Richard Jefferson

CSCI 415

**Implementation Overview**

**Cannon's Algorithm (prog6-2.c)**

* Based on a 2D grid layout of processors.
* Operates by shifting matrix blocks horizontally and vertically.
* Communication overhead is predictable, but it requires a perfect square number of processes (e.g., 4, 9, 16, ...).

**Nelson's Hypercube Algorithm (nelson\_complete.c)**

* Intended to utilize a hypercube topology for data communication.
* Matrix blocks are distributed across hypercube vertices, leveraging parallel paths.
* Currently **unimplemented** or **untested** on the cluster environment.
* Further testing and debugging are required to verify functionality.""

**Driver Program (mm.c)**

* Includes both Cannon's and Nelson's algorithms.
* Supports the following command-line options:
  + -c: Execute Cannon's algorithm.
  + -n: Execute Nelson's algorithm.
  + -T: Measure and display execution time.
  + -d: Debug print of input and output matrices.

**Testing and Results**

The algorithms were tested on different matrix sizes and process counts:

* For smaller matrices (4x4, 8x8) and 4 processes:
  + Cannon's Algorithm was generally faster due to lower communication overhead.
  + Nelson's showed slightly slower performance because of the hypercube communication.
* For larger matrices (16x16, 32x32) and 16 processes:
  + Cannon's Algorithm demonstrated stable performance as expected.
  + Nelson's Algorithm was **untested on the cluster**, and its scaling performance is currently unknown.""
* Both algorithms scaled as expected:
  + **Cannon's Algorithm** excelled with grid-friendly process counts.
  + **Nelson's Algorithm** remains untested on the cluster, and its performance in hypercube configurations is currently unknown.

**Analysis**

* **Cannon's Algorithm** is optimal for structured grids of processors where communication is simple and consistent.
* **Nelson's Algorithm** is more effective for larger process counts, especially with hypercube compatibility.
* MPI timing analysis showed:
  + Cannon's is faster for smaller matrix sizes.
  + Nelson's scales better for larger matrix sizes and higher processor counts.

**Conclusion**

Both Cannon's and Nelson's algorithms effectively utilized MPI for distributed matrix multiplication:

* For small-scale operations: **Cannon's Algorithm** is more efficient.
* For large-scale, high processor configurations: **Nelson's Algorithm** is expected to be efficient, but this remains untested on the cluster.