

Example Application: Matrix Multiplication

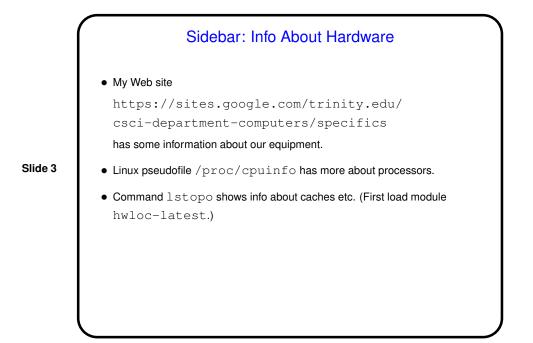
• Basic problem is straightforward: For two N by N matrices A and B, compute the matrix product C with elements defined thus (assuming 0-based indexing):

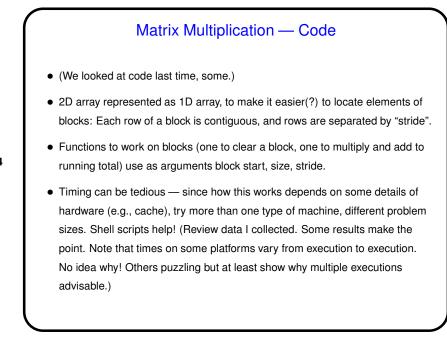
$$c_{i,j} = \sum_{k=0}^{N-1} a_{i,k} \cdot b_{k,j}$$

Slide 2

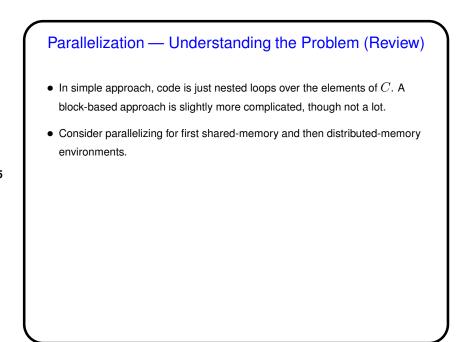
- (Actually $A \mbox{ and } B \mbox{ don't have to be square and the same size, but for the moment let's assume they are.)$
- Simple approach to calculating this is obvious just do the above calculation for all i and j between 0 and N 1.
- Less obvious approach: Decompose A, B, and C into blocks and think of the calculation in terms of these blocks (equation similar to the above, but for blocks rather than individual elements).

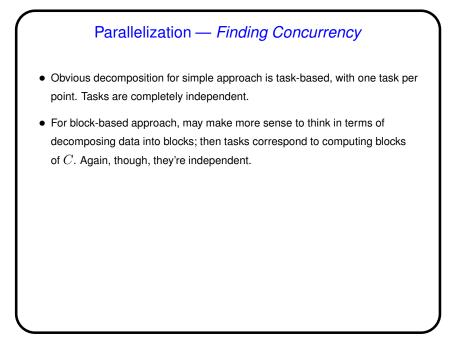
Why? often makes better use of cache and therefore is faster.

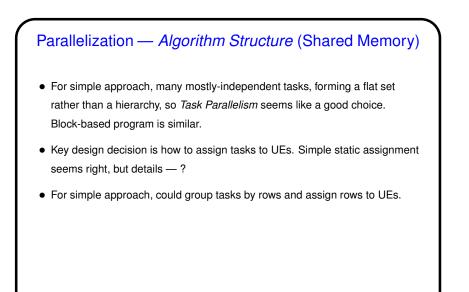




2







Parallelization — *Algorithm Structure* (Shared Memory), Continued

 For block-based approach, probably want to assign (groups of) blocks to UEs: Multithreading within block seems like might lose improvement in use of cache?

- Giving each UE rows of blocks is simple but may limit concurrency too much.
- Giving each UE individual blocks is more trouble but less limiting.

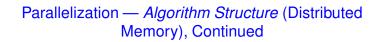


- For program structure, *Loop Parallelism* makes sense.
- Code in OpenMP is straightforward.
- (Look at code, timing data.)

Parallelization — *Algorithm Structure* (Distributed Memory)

• For distributed memory, have to think about how to distribute C and how to duplicate/distribute A and B. Might work better to think in terms of block-based approach and data decomposition — so *Geometric Decomposition* might be a better fit.

- Key design decisions here are how to decompose data and assign chunks to UEs, and then how to manage synchronization/communication for update operation.
- Probably makes sense to decompose data so we can assign one block of C to each UE amount of work per block is pretty much constant.



• For each block of *C*, computation can be thought of a sequence of update operations, each involving a different combination of blocks of *A* and *B*. (Compare how this fits overall idea of *Geometric Decomposition* with how heat-diffusion example fits.)

Slide 11

 This tells us what kind of communication we need. Simple approach is to broadcast two blocks at each step, one for "row" and one for "column". More complex, but more efficient, version involves rotating blocks among processes.



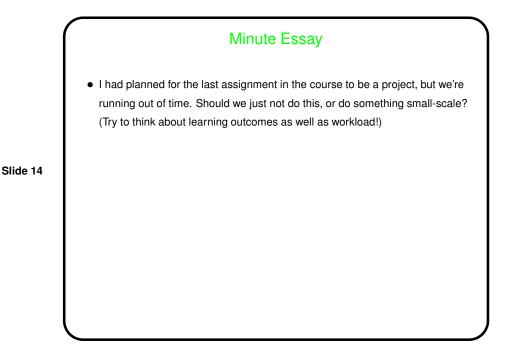
- For program structure, we probably want *SPMD* (especially if using MPI or similar programming environment).
- *Distributed Array* is relevant, especially for parts of sample/test program that initialize and print array (since they use each array element's global indices).

Parallelization — Code (Distributed Memory)

 If we distribute all three arrays (which seems like a good idea), have to make changes in code to initialize and print as well as matrix-multiplication. As is often the case with programs using *Distributed Array*, ideas are simple but code inclined to be messy!

Slide 13

- For actual multiplication, each process will update one "chunk", doing same computation done in the block-based sequential program, but with communication operations to broadcast two blocks per step.
- (Look at code, timing data. Note printing strategy: Each process prints its data, but with global indices; then can use external tools (cat, sort) to combine. Good idea to do a small test run for correctness in addition to timing.)



7