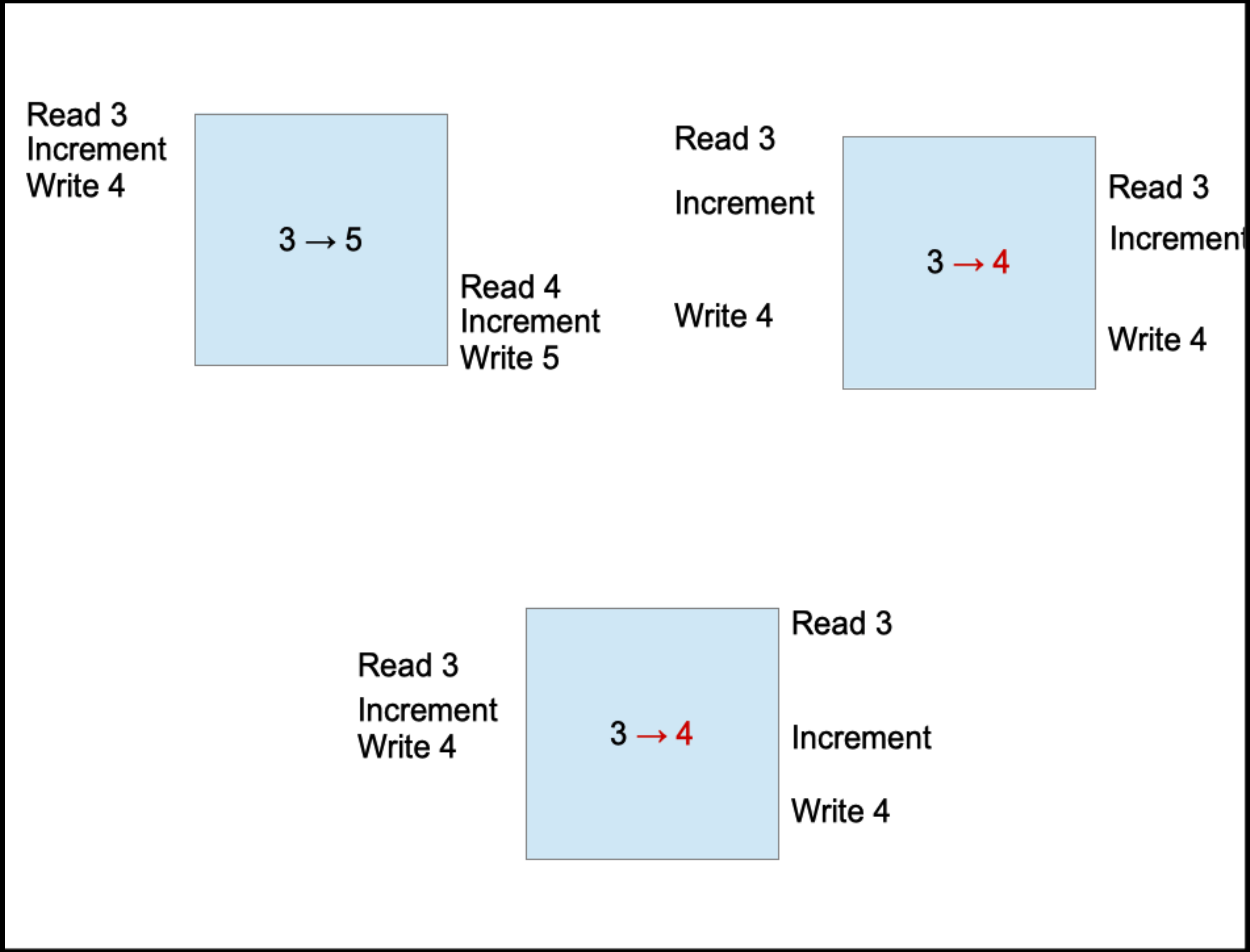
Getting Data to Processors

Two Schemes

- Distributed memory
  - An initial processor must explicitly send data to any processor that must act upon those data.
  - Each processor sees its own *copy* of the data sent to it.
  - Data are communicated under *constraints*, such as the number of interconnects (typically one) and the speed of the connection.
  - Updates for other proessors must be explicitly shared.
- Shared memory
  - All processors have equal access to the data.
  - Processors share a *single instance* of the data.
  - Data accesses may result in *race conditions*.

Race Conditions

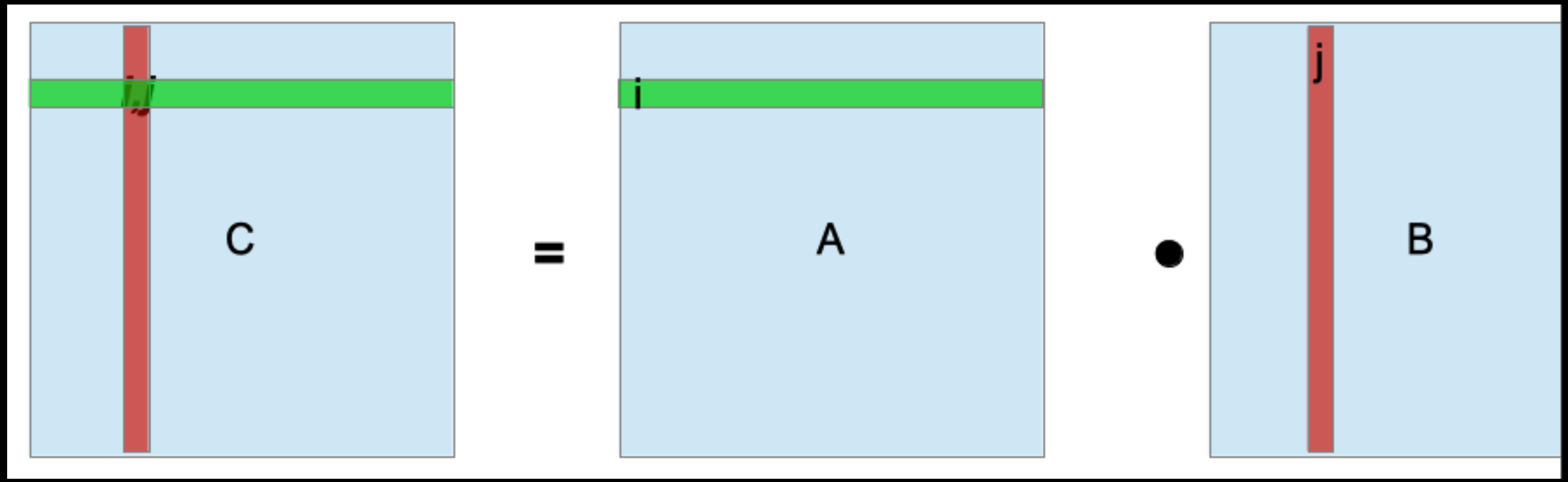
Example: Concurrent Increment



Ideal for Shared Memory

Data and Algorithm

- The data set for each process is independent of all other data sets during the lifetime of the process.
- The program partitions the data fairly (evenly) among the concurrent processes.



General Considerations

- Partitioning
- Communication
- Synchronization
- Load Balancing

Challenge: Prefix Sum

Shared Memory with Loop-Carried Dependency

```c
for (int i  = 1; i < n; ++i) {
    A[i] += A[i - 1];
}
```

```python
for i in range(1, n):
   A[i] += A[i - 1]
```

Parallel Prefix Sum