CMSC416: Introduction to Parallel Computing

Topic: Parallel Algorithms
Date: March 7, 2024

Blocking is a technique used to improve cache performance by splitting a matrix into smaller blocks that individually fit in cache, leading to cache reuse.
Example: Say we have a 16x16 matrix and our cache size is 8 elements. A full row or column will not fit in our cache, so we will need to constantly take things in and out of cache. This is inefficient and causes a lot of overhead.
With blocking, we can instead do partial matrix multiplication for each block in C. For instance, we can split the matrix in 4x4 blocks. Block C12 can be calculated with block row 1 of A and block column 2 of B. Notice that adjacent cells in the same block as C reuses the same row of A (cache reuse).

Parallel Matrix Multiplication

At a high level, we borrow ideas from blocking and communicating between threads.
Say we wish to calculate matrix C, where C=AxB. Assume that A and B are too large to fit in memory and/or take too long.

First algorithm: Cannon's 2d matrix multiply
2d here refers to how we arrange our processes. Our only requirement is that the first dimension of A matches the 2nd dimension of B. Results can be generalized to non-square matrices.
Steps:

- We arrange processes in a 2d virtual grid and assign sub-blocks of A and B to each process. Each process is responsible for computing a sub-block of C.
- Take the same 4x4 example with 16 processes: Sub-locks A00 and B00 goes to process 0, A01 and B01 goes to proc 1, etc. Consider calculating sub-block C12 on process 6. Partial matrix multiplication formula: C12 = A10*B02+A11*B12+A12*B22+A13*B32
- Processor 6 has 2 of the required sub-blocks (A12 and B12) but we need other sub-blocks to complete the equation. We can use point to point message sending but if every process is communicating with every other process, there will be too much data flow.
- Solution: displace sub-blocks in row i in A by i, displace sub-blocks in row j in B by j (displacement means sending data to the other sub-blocks)
- Initial skew:
    - A02 is displaced by 0, A12 is displaced by 1 to the left, A22 is displaced by 2 to the left, A32 is displaced by 3 to the left but wrapping around to processor 15. Repeat for all columns in A.
    - B10 is displaced by 0, B11 displaced upward by 1, B12 is displaced upward by 2 wrapping around to processor 14, B13 is displaced upwards by 3 wrapping to proc 7. Repeat for all rows in B.
    - Processor 6 now has access to A13 and B32

- Shift-by-1: Shift all columns in A to the left by 1 with wrap around. Shift all rows in B upwards by 1 with wrap around.
    - A11 A12 A13 A10 -> A12 A13 A10 A11
    - B10 B21 B32 B03 -> B32 B02 B12 B22
    - Processor 6 now has access to A10 and B02
- Continue doing shift-by-1 until all processes get all the sub-blocks they need

Time complexity: O(n^3)[computation where n=length of sub-block] + O($\sqrt{p}$)[communication]
Algorithm has the same amount of computations as the serial version, i.e no overhead other than additional communication. Unfortunately, its main drawback is lots of communication.

2nd algorithm: Agarwal's 3d matrix multiply
This algorithm arranges processes in a 3d virtual grid. Unlike Cannon's algorithm, each process only computes a partial sub-block instead of a full sub-block, and data movement is only done once before and once after computation.
Steps:

- We arrange processes in a 3d virtual grid and assign sub-blocks of A and B to each process. Each process is responsible for computing a partial sub-block of C.
- Copy A to all i-k planes and B to all j-k planes. i, j, and k are axes in 3d space.
- For example, consider a 3d process grid with dimensions 3x3x2. We pick the front facing plane to assign values of A, divide A into 9 sub-blocks, and broadcast the plane to all other planes in the j direction. Next we pick a plane orthogonal to A. Here we pick the bottom plane to assign values of B and broadcast it to other planes in the i direction.
- After doing initial setup, every sub-block has something to compute (A cell has corresponding B cell to multiply with). In this example, processor 3 has A10 and B01, processor 4 has A11 and B11, and processor has A12 and B21. Each processor performs a unique partial sub-block computation.
- Notice that C11 = A10*B01+A11*B11+12*B21. Use allreduce along the k direction to add sub-blocks together to get a full sub-block of C.

We only performed 2 broadcasts in the beginning and a single reduce at the end, which is much less communication than in Cannon's algorithm. However, the more complex processor setup means it has a higher memory requirement.

Communication Algorithms

Reduction

There are 2 types of reduction:

- Scalar reduction is where every process contributes one number (ex: adding times)
- Vector reduction is where every process contributes an array of numbers.

How does MPI do parallelization for reduction?

A naïve algorithm would be to have every process send their data to the root. Problems with this approach include communication traffic, bottleneck at root (a given process can only handle messages at a certain rate), and heavy computation on root.

A better approach is to use a spanning tree that organizes processes in a k-ary tree. Each process has k children. Each intermediate node does a partial sum of its leaves and sends the result to their parent. The number of phases or height of a k-ary tree is log_k(p). k can be customized to fit one's needs, i.e if the network is slow and compute is fast, a larger k can be better.

All-to-all: Each process sends a distinct message to every other process

A naïve algorithm would be to have every process send the data pair-wise to all other processes. Number of messages is O(n^2). As n grows large, there can be too much communication happening at once.