The Lecture Contains:

Previous Next

Data Access and Communication

- The memory hierarchy (caches and main memory) plays a significant role in determining communication cost
  - May easily dominate the inherent communication of the algorithm
- For uniprocessor , the execution time of a program is given by useful work time + data access time
  - Useful work time is normally called the busy time or busy cycles
  - Data access time can be reduced either by architectural techniques (e.g., large caches) or by cache-aware algorithm design that exploits spatial and temporal locality

Data Access

- In multiprocessors
  - Every processor wants to see the memory interface as its own local cache and the main memory
  - In reality it is much more complicated
  - If the system has a centralized memory (e.g., SMPs), there are still caches of other processors; if the memory is distributed then some part of it is local and some is remote
  - For shared memory, data movement from local or remote memory to cache is transparent while for message passing it is explicit
  - View a multiprocessor as an extended memory hierarchy where the extension includes caches of other processors, remote memory modules and the network topology

Previous Next

Artifactual Comm.

- Communication caused by artifacts of extended memory hierarchy
  - Data accesses not satisfied in the cache or local memory cause communication
  - Inherent communication is caused by data transfers determined by the program
  - Artifactual communication is caused by poor allocation of data across distributed memories, unnecessary data in a transfer, unnecessary transfers due to system-dependent transfer granularity, redundant communication of data, finite replication capacity (in cache or memory)
- Inherent communication assumes infinite capacity and perfect knowledge of what should be transferred

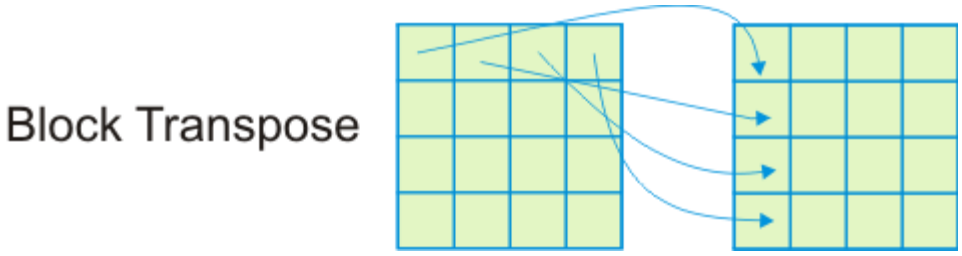
Capacity Problem

- Most probable reason for artifactual communication
  - Due to finite capacity of cache, local memory or remote memory
  - May view a multiprocessor as a three-level memory hierarchy for this purpose: local cache, local memory, remote memory
  - Communication due to cold or compulsory misses and inherent communication are independent of capacity
  - Capacity and conflict misses generate communication resulting from finite capacity
  - Generated traffic may be local or remote depending on the allocation of pages
  - General technique: exploit spatial and temporal locality to use the cache properly

Previous Next

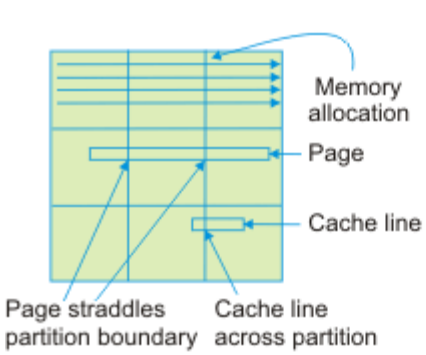
Temporal Locality

- Maximize reuse of data
  - Schedule tasks that access same data in close succession
  - Many linear algebra kernels use blocking of matrices to improve temporal (and spatial) locality
  - Example: Transpose phase in Fast Fourier Transform (FFT); to improve locality, the algorithm carries out blocked transpose i.e. transposes a block of data at a time



Spatial Locality

- Consider a square block decomposition of grid solver and a C-like row major layout i.e. A[i][j] and A[i][j+1] have contiguous memory locations



The same page is local to a processor while remote to others; same applies to straddling cache lines. Ideally, I want to have all pages within a partition local to a single processor. Standard trick is to covert the 2D array to 4D.

2D to 4D Conversion

- Essentially you need to change the way memory is allocated
  - The matrix A needs to be allocated in such a way that the elements falling within a partition are contiguous
  - The first two dimensions of the new 4D matrix are block row and column indices i.e. for the partition assigned to processor P 6 these are 1 and 2 respectively (assuming 16 processors)
  - The next two dimensions hold the data elements within that partition
  - Thus the 4D array may be declared as float B[vP][vP][N/vP][N/vP]
  - The element B[3][2][5][10] corresponds to the element in 10 th column, 5 th row of the partition of P 14
  - Now all elements within a partition have contiguous addresses

Transfer Granularity

- How much data do you transfer in one communication?
  - For message passing it is explicit in the program
  - For shared memory this is really under the control of the cache coherence protocol: there is a fixed size for which transactions are defined (normally the block size of the outermost level of cache hierarchy)
- In shared memory you have to be careful
  - Since the minimum transfer size is a cache line you may end up transferring extra data e.g., in grid solver the elements of the left and right neighbors for a square block decomposition (you need only one element, but must transfer the whole cache line): no good solution

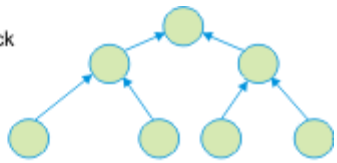
Previous Next

Worse: False Sharing

- If the algorithm is designed so poorly that
  - Two processors write to two different words within a cache line at the same time
  - The cache line keeps on moving between two processors
  - The processors are not really accessing or updating the same element, but whatever they are updating happen to fall within a cache line: not a true sharing, but false sharing
  - For shared memory programs false sharing can easily degrade performance by a lot
  - Easy to avoid: just pad up to the end of the cache line before starting the allocation of the data for the next processor (wastes memory, but improves performance)

Contention

- It is very easy to ignore contention effects when designing algorithms
  - Can severely degrade performance by creating hot-spots
- Location hot-spot:
  - Consider accumulating a global variable; the accumulation takes place on a single node i.e. all nodes access the variable allocated on that particular node whenever it tries to increment it



Scalable tree accumulation

Hot-spots

- Avoid location hot-spot by either staggering accesses to the same location or by designing the algorithm to exploit a tree structured communication
- Module hot-spot
  - Normally happens when a particular node saturates handling too many messages (need not be to same memory location) within a short amount of time
  - Normal solution again is to design the algorithm in such a way that these messages are staggered over time
- Rule of thumb: design communication pattern such that it is not bursty ; want to distribute it uniformly over time

Overlap

- Increase overlap between communication and computation
  - Not much to do at algorithm level unless the programming model and/or OS provide some primitives to carry out prefetching , block data transfer, non-blocking receive etc.
  - Normally, these techniques increase bandwidth demand because you end up communicating the same amount of data, but in a shorter amount of time (execution time hopefully goes down if you can exploit overlap)

Summary

- Parallel programs introduce three overhead terms: busy overhead (extra work), remote data access time, and synchronization time
  - Goal of a good parallel program is to minimize these three terms
  - Goal of a good parallel computer architecture is to provide sufficient support to let programmers optimize these three terms (and this is the focus of the rest of the course)

Previous Next