High-Level Semantic Optimization of Numerical Codes

Vijay Menon and Keshav Pingali

Cornell University

The MaJIC Project

at Illinois/Cornell

- George Almasi
- Luiz De Rose
- David Padua

Motivation

Scalar algebraic properties enable/enhance:

- Strength Reduction
- Common Subexpression Elimination
- Loop Tiling

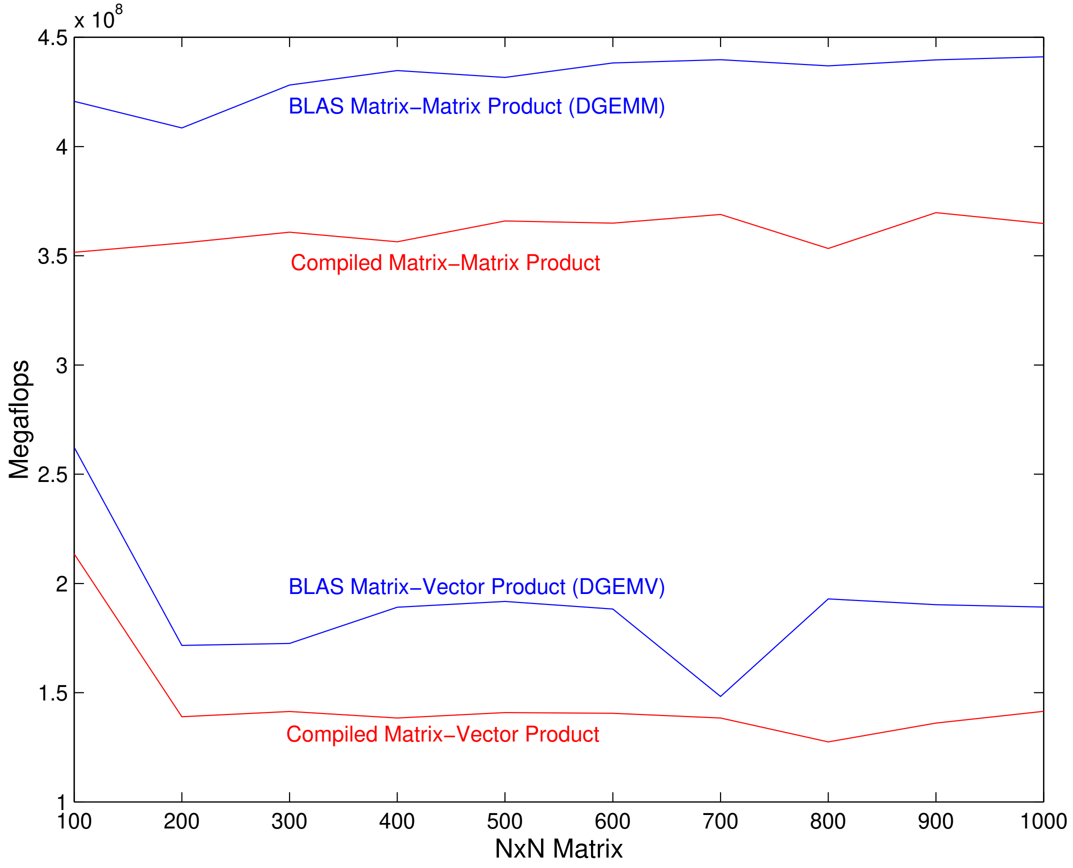
How to use *Matrix* properties in loop-based codes?

- Detect Matrix Operations
  - Map to hand-optimized libraries
  - Eliminate interpreter overhead
- Exploit Matrix Semantics
  - Improve asymptotic complexity

Detecting Matrix Operations: Mapping to Efficient Library Calls

Hand-optimized libraries outperform compiled-generated codes

Example: BLAS



Detection Matrix Operations: Elimination of Loops in Interpreted Codes

MATLAB : Interpreted numerical computing language

Code	Time (ms)

`for i = 1:100`
`  k = k + x(i)*y(i);`
`end`
	7.5

`k = k + x(1:100)'*y(1:100);`	2.0

Interpreted loops are slow: Type checks, Bounds checks, ...

Exploiting Matrix Semantics: Improving Asymptotic Complexity

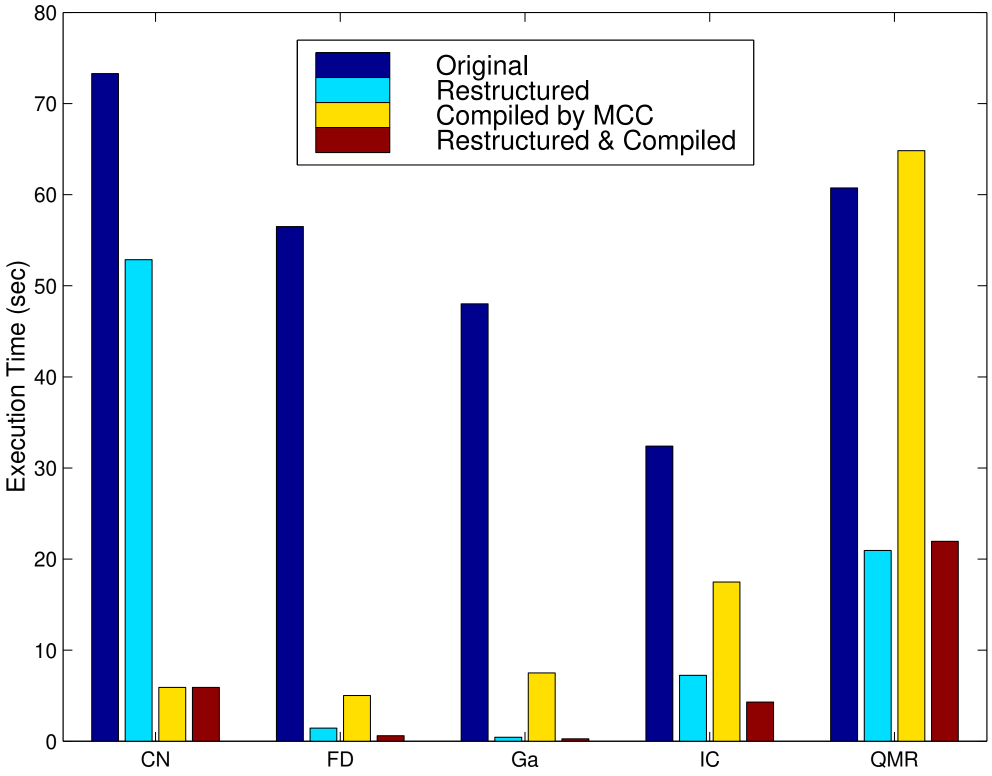
Semantically equivalent forms:	Floating Point Operations

1. `for i = 1 : n`
`  for j = 1 : n`
`    for k = 1 : n`
`        y(i) = y(i) + x(k)*A(i,j)*B(k,j);`
	$3n^3$

2. `y = y + (A*B')*x;`	$2n^3$
3. `y = y + A*(B'*x);`	$4n^2$

Importance of Optimizations in Real Codes

- FALCON Benchmarks from University of Illinois



- MCC, MATCOM, etc., do not detect/exploit matrix operations

Utilizing Matrix Properties in Compilers

Summary of benefits:

+ Profit from hand-optimized libraries
+ Eliminate interpretation overheads
+ Improve asymptotic complexity

How do we automate this:

- Detect matrix operations in loops?
- Exploit matrix semantics?
  - Ring Properties of Matrix $(+, \times)$

Conventional Approaches

- Syntactic Pattern Replacement (KAPF, VAST, FALCON)
  - \+ Can encode high-level properties
  - \- Limited use on loops
- Vector Code Generation
  - \+ Can detect array operations in loops
  - \- Cannot detect/exploit matrix products

Our Approach

1. Map code to: *Abstract Matrix Form (AMF)*
2. Convert to *Symmetric* AMF formulation
3. Optimize AMF expression
   - Factorization
   - Invariant Elimination
4. Detect Matrix Products
5. Map AMF to MATLAB/BLAS

Abstract Matrix Form

AMF Expressions:

- s : scalar variable or constant
- a_{ij} : scalar value in array
- $f(e_1, e_2, \ldots, e_n)$: element-wise scalar operation

AMF Expansion

- $\underset{i \to \hat{i}}{\overset{l:u}{\forall}} e$: expansion of AMF expression e in $\hat{i}$ dimension

$$\underset{i \to \hat{i}}{\overset{1:3}{\forall}} \begin{pmatrix} i \\ 2i \\ 3i \end{pmatrix} = \begin{array}{c} \xrightarrow{\quad}{}^{\hat{i}} \\ \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{pmatrix} \end{array}$$

Note: $\forall$ maps an n dim. object into an $n+1$ dim. object

Examples of ∀-Expansion

- Array Expressions

$$\text{a(1:m,1:n)} \longrightarrow \overset{1:m}{\underset{i\to\hat{r}}{\forall}}\ \overset{1:n}{\underset{j\to\hat{c}}{\forall}}\ a_{i,j}$$

- Forall Loops

```
for i = 1:n
  x(i) = x(i) + y(i);
end
```

$$\longrightarrow \overset{1:n}{\underset{i\to\hat{i}}{\forall}}\ x_i = \overset{1:n}{\underset{i\to\hat{i}}{\forall}}(x_i + y_i)$$

AMF Reduction

- $\sum_{\hat{i}} e$: additive reduction of AMF expression e in $\hat{i}$ dimension

$$\sum_{\hat{i}} \begin{array}{c} \xrightarrow{\quad}{}^{\hat{i}} \\ \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{pmatrix} \end{array} = \begin{pmatrix} 6 \\ 12 \\ 18 \end{pmatrix}$$

Note: $\sum$ maps an n dim. object into an $n-1$ dim. object

Example of Σ-Reduction

- Sum Reduction Loops

```
for i = 1:n
  k = k + x(i);
end
```

$$\longrightarrow \quad k = k + \sum_{\hat{i}} \overset{1:n}{\underset{i \to \hat{i}}{\forall}} x_i$$

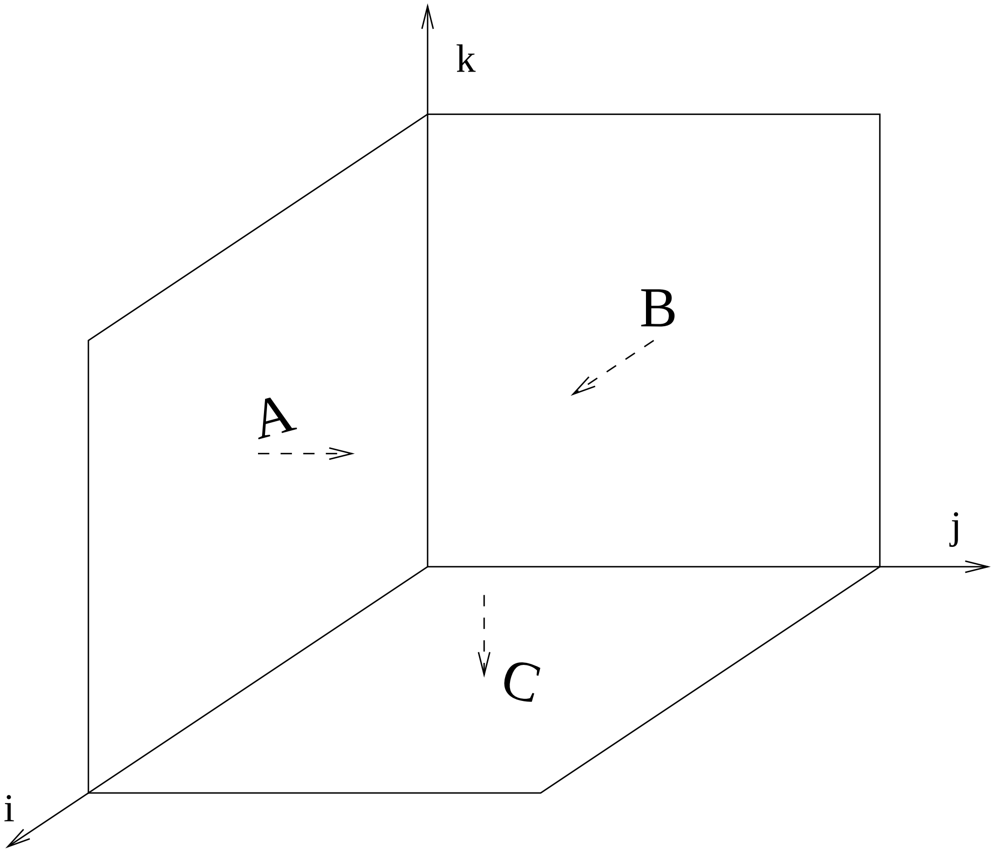
AMF Product

- $\underset{\hat{i}}{\mathrm{P}}(e_1, e_2)$: the product of AMF expressions e_1 and e_2

$$\underset{\hat{k}}{\mathrm{P}}\left(\begin{array}{cc} & \longrightarrow^{\hat{k}} \\ \downarrow_{\hat{i}} & \begin{pmatrix} 1 & 0 & -2 & 0 \\ 0 & 3 & 0 & -1 \end{pmatrix} \end{array}, \begin{array}{cc} & \longrightarrow^{\hat{k}} \\ \downarrow_{\hat{j}} & \begin{pmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 2 & 0 & 1 & -2 \end{pmatrix} \end{array} \right) =$$

$$\begin{array}{cc} & \longrightarrow^{\hat{j}} \\ \downarrow_{\hat{i}} & \begin{pmatrix} 1 & 0 & 0 \\ -3 & 1 & 2 \end{pmatrix} \end{array}$$

AMF Product: 3D View



- Matrix Product

`C = A*B` $\longrightarrow$ $\forall_i \forall_j C_{i,j} = \mathrm{P}_{\hat{k}}(\forall_i \forall_k A_{i,k}, \forall_j \forall_k B_{k,j})$

AMF Product-Reduction Equivalence

- Product-Reduction Equivalence Property:

$$\sum_{\hat{j}}(\forall_{i_1} \cdots \forall_{i_n} e_1 \cdot \forall_{i_1} \cdots \forall_{i_n} e_2) = \mathrm{P}_{\hat{j}}(\forall_{i_1} \cdots \forall_{i_k} e_1, \forall_{i_k} \cdots \forall_{i_n} e_2)$$

when

e_1 and e_2 are scalar

e_1 is constant w.r.t. $i_{k+1}...i_n$

e_2 is constant w.r.t. $i_1...i_{k-1}$

Other AMF Properties

- Interchange Properties
  - $\underset{i}{\forall} f(e_1, e_2, \ldots, e_n) = f(\underset{i}{\forall} e_1, \underset{i}{\forall} e_2, \ldots, \underset{i}{\forall} e_n)$
  - $\underset{i}{\forall} \underset{\hat{j}}{\sum} e = \underset{\hat{j}}{\sum} \underset{i}{\forall} e$ where $\hat{i} \neq \hat{j}$
  - $\underset{i}{\forall} \underset{j}{\forall} e = \underset{j}{\forall} \underset{i}{\forall} e$
  - $\underset{\hat{i}}{\sum} \underset{\hat{j}}{\sum} e = \underset{\hat{j}}{\sum} \underset{\hat{i}}{\sum} e$
- Distributive Property
  - $e_1 \cdot \underset{\hat{i}}{\sum} e_2 = \underset{\hat{i}}{\sum}((\underset{i}{\forall} e_1) \cdot e_2)$

    when e_1 is constant w.r.t. i

Working Example

Reconsider our earlier example::

```
for i = 1 : n
  for j = 1 : n
    for k = 1 : n
        y(i) = y(i) + x(k)*A(i,j)*B(k,j);
```

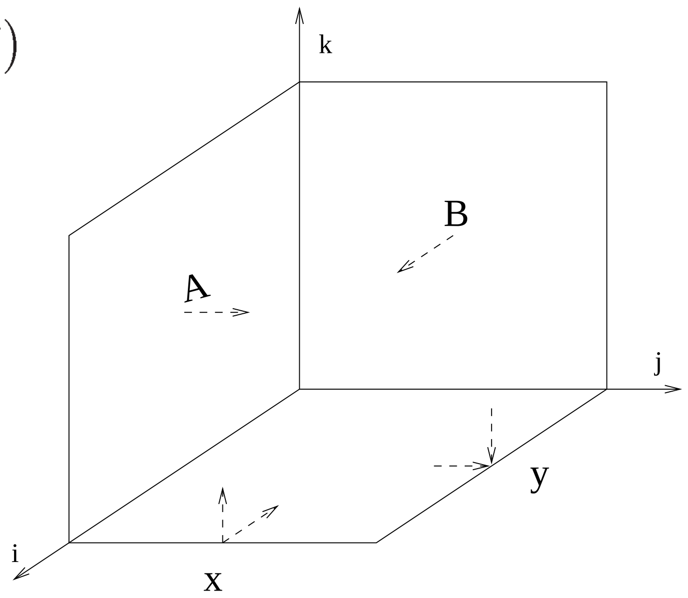
converts to:

$$\underset{i}{\forall} y_i = \underset{i}{\forall} y_i + \underset{i}{\forall} \sum_{\hat{j}} \underset{j}{\forall} \sum_{\hat{k}} \underset{k}{\forall} (x_k \cdot A_{i,j} \cdot B_{k,j})$$

Flop Count: $3n^3$ operations

Obtain Symmetric Formulation

- Sink Expansion Operations ($\forall$)



Our example converts to:

$$\underset{i}{\forall} y_i = \underset{i}{\forall} y_i + \underset{i}{\forall} \underset{\hat{j}}{\sum} \underset{j}{\forall} \underset{\hat{k}}{\sum} \underset{k}{\forall} (x_k \cdot A_{i,j} \cdot B_{k,j})$$

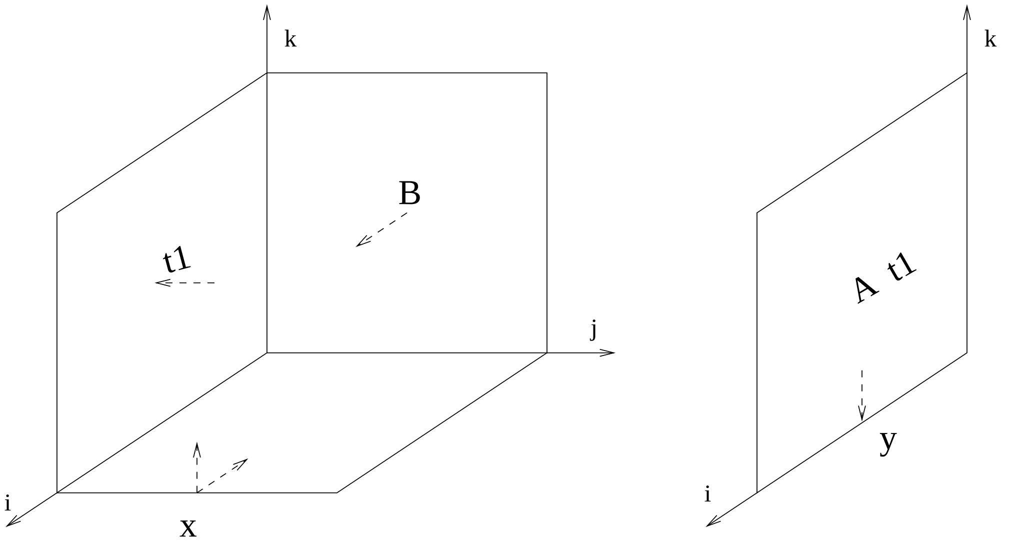
$$\downarrow$$

$$\underset{i}{\forall} y_i = \underset{i}{\forall} y_i + \underset{\hat{j}}{\sum} \underset{\hat{k}}{\sum} (\underset{i}{\forall} \underset{j}{\forall} \underset{k}{\forall} x_k \cdot \underset{i}{\forall} \underset{j}{\forall} \underset{k}{\forall} A_{i,j} \cdot \underset{i}{\forall} \underset{j}{\forall} \underset{k}{\forall} B_{k,j})$$

Flop Count: $3n^3$ operations

Optimization #1: Factorization

Use distributive property to decompose into 2 box products:



$$\underset{i}{\forall} y_i = \underset{i}{\forall} y_i + \sum_{\hat{j}} \sum_{\hat{k}} \left(\underset{i}{\forall} \underset{j}{\forall} \underset{k}{\forall} x_k \cdot \underset{i}{\forall} \underset{j}{\forall} \underset{k}{\forall} A_{i,j} \cdot \underset{i}{\forall} \underset{j}{\forall} \underset{k}{\forall} B_{k,j} \right)$$

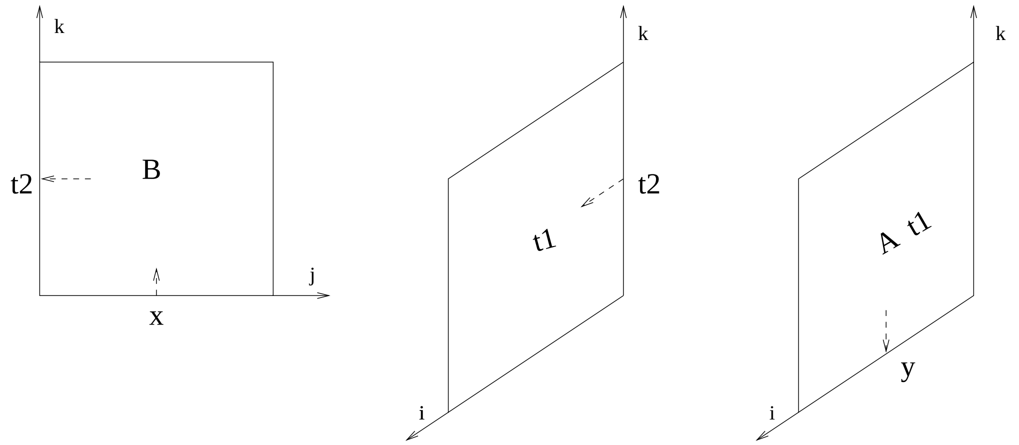
$$\downarrow$$

$$\underset{i}{\forall} y_i = \underset{i}{\forall} y_i + \sum_{\hat{j}} \left(\underset{i}{\forall} \underset{j}{\forall} A_{i,j} \right) \cdot \left(\sum_{\hat{k}} \left(\underset{i}{\forall} \underset{j}{\forall} \underset{k}{\forall} x_k \right) \cdot \left(\underset{i}{\forall} \underset{j}{\forall} \underset{k}{\forall} B_{k,j} \right) \right)$$

Flop Count: $2n^3$ operations

Optimization #2: Invariant Elimination

Hoist invariant expansion out of reduction:



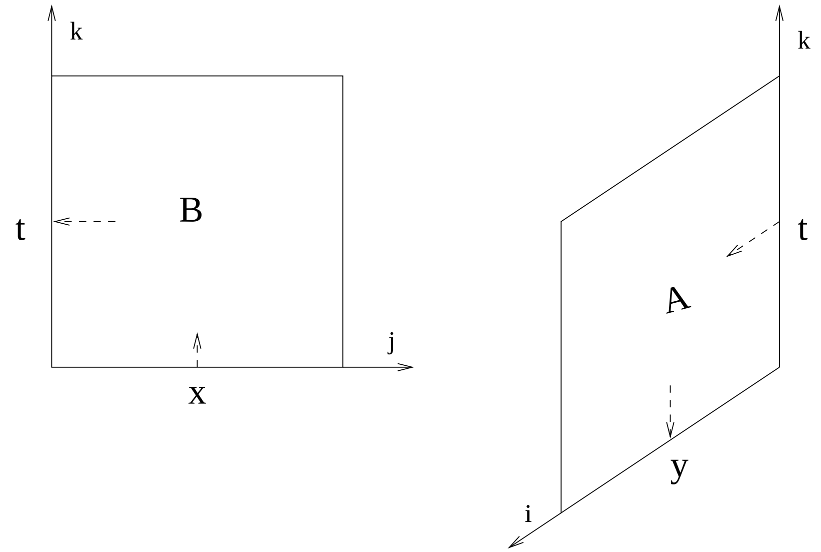
$$\underset{i}{\forall} y_i = \underset{i}{\forall} y_i + \underset{\hat{j}}{\sum}(\underset{i}{\forall}\underset{j}{\forall} A_{i,j}) \cdot (\underset{\hat{k}}{\sum}(\underset{i}{\forall}\underset{j}{\forall}\underset{k}{\forall} x_k) \cdot (\underset{i}{\forall}\underset{j}{\forall}\underset{k}{\forall} B_{k,j}))$$

$$\downarrow$$

$$\underset{i}{\forall} y_i = \underset{i}{\forall} y_i + \underset{\hat{j}}{\sum}(\underset{i}{\forall}\underset{j}{\forall} A_{i,j}) \cdot (\underset{i}{\forall}\underset{\hat{k}}{\sum}(\underset{j}{\forall}\underset{k}{\forall} x_k) \cdot (\underset{j}{\forall}\underset{k}{\forall} B_{k,j}))$$

Flop Count: $4n^2$ operations

Mapping to Matrix products



$$\underset{i}{\forall}\, y_i = \underset{i}{\forall}\, y_i + \sum_{\hat{j}} (\underset{i}{\forall}\,\underset{j}{\forall}\, A_{i,j}) \cdot \underset{i}{\forall} \sum_{\hat{k}} (\underset{j}{\forall}\,\underset{k}{\forall}\, x_k) \cdot (\underset{j}{\forall}\,\underset{k}{\forall}\, B_{k,j}))$$

$$\downarrow$$

$$\underset{i}{\forall}\, y_i = \underset{i}{\forall}\, y_i + \underset{\hat{j}}{\mathrm{P}} (\underset{i}{\forall}\,\underset{j}{\forall}\, A_{i,j}, \underset{\hat{k}}{\mathrm{P}} (\underset{k}{\forall}\, x_k, \underset{j}{\forall}\,\underset{k}{\forall}\, B_{k,j}))$$

$$\downarrow$$

```
y = y + A*(B'*x);
```

Example #2

- Taken from USENET:

```
for i = 1:n
  for j = 1:n
    C(i,j) = A(i,j) * x(i);
  end
end
```

$$\downarrow$$

```
C(1:n,1:n) = A(1:n,1:n) .* repmat(x(1:n),1,n);
```

Example #3

- Taken from USENET:

```
for i = 1:n
  x(i) = y(:,i)*A*y(:,i)';
end
```

$$\downarrow$$

```
x(1:n) = sum(y.*(y*A'),2);
```

Conclusions

- Compiler framework to detect/exploit:
  - Matrix Addition
  - Matrix Multiplication

  $\Big\rangle$ *Ring Properties of Matrix* $(+, \times)$

- Implemented in MaJIC system

- Future Work:
  - Loops with dependences
    - Matrix Factorizations
    - Recurrences
  - Further matrix properties
    - Symmetry
    - Orthogonality