Communication-Avoiding Compilers (?)

Kathy Yelick
Associate Laboratory Director of Computing Sciences
Lawrence Berkeley National Laboratory

EECS Professor, UC Berkeley
Challenge #4: Communication is expensive

Communication is expensive…
… time and energy

Cost components:
• Bandwidth: # of words
• Latency: # messages

Strategies
• Overlap: hide latency
• Avoid: algorithms to reduce bandwidth use and number of messages (latency)

Hard to change: Latency is physics; bandwidth is money!

<table>
<thead>
<tr>
<th>Annual improvements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flops</td>
</tr>
<tr>
<td>Network</td>
</tr>
<tr>
<td>59%</td>
</tr>
</tbody>
</table>

Hard to change: Latency is physics; bandwidth is money!
Because of cost and power issues, we cannot have both high memory bandwidth and large memory capacity.

The colored region is feasible in 2017.

**Compute intensive architecture focus on upper-left**
**Data Intensive architecture focus on lower right**

Slide source: John Shalf
Finding Good Performance is like finding the Needle in a Haystack

OSKI sparse matrix library: offline search + online evaluation: adding zeros can reduce storage in blocked format

Dense: \( MFlops(r,c) \)

\[
\frac{MFlops(r,c)}{Tsopf: \ Fill(r,c)} = Effective\_MFlops(r,c)
\]

Selected RB(5x7) with a sample dense matrix

Work by Im, Vuduc, Williams, Kamil, Ho, Demmel, Yelick…
Autotuning: Write Code Generators

- Autotuners are code generators plus search
- Avoids two unsolved compiler problems: dependence analysis and accurate performance models
- Popular in libraries: Atlas, FFTW, OSKI,…

Work by Williams, Oliker, Shalf, Madduri, Kamil, Im, Ethier,…
Approaches to Autotuning

How do we produce all of these (correct) versions?

• Using scripts (Python, perl, C,..)
• Transform high level representation (FFTW, Spiral)
• Compiling a domain-specific language (D-TEC)
• Compiling a general-purpose language (X-Tune)
• Dynamic compilation of a domain-specific (SEJITS)
Avoiding Communication in Iterative Solvers

- Consider Sparse Iterative Methods for $Ax=b$
  - Krylov Subspace Methods: GMRES, CG,…

- Solve time dominated by:
  - Sparse matrix-vector multiple (SPMV)
    - Which even on one processor is dominated by “communication” time to read the matrix
  - Global collectives (reductions)
    - Global latency-limited

- Can we lower the communication costs?
  - Latency: reduce # messages by computing multiple reductions at once
  - Bandwidth to memory, i.e., compute $Ax, A^2x, \ldots$ $A^kx$ with one read of $A$
Multiplying by a matrix is equivalent to nearest neighbor relaxation on a grid.

Simplest example: a tridiagonal matrix

Is the same as relaxation on a line grid:

\[ y[i] = \ldots x[i-1] + \ldots x[i] + \ldots x[i+1] \]
Communication Avoiding Kernels

The Matrix Powers Kernel: \([Ax, A^2x, \ldots, A^kx]\)

- Replace \(k\) iterations of \(y = A \cdot x\) with \([Ax, A^2x, \ldots, A^kx]\)

- Idea: pick up part of \(A\) and \(x\) that fit in fast memory, compute each of \(k\) products

- Example: A tridiagonal matrix (a 1D “grid”), \(n=32\), \(k=3\)

- General idea works for any “well-partitioned” \(A\)
Communication Avoiding Kernels
(Sequential case)

The Matrix Powers Kernel: \([Ax, A^2x, \ldots, A^kx]\)

- Replace \(k\) iterations of \(y = A \cdot x\) with \([Ax, A^2x, \ldots, A^kx]\)
- **Sequential Algorithm**

- Example: A tridiagonal, \(n=32, k=3\)
- Saves bandwidth (one read of \(A\&x\) for \(k\) steps)
- Saves latency (number of independent read events)
Communication Avoiding Kernels: (Parallel case)

The Matrix Powers Kernel: \([Ax, A^2x, \ldots, A^kx]\)

- Replace \(k\) iterations of \(y = A \cdot x\) with \([Ax, A^2x, \ldots, A^kx]\)
- **Parallel Algorithm**

- Example: A tridiagonal, \(n=32, k=3\)
- Each processor works on (overlapping) trapezoid
- Saves latency (# of messages); Not bandwidth

But adds redundant computation
Matrix Powers Kernel on a General Matrix

• Saves communication for “well partitioned” matrices
  • Serial: $O(1)$ moves of data moves vs. $O(k)$
  • Parallel: $O(\log p)$ messages vs. $O(k \log p)$

For implicit memory management (caches) uses a TSP algorithm for layout

Joint work with Jim Demmel, Mark Hoemman, Marghoob Mohiyuddin
$A^k x$ has higher performance than $Ax$

Speedups on Intel Clovertown (8 core)
Minimizing Communication of GMRES to solve $Ax=b$

- **GMRES**: find $x$ in $\text{span}\{b, Ab, \ldots, A^k b\}$ minimizing $\|Ax-b\|_2$

**Standard GMRES**

for $i=1$ to $k$

$w = A \cdot v(i-1)$  ...  $\text{SpMV}$

$\text{MGS}(w, v(0), \ldots, v(i-1))$

update $v(i)$, $H$

endfor

solve LSQ problem with $H$

**Communication-avoiding GMRES**

$W = [v, Av, A^2 v, \ldots, A^k v]$

$[Q, R] = \text{TSQR}(W)$

... "Tall Skinny QR"

build $H$ from $R$

solve LSQ problem with $H$

Sequential case: #words moved decreases by a factor of $k$

Parallel case: #messages decreases by a factor of $k$

- **Oops** – $W$ from power method, precision lost!
TSQR: An Architecture-Dependent Algorithm

Parallel: \( W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow \begin{bmatrix} R_{00} \\ R_{10} \\ R_{20} \\ R_{30} \end{bmatrix} \rightarrow \begin{bmatrix} R_{01} \\ R_{11} \end{bmatrix} \rightarrow R_{02} \)

Sequential: \( W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow \begin{bmatrix} R_{00} \\ R_{01} \end{bmatrix} \rightarrow \begin{bmatrix} R_{02} \\ R_{03} \end{bmatrix} \)

Dual Core: \( W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow \begin{bmatrix} R_{00} \\ R_{01} \\ R_{11} \end{bmatrix} \rightarrow \begin{bmatrix} R_{02} \\ R_{03} \end{bmatrix} \)

Work by Laura Grigori, Jim Demmel, Mark Hoemmen, Julien Langou

Multicore / Multisocket / Multirack / Multisite / Out-of-core: ?
Can choose reduction tree dynamically
Matrix Powers Kernel (and TSQR) in GMRES

Jim Demmel, Mark Hoemmen, Marghoob Mohiyuddin, Kathy Yelick
Communication-Avoiding Krylov Method (GMRES)

Performance on 8 core Clovertown

Runtime per kernel, relative to CA-GMRES(k,t), for all test matrices, using 8 threads and restart length 60

Relative runtime, for best (k,t), with floor(restart length / k) = t

Sparse matrix name

k=5
2.3x
k=5
2.1x
k=5
1.7x
k=5
2.1x
k=5
4.3x
k=5
1.7x
k=4
1.6x

Matrix powers
kernel
TSQR
Block Gram-Schmidt
Small dense operations
Sparse matrix-vector product
Modified Gram-Schmidt
Can we do better? How do we know?

Lower bounds, (matching) upper bounds (algorithms) and a question:

Can we train compilers to do this?

See: http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-61.pdf
Beyond Domain Decomposition
2.5D Matrix Multiply on BG/P, 16K nodes / 64K cores

\[ c = 16 \text{ copies} \]
Matrix multiplication on 16,384 nodes of BG/P

**Surprises:**
- Even Matrix Multiply had room for improvement
- Idea: make copies of C matrix (as in prior 3D algorithm, but not as many)
- Result is provably optimal in communication

**Lesson:** Never waste fast memory

**Can we generalize for compiler writers?**

*EuroPar’11 (Solomonik, Demmel)*
*SC’11 paper (Solomonik, B hatele, Demmel)*
Towards Communication-Avoiding Compilers: Deconstructing 2.5D Matrix Multiply

Tiling the iteration space
- Compute a subcube
- Will need data on faces (projection of cube, subarrays)
  - For s loops in the nest $\Rightarrow$ s dimensional space
- For x dimensional arrays, project to x dim space

Matrix Multiplication code has a 3D iteration space
Each unit cube in the space is a constant computation ($*/+$)

for i
  for j
    for k
      $C[i,j] \ldots A[i,k] \ldots B[k,j] \ldots$
Deconstructing 2.5D Matrix Multiply
Solomonik & Demmel

Tiling in the k dimension
- k loop has dependencies because C (on the top) is a Left-Hand-Side variable
  \[ C += .. \]
- Advantages to tiling in k:
  - More parallelism \( \rightarrow \) Less synchronization
  - Less communication

What happens to these dependencies?
- All dependencies are vertical k dim (updating C matrix)
- Serial case: compute vertical block column in order
- Parallel case:
  - 2D algorithm (and compilers): never chop k dim
  - 2.5 or 3D: Assume + is associative; chop k, which implies replication of C matrix
Short Digression on Legality

for $k = 1$ to $n$

$C(i,j) = C(i,j) + A(i,k) \times B(k,j)$

- The $k$ dimension is a dot product with the $+= $ as the “interesting” operation
- If we assume (which is not strictly true for floating point) that $+$ is associate
  - We can use a tree reduction for $+$
  - It can be deterministic (even with fp) if we use the same tree, independent of the number of processors (even on 1)
  - Any tree / serial combination may be OK in some settings
- If we assume $+$ is commutative (also not true for fp)
  - We can do “atomic” asynchronous updates

(Harsha’s hyperedges were these set of vertices)
Beyond Domain Decomposition

• Much of the work on compilers is based on owner-computes
  – For MM: Divide C into chunks, schedule movement of A/B
  – Data-driven domain decomposition partitions data; but we can partition work instead

• Ways to compute C “pencil”
  1. Serially
  2. Parallel reduction *Standard vectorization trick*
  3. Parallel asynchronous (atomic) updates
  4. Or any hybrid of these

• For what types / operators does this work?
  – “+” is associative for 1,2 rest of RHS is “simple”
  – and commutative for 3

Using x for C[i,j] here
Lower Bound Idea on $C = A*B$

Iromy, Toledo, Tiskin

"Unit cubes" in black box with side lengths $x$, $y$ and $z$

= Volume of black box

= $x*y*z$

= $(\#A\square s \times \#B\square s \times \#C\square s)^{1/2}$

= $(xz * zy * yx)^{1/2}$

(i,k) is in "A shadow" if (i,j,k) in 3D set

(j,k) is in "B shadow" if (i,j,k) in 3D set

(i,j) is in "C shadow" if (i,j,k) in 3D set

Thm (Loomis & Whitney, 1949)

# cubes in 3D set = Volume of 3D set

$\leq (\text{area}(A\ shadow) \times \text{area}(B\ shadow) \times \text{area}(C\ shadow))^{1/2}$
Lower Bound: What is the minimum amount of communication required?


- Assume fast memory of size M

- Outline (big-O reasoning):
  - Segment instruction stream, each with M loads/stores
  - Somehow bound the maximum number of flops that can be done in each segment, call it F
  - So \( F \cdot \# \text{segments} \geq T = \text{total flops} = 2 \cdot n^3 \), so \( \# \text{segments} \geq T / F \)
  - So \( \# \text{loads & stores} = M \cdot \# \text{segments} \geq M \cdot T / F \)

- How much work (F) can we do with \( O(M) \) data?
Recall optimal sequential Matmul

• Naïve code
  for i=1:n, for j=1:n, for k=1:n, C(i,j)+=A(i,k)*B(k,j)

• “Blocked” code
  for i1 = 1:b:n, for j1 = 1:b:n, for k1 = 1:b:n
  for i2 = 0:b-1, for j2 = 0:b-1, for k2 = 0:b-1
  i=i1+i2, j = j1+j2, k = k1+k2
  C(i,j)+=A(i,k)*B(k,j)

• Thm: Picking b = M^{1/2} attains lower bound:
  #words_moved = \Omega(n^3/M^{1/2})

• Memory-constrained replication (compare to 3D)
• Where does 1/2 come from? Can we compute these for arbitrary programs?
Generalizing Communication Lower Bounds and Optimal Algorithms

• For serial matmul, we know \( \#\text{words\_moved} = \Omega \left( \frac{n^3}{M^{1/2}} \right) \), attained by tile sizes \( M^{1/2} \times M^{1/2} \).

• Thm 1: (Christ, Demmel, Knight, Scanlon, Yelick): For any program that “smells like” nested loops, accessing arrays with subscripts that are linear functions of the loop indices

\[
\#\text{words\_moved} = \Omega \left( \frac{\#\text{iterations}}{M^e} \right)
\]

for some \( e \) we can determine

• Thm 2: (C/D/K/S/Y): Under some assumptions, we can determine the optimal tiles sizes
  – E.g., index expressions are just subsets of indices

• Long term goal: All compilers should generate communication optimal code from nested loops
Lower Bounds on Communication

**Discrete HBL Linear Program (D-HBL-LP):**
for all subgroups $H \leq \mathbb{Z}^d$, $\text{rank}(H) \leq \sum_{i=1}^{m} s_i \cdot \text{rank}(\phi_i(H))$

Note: There exist infinitely many $H$, but only finitely many possible constraints in D-HBL-LP (at most $(d+1)^{m+1}$)

**Thm (B/C/C/T):** $s_i \geq 0$ satisfy D-HBL-LP if and only if for any finite set $E \subset \mathbb{Z}^d$ its cardinality $|E|$ is bounded by

$$|E| \leq \prod_{i=1}^{m} |\phi_i(E)|^{s_i} \quad \ldots \quad C = 1!$$

We want tightest bound when $|\phi_i(E)| \leq 2M$, i.e. $|E| \leq (2M)^{\sum_{i=1}^{m} s_i}$

$\implies$ Compute $s_{HBL} \equiv \min \sum_{i=1}^{m} s_i$ subject to D-HBL-LP

**Thm:** \#words\_moved $= \Omega(\#\text{iterations}/M^{s_{HBL}-1})$
How general is this?

- General model:

\[
\text{for all } \mathcal{I} \in \mathcal{Z} \subset \mathbb{Z}^d, \text{ in some order}
\]
\[
\text{inner\_loop}(\mathcal{I}, A_1(\phi_1(\mathcal{I})), \ldots, A_m(\phi_m(\mathcal{I})))
\]

- Ex: LU inner loop: \( A(i, j) = A(i, j) - L(i, k) \times U(k, j) \)
  - Ok to ignore loop scaling columns of \( L \)
  - Ok to overwrite \( A \): \( L(i, k) = A(i, k) \) for \( i > k \), ditto for \( U \)
  - Same idea applies to BLAS, Cholesky, \( LDL^T \), ...
  - Same idea applies to tensor contractions
  - QR, eig, SVD need another idea
Decidability of the Lower Bound

- What about Discrete HBL-LP?
  \[ \forall H \leq \mathbb{Z}^d, \text{rank}(H) \leq \sum_{i=1}^{m} s_i \cdot \text{rank}(\phi_i(H)) \]

- Constraints define polytope \( \mathcal{P} \) in space of \([s_1, \ldots, s_m] \in \mathbb{R}^m\)

- Enough to get any subset of subgroups \( H \) defining \( \mathcal{P} \)

- Let \((H_1, H_2, H_3, \ldots) \) be any enumeration of all \( H \leq \mathbb{Z}^d \)

- Let \( \mathcal{P}_i \) be polytope defined by \((H_1, \ldots, H_i)\)

- “Simple” decidability algorithm:
  \[ i = 0, \text{repeat } i = i + 1 \text{ until } \mathcal{P}_i = \mathcal{P} \]

- Thm: Decidable whether a vertex of \( \mathcal{P}_i \) in \( \mathcal{P} \)
  - Similar induction idea as before

- Better algorithm: which subgroups \( H \) to try first?
Special case: Subsets of Indices

- $i_1, \ldots, i_d$ be indices, $\phi_1, \ldots, \phi_m$ be projections
- Let $\Delta_{j,k} = 1$ if $i_k$ in range of $\phi_j$, else 0
- Thm: Let $s = [s_1, \ldots, s_m]$ minimize $1^T s \equiv s_{HBL}$ such that $s^T \Delta \geq 1^T$. Then
  \[
  \#\text{words\_moved} = \Omega(\#\text{loop\_iterations}/M^{s_{HBL}^{-1}})
  \]
- Proof idea
  - Constraints $s^T \Delta \geq 1$ are subset of Discrete HBL-LP, for all $H$ spanned by $(0, \ldots, 0, 1, 0, \ldots, 0)$ ($k$-th entry = 1)
  - Show this subset implies $\text{rank}(H) \leq \sum_{j=1}^{m} s_j \text{rank}(\phi_j(H))$ for all $H \leq \mathbb{Z}^d$
Upper Bound for Subset of Indices Case

• $i_1, \ldots, i_d$ be indices, $\phi_1, \ldots, \phi_m$ be projections

• Let $\Delta_{j,k} = 1$ if $i_k$ in range of $\phi_j$, else 0

• Dual LP: Let $x = [x_1, \ldots, x_d]$ maximize $1^T x \equiv s_{HBL}$ such that $\Delta x \leq 1^T$.

• Thm: The solution $x$ of the Dual LP gives the optimal block sizes to minimize communication: $i_k$ blocked by $M^{x_k}$

• Proof idea

  – Each constraint in $\Delta x \leq 1$ bounds number of entries of each array by $M$

  – $1^T x = s_{HBL}$ says number of inner loop iterations per block is $M^{s_{HBL}}$.

• Extends to parallel case, “n.5D” algorithms
New Theorem 2 applied to Matmul

- for i=1:n, for j=1:n, for k=1:n, C(i,j) += A(i,k)*B(k,j)
- Record array indices in matrix Δ

\[
\Delta = \begin{pmatrix}
i & j & k \\
1 & 0 & 1 \\
0 & 1 & 1 \\
1 & 1 & 0 \\
\end{pmatrix}
\]

\[
\Delta x \leq 1
\]

Solve LP for \(x = [x_i, x_j, x_k]^T\): max \(1^T x\) s.t. \(\Delta x \leq 1\)
- Result: \(x = [1/2, 1/2, 1/2]^T\), \(1^T x = 3/2 = s_{HBL}\)
- Thm: 
  \#words\_moved = \(\Omega(n^3/M_{s_{HBL}}^{-1}) = \Omega(n^3/M^{1/2})\)
- Attained by block sizes \(M^{x_i}, M^{x_j}, M^{x_k} = M^{1/2}, M^{1/2}, M^{1/2}\)
What does this tell us?

Lower bound attained by block sizes

\[ M^{x_i}, M^{x_j}, M^{x_k} = M^{1/2}, M^{1/2}, M^{1/2} \]

- Need to cut in all three dimensions for optimality: not “owner-computes” aka 2D
  - This is the 3D algorithm (if \( M \) is large enough) or 2.5D otherwise

- Tile shape is roughly a cube (within constants)

- Scales with cache / memory size

- This works for parallel code
New Theorem applied to Direct N-Body

- for \( i=1:n, \) for \( j=1:n, \) \( F(i) += \) force( \( P(i), P(j) \) )
- Record array indices in matrix \( \Delta \)

\[
\Delta = \begin{pmatrix}
i & j \\
1 & 0 & F \\
1 & 0 & P(i) \\
0 & 1 & P(j)
\end{pmatrix}
\]

- Solve LP for \( x = [x_i, x_j]^T: \) max \( 1^T x \) s.t. \( \Delta x \leq 1 \)
  - Result: \( x = [1, 1], \) \( 1^T x = 2 = s_{HBL} \)
- Thm: \#words\_moved = \( \Omega(n^2/M^{S_{HBL}-1}) = \Omega(n^2/M^1) \)
  Attained by block sizes \( M^{x_i}, M^{x_j} = M^1, M^1 \)
**Generalizing Communication Optimal Transformations to Arbitrary Loop Nests**

The same idea (replicate and reduce) can be used on (direct) N-Body code:

1D decomposition $\rightarrow$ “1.5D”

### Does this work in general?
- Yes, for certain loops and array expressions
- Relies on basic result in group theory
- Compiler work TBD

### Speedup of 1.5D N-Body over 1D

<table>
<thead>
<tr>
<th># of cores</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>32K</td>
<td>2.0x</td>
</tr>
<tr>
<td>24K</td>
<td>1.7x</td>
</tr>
<tr>
<td>8K</td>
<td>1.8x</td>
</tr>
<tr>
<td>6K</td>
<td>3.7x</td>
</tr>
</tbody>
</table>

*IPDPS’13 paper (Driscoll, Georganas, Koanantakool, Solomonik, Yelick)*
N-Body Speedups on IBM-BG/P (Intrepid)
8K cores, 32K particles

K. Yelick, E. Georganas, M. Driscoll, P. Koanantakool, E. Solomonik

Execution Time vs. Replication Factor

- Blue: Communication (Reduce)
- Green: Communication (Shift)
- Red: Computation

Execution Time Per Timestep (sec)

Replication Factor

11.8x speedup
New Theorem applied to Random Code

- for $i_1=1:n$, for $i_2=1:n$, ... , for $i_6=1:n$
  $A_1(i_1,i_3,i_6) \leftarrow \text{func1}(A_2(i_1,i_2,i_4),A_3(i_2,i_3,i_5),A_4(i_3,i_4,i_6))$
  $A_5(i_2,i_6) \leftarrow \text{func2}(A_6(i_1,i_4,i_5),A_3(i_3,i_4,i_6))$

- Record array indices in matrix $\Delta$

- Solve LP for $x = [x_1, ..., x_7]^T$: max $1^T x$ s.t. $\Delta x \leq 1$
  - Result: $x = [2/7, 3/7, 1/7, 2/7, 3/7, 4/7]$, $1^T x = 15/7 = s_{HBL}$

- Thm: $\#\text{words}_\text{moved} = \Omega(n^6/M_{SHBL}^{-1}) = \Omega(n^6/M^{8/7})$
  Attained by block sizes $M^{2/7}, M^{3/7}, M^{1/7}, M^{2/7}, M^{3/7}, M^{4/7}$
General Communication Bound

• Given $S$ subset of $\mathbb{Z}^k$, group homomorphisms $\phi_1, \phi_2, \ldots$, bound $|S|$ in terms of $|\phi_1(S)|$, $|\phi_2(S)|$, $\ldots$, $|\phi_m(S)|$

• Def: Hölder-Brascamp-Lieb LP (HBL-LP) for $s_1, \ldots, s_m$:
  for all subgroups $H < \mathbb{Z}^k$, $\text{rank}(H) \leq \sum_j s_j \cdot \text{rank}(\phi_j(H))$

• Thm (Christ/Tao/Carbery/Bennett): Given $s_1, \ldots, s_m$
  \[ |S| \leq \prod_j |\phi_j(S)|^{s_j} \]

• Thm: Given a program with array refs given by $\phi_j$, choose $s_j$ to minimize $s_{\text{HBL}} = \sum_j s_j$ subject to HBL-LP. Then
  \[ \#\text{words}_{\text{moved}} = \Omega (\#\text{iterations} / M^{s_{\text{HBL}}^{-1}}) \]
Comments

• Thm: (bad news) HBL-LP reduces to Hilbert’s 10th problem over Q (conjectured to be undecidable)

• Thm: (good news) Another LP with same solution is decidable (but expensive, so far)

• Thm: (better news) Easy to write down LP explicitly in many cases of interest (eg all $\varphi_j = \{\text{subset of indices}\})

• Thm: (good news) Easy to approximate, i.e. get upper or lower bounds on $s_{\text{HBL}}$
  
   • If you miss a constraint, the lower bound may be too large (i.e. $s_{\text{HBL}}$ too small) but still worth trying to attain
   
   • Tarski-decidable to get superset of constraints (may get $s_{\text{HBL}}$ too large)
Comments

• Attainability depends on loop dependencies
  Best case: none, or associative operators (matmul, nbody)

• Thm: When all $\phi_j = \{\text{subset of indices}\}$, dual of HBL-LP gives optimal tile sizes:
  
  HBL-LP: $\text{minimize } 1^T s \text{ s.t. } s^T \Delta \geq 1^T$
  
  Dual-HBL-LP: $\text{maximize } 1^T x \text{ s.t. } \Delta^* x \leq 1$

Then for sequential algorithm, tile $i_j$ by $M^{x_j}$

• Ex: Matmul: $s = [1/2, 1/2, 1/2]^T = x$

• Generality:
  – Extends to unimodular transforms of indices
  – Does not require arrays (as long as the data structures are injective containers)
  – Does not require loops as long as they can model computation
Stepping Back

• Communication avoidance as old at tiling
• Communication optimality as old as Hong/Kung
  – But many of those algorithms assume unlimited memory (3D Matmul, 2D N-body; rather than memory-constrained .5D)

• What’s new?
  – Raising the level of abstraction at which we optimize
  – BLAS2 → BLAS3 → LU or SPMV/DOT → Krylov
  – Changing numerics in non-trivial ways
  – Rethinking methods to models

• Communication and synchronization avoidance
• Software engineering: breaking abstraction
• Compilers: inter-procedural optimizations
Communication Optimization Summary

1. Compress Data Structures
2. Target Higher Level Loops
3. Understand theory / numerics
4. Replicate data
5. Understand theory / lower bounds
6. Aggregate communication
7. Overlap communication
8. Use one-sided communication
9. Synchronization strength reduction
10. Combine the techniques
Optimal Tiling for N-Body

- $M = \text{cache size (} \#\text{words)}$
- $b \times b = \text{block size}$

Unblocked (given) code:

```plaintext
for i = 1:N, for j = 1:N
    F(i) += force(P(i), P(j))
```

Blocked code (optimal)

- $b = O(M) \rightarrow O(M^2) \text{ reuse}$

```plaintext
for j_1 = 1:N/b, for j_2 = 1:N/b
    for k_1 = 1:b, for k_2 = 1:b
        (i, j) = b \cdot (j_1, j_2) + (k_1, k_2)
        F(i) += force(P(i), P(j))
```

b = 6
(Sub)optimal tiling for “Twisted” N-body

- $M =$ cache size (#words)
- $b \times b =$ block size

Unblocked (given) code:

```plaintext
for i = 1:N, for j = 1:N
    F(i+c\cdot j) += force(P(i+c\cdot j), P(i-c\cdot j))
```

Naively blocked code (suboptimal)

- $b = O(M/c)$ to fit $P, F$ in cache
- $\rightarrow O(M^2/c^2)$ reuse, not $O(M^2)$

```plaintext
for j_1 = 1:N/b, for j_2 = 1:N/b
    for k_1 = 1:b, for k_2 = 1:b
        (i, j) = b \cdot (j_1, j_2) + (k_1, k_2)
        F(i+c\cdot j) += force(P(i+c\cdot j), P(i-c\cdot j))
```

$c = 2, b = 3$
Optimal Tiling for “Twisted” N-Body

- $M =$ cache size (#words)
- $b \times b =$ block size

Unblocked (given) code:

```plaintext
for i = 1:N, for j = 1:N
    F(i+c\cdot j) += force(P(i+c\cdot j), P(i-c\cdot j))
```

Blocked code (optimal)

- $b = O(M) \to O(M^2) =$ reuse
  
  ```plaintext
  for j_1 = (*), for j_2 = (*)
      for t = 1:2 \cdot c
          for k_1 = 1:b, for k_2 = 1:b
              (i,j) = b \cdot (j_1, j_2) + (k_1, k_2)
              (m,n) = i \cdot (c,-1) + j \cdot (c,1) + t \cdot (1,0)
              F(m+c\cdot n) += force(P(m+c\cdot n), P(m-c\cdot n))
  ```

(*) denotes a subset of 1:N/b (ignore fringe cases)
$c = 2,$
$b = 6$

$t = 0, 1, 2, 3$