Extending Distributed Functionality in Phylanx

Scientific Computing Around Louisiana

Maxwell Reeser

February 8, 2020

Division of Computer Science and Engineering
School of Electrical Engineering and Computer Science
Louisiana State University

LSU Center for Computation & Technology

STE||AR GROUP

Outline

Introduction to Phylanx

Phylanx: An Asynchronous Distributed C++ Array Processing Toolkit

- Write Python code, run it in distributed
- Targeting machine learning mainly
    - Focus on linear algebra
- Heavy use of HPX
    - Standard's compliant distributed C++ runtime
    - Product of Stellar Group
- Blaze data structures
    - Parallelism already implemented

Tiling in Phylanx

- Distributed computation means distributed data
  - Distributed data structures have local (single node) tiles
  - These can be tiles of vectors, matrices, or other data structures
  - Splitting up of data must be intentional
  - We mostly focus on tiling of matrices
- Different tilings can have different costs
  - Eventually we want to minimize the cost
  - This problem is NP-hard

Distributed Phylanx

- Map operations
    - No Data Dependencies
- Distributed Data Structures
    - distributed_object
        - UPC++
    - distributed_vector/matrix
    - Annotations
- Distributed Primitives
- Tiling testing
- Tiling Optimizer

distributed_matrix

- Object-oriented distributed data organization
- Enabled by HPX's Active Global Address Space (AGAS)
- Maintains a list of participating nodes
- Allows one local tile to "fetch" a non-local tile over the network

Algorithms

dot_d 2d2d

- Iterates through all tiles of RHS
    - Performs multiplication if intersection detected
- Result matrices may be large
- May require row-aggregation in order to compute result

Cannon's Algorithm

- Matrix Matrix multiplication algorithm
- Moves both input matrix tiles at every step
- End result does not require row-aggregation

Cannon's Algorithm

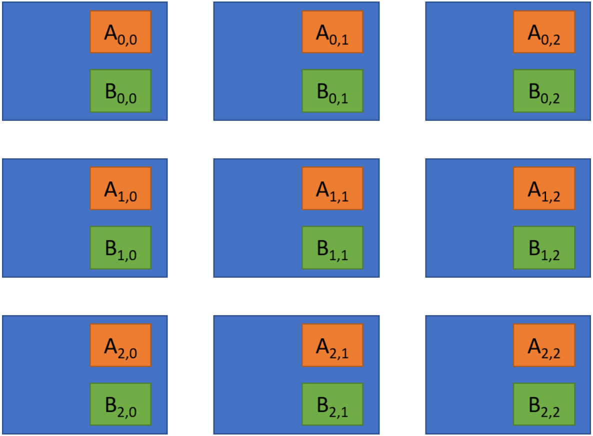

Figure 1: Alignment

Cannon's Algorithm

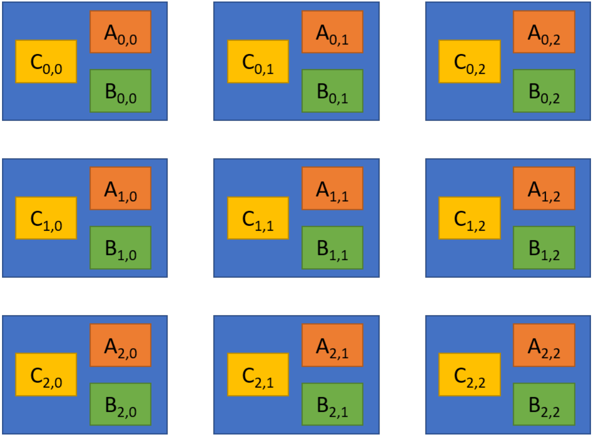

Figure 2: Multiply Local Values

Cannon's Algorithm

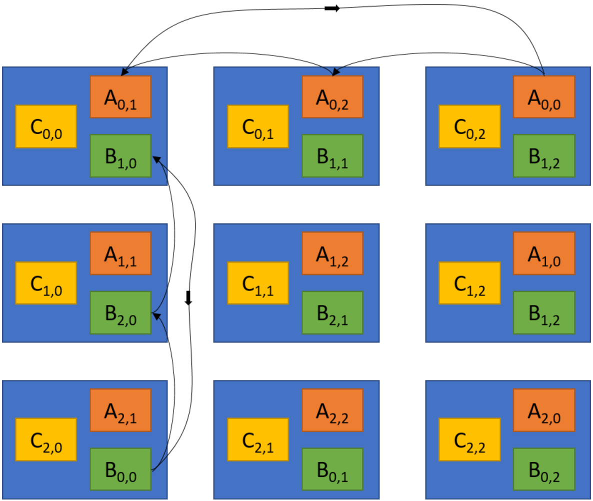

Figure 3: Shift Data & multiply

Cannon's Algorithm

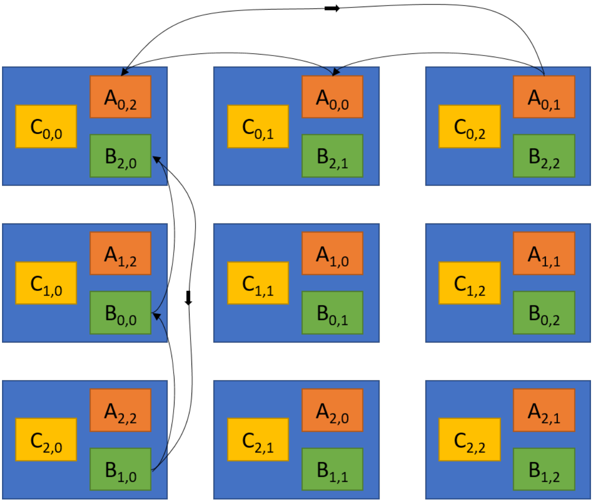

Figure 4: Shift Data & multiply

Cannon's Algorithm

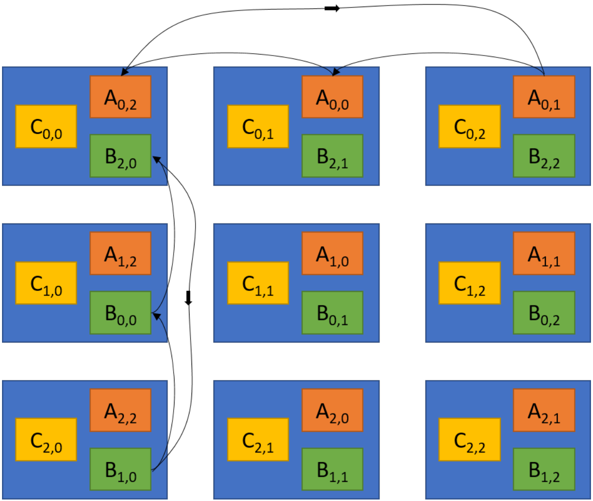

Figure 5: Shift Data & multiply

Futurizing Cannon's Algorithm

- No permanent data moving
    - Pulling instead
    - Doubles memory cost
- Allows pulling to be done one cycle ahead

Results

Preliminary Results

- Both distributed algorithms outperformed the pure serial version
- Cannon performed substantially better than dot_d

Matrix Size	dot_d	cannon	dot (serial)
500	5589.92	2519.04	8566.63
1000	36991.6	25815.2	67518.4
2000	283583	203211	538530

Questions?