On Characterizing the Data Access Complexity of Programs

Venmugil Elango[1] Fabrice Rastello[2] Louis-Noël Pouchet[1]
J. Ramanujam[3] P. Sadayappan[1]

[1]The Ohio State University
[2]Inria
[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

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?

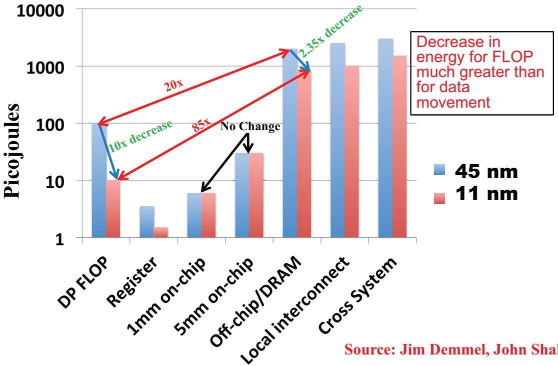
- Computational cost: number of operations executed by the algorithm
  - Objective: reduce the operation complexity

- 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

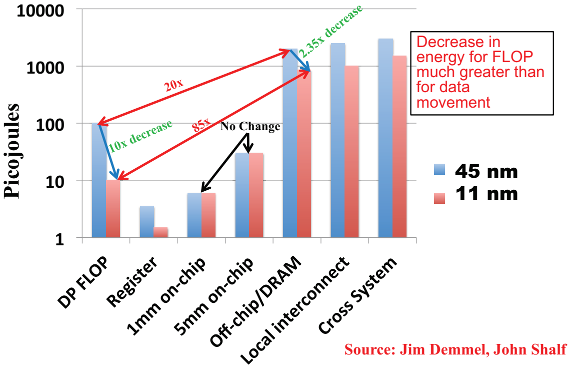


Source: Jim Demmel, John Shalf

➡ 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!**

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: $(N-1)^2$

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++)
    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?

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

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: $(N-1)^2$

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++)
    A[i][j] = A[i][j-1] + A[i-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 cost

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];
```

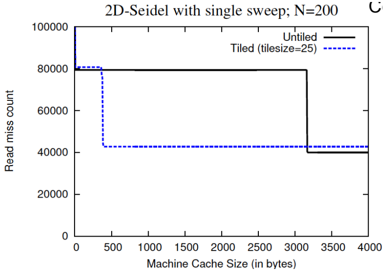
Comp. cost: $(N-1)^2$

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++)
    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?

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

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];
```

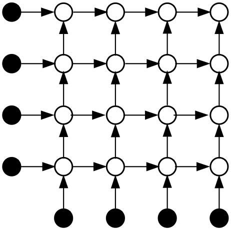
Comp. cost: $(N-1)^2$

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++)
    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?

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

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];
```

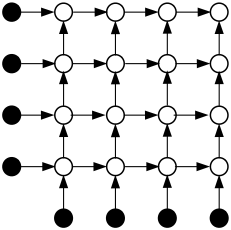
Comp. cost: $(N-1)^2$

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++)
    A[i][j] = A[i][j-1] + A[i-1][j];
```

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

Data movement cost



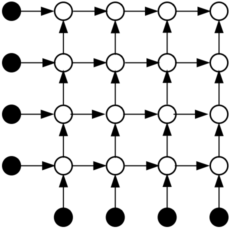
- Data movement cost different for two versions
- Also depends on cache size
- Question: What is data movement complexity?

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

Outline

Graph based

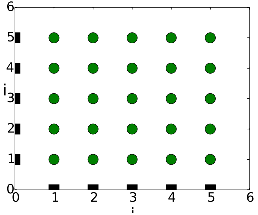
- Arbitrary CDAGs



- [Hong and Kung, 1981]: all valid schedules ⇝ all valid "$2S$-partitions" of CDAG
- (+) Generality
- (-) manual reasoning $\Longrightarrow$ challenge to automate

Geometric data footprint

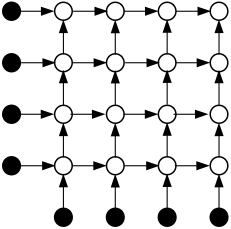
- Linear algebra like algorithms



- [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 $\Longrightarrow$ weakness of bounds or inapplicability

Graph based

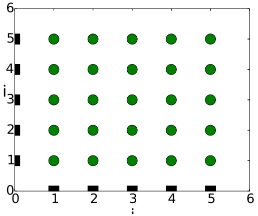
- Arbitrary CDAGs



- [Hong and Kung, 1981]: all valid schedules $\rightsquigarrow$ all valid "$2S$-partitions" of CDAG
- (+) Generality
- (-) manual reasoning $\Longrightarrow$ challenge to automate

Geometric data footprint

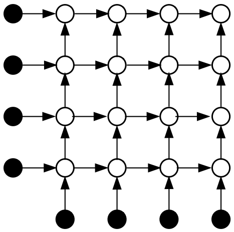
- Linear algebra like algorithms



- [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 $\Longrightarrow$ weakness of bounds or inapplicability

Graph based

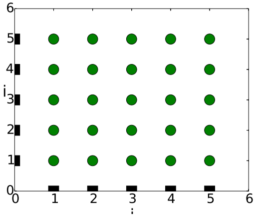
- Arbitrary CDAGs



- [Hong and Kung, 1981]: all valid schedules $\rightsquigarrow$ all valid "$2S$-partitions" of CDAG
- (+) Generality
- (-) manual reasoning $\Longrightarrow$ challenge to automate

Geometric data footprint

- Linear algebra like algorithms



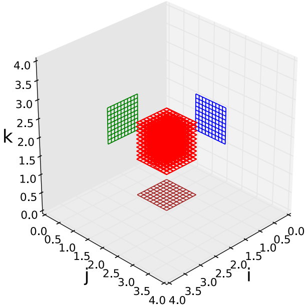
- [Irony et al., 2004], [Ballard et al., 2011]: Geom. approach based on Loomis-Whitney (LW) inequality
- [Christ et al., 2013]: Automation based on Holder-Brascamp-Leib (HBL) ineq.
- (+) Automated
- (-) Restricted model $\Longrightarrow$ weakness of bounds or inapplicability

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 ($d = 3$):



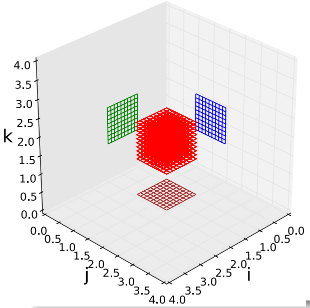
$$|E| \leq |\phi_1(E)|^{1/2} \times |\phi_2(E)|^{1/2} \times |\phi_3(E)|^{1/2}$$

$$|E| \leq |\phi_1(E)|^{1/(d-1)} \times \cdots \times |\phi_d(E)|^{1/(d-1)}$$

Loomis-Whitney inequality

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

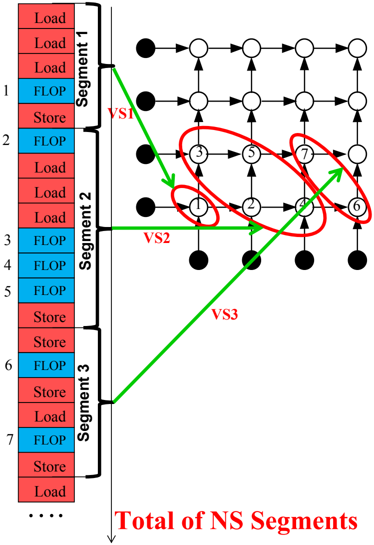
Example ($d = 3$):



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

$$|E| \leq |\phi_1(E)|^{1/(d-1)} \times \cdots \times |\phi_d(E)|^{1/(d-1)}$$

Hong & Kung $2S$-partioning



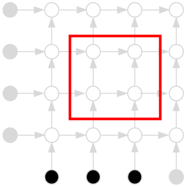
- Any valid schedule is asociated with a $2S$-partition

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** $\forall i, \ \ |\mathrm{In}(V_i)| \leq S$

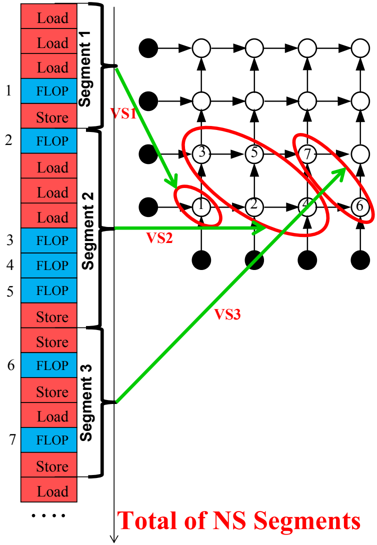
- Largest vertex-set: P



Data movement complexity

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

Hong & Kung $2S$-partioning



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

S-partition

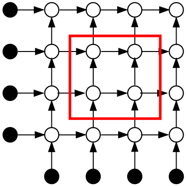
Collection of h subsets $(V_1, \ldots, V_h)$ of $V \setminus I$ s.t:

P1 pairwise disjoint

P2 no cyclic dependence

P3 $\forall i, \ \ |\mathrm{In}(V_i)| \leq S$

- Largest vertex-set: P



Data movement complexity

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

Outline

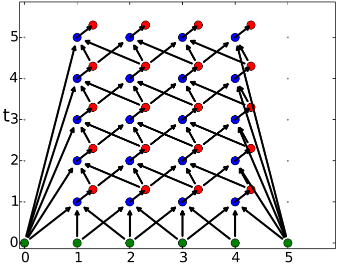
1. Motivation
2. Prior work & Challenges
3. Static analysis of affine programs

Affine computations

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

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

```
for (i=0; i<N; i++)
S1:  A[i] = I[i];
for (t=1; t<T; t++)
{
  for (i=1; i<N-1; i++)
S2:  B[i] = A[i-1]+A[i]+A[i+1];
  for (i=1; i<N-1; i++)
S3:  A[i] = B[i];
}
```



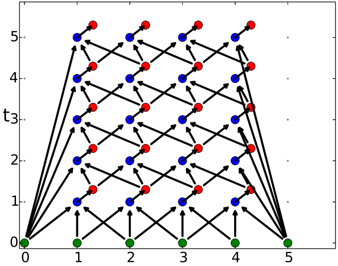
➡ Apply geometric reasoning on $\mathcal{Z}$-polyhedra to bound $|P|$

Affine computations

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

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

```
for (i=0; i<N; i++)
S1:  A[i] = I[i];
for (t=1; t<T; t++)
{
  for (i=1; i<N-1; i++)
S2:  B[i] = A[i-1]+A[i]+A[i+1];
  for (i=1; i<N-1; i++)
S3:  A[i] = B[i];
}
```

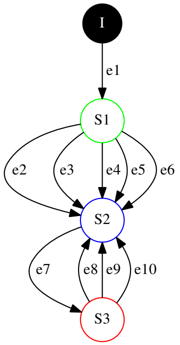


➡ **Apply geometric reasoning on $\mathcal{Z}$-polyhedra to bound $|P|$**

Example 1: Jacobi 1D

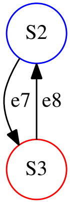
```
Parameters: N, T; Inputs: I[N]; Outputs: A[N]
for (i=0; i<N; i++)
S1:  A[i] = I[i];
for (t=1; t<T; t++)
{
  for (i=1; i<N-1; i++)
S2:  B[i] = A[i-1] + A[i] + A[i+1];
  for (i=1; i<N-1; i++)
S3:  A[i] = B[i];
}
```

DFG:

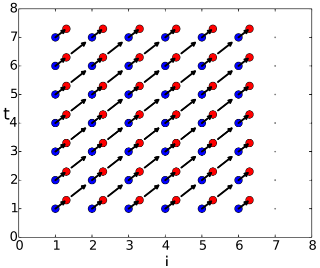


Example 1: Jacobi 1D

Injective (DFG) circuit:



Set of disjoint (CDAG) paths:

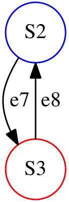


From disjoint paths to projections

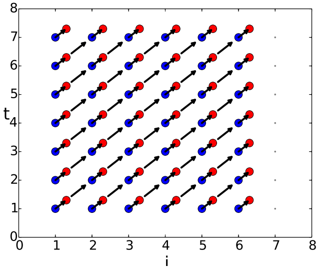
- For any $P \rightsquigarrow$ at most one element per disjoint path in $\mathrm{In}(P)$.
- $\vec{b}$ as projection vector for ϕ_i $\rightsquigarrow |\phi_i(P)| \leq |\mathrm{In}(P)| \leq 2S$.

Example 1: Jacobi 1D

Injective (DFG) circuit:



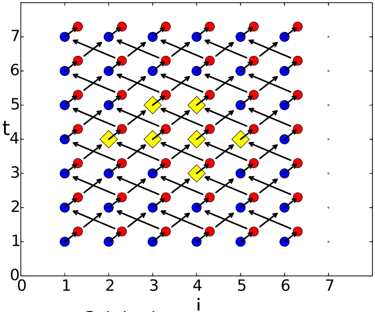
Set of disjoint (CDAG) paths:



From disjoint paths to projections

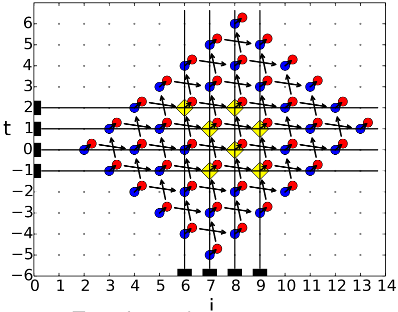
- For any $P \leadsto$ at most one element per disjoint path in $\mathrm{In}(P)$.
- $\vec{b}$ as projection vector for ϕ_i $\leadsto |\phi_i(P)| \leq |\mathrm{In}(P)| \leq 2S$.

Example 1: Jacobi 1D



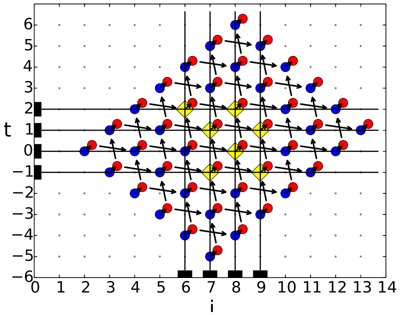
Original space

⇝



Transformed space

Example 1: Jacobi 1D



Apply Loomis-Whitney inequality:

$$|P| \leq (2S)^2 \qquad \rightsquigarrow \qquad Q = \Omega\left(\frac{NT}{S}\right) - (N + T)$$

High-Level algorithm

1. Extract data flow graph (DFG) from source code.
2. Identify paths of interest in DFG
3. Obtain projections that satisfy $|\phi_j(P)| \leq |\mathrm{In}(P)|$.
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 $m, n \gg \sqrt{2S}$ and $p \ll \sqrt{2S}$ (mat-vect):

$$Q = \Omega(mn)$$

else if $\ldots$:

$$\ldots$$

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 irreglar 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 irreglar 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 irreglar CDAGs
  - developing corresponding upper bounds of algorithms

Thank you

- Ballard, G., Demmel, J., Holtz, O., and Schwartz, O. (2011).
  Minimizing communication in numerical linear algebra.
  SIAM J. Matrix Analysis Applications, 32(3):866–901.
- Christ, M., Demmel, J., Knight, N., Scanlon, T., and Yelick, K. (2013).
  Communication Lower Bounds and Optimal Algorithms for Programs That Reference Arrays Part 1.
  EECS Technical Report EECS–2013-61, UC Berkeley.
- Hong, J.-W. and Kung, H. T. (1981).
  I/O complexity: The red-blue pebble game.
  In Proc. of the 13th annual ACM sympo. on Theory of computing (STOC'81), pages 326–333. ACM.
- Irony, D., Toledo, S., and Tiskin, A. (2004).
  Communication lower bounds for distributed-memory matrix multiplication.
  J. Parallel Distrib. Comput., 64(9):1017–1026.