Lecture 15: Dense Linear Algebra II

David Bindel

19 Oct 2011
Matrix multiply is graded

HW 2 logistics
- Viewer issue was a compiler bug – change Makefile to switch gcc version or lower optimization. Or grab the revised tarball.
- It is fine to use binning vs. quadtrees
Review: Parallel matmul

- Basic operation: $C = C + AB$
- Computation: $2n^3$ flops
- Goal: $2n^3/p$ flops per processor, minimal communication
- Two main contenders: SUMMA and Cannon
Outer product algorithm

Serial: Recall outer product organization:

```matlab
for k = 0:s-1
    C += A(:,k)*B(k,:);
end
```

Parallel: Assume $p = s^2$ processors, block $s \times s$ matrices. For a $2 \times 2$ example:

\[
\begin{bmatrix}
C_{00} & C_{01} \\
C_{10} & C_{11}
\end{bmatrix} = \begin{bmatrix}
A_{00}B_{00} & A_{00}B_{01} \\
A_{10}B_{00} & A_{10}B_{01}
\end{bmatrix} + \begin{bmatrix}
A_{01}B_{10} & A_{01}B_{11} \\
A_{11}B_{10} & A_{11}B_{11}
\end{bmatrix}
\]

- Processor for each $(i, j) \implies$ parallel work for each $k$!
- Note everyone in row $i$ uses $A(i, k)$ at once, and everyone in row $j$ uses $B(k, j)$ at once.
Parallel outer product (SUMMA)

for $k = 0:s-1$
  for each $i$ in parallel
    broadcast $A(i,k)$ to row
  for each $j$ in parallel
    broadcast $A(k,j)$ to col
  On processor $(i,j)$, $C(i,j) += A(i,k) \times B(k,j)$;
end

If we have tree along each row/column, then
  - $\log(s)$ messages per broadcast
  - $\alpha + \beta n^2/s^2$ per message
  - $2\log(s)(\alpha s + \beta n^2/s)$ total communication
  - Compare to 1D ring: $(p - 1)\alpha + (1 - 1/p)n^2\beta$

Note: Same ideas work with block size $b < n/s$
Parallel outer product (SUMMA)

If we have tree along each row/column, then

- \( \log(s) \) messages per broadcast
- \( \alpha + \beta \frac{n^2}{s^2} \) per message
- \( 2 \log(s)(\alpha s + \beta \frac{n^2}{s}) \) total communication

Assuming communication and computation can potentially overlap \textit{completely}, what does the speedup curve look like?
Reminder: Why matrix multiply?

LAPACK structure

Build fast serial linear algebra (LAPACK) on top of BLAS 3.
Reminder: Why matrix multiply?

ScaLAPACK builds additional layers on same idea.
Reminder: Evolution of LU

On board...
Blocked GEPP

Find pivot
Blocked GEPP

Swap pivot row
Blocked GEPP

Update within block
Blocked GEPP

Delayed update (at end of block)
Big idea

- *Delayed update* strategy lets us do LU fast
  - Could have also delayed application of pivots
- Same idea with other one-sided factorizations (QR)
- Can get decent multi-core speedup with parallel BLAS!
  ... assuming $n$ sufficiently large.

There are still some issues left over (block size? pivoting?)...
Explicit parallelization of GE

What to do:
- *Decompose* into work chunks
- *Assign* work to threads in a balanced way
- *Orchestrate* the communication and synchronization
- *Map* which processors execute which threads
Possible matrix layouts

1D column blocked: bad load balance

\[
\begin{bmatrix}
0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\
0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\
0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\
0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\
0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\
0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\
0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\
0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\
0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\
0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\
\end{bmatrix}
\]
Possible matrix layouts

1D column cyclic: hard to use BLAS2/3

\[
\begin{bmatrix}
0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\
0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\
0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\
0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\
0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\
0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\
0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\
0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\
0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\
0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\
\end{bmatrix}
\]
Possible matrix layouts

1D column block cyclic: block column factorization a bottleneck

\[
\begin{bmatrix}
0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 2 & 2 & 0 & 0 & 1 & 1 \\
\end{bmatrix}
\]
## Possible matrix layouts

Block skewed: indexing gets messy

$$
\begin{bmatrix}
0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\
0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\
0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\
2 & 2 & 2 & 0 & 0 & 0 & 1 & 1 & 1 \\
2 & 2 & 2 & 0 & 0 & 0 & 1 & 1 & 1 \\
2 & 2 & 2 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 2 & 2 & 2 & 0 & 0 & 0 \\
1 & 1 & 1 & 2 & 2 & 2 & 0 & 0 & 0 \\
1 & 1 & 1 & 2 & 2 & 2 & 0 & 0 & 0
\end{bmatrix}
$$
Possible matrix layouts

2D block cyclic:

\[
\begin{bmatrix}
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
2 & 2 & 3 & 3 & 2 & 2 & 3 & 3 \\
2 & 2 & 3 & 3 & 2 & 2 & 3 & 3 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
2 & 2 & 3 & 3 & 2 & 2 & 3 & 3 \\
2 & 2 & 3 & 3 & 2 & 2 & 3 & 3
\end{bmatrix}
\]
Possible matrix layouts

- 1D column blocked: bad load balance
- 1D column cyclic: hard to use BLAS2/3
- 1D column block cyclic: factoring column is a bottleneck
- Block skewed (a la Cannon): just complicated
- 2D row/column block: bad load balance
- 2D row/column block cyclic: win!
Distributed GEPP

Find pivot (column broadcast)
Distributed GEPP

Swap pivot row within block column + broadcast pivot
Distributed GEPP

Update within block column
Distributed GEPP

At end of block, broadcast swap info along rows
Distributed GEPP

Apply all row swaps to other columns
Distributed GEPP

Broadcast block $L_{//}$ right
Distributed GEPP

Update remainder of block row
Distributed GEPP

Broadcast rest of block row down
Distributed GEPP

Broadcast rest of block col right
Distributed GEPP

Update of trailing submatrix
Cost of ScaLAPACK GEPP

Communication costs:

- Lower bound: $O(n^2/\sqrt{P})$ words, $O(\sqrt{P})$ messages
- ScaLAPACK:
  - $O(n^2 \log P/\sqrt{P})$ words sent
  - $O(n \log p)$ messages
  - Problem: reduction to find pivot in each column
- Recent research on stable variants without partial pivoting
What if you don’t care about dense Gaussian elimination?
Let’s review some ideas in a different setting...
**Floyd-Warshall**

Goal: Find shortest path lengths between all node pairs.

Idea: Dynamic programming! Define

\[
d_{ij}^{(k)} = \text{shortest path } i \text{ to } j \text{ with intermediates in } \{1, \ldots, k\}.
\]

Then

\[
d_{ij}^{(k)} = \min \left( d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \right)
\]

and \(d_{ij}^{(n)}\) is the desired shortest path length.
The same and different

Floyd’s algorithm for all-pairs shortest paths:

\[
\text{for } k=1:\text{n} \\
\quad \text{for } i = 1:\text{n} \\
\quad \quad \text{for } j = 1:\text{n} \\
\quad \quad \quad D(i,j) = \min(D(i,j), D(i,k)+D(k,j));
\]

Unpivoted Gaussian elimination (overwriting A):

\[
\text{for } k=1:\text{n} \\
\quad \text{for } i = k+1:\text{n} \\
\quad \quad A(i,k) = A(i,k) / A(k,k); \\
\quad \quad \text{for } j = k+1:\text{n} \\
\quad \quad \quad A(i,j) = A(i,j)-A(i,k)*A(k,j);
\]
The same and different

- The same: $O(n^3)$ time, $O(n^2)$ space
- The same: can’t move $k$ loop (data dependencies)
  - ... at least, can’t without care!
  - Different from matrix multiplication
- The same: $x_{ij}^{(k)} = f \left( x_{ij}^{(k-1)}, g \left( x_{ik}^{(k-1)}, x_{kj}^{(k-1)} \right) \right)$
  - Same basic dependency pattern in updates!
  - Similar algebraic relations satisfied
- Different: Update to full matrix vs trailing submatrix
How far can we get?

How would we

▶ Write a cache-efficient (blocked) *serial* implementation?
▶ Write a message-passing *parallel* implementation?

The full picture could make a fun class project...
Onward!

Next up: Sparse linear algebra and iterative solvers!