CMSC416 Instructor Notes Spring 2024
Topics: Parallel Algorithms

1 Matrix Multiplication

1.1 Blocking (Tiling) in Matrix Multiplication

Blocking or tiling is a technique used to optimize matrix multiplication by dividing the matrices into smaller sub-matrices or blocks. This approach improves cache utilization and reduces cache misses by ensuring that data used in computations is more likely to be held in the cache.

The lecture emphasized the importance of understanding the memory hierarchy to improve matrix multiplication efficiency. It described a simplified memory model consisting of a larger, slower memory (e.g., RAM) and a smaller, faster memory (cache). The process of computing matrix multiplication involves moving data from the larger memory to the cache. However, due to the limited size of the cache, it is crucial to manage the data efficiently to avoid frequent evictions and reloads, which slow down computation.

Blocking is a technique used to optimize matrix multiplication by dividing matrices into smaller subblocks. This approach allows for more efficient use of the cache by bringing smaller portions of the data into the cache and performing computations on these subblocks. The lecture explained that by working on smaller subblocks, it is possible to keep the necessary data in the cache longer, reducing the need to reload data from the larger memory. The process involves dividing matrices A, B, and C into subblocks and then computing the multiplication of these subblocks to obtain partial results for matrix C.

1.2 Parallel Matrix Multiplication

Sequentially, matrix multiplication involves three nested loops iterating over the rows and columns of the input matrices. Parallel matrix multiplication involves dividing the computation across multiple processes or threads, each handling a portion of the data.

The principle of blocking can be applied to parallel matrix multiplication by assigning specific blocks of matrices to different threads or processes.

In a distributed memory setting, the input matrices are divided among different processes, each computing a portion of the result matrix. This requires careful distribution of data and potentially communication between processes to gather necessary parts of the input matrices for local computation.

Canon's 2D Matrix Multiply: This algorithm arranges processes in a 2D virtual grid and divides matrices A and B into subblocks distributed among the processes. It involves an initial displacement of subblocks (skew) and subsequent phases of data movement and computation to ensure each process gets the necessary subblocks for computation. The algorithm effectively reduces the communication overhead by structuring data movement and computation in a coordinated manner.

Agrawal's 3D Matrix Multiply: This more sophisticated algorithm arranges processes in a 3D virtual grid and requires multiple copies of matrices A and B across different planes. Each process computes a portion of the result matrix C, and a final reduction step combines these partial results. The 3D arrangement and initial distribution of data reduce the need for extensive data movement during computation, focusing on optimizing communication.

2 Communication Algorithms

Communication algorithms are crucial for parallel processing, enabling efficient data sharing and coordination among processors. Two main topics covered include MPPI reduction and all-to-all communication.

2.1 Reduction

Reduction involves aggregating data from multiple processors to a single value. A naive approach would have all processors send their data to a single root processor, causing a bottleneck. A more efficient method uses a spanning tree structure, distributing the aggregation workload among several processors and reducing communication overhead.

2.2 MPI All-to-All

MPI All-to-All communication is more complex, requiring each processor to send unique data to every other processor.

Naive Approach: The straightforward method would have each processor directly send data to all others, resulting in a high number of messages and potential network congestion.

Optimized Approach: By arranging processors in a 2D grid, communication can be split into two phases: row-wise and column-wise. This reduces the total number of messages sent and avoids network congestion.

Hypercube Topology: Processes can also be arranged in a hypercube topology, allowing for even more efficient communication patterns. This method is suitable for very large numbers of processors.