Work Allocation

Mark Greenstreet

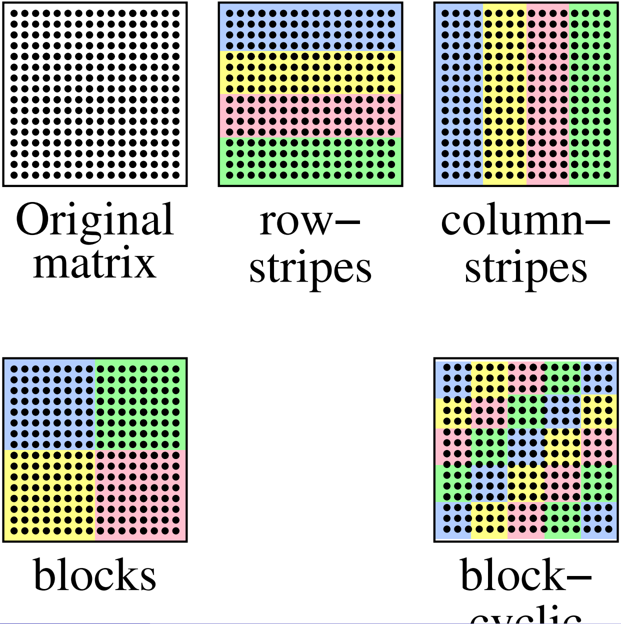
CpSc 418 – Oct. 25, 2012

Lecture Outline

Work Allocation

- Static Allocation (matrices and other arrays)
  - Stripes
  - Blocks
  - Block-Cyclic
  - Irregular meshes
- Dynamic Allocation
  - Work Queues
  - Work Stealing
  - Trees

Static Allocation: Paritioning Matrices



Matrix Multiplication

- Examined in September 25 lecture.
- Consider distributing a $N \times N$ matrix over P processors:
  - If arranged as P strips of N/P rows,
    - then computing a matrix multiplication requires each process to send and receive $P - 1$ messages of size N^2/P.
  - If arranged as $\sqrt{P} \times \sqrt{P}$ blocks of size $(N/\sqrt{P}) \times (N/\sqrt{P})$,
    - then computing a matrix multiplication requires each process to send and receive $\sqrt{P}$ messages of size N^2/P.
  - In practice, communication cost much more than computation.
    - Thus, the second arrangement achieves good speed-ups for smaller matrices than the first.
    - Both approaches have the same asymptotic performance.
    - What does this say about Amdahl's law?

LU-Decomposition

- Given a matrix, A, factor into matrices L, U, and P such that $PA = LU$ where
  - L is lower-triangular (all elements above the main diagonal are 0).
  - U is upper-triangular (all elements below the main diagonal are 0).
  - P is a permutaion matrix (rearranges the rows of A).
- Why?
  - We often want to solve linear systems:
    Given A and y, find x such that $Ax = y$.
  - If we can factor A so that $PA = LU$, then we get:

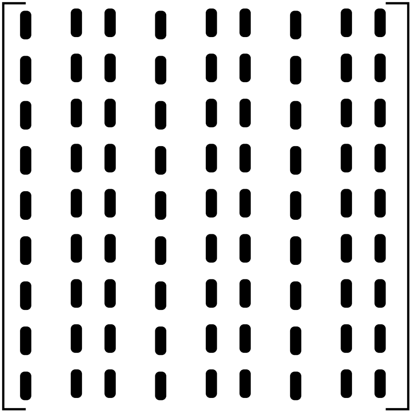
$$x = U^{-1}L^{-1}Py$$

    - ★ Computing $w = Py$ is very easy (just a permutation).
    - ★ Computing $z = L^{-1}w$ is easy $O(N^2)$ operations.
    - ★ Computing $x = U^{-1}z$ is easy $O(N^2)$ more operations.

LU-Decomposition

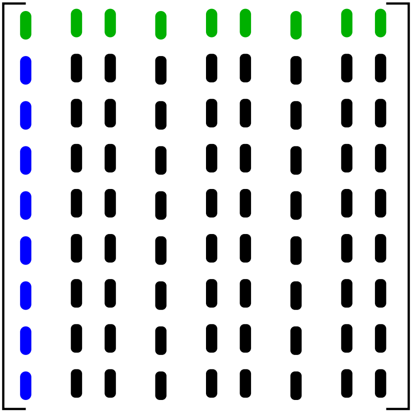
- Find the largest element in the first column (a reduce operation).
- Swap the row for that column with the first row, and scale to make the $A_{1,1} = 1$.
- Eliminate all elements in the first column except for $A_{1,1}$.
  - The multipliers for this form a column of the L matrix.
  - The main diagonal and the elements above it form the U matrix.
- Now, repeat for the $(N-1) \times (N-1)$ submatrix.

LU animated



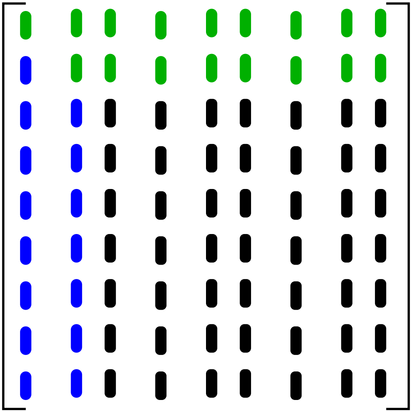
Initial matrix

LU animated



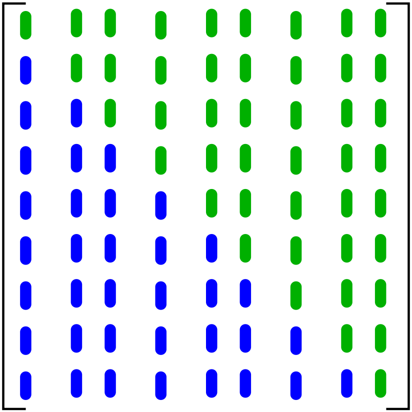
After first LU-decomp step

LU animated



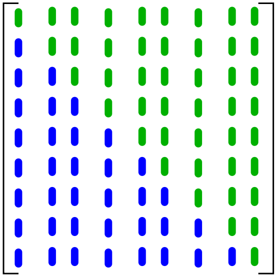
After second LU-decomp step Matrix

LU animated

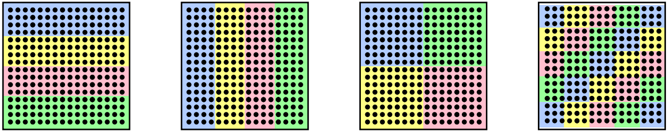


After final LU-decomp step Matrix

LU animated



After final LU-decomp step Matrix



row–stripes | column–stripes | blocks | block–cyclic

More meshes

- matrices used for linear algebra problems
- also used for representing spatial data and finite element computation.

```
repeat {
    Each grid location updates its value based on:
        • its current value;
        • the current values of its neighbours.
} until (convergence target reached)
```

- multi-resolution methods are common, but present extra challenges for distributing data and work.
- This isn't a scientific computing course:
  - So, I'll just let you know that the issues are there.
  - Lots of work has been done in this area.
  - When/if you need it, you can check the current state-of-the-art.

Dynamic Scheduling

- Work queues
- Trees and capping
- Work Stealing
- An example: PReach

Work Queues

```
while(the work queue is not empty) {
    wait for a free worker process;
    textrmAssign a task from the queue to the worker;
}

worker(Task) {
    W = estimate of work required to perform Task;
    if(W ≤ threshold)
        perform Task;
    else {
        {Task1, Task2} = divide(Task);
        insert(WorkQueue, Task1);
        insert(WorkQueue, Task2);
    }
}
```

- A reasonable model if tasks are relatively independent.
- Can be extended to handle simple dependencies between tasks.

Trees and Capping

Example: PReach

```
insert initial states into work queue
while(any process has a non-empty work-queue) {
    Each process:
        receive any incoming states
        dequeue a state if one is waiting
        if this state is new {
            compute successors of this state
            send these successors to their owner processes
        }
}
```

Work Stealing

Summary

- Work allocation determines how parallel taskw will be distributed between processes.
- What is the difference between static and dynamic work allocation?
- Why might we create more processes than we have processors?
- What is block-cyclic allocation?
  Give an example of where block-cyclic allocation is useful.
- What is a work queue?