# Maunam:
# A Communication-Avoiding Compiler

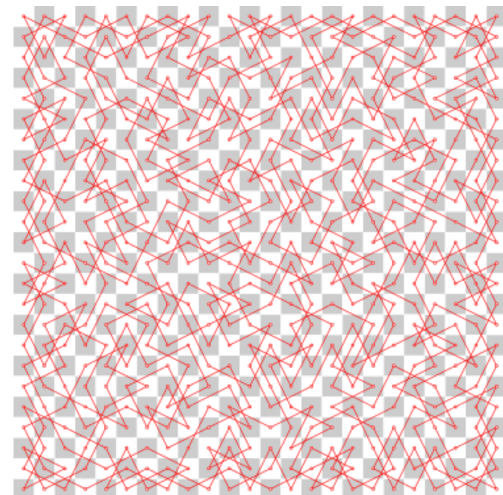
**Karthik Murthy**

**Advisor: John Mellor-Crummey**

**Department of Computer Science**
**Rice University**

**karthik.murthy@rice.edu**

# Knights Tour



Closed Knight's Tour on a 24 × 24 board

- **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**
- **Conrad, A.; Hindrichs, T.; Morsy, H. & Wegener, I. (1994). "Solution of the Knight's Hamiltonian Path Problem on Chessboards". Discrete Applied Mathematics**

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

| Annual improvements | | | |
|---|---|---|---|
| Time_per_flop | | Bandwidth | Latency |
| 59% | Network | 26% | 15% |
| | DRAM | 23% | 5% |

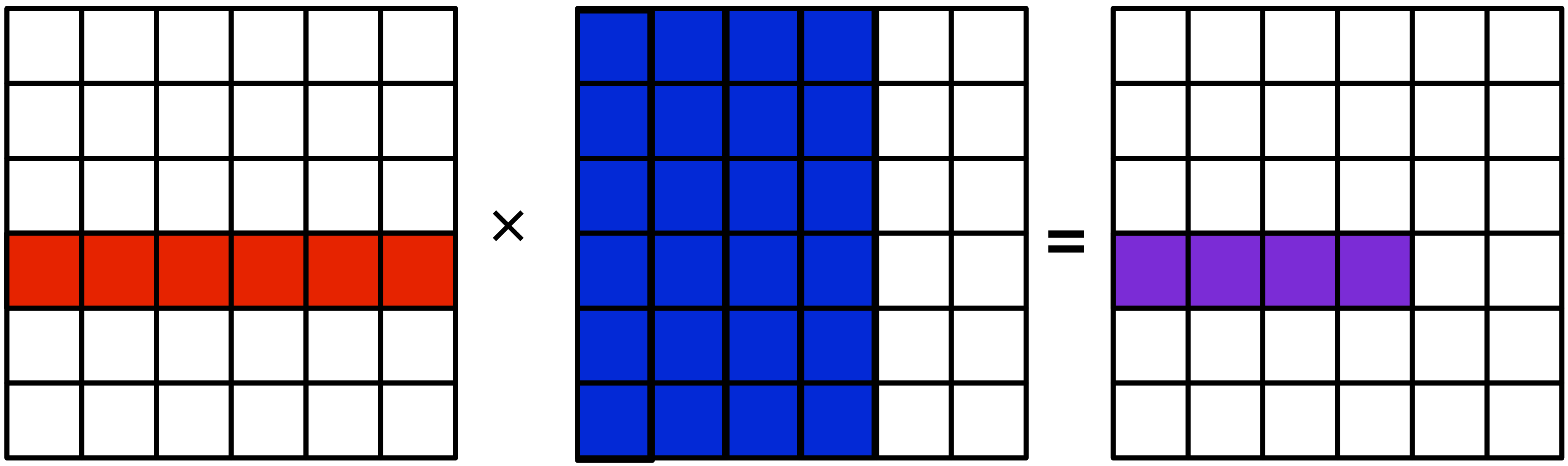
**reduced communication saves time**

Figure credits: Demmel, IPDPS13 keynote

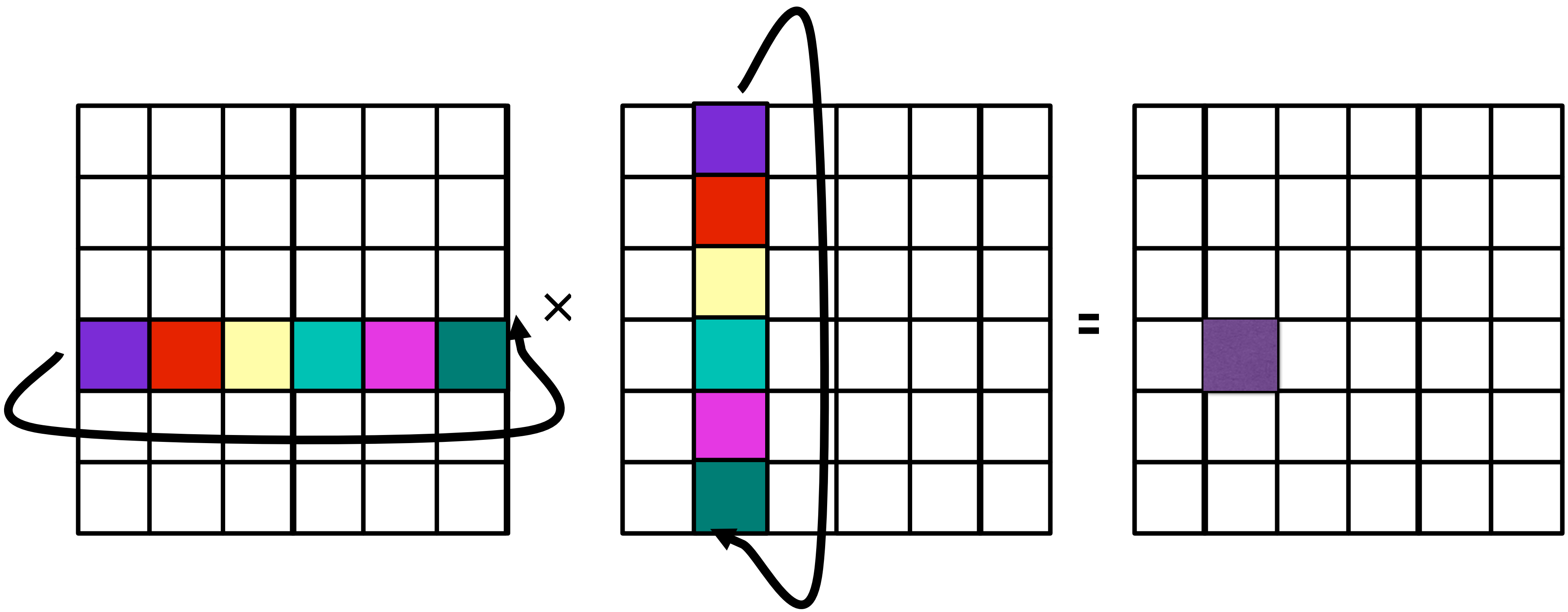
## Kuldeep tries to understand CA Matrix Multiplication before trying CA GE

# Matrix Multiplication

A X B = C

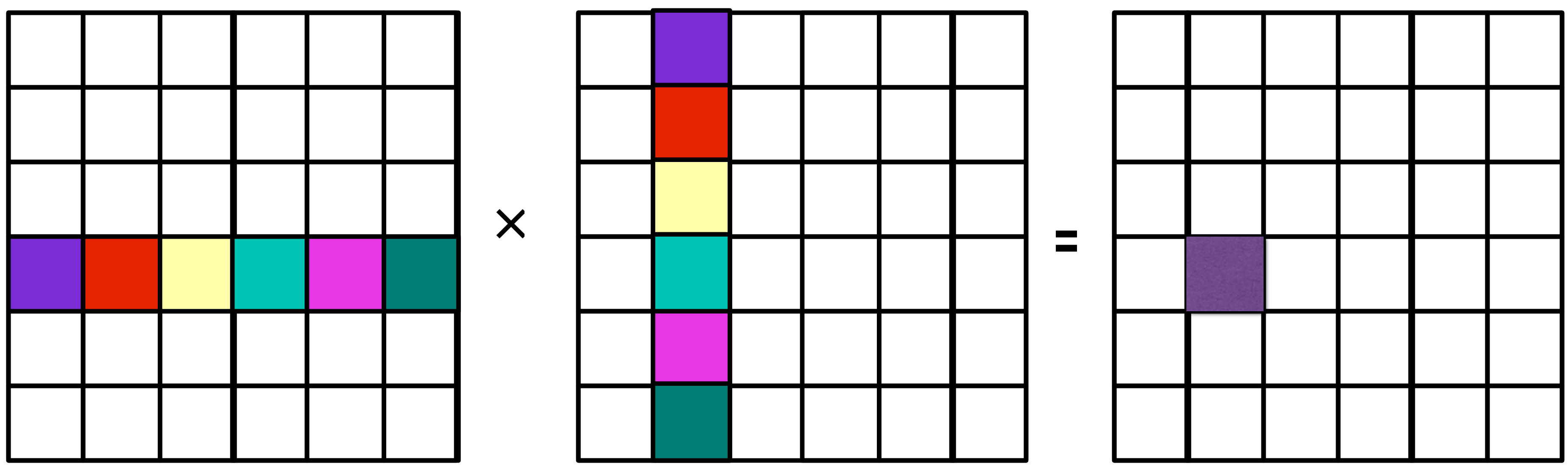


# Cannon's Matrix Multiplication



A, B are distributed on $\sqrt{p}$ x $\sqrt{p}$ processor grid

# Communication-Avoiding 2.5D Matrix Multiplication



## Mathematica Demo

# Communication-Avoiding 2.5D Matrix Multiplication

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

**Input**: square $n$-by-$n$ matrices $A$, $B$ distributed so that $P_{ij0}$ owns $\frac{n}{\sqrt{p/c}}$-by-$\frac{n}{\sqrt{p/c}}$ 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 $\frac{n}{\sqrt{p/c}}$-by-$\frac{n}{\sqrt{p/c}}$ 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 $P_{ijk}$
- $s := \mod(j - i + k\sqrt{p/c^3}, \sqrt{p/c})$
- $P_{ijk}$ 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 $P_{s'jk}$
- $C_{ijk} := A_{\text{local}} \cdot B_{\text{local}}$
- $s := \mod(j + 1, \sqrt{p/c})$
- $s' := \mod(i + 1, \sqrt{p/c})$
- **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**

broadcast front plane's blocks to the back planes of the cube

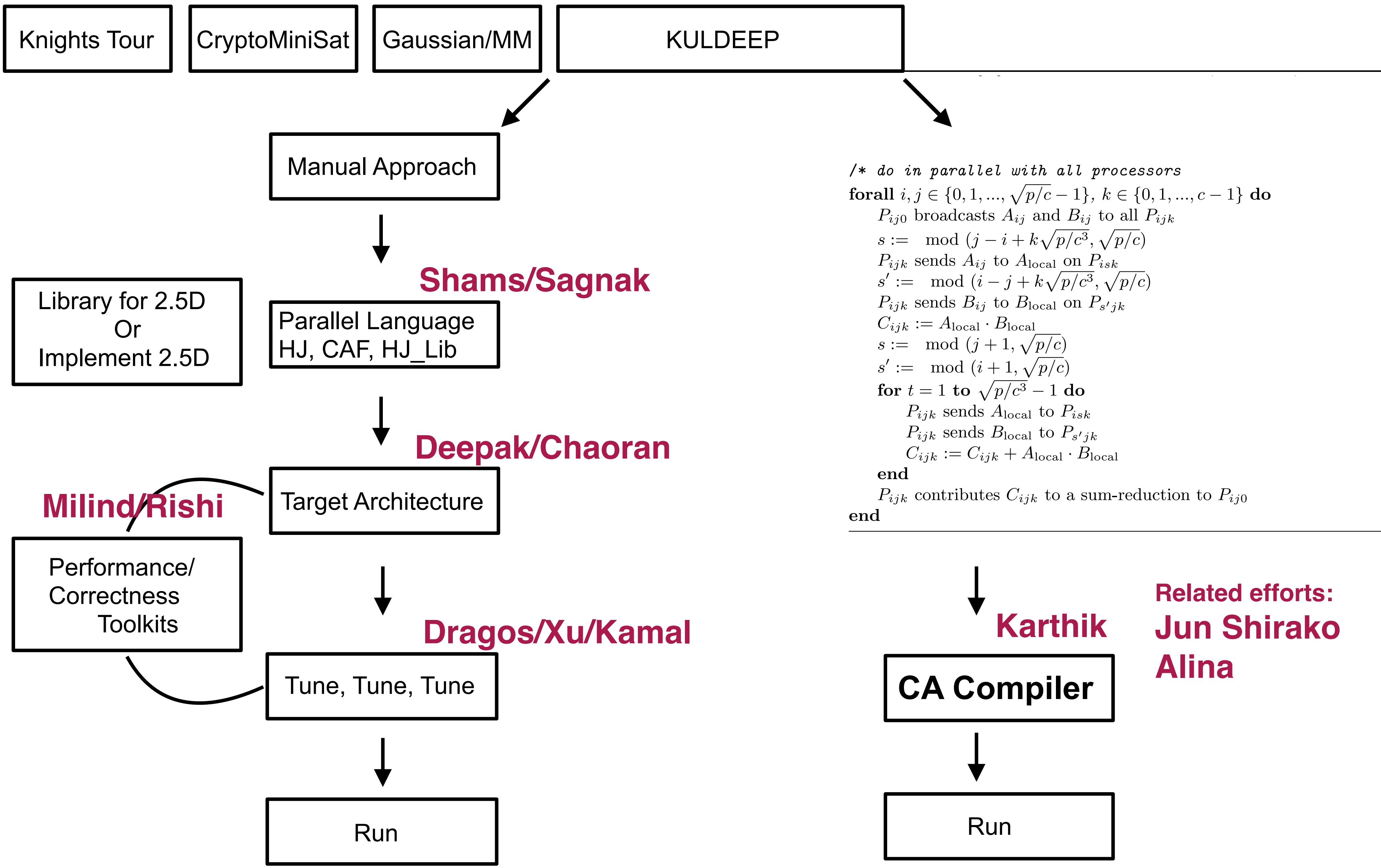
analogous to Cannon's alignment

analogous to Cannon's multiply and shift

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 ?



# Compiling 2.5D Matrix Multiplication Algorithm

```
/* 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 $P_{ijk}$

- **Identify that broadcasts occur along the 'c' dimension**
- **Create sub-teams**
  - **given i ∈ 1 ..√(p/c), j ∈ 1 ..√(p/c), k ∈ 1 .. c**
    - **sub-team$_{ij}$ = ∀k [i,j,k]**
  - **vector of processors along the 'c' dimension**
- **Perform a broadcast within each sub-team**

**end**

# Compiling 2.5D Matrix Multiplication Algorithm

```
/* 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 $A_{\mathrm{local}}$ on $P_{isk}$

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

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

$C_{ijk} := A_{\mathrm{local}} \cdot B_{\mathrm{local}}$

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

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

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

$P_{ijk}$ sends $A_{\mathrm{local}}$ to $P_{isk}$

$P_{ijk}$ sends $B_{\mathrm{local}}$ to $P_{s'jk}$

$C_{ijk} := C_{ijk} + A_{\mathrm{local}} \cdot B_{\mathrm{local}}$

**end**

$P_{ijk}$ contributes $C_{ijk}$ to a sum-reduction to $P_{ij0}$

**end**

- **Recognize reduction along the 'c' dimension**

  —use existing sub-teams

# CA Compiler Status Update

*/* 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 $P_{ijk}$

- **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

$P_{ijk}$ contributes $C_{ijk}$ to a sum-reduction to $P_{ij0}$

**end**

# Compiling 2.5D Matrix Multiplication Algorithm

```
/* 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 $P_{ijk}$

$$\left.\begin{array}{l} s := \mod (j - i + k\sqrt{p/c^3}, \sqrt{p/c}) \\ P_{ijk} \text{ sends } A_{ij} \text{ to } A_{\text{local}} \text{ on } P_{isk} \\ s' := \mod (i - j + k\sqrt{p/c^3}, \sqrt{p/c}) \\ P_{ijk} \text{ sends } B_{ij} \text{ to } B_{\text{local}} \text{ on } P_{s'jk} \end{array}\right\} \longrightarrow \textbf{Recognize shift communication}$$

$C_{ijk} := A_{\text{local}} \cdot B_{\text{local}}$ $\longrightarrow$ **Local multiplication (no communication)**

**end**

# CA Compiler Status Update

- **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;

# CA Compiler Status Update

**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

# Compiling 2.5D Matrix Multiplication Algorithm

```
/* 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 $P_{s'jk}$

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

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

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

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

$P_{ijk}$ sends $A_{\text{local}}$ to $P_{isk}$ } **shift operation**

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

$C_{ijk} := C_{ijk} + A_{\text{local}} \cdot B_{\text{local}}$ } **local multiplication**

**end**

$P_{ijk}$ contributes $C_{ijk}$ to a sum-reduction to $P_{ij0}$

**end**

- **Recognize opportunity to overlap computation with communication**
  - —employ extra buffers for holding A, B blocks
  - —switch between buffers each iteration

# 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

# Communication Avoiding Compiler

- 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**