Automatic Optimization Of Arrays In Affine Loop-Nests

Somashekaracharya G. Bhaskaracharya[1,2]
Advisor: Uday Bondhugula[1]

CSA, Indian Institute of Science[1]
National Instruments[2]

April 29, 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 $\mathbf{2} \times \mathbf{N}$

```
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 $\mathbf{N^2} + \mathbf{N}$.

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

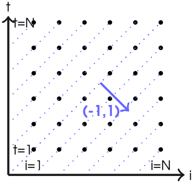
  - Thoroughly studied by Lefebvre and Feautrier(1998)

- No good heuristics for **Step 1**
- Darte et al(2005), Lefebvre and Feautrier(1998)
  - work with canonical basis or assume that directions are given.

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? $fib(n)$

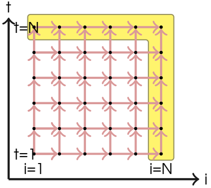
Conflicts? Each array index conflicts with its adjacent index: $i \bowtie (i-1)$

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

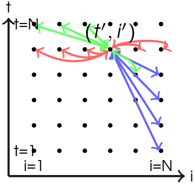
Array Space Partitioning Through Conflict Satisfaction

```
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 Satisfaction

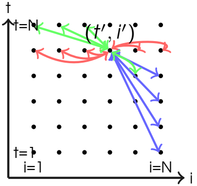
A conflict $\vec{i} \bowtie \vec{j}$ is said to be satisfied by a hyperplane $\vec{\Gamma}$ if $\vec{\Gamma}.\vec{i} - \vec{\Gamma}.\vec{j} \neq 0$.

– Conflicting indices must be mapped to different partitions

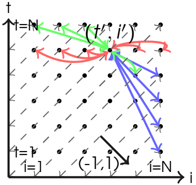
Example Revisited

```
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

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

Storage Mappings Obtained Using SMO tool

Table: Our approach (SMO) compared to the baseline (Lefebvre and Feautrier(1998)) with B being the loop blocking factor

Benchmark		Modulo storage mapping	Reduction (approx.)	SMO time
produce-consume	baseline	$A[t \bmod N, i \bmod N]$	$\frac{N}{2}$	0.17s
	SMO	$A[(i-t) \bmod (2N-1)]$		
blur-interleaved	baseline	$blurx[y \bmod 3, x \bmod N]$	1.5	0.14s
	SMO	$blurx[(2x-y) \bmod (2N+1)]$		
blur-tiled	baseline	$A[tx, ty, x \bmod B, y \bmod B]$	$\frac{B}{3}$	0.11s
	SMO	$A[tx, ty, (y-2x) \bmod (3B-2)]$		
harris-corner-tiled	baseline	$sobel[tx, ty, x \bmod B, y \bmod B]$	$\frac{B}{3}$	0.12s
	SMO	$sobel[tx, ty, (y-2x) \bmod (3B-2)]$		
unsharp-mask-tiled	baseline	$A[z, tx, ty, x \bmod B, y \bmod B]$	$\frac{B}{5}$	0.82s
	SMO	$A[z, tx, ty, (y-4x) \bmod (5B-4)]$		
LBM-D2Q9	baseline	$A[t \bmod 2, i \bmod N, j \bmod N]$	2	0.61s
	SMO	$A[(i-2t) \bmod (N+2), j \bmod N]$		
LBM-D3Q19	baseline	$A[t \bmod 2, i \bmod N, j \bmod N, k \bmod N]$	2	3.32s
	SMO	$A[(i-2t) \bmod (N+2), j \bmod N, k \bmod N]$		
LBM-D3Q27	baseline	$A[t \bmod 2, i \bmod N, j \bmod N, k \bmod N]$	2	3.33s
	SMO	$A[(i-2t) \bmod (N+2), j \bmod N, k \bmod N]$		
diamond-tile	baseline	$A_B[tt \bmod B, ii \bmod (2B-1)]$	$\frac{B}{3}$	0.44s
	SMO	$A_B[(tt-3ii) \bmod (6B-5)]$		
stencil-1d-llelogram-tile	baseline	$A_B[tt \bmod B, ii \bmod B]$	$\frac{B}{3}$	0.29s
	SMO	$A_B[(tt-ii) \bmod (3B-2)]$		
stencil-1d-hex-tile	baseline	$A_B[tt \bmod B, ii \bmod (3B-2)]$	$\frac{B}{3}$	1.15s
	SMO	$A_B[(-tt+3ii) \bmod (9B-7)]$		

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.

Acknowledgements And Publications

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

1. Somashekaracharya G. Bhaskaracharya, Uday Bondhugula, Albert Cohen *Automatic Storage Optimization for Arrays*, ACM Transactions on Programming Languages and Systems (TOPLAS), accepted in 2015.
2. Somashekaracharya G. Bhaskaracharya, Uday Bondhugula, Albert Cohen *SMO: An Integrated Approach to Intra-Array and Inter-Array Storage Optimization*, ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), St.Petersberg, USA, pages 526 - 538, Jan 2016.