SMO: An Integrated Approach To Intra-Array And Inter-Array Storage Optimization

Somashekaracharya G. Bhaskaracharya[1,3]
Uday Bondhugula[1]
Albert Cohen[2]

Indian Institute of Science[1]
ENS Paris, INRIA[2]
National Instruments[3]

January 21, 2016

Outline

Storage Optimization

Basic Goal Reuse memory locations for values without overlapping lifetimes

- Reuse within a given array or across different arrays
- Crucial for data-intensive programs
  - run larger problem size with a fixed amount of main memory
  - stencils, image processing applications, DSL compilers
    - affine loop-nests

Contracting A Particular Array

```
for(t=1;t<=N;i++)
 for(i=1;i<=N;i++)
  /*S*/ A[t,i]=f(A[t-1,i-1]
               +A[t-1,i]
               +A[t-1,i+1]);
```

(a) 1-d stencil using $\mathbf{N^2}$ storage

Dependences $(1,-1)$, $(1,0)$ and $(1,1)$

Live-out $A[T,*]$

```
for(t=1;t<=N;i++)
 for(i=1;i<=N;i++)
  /*S*/ A[t%2,i]=f(A[(t-1)%2,i-1]
                  +A[(t-1)%2,i]
                  +A[(t-1)%2,i+1]);
```

(b) Array contracted to size $\mathbf{2 \times N}$

```
for(t=1;t<=N;i++)
 for(i=1;i<=N;i++)
  /*S*/ A[(i-t+N)%(N+1)]=f(A[(i-t+N)%(N+1)]
                          +A[(i-t+1+N)%(N+1)]
                          +A[(i-t+2+N)%(N+1)]);
```

(c) Array contracted to **N+1** cells.
Storage optimal!

Contracting A Particular Array

```
for(t=1;t<=N;i++)
 for(i=1;i<=N;i++)
  /*S*/ A[t,i]=f(A[t-1,i-1]
               +A[t-1,i]
               +A[t-1,i+1]);
```

(a) 1-d stencil using $\mathbf{N^2}$ storage

Dependences $(1,-1)$, $(1,0)$ and $(1,1)$

Live-out $A[T,*]$

```
for(t=1;t<=N;i++)
 for(i=1;i<=N;i++)
  /*S*/ A[t%2,i]=f(A[(t-1)%2,i-1]
                  +A[(t-1)%2,i]
                  +A[(t-1)%2,i+1]);
```

(b) Array contracted to size $\mathbf{2} \times \mathbf{N}$

```
for(t=1;t<=N;i++)
 for(i=1;i<=N;i++)
  /*S*/ A[(i-t+N)%(N+1)]=f(A[(i-t+N)%(N+1)]
                         +A[(i-t+1+N)%(N+1)]
                         +A[(i-t+2+N)%(N+1)]);
```

(c) Array contracted to **N+1** cells.
Storage optimal!

Contracting A Particular Array

```
for(t=1;t<=N;i++)
 for(i=1;i<=N;i++)
  /*S*/ A[t,i]=f(A[t-1,i-1]
               +A[t-1,i]
               +A[t-1,i+1]);
```

(a) 1-d stencil using $\mathbf{N^2}$ storage

Dependences $(1,-1)$, $(1,0)$ and $(1,1)$

Live-out $A[T,*]$

```
for(t=1;t<=N;i++)
 for(i=1;i<=N;i++)
  /*S*/ A[t%2,i]=f(A[(t-1)%2,i-1]
                  +A[(t-1)%2,i]
                  +A[(t-1)%2,i+1]);
```

(b) Array contracted to size $\mathbf{2} \times \mathbf{N}$

```
for(t=1;t<=N;i++)
 for(i=1;i<=N;i++)
  /*S*/ A[(i-t+N)%(N+1)]=f(A[(i-t+N)%(N+1)]
                          +A[(i-t+1+N)%(N+1)]
                          +A[(i-t+2+N)%(N+1)]);
```

(c) Array contracted to **N+1** cells.
Storage optimal!

Reuse Across Arrays - Image Processing Applications

```
#define isbound(i,j) (i==0)||(i==(N-1))
                    ||(j==0)||(j==(N-1))
for(int i=0; i<N; ++i)
  for(int j=0; j<N: ++j)
/*S0*/ A0[i,j]= isbound(i,j) ? a[i,j]
               :a[i,j]+(a[i-1,j]+a[i+1,j]
                +a[i,j-1]+a[i][j+1]);

for(int i=0; i<N; ++i)
  for(int j=0; j<N: ++j)
/*S1*/ A1[i,j]=isbound(i,j) ? A0[i,j]
              :A0[i,j]+(A0[i-1,j]+A0[i+1,j]
               +A0[i,j-1]+A0[i,j+1]);

for(int i=0; i<N; ++i)
  for(int j=0; j<N: ++j)
/*S2*/ A2[i,j]=!isbound(i,j) ? A1[i][j]
              :A1[i,j]+(A1[i-1,j]+A1[i+1,j]
               +A1[i,j-1]+A1[i,j+1]);
```

(a) A_0, A_1 are just intermediate arrays which are not live-out.

$$S_0 : A_0[i,j] \rightarrow A[(i+3) \mod (N+2), j \mod N]$$
$$S_1 : A_1[i,j] \rightarrow A[(i+1) \mod (N+2), j \mod N]$$
$$S_2 : A_2[i,j] \rightarrow A[(i-1) \mod (N+2), j \mod N]$$

(b) The storage mapping enabling inter-array reuse. Overall storage requirement is reduced from $\mathbf{3N^2}$ to $\mathbf{N^2 + N}$.

Reuse Across Arrays - Image Processing Applications

```
#define isbound(i,j) (i==0)||(i==(N-1))
                    ||(j==0)||(j==(N-1))
for(int i=0; i<N; ++i)
  for(int j=0; j<N: ++j)
/*S0*/ A0[i,j]= isbound(i,j) ? a[i,j]
               :a[i,j]+(a[i-1,j]+a[i+1,j]
                +a[i,j-1]+a[i][j+1]);

for(int i=0; i<N; ++i)
  for(int j=0; j<N: ++j)
/*S1*/ A1[i,j]=isbound(i,j) ? A0[i,j]
              :A0[i,j]+(A0[i-1,j]+A0[i+1,j]
               +A0[i,j-1]+A0[i,j+1]);

for(int i=0; i<N; ++i)
  for(int j=0; j<N: ++j)
/*S2*/ A2[i,j]=!isbound(i,j) ? A1[i][j]
              :A1[i,j]+(A1[i-1,j]+A1[i+1,j]
               +A1[i,j-1]+A1[i,j+1]);
```

(a) A_0, A_1 are just intermediate arrays which are not live-out.

$$S_0 : A_0[i,j] \rightarrow A[(i+3) \mod (N+2), j \mod N]$$
$$S_1 : A_1[i,j] \rightarrow A[(i+1) \mod (N+2), j \mod N]$$
$$S_2 : A_2[i,j] \rightarrow A[(i-1) \mod (N+2), j \mod N]$$

(b) The storage mapping enabling inter-array reuse. Overall storage requirement is reduced from $\mathbf{3N^2}$ to $\mathbf{N^2 + N}$.

Outline

General Approach To The Problem

- Contract array along one or more directions to fixed sizes
  - **Step 1:** Determine *good* directions
    - Canonical directions need not be good ones
    - Affects dimensionality and storage size
    - Can be the difference between $N^2, 2N, N+1$ storage for a given $N \times N$ array
  - **Step 2:** Minimize the array size along these directions
    - Thoroughly studied by Lefebvre and Feautrier (1998)
- No good heuristics for **Step 1**
- Darte et al (2005), Lefebvre and Feautrier (1998)
  - work with canonical basis or assume that directions are given.

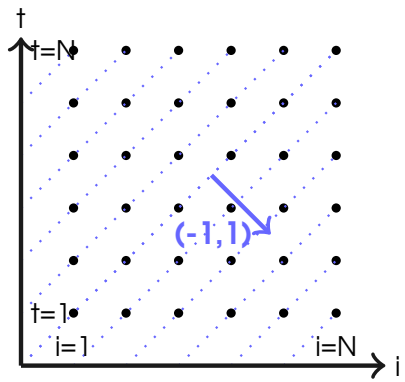
General Approach To The Problem

- Contract array along one or more directions to fixed sizes
  - **Step 1:** Determine *good* directions
    - Canonical directions need not be good ones
    - Affects dimensionality and storage size
    - Can be the difference between $N^2, 2N, N+1$ storage for a given $N \times N$ array
  - **Step 2:** Minimize the array size along these directions
    - Thoroughly studied by Lefebvre and Feautrier (1998)
- No good heuristics for **Step 1**
- Darte et al (2005), Lefebvre and Feautrier (1998)
  - work with canonical basis or assume that directions are given.

An Array Space Partitioning Approach

Storage Partitioning Hyperplane

Partitions the iteration space such that each partition uses a single memory location.



Hyperplane $(-1, 1)$ creating $(2N - 1)$ partitions.

Good Directions? Storage hyperplanes with good orientations

Contraction? Minimize the number of partitions created

- Affects the resulting storage size

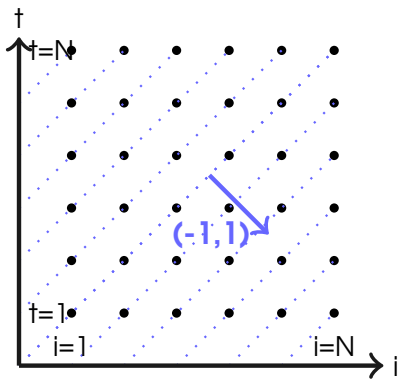
Dimensionality? Number of storage hyperplanes found

- Iteratively found until some criterion is met

An Array Space Partitioning Approach

Storage Partitioning Hyperplane

Partitions the iteration space such that each partition uses a single memory location.



Hyperplane $(-1, 1)$ creating $(2N - 1)$ partitions.

Good Directions? Storage hyperplanes with good orientations

Contraction? Minimize the number of partitions created

- Affects the resulting storage size

Dimensionality? Number of storage hyperplanes found

- Iteratively found until some criterion is met

Outline

Conflicts Within An Array Space

Conflicting indices $\vec{i} \bowtie \vec{j}$

Two array indices $\vec{i}, \vec{j}$, $(\vec{i} \neq \vec{j})$, conflict with each other and the conflict relation $\vec{i} \bowtie \vec{j}$ holds if the corresponding array elements are simultaneously live under the given schedule θ.

```
for(i=2;i<=n;i++)
  fib[i]=fib[i-1]+fib[i-2];
   result=fib[n];
```

(a) Before contraction

```
for(i=2;i<=n;i++)
  fib[i%2]=fib[(i-1)%2]+fib[(i-2)%2];
   result=fib[n%2];
```

(a) After contraction

Dependences? $(i-2) \rightarrow_{RAW} i, (i-1) \rightarrow_{RAW} i$

Live Out? $fib(n)$

Conflicts? Each array index conflicts with its adjacent index: $i \bowtie (i-1)$

$\Longrightarrow$ fib can be contracted to a 2-element array

Modulo storage mapping: $fib[i] \rightarrow fib[i \bmod 2]$

Conflicts Within An Array Space

Conflicting indices $\vec{i} \bowtie \vec{j}$

Two array indices $\vec{i}, \vec{j}$, $(\vec{i} \neq \vec{j})$, conflict with each other and the conflict relation $\vec{i} \bowtie \vec{j}$ holds if the corresponding array elements are simultaneously live under the given schedule θ.

```
for(i=2;i<=n;i++)
  fib[i]=fib[i-1]+fib[i-2];
   result=fib[n];
```

(a) Before contraction

```
for(i=2;i<=n;i++)
  fib[i%2]=fib[(i-1)%2]+fib[(i-2)%2];
   result=fib[n%2];
```

(a) After contraction

Dependences? $(i-2) \rightarrow_{RAW} i, (i-1) \rightarrow_{RAW} i$

Live Out? $fib(n)$

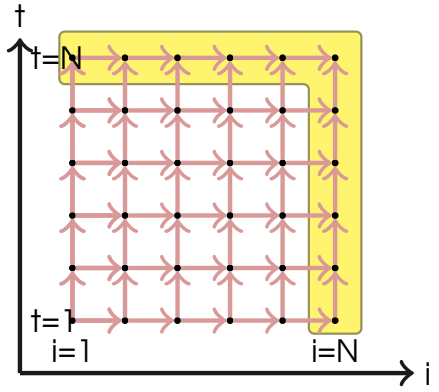
Conflicts? Each array index conflicts with its adjacent index: $i \bowtie (i-1)$

$\Longrightarrow$ fib can be contracted to a 2-element array
Modulo storage mapping: $fib[i] \rightarrow fib[i \bmod 2]$

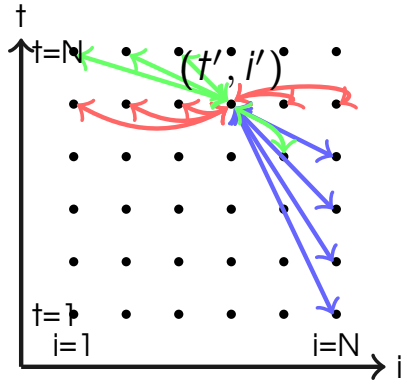
Array Space Partitioning Through Conflict Satisfaction

```
for(t=1;t<=N;i++)
   for(i=1;i<=N;i++)
/*S*/ A[t,i]=A[t,i-1]+A[t-1,i];
for(i=1;i<=N;i++)
   result=result+A[i,N]+A[N,i];
```

(a) A producer-consumer loop-nest



The flow dependences.
Live-out portion in yellow.



Conflicts in different conflict polyhedra.

Conflict set can be specified as union of convex polyhedra

Conflict Satisfaction

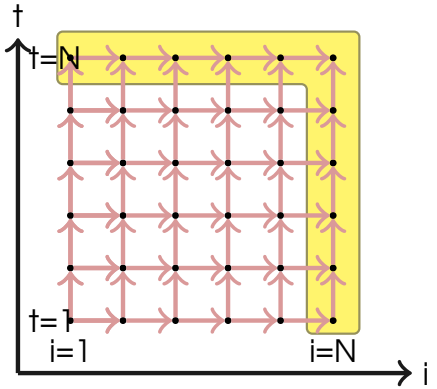
A conflict $\vec{I} \bowtie \vec{J}$ is said to be satisfied by a hyperplane $\vec{\Gamma}$ if $\vec{\Gamma}.\vec{I} - \vec{\Gamma}.\vec{J} \neq 0$.

→ Conflicting indices must be mapped to different partitions

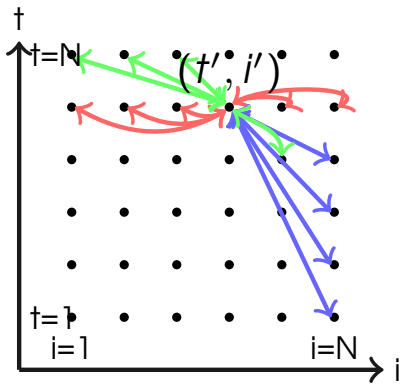
Array Space Partitioning Through Conflict Satisfaction

```
for(t=1;t<=N;i++)
   for(i=1;i<=N;i++)
/*S*/ A[t,i]=A[t,i-1]+A[t-1,i];
for(i=1;i<=N;i++)
   result=result+A[i,N]+A[N,i];
```

(a) A producer-consumer loop-nest



The flow dependences.
Live-out portion in yellow.



Conflicts in different
conflict polyhedra.

Conflict set can be specified as union of convex polyhedra

Conflict Satisfaction

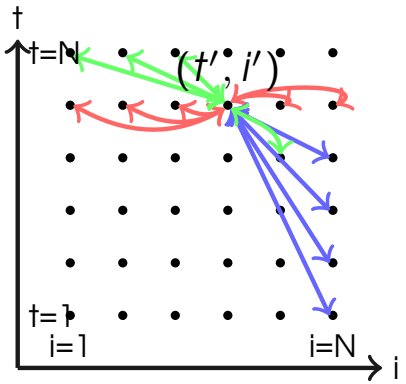
A conflict $\vec{i} \bowtie \vec{j}$ is said to be satisfied by a hyperplane $\vec{\Gamma}$ if $\vec{\Gamma}.\vec{i} - \vec{\Gamma}.\vec{j} \neq 0$.

- Conflicting indices must be mapped to different partitions

Finding Storage Hyperplanes

```
for(t=1;t<=N;i++)
   for(i=1;i<=N;i++)
/*S*/ A[t,i]=A[t,i-1]+A[t-1,i];
for(i=1;i<=N;i++)
   result=result+A[i,N]+A[N,i];
```

(a) A producer-consumer loop-nest



Conflicts in different conflict polyhedra

$(-1, 1)$ satisfies all conflicts

Heuristic for finding storage partitioning hyperplanes

- Primary objective: maximize conflict satisfaction
- Secondary objective: minimize number of partitions

Candidate hyperplanes . . .

- $(\mathbf{1}, \mathbf{0})$ Satisfies only blue, green conflicts
- $(\mathbf{0}, \mathbf{1})$ Satisfies only red, green conflicts
- $(-1, 1)$ Satisfies all conflicts
- $(-2, 1)$ Satisfies all conflicts creating $3N - 2$ partitions
- $(-3, 1)$ Satisfies all conflicts creating $4N - 2$ partitions

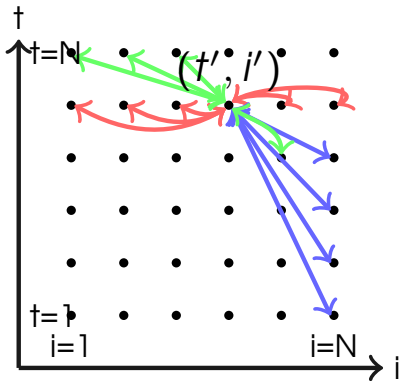
Modulo Storage Mapping $A[t, i] \rightarrow A[(i - t) \bmod (2N - 1)]$

Storage as well as dimension optimal!

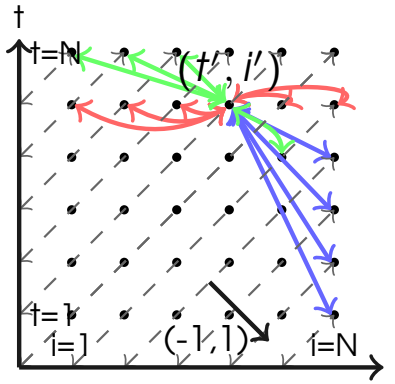
Finding Storage Hyperplanes

```
for(t=1;t<=N;i++)
   for(i=1;i<=N;i++)
/*S*/ A[t,i]=A[t,i-1]+A[t-1,i];
for(i=1;i<=N;i++)
   result=result+A[i,N]+A[N,i];
```

(a) A producer-consumer loop-nest



Conflicts in different conflict polyhedra



$(-1, 1)$ satisfies all conflicts

Heuristic for finding storage partitioning hyperplanes

- Primary objective: maximize conflict satisfaction
- Secondary objective: minimize number of partitions

Candidate hyperplanes . . .

$(\mathbf{1}, \mathbf{0})$ Satisfies only blue, green conflicts
$(\mathbf{0}, \mathbf{1})$ Satisfies only red, green conflicts
$(-\mathbf{1}, \mathbf{1})$ Satisfies all conflicts creating $\mathbf{2N} - \mathbf{1}$ partitions
$(-\mathbf{2}, \mathbf{1})$ Satisfies all conflicts creating $3N - 2$ partitions
$(-\mathbf{3}, \mathbf{1})$ Satisfies all conflicts creating $4N - 2$ partitions

Modulo Storage Mapping $A[t, i] \rightarrow A[(i - t) \bmod (2N - 1)]$
Storage as well as dimension optimal!

Outline

Typical Approach

- Decoupling intra-array from inter-array reuse
  - e.g. Lefebvre and Feutrier (1998), De Greef et al (1997)

1. Contract each indidivual array separately (successive modulo technique)

2. Exploit inter-array reuse opportunities

   - Build the array interference graph
     - Each node represents a statement in the affine loop-nest
     - Edge between statements S_i and S_j
       $\Longrightarrow$ S_i prematurely overwrites value computed by S_j (or vice-versa)

   - Greedy coloring of array interference graph

     Statements with same colour write to same data structure
     - rectangular hull of their contracted arrays

Typical Approach

- Decoupling intra-array from inter-array reuse
  - e.g. Lefebvre and Feutrier (1998), De Greef et al (1997)

1. Contract each indidivual array separately (successive modulo technique)

2. Exploit inter-array reuse opportunities

   - Build the array interference graph
     - Each node represents a statement in the affine loop-nest
     - Edge between statements S_i and S_j
       $\Longrightarrow$ S_i prematurely overwrites value computed by S_j (or vice-versa)

   - Greedy coloring of array interference graph

     Statements with same colour write to same data structure
     – rectangular hull of their contracted arrays

Ping-pong style stencil – an example

```
for (t=1; t<=N; t++){
  for (i=1; i<=N; i++)
    P[i] = f(Q[i-1],Q[i],Q[i+1]); /*S0*/
  for (i=1; i<=N; i++)
    Q[i] = P[i]; /*S1*/
}
for(i=1;i<=N;i++)
  result += Q[N][i];
```

(a) 1-d stencil using ping-pong buffer

$P[i]$ and $Q[i]$ are simultaneously live.

On graph colouring.
S_1 and S_2 cannot write to the same data structure

Arrays P and Q are already contracted to size N

Need unified approach to exploit intra-array and inter-array reuse

Better Solution $S_j(t, i) \rightarrow A[(i - t) \bmod (N + 1)], \quad j = 1, 2$

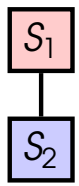
Ping-pong style stencil – an example

```
for (t=1; t<=N; t++){
  for (i=1; i<=N; i++)
    P[i] = f(Q[i-1],Q[i],Q[i+1]); /*S0*/
  for (i=1; i<=N; i++)
    Q[i] = P[i]; /*S1*/
}
for(i=1;i<=N;i++)
  result += Q[N][i];
```

(a) 1-d stencil using ping-pong buffer

Arrays P and Q are already contracted to size N

$P[i]$ and $Q[i]$ are simultaneously live.



On graph colouring.
S_1 and S_2 cannot write to the same data structure

Need unified approach to exploit intra-array and inter-array reuse

Better Solution $S_j(t,i) \rightarrow A[(i-t) \bmod (N+1)], \quad j = 1,2$

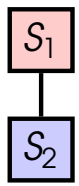
Ping-pong style stencil – an example

```
for (t=1; t<=N; t++){
  for (i=1; i<=N; i++)
    P[i] = f(Q[i-1],Q[i],Q[i+1]); /*S0*/
  for (i=1; i<=N; i++)
    Q[i] = P[i]; /*S1*/
}
for(i=1;i<=N;i++)
  result += Q[N][i];
```

(a) 1-d stencil using ping-pong buffer

Arrays P and Q are already contracted to size N

$P[i]$ and $Q[i]$ are simultaneously live.



On graph colouring.
S_1 and S_2 cannot write to the same data structure

Need unified approach to exploit intra-array and inter-array reuse

Better Solution $S_j(t, i) \rightarrow A[(i - t) \bmod (N + 1)], \quad j = 1, 2$

Outline

Global Unified Array Space

I. Convert to single-assignment form

- Each statement S_j writes to its own local array space A_j ($S_j(\vec{i})$ writes to $A_j[\vec{i}]$)

II. Unify local array spaces into a $(d+1)$-dimensional global array space A

"d" max dimensionality of a local array space

- $A[j] = A_j$, padded with $(d - d_j)$ dimensions

```
for (t=1; t<=N; t++){
  for (i=1; i<=N; i++)
    P[i] = f(Q[i-1],Q[i],Q[i+1]); /*S0*/
  for (i=1; i<=N; i++)
    Q[i] = P[i]; /*S1*/
}
for(i=1;i<=N;i++)
  result += Q[N][i];
```

(a) 1-d stencil using ping-pong buffer

```
for(t=1;t<=N;t++){
  for(i=1;i<=N;i++)
/*S0*/A[0,t,i]=f((i>1&&t>1?A[1,t-1,i-1]:Q[i-1]),
                 (t>1?A[1,t-1,i]:Q[i]),
                 (i<N&&t>1?A[1,t-1,i+1]:Q[i+1]));
  for(i=1;i<=N;i++)
/*S1*/A[1,t,i] = A[0,t,i];
}
for(i=1;i<=N;i++) result += A[1,N,i];
```

(b) Outermost dimension to index local array spaces

Global Unified Array Space

I. Convert to single-assignment form

- Each statement S_j writes to its own local array space A_j ($S_j(\vec{i})$ writes to $A_j[\vec{i}]$)

II. Unify local array spaces into a $(d+1)$-dimensional global array space A

"d" max dimensionality of a local array space

- $A[j] = A_j$, padded with $(d - d_j)$ dimensions

```
for (t=1; t<=N; t++){
  for (i=1; i<=N; i++)
    P[i] = f(Q[i-1],Q[i],Q[i+1]); /*S0*/
  for (i=1; i<=N; i++)
    Q[i] = P[i]; /*S1*/
}
for(i=1;i<=N;i++)
  result += Q[N][i];
```

(a) 1-d stencil using ping-pong buffer

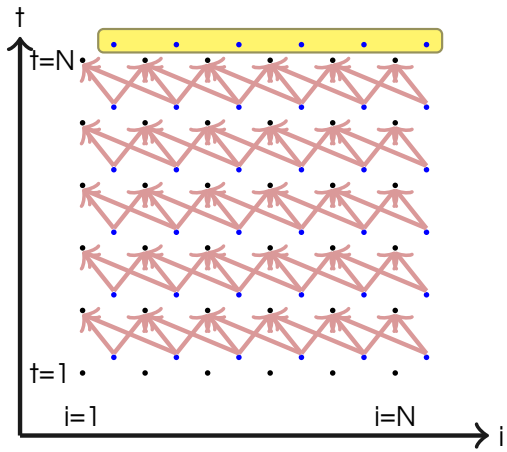
```
for(t=1;t<=N;t++){
  for(i=1;i<=N;i++)
/*S0*/A[0,t,i]=f((i>1&&t>1?A[1,t-1,i-1]:Q[i-1]),
                 (t>1?A[1,t-1,i]:Q[i]),
                 (i<N&&t>1?A[1,t-1,i+1]:Q[i+1]));
  for(i=1;i<=N;i++)
/*S1*/A[1,t,i] = A[0,t,i];
}
for(i=1;i<=N;i++) result += A[1,N,i];
```

(b) Outermost dimension to index local array spaces

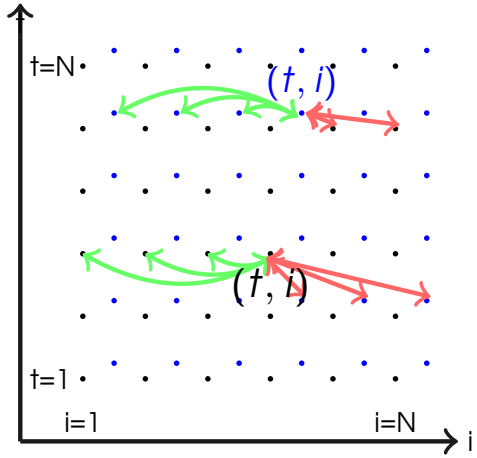
Conflict Satisfaction In Global Array Space

```
for(t=1;t<=N;t++){
  for(i=1;i<=N;i++)
/*S0*/A[0,t,i]=f((i>1&&t>1?
                  A[1,t-1,i-1]:Q[i-1]),
                (t>1?A[1,t-1,i]:Q[i]),
                (i<N&&t>1?
                  A[1,t-1,i+1]:Q[i+1]));
  for(i=1;i<=N;i++)
/*S1*/A[1,t,i]=A[0,t,i];
}
for(i=1;i<=N;i++) result += A[1,N,i];
```

(a) In single-assignment form



(b) Last use of $A[1, t, i]$ in $S0(t+1, i+1)$



(c) Intra-statement and inter-statement conflicts.

- Partition global array space separately with hyperplanes Γ_s, Γ_t for statements S_s, S_t
- Hyperplane also characterized by its offset
  - Constant shift of a local array space can enable inter-array reuse

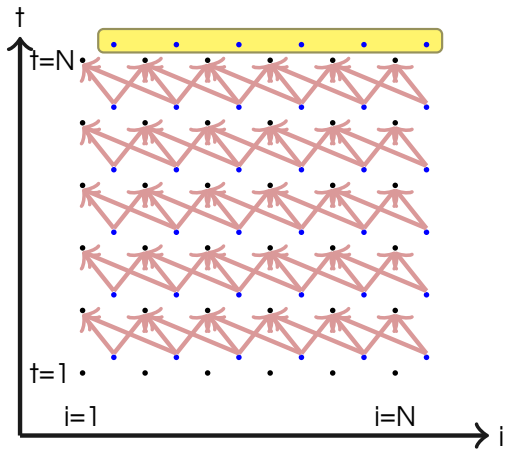
Conflict Satisfaction

A conflict $\vec{i} \bowtie \vec{j}$ in global array space such that $\vec{i} \in A[s]$ and $\vec{j} \in A[t]$ is said to be satisfied by hyperplanes $\vec{\Gamma}_s$ and $\vec{\Gamma}_t$ with offsets δ_s and δ_t if $\vec{\Gamma}_s.\vec{i} + \delta_s - \vec{\Gamma}_t.\vec{j} - \delta_t \neq 0$.

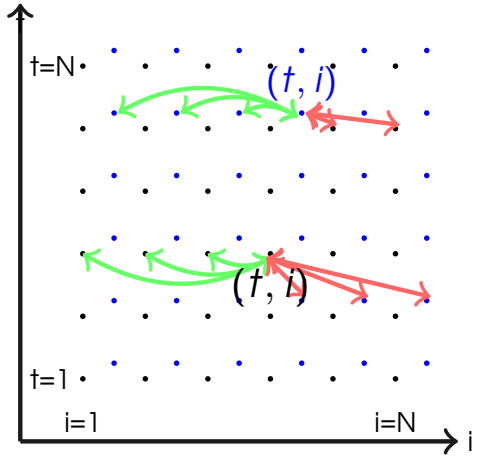
Conflict Satisfaction In Global Array Space

```
for(t=1;t<=N;t++){
  for(i=1;i<=N;i++)
/*S0*/A[0,t,i]=f((i>1&&t>1?
                  A[1,t-1,i-1]:Q[i-1]),
                (t>1?A[1,t-1,i]:Q[i]),
                (i<N&&t>1?
                  A[1,t-1,i+1]:Q[i+1]));
  for(i=1;i<=N;i++)
/*S1*/A[1,t,i]=A[0,t,i];
}
for(i=1;i<=N;i++) result += A[1,N,i];
```

(a) In single-assignment form



(b) Last use of $A[1, t, i]$ in $S0(t+1, i+1)$



(c) Intra-statement and inter-statement conflicts.

- Partition global array space separately with hyperplanes Γ_s,Γ_t for statements S_s,S_t
- Hyperplane also characterized by its offset
  - Constant shift of a local array space can enable inter-array reuse

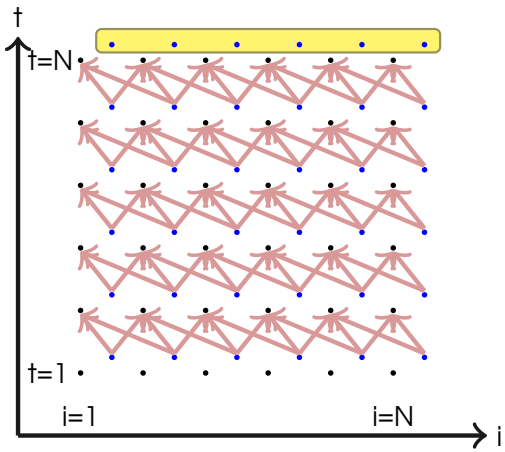
Conflict Satisfaction

A conflict $\vec{i} \bowtie \vec{j}$ in global array space such that $\vec{i} \in A[s]$ and $\vec{j} \in A[t]$ is said to be satisfied by hyperplanes $\vec{\Gamma}_s$ and $\vec{\Gamma}_t$ with offsets δ_s and δ_t if $\vec{\Gamma}_s.\vec{i} + \delta_s - \vec{\Gamma}_t.\vec{j} - \delta_t \neq 0$.

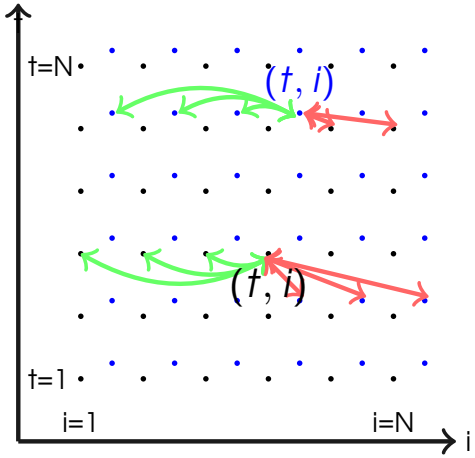
Conflict Satisfaction In Global Array Space

```
for(t=1;t<=N;t++){
  for(i=1;i<=N;i++)
/*S0*/A[0,t,i]=f((i>1&&t>1?
                  A[1,t-1,i-1]:Q[i-1]),
                (t>1?A[1,t-1,i]:Q[i]),
                (i<N&&t>1?
                  A[1,t-1,i+1]:Q[i+1]));
  for(i=1;i<=N;i++)
/*S1*/A[1,t,i]=A[0,t,i];
}
for(i=1;i<=N;i++) result += A[1,N,i];
```

(a) In single-assignment form



(b) Last use of $A[1, t, i]$ in $S0(t+1, i+1)$



(c) Intra-statement and inter-statement conflicts.

- Partition global array space separately with hyperplanes Γ_s,Γ_t for statements S_s,S_t
- Hyperplane also characterized by its offset
  - Constant shift of a local array space can enable inter-array reuse

Conflict Satisfaction

A conflict $\vec{i} \bowtie \vec{j}$ in global array space such that $\vec{i} \in A[s]$ and $\vec{j} \in A[t]$ is said to be satisfied by hyperplanes $\vec{\Gamma}_s$ and $\vec{\Gamma}_t$ with offsets δ_s and δ_t if $\vec{\Gamma_s}.\vec{i} + \delta_s - \vec{\Gamma_t}.\vec{j} - \delta_t \neq 0$.

Outline

Partitioning Hyperplanes For Global Array Space

Conflict Set $CS = CS_{intra} \cup CS_{inter} = K_1 \cup K_2 \cup \cdots \cup K_l$

To Find For each statement S_j,

m partitioning hyperplanes $\vec{\Gamma}_j^{(0)}, \vec{\Gamma}_j^{(1)}, \ldots, \vec{\Gamma}_j^{(m-1)}$

with offsets $\delta_j^{(0)}, \delta_j^{(1)}, \ldots, \delta_j^{(m-1)}$

- An intra-statement conflict associated with statement S_j
  - Satisfied by atleast one of the hyperplanes found for S_j
- An inter-statement conflict associated with statements S_j and S_k
  - Satisfied by pair of hyperplanes $\vec{\Gamma}_j^{(l)}$ and $\vec{\Gamma}_k^{(l)}$ found at same level l

Partitioning Hyperplanes For Global Array Space

Conflict Set $CS = CS_{intra} \cup CS_{inter} = K_1 \cup K_2 \cup \cdots \cup K_l$

To Find For each statement S_j,

m partitioning hyperplanes $\vec{\Gamma}_j^{(0)}, \vec{\Gamma}_j^{(1)}, \ldots, \vec{\Gamma}_j^{(m-1)}$

with offsets $\delta_j^{(0)}, \delta_j^{(1)}, \ldots, \delta_j^{(m-1)}$

- An intra-statement conflict associated with statement S_j
  - Satisfied by atleast one of the hyperplanes found for S_j
- An inter-statement conflict associated with statements S_j and S_k
  - Satisfied by pair of hyperplanes $\vec{\Gamma}_j^{(l)}$ and $\vec{\Gamma}_k^{(l)}$ found at same level l

Partitioning Hyperplanes For Global Array Space

Conflict Set $CS = CS_{intra} \cup CS_{inter} = K_1 \cup K_2 \cup \cdots \cup K_l$

To Find For each statement S_j,

m partitioning hyperplanes $\vec{\Gamma}_j^{(0)}, \vec{\Gamma}_j^{(1)}, \ldots, \vec{\Gamma}_j^{(m-1)}$

with offsets $\delta_j^{(0)}, \delta_j^{(1)}, \ldots, \delta_j^{(m-1)}$

- An intra-statement conflict associated with statement S_j
  - Satisfied by atleast one of the hyperplanes found for S_j
- An inter-statement conflict associated with statements S_j and S_k
  - Satisfied by pair of hyperplanes $\vec{\Gamma}_j^{(l)}$ and $\vec{\Gamma}_k^{(l)}$ found at same level l

Integrated Heuristic For Intra-Array and Inter-Array Reuse

Objective I Maximize intra-statement conflict satisfaction

Objective II Minimize #partitions due to intra-statement conflict satisfaction

Satisfy inter-statement conflicts as well (as side effects)

Objective III Maximize inter-statement conflict satisfaction

Objective IV Minimize #partitions due to inter-statement conflict satisfaction

Iterate after eliminating satisfied conflicts from the conflict set

Integrated Heuristic For Intra-Array and Inter-Array Reuse

Objective I Maximize intra-statement conflict satisfaction

Objective II Minimize #partitions due to intra-statement conflict satisfaction

Satisfy inter-statement conflicts as well (as side effects)

Objective III Maximize inter-statement conflict satisfaction

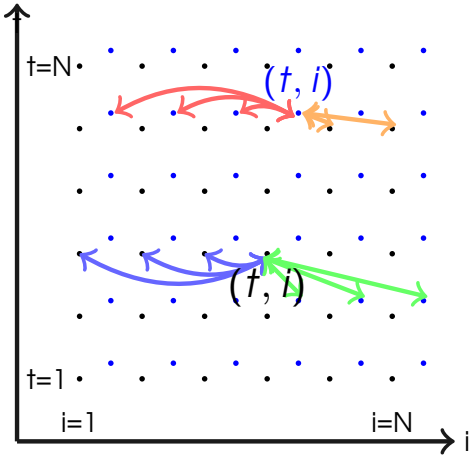
Objective IV Minimize #partitions due to inter-statement conflict satisfaction

Iterate after eliminating satisfied conflicts from the conflict set

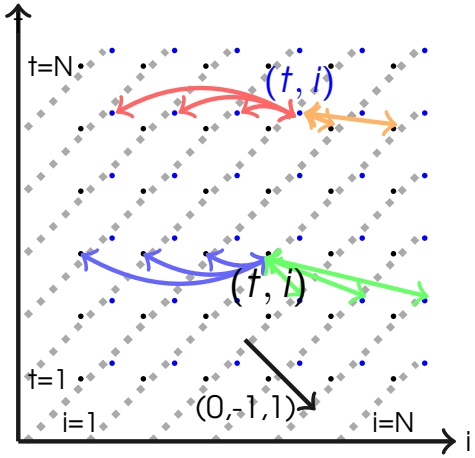
Example Revisited

```
for(t=1;t<=N;t++){
  for(i=1;i<=N;i++)
/*S0*/A[0,t,i]=f((i>1&&t>1?
                  A[1,t-1,i-1]:Q[i-1]),
                (t>1?A[1,t-1,i]:Q[i]),
                (i<N&&t>1?
                  A[1,t-1,i+1]:Q[i+1]));
  for(i=1;i<=N;i++)
/*S1*/A[1,t,i]=A[0,t,i];
}
for(i=1;i<=N;i++) result += A[1,N,i];
```

(a) In single-assignment form



(b) Intra-statement and inter-statement conflicts.



(c) $(0, -1, 1)$ satisfying all conflicts

Candidate hyperplanes for statements S_0 and S_1 ...

- $(\mathbf{0}, \mathbf{0}, \mathbf{1})$ Satisfies blue, red and orange conflicts
- $(\mathbf{0}, \mathbf{1}, \mathbf{0})$ Satisfies only green inter-statement conflicts
- $(\mathbf{0}, -\mathbf{1}, \mathbf{1})$ Satisfies all conflicts
- $(\mathbf{0}, -\mathbf{2}, \mathbf{1})$ Satisfies all conflicts
- $(\mathbf{0}, -\mathbf{3}, \mathbf{1})$ Satisfies all conflicts

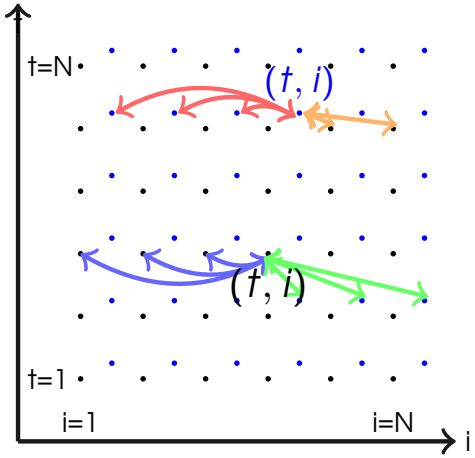
Modulo Storage Mapping $A[j, t, i] \rightarrow A[(i - t) \bmod (N + 1)]$

Statement $S1$ is a redundant copy statement!

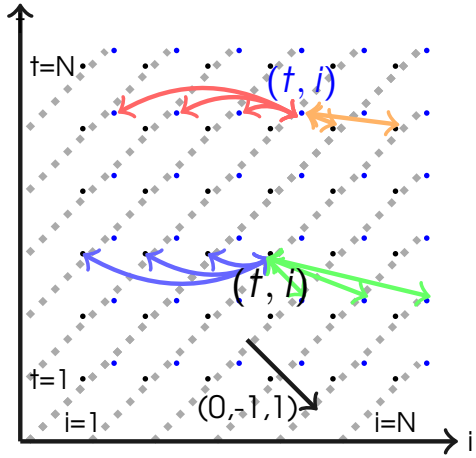
Example Revisited

```
for(t=1;t<=N;t++){
  for(i=1;i<=N;i++)
/*S0*/A[0,t,i]=f((i>1&&t>1?
                  A[1,t-1,i-1]:Q[i-1]),
                (t>1?A[1,t-1,i]:Q[i]),
                (i<N&&t>1?
                  A[1,t-1,i+1]:Q[i+1]));
  for(i=1;i<=N;i++)
/*S1*/A[1,t,i]=A[0,t,i];
}
for(i=1;i<=N;i++) result += A[1,N,i];
```

(a) In single-assignment form



(b) Intra-statement and inter-statement conflicts.



(c) $(0, -1, 1)$ satisfying all conflicts

Candidate hyperplanes for statements S_0 and S_1 ...

- $(\mathbf{0}, \mathbf{0}, \mathbf{1})$ Satisfies blue, red and orange conflicts
- $(\mathbf{0}, \mathbf{1}, \mathbf{0})$ Satisfies only green inter-statement conflicts
- $(\mathbf{0}, -\mathbf{1}, \mathbf{1})$ Satisfies all conflicts creating $\mathbf{N} + \mathbf{1}$ partitions
- $(\mathbf{0}, -\mathbf{2}, \mathbf{1})$ Satisfies all conflicts creating $N + 2$ partitions
- $(\mathbf{0}, -\mathbf{3}, \mathbf{1})$ Satisfies all conflicts creating $N + 3$ partitions

Modulo Storage Mapping $A[j, t, i] \rightarrow A[(i - t) \bmod (N + 1)]$

Statement $S1$ is a redundant copy statement!

Outline

Storage Mappings Obtained Using SMO tool

Table : (SMO) against baseline (Lefebvre and Feautrier (1998)) with B as blocking factor

Benchmark		Modulo storage mapping	Reduction (approx.)	SMO time
1-d stencil	baseline	$S_0 : A_0[t \bmod 1, i \bmod N]$ $S_1 : A_1[t \bmod 1, i \bmod N]$	2	0.055s
	SMO	$S_0 : A[(i-t) \bmod (N+1)]$ $S_1 : A[(i-t) \bmod (N+1)]$		
2-d stencil	baseline	$S_0 : A_0[t \bmod 1, i \bmod N, j \bmod N]$ $S_1 : A_1[t \bmod 1, i \bmod N, j \bmod N]$	2	0.633s
	SMO	$S_0 : A[(i-3t+1) \bmod (N+2), j \bmod N]$ $S_1 : A[(i-3t) \bmod (N+2), j \bmod N]$		
3-d stencil	baseline	$S_0 : A_0[t \bmod 1, i \bmod N, j \bmod N, k \bmod N]$ $S_1 : A_1[t \bmod 1, i \bmod N, j \bmod N, k \bmod N]$	2	22.57s
	SMO	$S_0 : A[(i-3t) \bmod (N+2), j \bmod N, k \bmod N]$ $S_1 : A[(i-3t-1) \bmod (N+2), j \bmod N, k \bmod N]$		
jacobi 2-d smoothing	baseline	$S_k : A_{k\%2}[i \bmod N, j \bmod N]$	2	4.846s
	SMO	$S_k : A[(i+3-2k) \bmod (N+2), j \bmod N]$		
blur-tiled	baseline	$S_0 : A_0[ty, tx, x \bmod B, y \bmod B]$	$\frac{B}{3}$	0.738s
	SMO	$S_0 : A'_0[ty, tx, (y-2x) \bmod (3B-2)]$		
	baseline	$S_1 : A_1[ty, tx, x \bmod B, y \bmod B]$	1	
	SMO	$S_1 : A'_1[ty, tx, x \bmod B, y \bmod B]$		
unsharp-tiled	baseline	$S_0 : A_0[z, ty, tx, x \bmod B, y \bmod B]$	$\frac{B}{5}$	1.013s
	SMO	$S_0 : A'_0[z, ty, tx, (y-4x) \bmod (5B-4)]$		
	baseline	$S_1 : A_1[z, ty, tx, x \bmod B, y \bmod B]$	1	
	SMO	$S_1 : A'_1[z, ty, tx, -y \bmod B, x \bmod B]$		

Performance Evaluation

- Compiled with Intel C compiler (version 15.0) with flags "-O3 -openmp"
- Run on all cores of an Intel Xeon E5-2680 dual-socket machine with 8 cores per socket and a total of 64 GB of non-ECC RAM

Table : Performance of various benchmarks with the storage mappings obtained

Benchmark	**Input size**	**Execution time**		**Speedup**
		baseline	**smo**	
1-d-stencil-ping-pong	N= 524288, T=256	0.411s	0.388s	1.059
2-d-stencil-ping-pong	N= 16384, T=16	39.65s	33.84s	1.172
2-d-stencil-ping-pong	N= 32768, T=8	85.07s	69.27s	1.228
3-d-stencil-ping-pong	N=128, T=512	22.70s	22.96s	0.988
3-d-stencil-ping-pong	N=256, T=32	11.17s	12.11s	0.922
3-d-stencil-ping-pong	N=512, T=32	88.71s	114.0s	0.778
jacobi-2d-smoothing	N=4096, 3 steps	2.455s	2.247s	1.092
jacobi-2d-smoothing	N=4096, 5 steps	2.896s	2.706s	1.070
jacobi-2d-smoothing	N=4096, 9 steps	3.820s	3.758s	1.016
unsharp-tiled	N=4096, B=256	1.337s	0.679s	1.969
blur-tiled	N=8192, T=512	0.046s	0.044s	1.045

Outline

Summary

- Intra-array and inter-array storage reuse
  - Array space partitioning by finding good storage hyperplanes
- Heuristic driven by a fourfold objective function.
  - greedy conflict satisfaction (impacts dimensionality)
  - minimizes the partitions (minimizes dimension sizes)
  - factors in inter-statement conflicts (exploits inter-statement reuse)
- Developed SMO tool—a polyhedral storage optimizer.
  - Effective on several real-world examples.
  - Storage mappings which are asymptotically better than those by existing techniques.

Summary

- Intra-array and inter-array storage reuse
  - Array space partitioning by finding good storage hyperplanes
- Heuristic driven by a fourfold objective function.
  - greedy conflict satisfaction (impacts dimensionality)
  - minimizes the partitions (minimizes dimension sizes)
  - factors in inter-statement conflicts (exploits inter-statement reuse)
- Developed SMO tool—a polyhedral storage optimizer.
  - Effective on several real-world examples.
  - Storage mappings which are asymptotically better than those by existing techniques.

Acknowledgements

- INRIA (France) for an associate team award PolyFlow
- National Instruments

Thanks

Questions?