On Characterizing the Data Access Complexity of Programs

Venmugil Elango\textsuperscript{1}  Fabrice Rastello\textsuperscript{2}  Louis-Noël Pouchet\textsuperscript{1}  J. Ramanujam\textsuperscript{3}  P. Sadayappan\textsuperscript{1}

\textsuperscript{1} The Ohio State University  \textsuperscript{2} Inria  \textsuperscript{3} Louisiana State University

POPL 2015: 42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages
Outline

1. Motivation
2. Prior work & Challenges
3. Static analysis of affine programs
Outline

1 Motivation

2 Prior work & Challenges

3 Static analysis of affine programs
What is a Good Algorithm?

- Computational cost: number of operations executed by the algorithm
  - Objective: reduce the operation complexity

- Execution time: time to execute the operations

- Actually, time to execute the operations **and** time to move the operands in the system
  - Example: moving data from disk to RAM at 3Gb/s
  - Example: moving data from RAM to CPU at 17Gb/s
  - ...
What is a Good Algorithm?

- Computational cost: number of operations executed by the algorithm
  - Objective: reduce the operation complexity

- Execution time: time to execute the operations
- Actually, time to execute the operations and time to move the operands in the system
  - Example: moving data from disk to RAM at 3Gb/s
  - Example: moving data from RAM to CPU at 17Gb/s
  - ...

⇒ Good algorithm: good execution time (computation + data movement)
A Look at Architectural Trends

- The relative cost of data movement vs. computation keeps increasing
  - Ex: Intel 80286: 2 MIPS, 13 MB/s for transfer RAM->CPU
  - Ex: Intel core i7: 50,000 MIPS, 16,000 MB/s for transfer RAM->CPU
- The relative energy cost of data movement vs. computation keeps increasing

Source: Jim Demmel, John Shalf

- Computational complexity alone is not sufficient. Data movement complexity matters!
A Look at Architectural Trends

- The relative cost of data movement vs. computation keeps increasing
  - Ex: Intel 80286: 2 MIPS, 13 MB/s for transfer RAM->CPU
  - Ex: Intel core i7: 50,000 MIPS, 16,000 MB/s for transfer RAM->CPU
- The relative energy cost of data movement vs. computation keeps increasing

Computational complexity alone is not sufficient. Data movement complexity matters!

Source: Jim Demmel, John Shalf
Motivation:

Data Movement vs. Computational Complexity

Untiled

\[
\begin{align*}
&\text{for}(i=1; \ i<N-1; \ i++) \\
&\quad \text{for}(j=1; \ j<N-1; \ j++) \\
&\quad \quad A[i][j] = A[i][j-1] \\
&\quad \quad + A[i-1][j];
\end{align*}
\]

Comp. cost: \((N - 1)^2\)

Tiled

\[
\begin{align*}
&\text{for}(\it=1; \ \it<N-1; \ \it+=B) \\
&\quad \text{for}(\jt=1; \ \jt<N-1; \ \jt+=B) \\
&\quad \quad \text{for}(i=\it; \ i<\min(\it+B,N-1); \ i++) \\
&\quad \quad \quad \text{for}(j=\jt; \ j<\min(\jt+B,N-1); \ j++) \\
&\quad \quad \quad \quad A[i][j] = A[i][j-1] + A[i-1][j];
\end{align*}
\]

Comp. cost: \((N - 1)^2\)

- Data movement cost different for two versions
- Also depends on cache size
- Question: What is data movement complexity?

Data movement complexity: Minimum data movement cost considering all possible valid schedules
**Data Movement vs. Computational Complexity**

**Untiled**

```plaintext
for(i=1; i<N-1; i++)
    for(j=1; j<N-1; j++)
```

Comp. cost: \((N - 1)^2\)

**Tiled**

```plaintext
for(it=1; it<N-1; it+=B)
    for(jt=1; jt<N-1; jt+=B)
        for(i=it; i<min(it+B,N-1); i++)
            for(j=jt; j<min(jt+B,N-1); j++)
```

Comp. cost: \((N - 1)^2\)

- Data movement cost different for two versions
- Also depends on cache size
- Question: What is data movement complexity?

**Data movement complexity:** Minimum data movement cost considering all possible valid schedules
**Data Movement vs. Computational Complexity**

**Untiled**

```plaintext
definition
for(i=1; i<N-1; i++)
    for(j=1; j<N-1; j++)
```

Comp. cost: $\left( N - 1 \right)^2$

**Tiled**

```plaintext
definition
for(it=1; it<N-1; it+=B)
    for(jt=1; jt<N-1; jt+=B)
        for(i=it; i<min(it+B,N-1); i++)
            for(j=jt; j<min(jt+B,N-1); j++)
```

Comp. cost: $\left( N - 1 \right)^2$

**Data movement cost**

- Data movement cost different for two versions
- Also depends on cache size
- Question: What is data movement complexity?
  - Data movement complexity: Minimum data movement cost considering all possible valid schedules
Motivation:

**Data Movement vs. Computational Complexity**

**Untiled**
```
for(i=1; i<N-1; i++)
  for(j=1; j<N-1; j++)
    A[i][j] = A[i][j-1]
      + A[i-1][j];
```

Comp. cost: \((N-1)^2\)

**Data movement cost**

- Data movement cost different for two versions
- Also depends on cache size
- **Question**: What is data movement complexity?

**Tiled**
```
for(it=1; it<N-1; it+=B)
  for(jt=1; jt<N-1; jt+=B)
    for(i=it; i<min(it+B,N-1); i++)
      for(j=jt; j<min(jt+B,N-1); j++)
```

Comp. cost: \((N-1)^2\)

- Data movement complexity: Minimum data movement cost considering all possible valid schedules
Data Movement vs. Computational Complexity

Untiled

\[
\begin{align*}
\text{for}(i=1; i<N-1; i++) \\
\quad \text{for}(j=1; j<N-1; j++) \\
\quad \quad A[i][j] &= A[i][j-1] \\
\quad \quad &+ A[i-1][j];
\end{align*}
\]

Comp. cost: \((N - 1)^2\)

Data movement cost

Tiled

\[
\begin{align*}
\text{for}(it=1; it<N-1; it+=B) \\
\quad \text{for}(jt=1; jt<N-1; jt+=B) \\
\quad \quad \text{for}(i=it; i<\text{min}(it+B,N-1); i++) \\
\quad \quad \quad \text{for}(j=jt; j<\text{min}(jt+B,N-1); j++) \\
\quad \quad \quad \quad A[i][j] &= A[i][j-1] + A[i-1][j];
\end{align*}
\]

Comp. cost: \((N - 1)^2\)

- Data movement cost different for two versions
- Also depends on cache size
- Question: What is data movement complexity?

Data movement complexity: Minimum data movement cost considering all possible valid schedules
Outline

1 Motivation

2 Prior work & Challenges

3 Static analysis of affine programs
**Prior work & Challenges:**

**Graph based**
- Arbitrary CDAGs
- [Hong and Kung, 1981]: all valid schedules $\leadsto$ all valid “2S-partitions” of CDAG
- (+) Generality
- (-) manual reasoning $\implies$ challenge to automate

**Geometric data footprint**
- Linear algebra like algorithms
- [Irony et al., 2004], [Ballard et al., 2011]: Geom. approach based on Loomis-Whitney (LW) inequality
- [Christ et al., 2013]: Automation based on Holder-Brascamp-Leib (HBL) ineq.
- (+) Automated
- (-) Restricted model $\implies$ weakness of bounds or inapplicability
Prior work & Challenges:

Graph based

- Arbitrary CDAGs

[Hong and Kung, 1981]: all valid schedules \(\rightsquigarrow\) all valid “2S-partitions” of CDAG

- (+) Generality

- (-) manual reasoning \(\Rightarrow\) challenge to automate

Geometric data footprint

- Linear algebra like algorithms

- [Irony et al., 2004], [Ballard et al., 2011]: Geom. approach based on Loomis-Whitney (LW) inequality

- [Christ et al., 2013]: Automation based on Holder-Brascamp-Leib (HBL) ineq.

- (+) Automated

- (-) Restricted model \(\Rightarrow\) weakness of bounds or inapplicability
Prior work & Challenges:

**Graph based**

- Arbitrary CDAGs

- [Hong and Kung, 1981]: all valid schedules $\rightsquigarrow$ all valid “2S-partitions” of CDAG

- (+) Generality

- (-) manual reasoning $\implies$ challenge to automate

**Geometric data footprint**

- Linear algebra like algorithms

- [Iryon et al., 2004], [Ballard et al., 2011]: Geom. approach based on Loomis-Whitney (LW) inequality

- [Christ et al., 2013]: Automation based on Holder-Brascamp-Leib (HBL) ineq.

- (+) Automated

- (-) Restricted model $\implies$ weakness of bounds or inapplicability
Our work: Static analysis to automate asymptotic parametric lower bounds analysis of affine codes for CDAG model.
Loomis-Whitney inequality

- \( E \subset \mathbb{R}^d \)
- \( \phi_1(E), \ldots, \phi_d(E) \) its projections on the coordinates hyperplanes

Example \((d = 3)\):

\[ |E| \leq |\phi_1(E)|^{1/2} \times |\phi_2(E)|^{1/2} \times |\phi_3(E)|^{1/2} \]

\[ |E| \leq |\phi_1(E)|^{1/(d-1)} \times \cdots \times |\phi_d(E)|^{1/(d-1)} \]
Loomis-Whitney inequality

- $E \subset \mathbb{R}^d$
- $\phi_1(E), \ldots, \phi_d(E)$ its projections on the coordinates hyperplanes

Example ($d = 3$):

$$|E| \leq |\phi_1(E)|^{1/2} \times |\phi_2(E)|^{1/2} \times |\phi_3(E)|^{1/2}$$

$$|E| \leq |\phi_1(E)|^{1/(d-1)} \times \cdots \times |\phi_d(E)|^{1/(d-1)}$$
Prior work & Challenges:

Hong & Kung 2S-partitioning

Any valid schedule is associated with a 2S-partition

S-partition

Collection of $h$ subsets ($V_1, \ldots, V_h$) of $V \setminus I$ s.t:

- **P1** pairwise disjoint
- **P2** no cyclic dependence
- **P3** $\forall i$, $|\text{In}(V_i)| \leq S$

Largest vertex-set: $P$

Data movement complexity

$$Q \geq \left(\frac{|V|}{|P|} - 1\right) \times S$$
Hong & Kung $2S$-partitioning

- Any valid schedule is associated with a $2S$-partition

$S$-partition
Collection of $h$ subsets $(V_1, \ldots, V_h)$ of $V \setminus I$ s.t:

- $P1$ pairwise disjoint
- $P2$ no cyclic dependence
- $P3$ $\forall i, |\text{In}(V_i)| \leq S$

- Largest vertex-set: $P$

Data movement complexity

$$Q \geq \left( \frac{|V|}{|P|} - 1 \right) \times S$$
Outline

1 Motivation

2 Prior work & Challenges

3 Static analysis of affine programs
Affine computations

Can be represented as (union of) \( \mathbb{Z} \)-polyhedra:

- **Space**: \( d \)-dimensional integer lattice (\( \mathbb{Z}^d \)).
- **Points**: Each instance of the statement.
- **Arrows**: True data dependencies.

```plaintext
for (i=0; i<N; i++)
S1: A[i] = I[i];
for (t=1; t<T; t++)
{
    for (i=1; i<N-1; i++)
    for (i=1; i<N-1; i++)
S3: A[i] = B[i];
}
```

⇒ Apply geometric reasoning on \( \mathbb{Z} \)-polyhedra to bound \(|P|\)
Affine computations

Can be represented as (union of) $\mathbb{Z}$-polyhedra:

- **Space**: $d$-dimensional integer lattice ($\mathbb{Z}^d$).
- **Points**: Each instance of the statement.
- **Arrows**: True data dependencies.

```plaintext
for (i=0; i<N; i++)
S1: A[i] = I[i];
for (t=1; t<T; t++)
{
    for (i=1; i<N-1; i++)
    for (i=1; i<N-1; i++)
S3: A[i] = B[i];
}
```

⇒ Apply geometric reasoning on $\mathbb{Z}$-polyhedra to bound $|P|$
Example 1: Jacobi 1D

Parameters: N, T; Inputs: I[N]; Outputs: A[N]
for (i=0; i<N; i++)
S1: A[i] = I[i];
for (t=1; t<T; t++)
{
    for (i=1; i<N-1; i++)
        for (i=1; i<N-1; i++)
    S3: A[i] = B[i];
}
Example 1: Jacobi 1D

Injective (DFG) circuit:

Set of disjoint (CDAG) paths:

From disjoint paths to projections

- For any $P \rightsquigarrow$ at most one element per disjoint path in $\text{In}(P)$.
- $\vec{b}$ as projection vector for $\phi_i$ $\rightsquigarrow |\phi_i(P)| \leq |\text{In}(P)| \leq 2S.$
Example 1: Jacobi 1D

Injective (DFG) circuit:

Set of disjoint (CDAG) paths:

From disjoint paths to projections

- For any $P \rightsquigarrow$ at most one element per disjoint path in $\text{In}(P)$.
- $\vec{b}$ as projection vector for $\phi_i$\hspace{1cm} $\rightsquigarrow |\phi_i(P)| \leq |\text{In}(P)| \leq 2S$. 
Example 1: Jacobi 1D

Original space

Transformed space

Static analysis of affine programs:
Example 1: Jacobi 1D

Apply Loomis-Whitney inequality:

$$|P| \leq (2S)^2 \quad \leadsto \quad Q = \Omega \left( \frac{NT}{S} \right) - (N + T)$$
High-Level algorithm

1. Extract data flow graph (DFG) from source code.
2. Identify paths of interest in DFG
3. Obtain projections that satisfy $|\phi_j(P)| \leq |\text{In}(P)|$.
4. Apply geometric reasoning to obtain the lower bounds.
more in the paper...

- use of “broadcast” paths to find projection directions
- use of generalized geometric Holder-Brascamp-Leib inequality
- Inherent multi-regime parametric characterization

Example: Rectangular matmult \((m \times n \times p)\)

If \(m, n, p \gg \sqrt{2S}\):

\[
Q = \Omega \left( \frac{mnp}{\sqrt{S}} \right)
\]

else if \(m, n \gg \sqrt{2S} \text{ and } p \ll \sqrt{2S}\) (mat-vect):

\[
Q = \Omega(mn)
\]

else if . . . :

\[
\ldots
\]
Conclusion

- **Challenge:** Computational complexity of algorithms is well understood, but data movement complexity is not.

- **Applications:**
  - **Algorithm analysis:** Which currently popular algorithms need rethinking due to high inherent data movement complexity?
  - **Compiler assessment:** Is further improvement of data locality possible?
  - **Algorithm-architecture co-design:** How to provision future architectures for the minimal data movement demands of algorithms?

- **Ongoing / future work:**
  - Methodologies
  - modeling / systems
  - handling irregular CDAGs
  - developing corresponding upper bounds of algorithms
Conclusion

▶ Challenge: Computational complexity of algorithms is well understood, but data movement complexity is not.

▶ Applications:
  ▶ **Algorithm analysis:** Which currently popular algorithms need rethinking due to high inherent data movement complexity?
  ▶ **Compiler assessment:** Is further improvement of data locality possible?
  ▶ **Algorithm-architecture co-design:** How to provision future architectures for the minimal data movement demands of algorithms?

▶ Ongoing / future work:
  ▶ Methodologies
  ▶ modeling / systems
  ▶ handling irregular CDAGs
  ▶ developing corresponding upper bounds of algorithms
Conclusion

- Challenge: Computational complexity of algorithms is well understood, but data movement complexity is not.

- Applications:
  - **Algorithm analysis**: Which currently popular algorithms need rethinking due to high inherent data movement complexity?
  - **Compiler assessment**: Is further improvement of data locality possible?
  - **Algorithm-architecture co-design**: How to provision future architectures for the minimal data movement demands of algorithms?

- Ongoing / future work:
  - Methodologies
  - modeling / systems
  - handling irregular CDAGs
  - developing corresponding upper bounds of algorithms
Thank you

