Knights Tour

- Find a path through the squares of a chess board
  — move like a knight (L)
  — visit each square exactly once
- Closed tours end a knight’s move from the original square

Figure credits: Wikipedia
Kuldeep Takes the Tour

- There are 33 trillion closed tours on an 8x8 board
- Exact number of open tours on an 8x8 board is unknown
- Recent result shows that there are at least 33 trillion + 5000 open tours
- Kuldeep’s (rising star of the department) solution to the open tours problem
  - employ an approximate counting technique
  - formulate the tour as a SAT problem
    - 60000 variables for an approximate count
    - 1M variables for an accurate count
  - employ the formulation in CryptoMiniSAT
  - won the gold in the “SAT Olympics” in 2011
Heart of the SAT solver

- Gaussian Elimination on a dense matrix of rank 10K Vs 60K
  — of course, it's parallel but only shared-memory parallelization
  — distributed memory implementations exist
  — not used in this solver
Communication-Avoiding Gaussian Elimination

- Berkeley Bebop group
- .5 D class of algorithms
- communication vs local computation
  - addition/multiplication is cheap
  - memory accesses are costly

<table>
<thead>
<tr>
<th>Annual improvements</th>
<th>Time_per_flop</th>
<th>Bandwidth</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>59% Network</td>
<td>26%</td>
<td>15%</td>
</tr>
<tr>
<td></td>
<td>59% DRAM</td>
<td>23%</td>
<td>5%</td>
</tr>
</tbody>
</table>

Reduced communication saves time

Figure credits: Demmel, IPDPS13 keynote
Kuldeep tries to understand CA Matrix Multiplication before trying CA GE
Matrix Multiplication

Consider data needed for output matrix block shown in purple

\[ \begin{array}{ccc}
  A & \times & B \\
  \times & = & C \\
\end{array} \]
Cannon’s Matrix Multiplication

A, B are distributed on $\sqrt{p} \times \sqrt{p}$ processor grid
Communication-Avoiding 2.5D Matrix Multiplication

Mathematica Demo
Algorithm 2: \([C] = 2.5\text{D-matrix-multiply}(A,B,n,p,c)\)

**Input:** square \(n\)-by-\(n\) matrices \(A\), \(B\) distributed so that \(P_{ij0}\) owns \(n/\sqrt{p/c}\)-by-\(n/\sqrt{p/c}\) blocks \(A_{ij}\) and \(B_{ij}\) for each \(i,j\)

**Output:** square \(n\)-by-\(n\) matrix \(C = A \cdot B\) distributed so that \(P_{ij0}\) owns \(n/\sqrt{p/c}\)-by-\(n/\sqrt{p/c}\) block \(C_{ij}\) for each \(i,j\)

/* do in parallel with all processors */

forall \(i,j \in \{0, 1, ..., \sqrt{p/c} - 1\}, k \in \{0, 1, ..., c - 1\}\) do

\(P_{ij0}\) broadcasts \(A_{ij}\) and \(B_{ij}\) to all \(P_{ijk}\)

\(s := \text{mod}\ (j - i + k\sqrt{p/c^3}, \sqrt{p/c})\)

\(P_{ijk}\) sends \(A_{ij}\) to \(A_{\text{local}}\) on \(P_{isk}\)

\(s' := \text{mod}\ (i - j + k\sqrt{p/c^3}, \sqrt{p/c})\)

\(P_{ijk}\) sends \(B_{ij}\) to \(B_{\text{local}}\) on \(P_{s'jk}\)

\(C_{ijk} := A_{\text{local}} \cdot B_{\text{local}}\)

\(s := \text{mod}\ (j + 1, \sqrt{p/c})\)

\(s' := \text{mod}\ (i + 1, \sqrt{p/c})\)

for \(t = 1\) to \(\sqrt{p/c^3} - 1\) do

\(P_{ijk}\) sends \(A_{\text{local}}\) to \(P_{isk}\)

\(P_{ijk}\) sends \(B_{\text{local}}\) to \(P_{s'jk}\)

\(C_{ijk} := C_{ijk} + A_{\text{local}} \cdot B_{\text{local}}\)

end

\(P_{ijk}\) contributes \(C_{ijk}\) to a sum-reduction to \(P_{ij0}\)

end

[Communication-Avoiding 2.5D Matrix Multiplication](Edgar Solomonik, James Demmel: Communication-Optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms. Euro-Par (2) 2011: 90-109)
So.. what does Kuldeep do here ?
**What’s the Role For Our Compiler ?**

- **Knights Tour**
- **CryptoMiniSat**
- **Gaussian/MM**
- **KULDEEP**

---

### Manual Approach

- **Library for 2.5D**
  - Or
  - Implement 2.5D

- **Parallel Language**
  - HJ, CAF, HJ_LIB

---

### Shams/Sagnak

- **Deepak/Chaoran**

---

### Milind/Rishi

- **Performance/Correctness Toolkits**

---

### Tuned Architecture

- **Dragos/Xu/Kamal**

---

### Related efforts:

- **Karthik**
- **Jun Shirako**
- **Alina**

---

```c
/* do in parallel with all processors */
forall i, j ∈ {0, 1, ..., √p/c − 1}, k ∈ {0, 1, ..., c − 1} do
    P_{ij0} broadcasts A_{ij} and B_{ij} to all P_{ijk}
    s := mod (j − i + k√p/c², √p/c)
    P_{ijk} sends A_{ij} to A_{local} on P_{isk}
    s' := mod (i − j + k√p/c², √p/c)
    P_{ijk} sends B_{ij} to B_{local} on P_{s'jk}
    C_{ijk} := A_{local} · B_{local}
    s := mod (j + 1, √p/c)
    s' := mod (i + 1, √p/c)
    for t = 1 to \sqrt{p/c^2} − 1 do
        P_{ijk} sends A_{local} to P_{js'jk}
        P_{ijk} sends B_{local} to P_{s'jk}
        C_{ijk} := C_{ijk} + A_{local} · B_{local}
    end
    P_{ijk} contributes C_{ijk} to a sum-reduction to P_{ij0}
end
```
Further, the cost of sacrificing flops for latency is large. Namely, if \( P \) is best to pick all \( P \) by counting the complexity along this path. The latency cost is given this dependency path (shown in Figure 2), we can lower bound the complexity of the algorithm. Factorizations of these blocks are on the critical path and must be done in strict sequence.

We now lower bound the communication cost for any algorithm that follows the above restrictions. Any such algorithm must compute a sequence of diagonal blocks. Evidently, if we want to do \( 2 \). The above condition holds recursively (for factorization of \( 1 \). Consider the largest

\[
\begin{align*}
\text{(a)} & \quad P \quad \text{(b)} \\
\text{(c)} & \quad P
\end{align*}
\]

\[
\begin{align*}
\text{2.5D LU communication lower bound}
\end{align*}
\]

end

/** do in parallel with all processors */
for all \( i, j \in \{0, 1, \ldots, \sqrt{p/c} - 1\} \), \( k \in \{0, 1, \ldots, c - 1\} \) do

\( P_{ij0} \) broadcasts \( A_{ij} \) and \( B_{ij} \) to all \( P_{ijk} \)


- Identify that broadcasts occur along the ‘c’ dimension
- Create sub-teams
  — given \( i \in 1 \ldots \sqrt{(p/c)} \), \( j \in 1 \ldots \sqrt{(p/c)} \), \( k \in 1 \ldots c \)
    - sub-team\(_{ij} = \forall k [i,j,k] \)
  — vector of processors along the ‘c’ dimension
- Perform a broadcast within each sub-team
Further, the cost of sacrificing flops for latency is large. Namely, if we see that the algorithmic costs are best to pick all $P$ by counting the complexity along this path. The latency cost is the latency lower bound is actually much higher, namely $4 \cdot 2.5D$ LU communication lower bound.

We argue that for Gaussian-elimination style LU algorithms that achieve the bandwidth lower bound, the $4 \cdot 2.5D$ LU communication lower bound must hold, pick some $P$.

Given a parallel LU factorization algorithm, we assume the algorithm must uphold the following properties:

Evidently, if we want to do in parallel with all processors:

```c
/* do in parallel with all processors */
forall $i, j \in \{0, 1, \ldots, \sqrt{p/c} - 1\}, k \in \{0, 1, \ldots, c - 1\}$ do
    $P_{ij0}$ broadcasts $A_{ij}$ and $B_{ij}$ to all $P_{ijk}$
    $s := \text{mod} (j - i + k \sqrt{p/c^3}, \sqrt{p/c})$
    $P_{ijk}$ sends $A_{ij}$ to $A_{\text{local}}$ on $P_{isk}$
    $s' := \text{mod} (i - j + k \sqrt{p/c^3}, \sqrt{p/c})$
    $P_{ijk}$ sends $B_{ij}$ to $B_{\text{local}}$ on $P_{s'jk}$
    $C_{ijk} := A_{\text{local}} \cdot B_{\text{local}}$
    $s := \text{mod} (j + 1, \sqrt{p/c})$
    $s' := \text{mod} (i + 1, \sqrt{p/c})$
    for $t = 1$ to $\sqrt{p/c^3} - 1$ do
        $P_{ijk}$ sends $A_{\text{local}}$ to $P_{isk}$
        $P_{ijk}$ sends $B_{\text{local}}$ to $P_{s'jk}$
        $C_{ijk} := C_{ijk} + A_{\text{local}} \cdot B_{\text{local}}$
    end
    $P_{ijk}$ contributes $C_{ijk}$ to a sum-reduction to $P_{ij0}$
end
```

- Recognize reduction along the ‘c’ dimension —use existing sub-teams
Further, the cost of sacrificing flops for latency is large. Namely, if we see that the algorithmic costs are best to pick all \( P \) by counting the complexity along this path. The latency cost is given this dependency path (shown in Figure 2), we can lower bound the complexity of the algorithm such that the blocks be

\[
\begin{align*}
\text{4 2.5D LU communication lower bound} & \quad \text{is actually much higher, namely} \\
\text{We argue that for Gaussian-elimination style LU algorithms that achieve} & \quad \text{the bandwidth lower bound, the} \\
\text{end} & \quad \text{of the above condition holds recursively (for factorization of} \\
\text{1. Consider the largest} & \quad \text{factorization of} \\
\text{2. The above condition holds recursively (for factorization of} & \quad \text{factorization of} \\
\end{align*}
\]

Evidently, if we want to do

\[
\begin{align*}
\text{/* do in parallel with all processors} & \quad \text{/* do in parallel with all processors} \\
\text{forall } i, j \in \{0, 1, \ldots, \sqrt{p/c} - 1\}, k \in \{0, 1, \ldots, c - 1\} \text{ do} & \quad \text{forall } i, j \in \{0, 1, \ldots, \sqrt{p/c} - 1\}, k \in \{0, 1, \ldots, c - 1\} \text{ do} \\
P_{ij0} \text{ broadcasts } A_{ij} \text{ and } B_{ij} \text{ to all } P_{ijk} & \quad P_{ij0} \text{ broadcasts } A_{ij} \text{ and } B_{ij} \text{ to all } P_{ijk} \\
\end{align*}
\]

- **Needs**
  - Understand broadcast/reduce along the ‘c’ dimension
  - Create sub-teams
  - Perform the collective operation using the sub-teams

- **Status**
  - Currently handle collectives over all processors, not projections of processor array
  - Need to support collectives over projections of processor array as well

\[
\begin{align*}
P_{ijk} \text{ contributes } C_{ijk} \text{ to a sum-reduction to } P_{ij0} & \quad P_{ijk} \text{ contributes } C_{ijk} \text{ to a sum-reduction to } P_{ij0} \\
\end{align*}
\]
Further, the cost of sacrificing flops for latency is large. Namely, if we see that the algorithmic costs are minimized, it is best to pick all factors of the blocks be non-overlapping. By counting the complexity along this path. The latency cost is defined as the number of messages broadcast before and after the local multiplication (no communication). Given this dependency path (shown in Figure 2), we can lower bound the complexity of the algorithm as follows:

\[ \text{Complexity} \approx \frac{2}{p/c} + \frac{2}{p/c} \]

We now lower bound the communication cost for any algorithm that follows the above restrictions. Any algorithm that achieves the bandwidth lower bound, the latency lower bound is actually much higher, namely \[ O(n^2) \]

We argue that for Gaussian-elimination style LU algorithms that achieve the bandwidth lower bound, the factorization cost is \[ O(n^2) \]. By the constraint, the following conditions must hold:

- The factorization cost is \[ O(n^2) \].
- As done in Gaussian Elimination and as required by our conditions, the factorization cost is \[ O(n^2) \].
- As required by our conditions, the factorization cost is \[ O(n^2) \].

\[ \forall i, j \in \{0, 1, ..., \sqrt{p/c} - 1\}, k \in \{0, 1, ..., c - 1\} \text{ do} \]

\[ P_{ij0} \text{ broadcasts } A_{ij} \text{ and } B_{ij} \text{ to all } P_{ijk} \]

\[ s := \text{mod}(j - i + k\sqrt{p/c^3}, \sqrt{p/c}) \]

\[ P_{ijk} \text{ sends } A_{ij} \text{ to } A_{\text{local}} \text{ on } P_{isk} \]

\[ s' := \text{mod}(i - j + k\sqrt{p/c^3}, \sqrt{p/c}) \]

\[ P_{ijk} \text{ sends } B_{ij} \text{ to } B_{\text{local}} \text{ on } P_{s'jk} \]

\[ C_{ijk} := A_{\text{local}} \cdot B_{\text{local}} \]

end

/* do in parallel with all processors */

forall i, j \in \{0, 1, ..., \sqrt{p/c} - 1\}, k \in \{0, 1, ..., c - 1\} do

P_{ij0} broadcasts A_{ij} and B_{ij} to all P_{ijk}

s := mod(j - i + k\sqrt{p/c^3}, \sqrt{p/c})

P_{ijk} sends A_{ij} to A_{\text{local}} on P_{isk}

s' := mod(i - j + k\sqrt{p/c^3}, \sqrt{p/c})

P_{ijk} sends B_{ij} to B_{\text{local}} on P_{s'jk}

C_{ijk} := A_{\text{local}} \cdot B_{\text{local}}
• Problem
  — mod expression encapsulates who-talks-to-whom information
  — mod expression needs to be inverted to determine \{sender, receiver\} tuples

• Solution
  — express the mod in *presburger arithmetic* before passing it to *omega* for code generation
  — consider dest = mod(me, p1) + 1

*Relations input to Omega*
  — known := \{ [X] : 1 <= me <= p1 \};
  — receiver := \{ [X] -> [dest] : 1 <= me,dest <= p1  
    &&
    ((me < p1  
    &&  
    dest = me + 1)  
    ||
    (me = p1  
    &&  
    dest = me + 1 - p1))\};
  — codegen receiver given known;
Would generating one-sided communication solve the problem?
— No!
— Must
   — Identify the points of synchronization
   — Identify the participants in the synchronization
— Equivalent to the problem of understanding sender/receiver pairs
Further, the cost of sacrificing flops for latency is large. Namely, if see that the algorithmic costs are is best to pick all by counting the complexity along this path. The latency cost is Given this dependency path (shown in Figure 2), we can lower bound the complexity of the algorithm factorizations of these blocks are on the critical path and must be done in strict sequence.

Given a parallel LU factorization algorithm, we assume the algorithm must uphold the following properties

- **2.5D LU communication lower bound**
- **—employ extra buffers for holding A, B blocks**
- **—switch between buffers each iteration**

```
/* do in parallel with all processors */
forall i, j ∈ {0, 1, ..., √p/c - 1}, k ∈ {0, 1, ..., c - 1} do
    P_{ij0} broadcasts A_{ij} and B_{ij} to all P_{ijk}
    s := mod (j + k√p/c^3, √p/c)
    P_{ijk} sends A_{ij} to A_{local} on P_{isk}
    s' := mod (i + j + k√p/c^3, √p/c)
    P_{ijk} sends B_{ij} to B_{local} on P_{s'jk}
    C_{ijk} := A_{local} · B_{local}
    s := mod (j + 1, √p/c)
    s' := mod (i + 1, √p/c)
    for t = 1 to √p/c^3 - 1 do
        P_{ijk} sends A_{local} to P_{isk}
        P_{ijk} sends B_{local} to P_{s'jk}
        C_{ijk} := C_{ijk} + A_{local} · B_{local}
    end
    P_{ijk} contributes C_{ijk} to a sum-reduction to P_{ij0}
end
```
Communication Avoiding Compiler

• Goal: Simplify development of communication-avoiding code
  — Understand important patterns of computation
    – e.g., stencils, matrix operations, …
  — Identify the feasibility of transformations and generate CA code

• Demonstrate applicability to a broad range of computations
  — Matrix Multiplication, N-Body (step 1)
  — All pairs-shortest path (step 2)
  — LU factorization (step 3)
  — RedBlack stencil (e.g. GSRB in GMG) (step 4)
  — Twisted N-Body, AKX kernel (multiple stencils + sparse computation), Krylov
• Generate code for CA parallel algorithms (reduce communication between processors)
  — CA .5D family of algorithms from the Berkeley group
  — high level algorithms needed to reduce communication
  — mechanically generate data movement details from a high level sketch

• Generate code to efficiently manage memory hierarchy (reduce communication on processor)
  — e.g., temporal skewing, threaded wavefronts
    – goal: generate sophisticated code from a high-level specification
  — efficient tiled code using LP to compute tile extents

THE END