Automatic Optimization of Arrays in Affine Loop Nests

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

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

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** $A[T,*]$

Array A can be contracted to size $\mathbf{2}\times\mathbf{N}$. Optimal?

```
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)]);
```

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

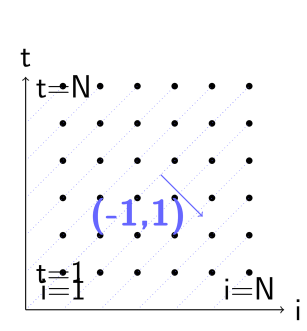
Intra-Array Reuse – Typical Approach

- Contract array along one or more directions to fixed sizes
  - **Step 1:** Determine *good* directions
    - canonical directions need not be good ones
    - can be difference between $N^2, 2N, N+1$ storage for 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 Partitioning Approach

Storage Partitioning Hyperplane

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



Storage hyperplane $(-1,1)$ creating $(2N-1)$ partitions.

Good Directions? Hyperplanes with good orientations
Contraction? Minimize the number of partitions created
Dimensionality? Number of storage hyperplanes found

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];
```

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

Live Out? $fib(n)$ **Conflicts?** $i\bowtie(i-1)$

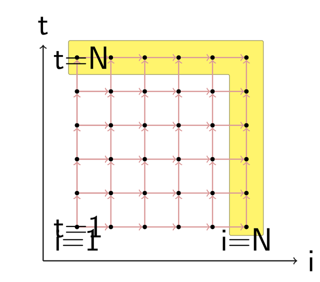
Modulo storage mapping: $fib[i]\rightarrow fib[i \bmod 2]$

Conflict Satisfaction

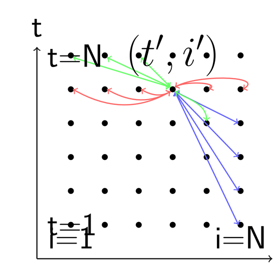
Conflict $\vec{i}\bowtie\vec{j}$ is satisfied by hyperplane $\vec{\Gamma}$ if $\vec{\Gamma}.\vec{i}-\vec{\Gamma}.\vec{j}\neq 0$.

Conflict Set Specification

```
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];
```



The flow dependences. Live-out portion in yellow.



Conflicts in different conflict polyhedra.

- Conflicting indices must be mapped to different partitions

Hyperplane $(1,0)$ Satisfies blue, green conflicts
Hyperplane $(0,1)$ Satisfies red conflicts

Modulo Storage Mapping $A[t,i]\rightarrow A[t \bmod N, i \bmod N]$
$\implies$ No contraction!

But... what about $A[t,i]\rightarrow A[(i-t) \bmod (2N-1)]$?

Heuristic To Find Storage Hyperplanes

Conflict Set $CS = K_1\cup K_2\cup\cdots\cup K_l$
Conflict satisfaction $(\vec{\Gamma}.\vec{s}-\vec{\Gamma}.\vec{t})\geq 1 \vee (\vec{\Gamma}.\vec{s}-\vec{\Gamma}.\vec{t})\leq -1$

Pair of decision variables x_{1i}, x_{2i} for each conflict polyhedron K_i

$$x_{1i} = \begin{cases} 1 \text{ if } (\vec{\Gamma}.\vec{s}-\vec{\Gamma}.\vec{t})\geq 1,\ \forall\, \vec{s}\bowtie\vec{t}\ \in\ K_i, \\ 0 \text{ if otherwise.}\end{cases}$$

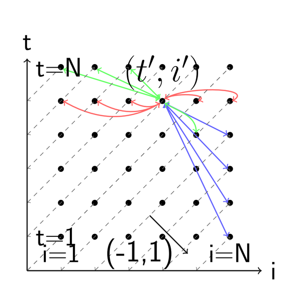
$$x_{2i} = \begin{cases} 1 \text{ if } (\vec{\Gamma}.\vec{s}-\vec{\Gamma}.\vec{t})\leq -1,\ \forall\, \vec{s}\bowtie\vec{t}\ \in\ K_i, \\ 0 \text{ if otherwise.}\end{cases}$$

Conflict satisfaction count η
$\eta=\Sigma_{i=1}^{i=l}(x_{1i}+x_{2i})$
$\uparrow\downarrow\eta \implies \downarrow\uparrow$ #unsatisfied polyhedra
Objective I Maximize η
Impacts Dimensionality

Bound on #partitions
$|\vec{\Gamma}.\vec{s}-\vec{\Gamma}.\vec{t}|\leq(\vec{u}.\vec{P}+w)$
$\forall\vec{s}\bowtie\vec{t}\in CS$
$\uparrow\downarrow(\vec{u}.\vec{P}+w)\implies\uparrow\downarrow$ #partitions
Objective II Minimize $(\vec{u}.\vec{P}+w)$
Affects Storage size

Iterate after eliminating satisfied conflicts from the conflict set.

Intra-Array Reuse Example Revisited



- $(\mathbf{1},\mathbf{0}),(\mathbf{0},\mathbf{1})$ don't satisfy all conflicts
- $(-\mathbf{1},\mathbf{1})$ Satisfies all conflicts creating $\mathbf{2N-1}$ partitions
- $(-\mathbf{2},\mathbf{1})$ Satisfies all conflicts creating $3N-2$ partitions
- $(-\mathbf{3},\mathbf{1})$ Satisfies all conflicts creating $4N-3$ partitions

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

Storage as well as dimension optimal!

Inter-Array Reuse – Typical Approach

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

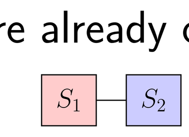
I. Contract each individual array separately

II. Exploit inter-array reuse opportunities
- Build the array interference graph
  - edge between nodes (statements) S_i and S_j
    - $\implies$ 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]); /*S1*/
  for (i=1; i<=N; i++)
    Q[i] = P[i]; /*S2*/
}
for(i=1; i<=N; i++) result += Q[N][i];
```

Arrays P and Q are already contracted to size N



Graph colouring: S_1,S_2 cannot write to same data structure
$\because P[i]$ and $Q[i]$ are simultaneously live.

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

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

Global Unified Array Space

I. Convert to single-assignment form
- statement S_j writes to own local array space A_j ($S_j(\vec{i})$ writes to $A_j[\vec{i}]$)

II. Unify local array spaces into $(d+1)$-d global array space A
- $A[j]=A_j$, padded with $(d-d_j)$ dimensions

```
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];
```

Outermost dimension to index local array spaces

- 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 In Global Array Space

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 $\vec{\Gamma}_s.\vec{i}+\delta_s-\vec{\Gamma}_t.\vec{j}-\delta_t\neq 0$.

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

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

An Integrated Heuristic

Conflict satisfaction $(\vec{\Gamma}_j.\vec{s}+\delta_j-\vec{\Gamma}_k.\vec{t}-\delta_k)\geq 1\ \vee$
$(\vec{\Gamma}_j.\vec{s}+\delta_j-\vec{\Gamma}_k.\vec{t}-\delta_k)\leq -1$

A pair of decision variables x_{1i}, x_{2i} for each conflict polyhedron K_i

$x_{1i}=1\ \ if\ \ (\vec{\Gamma}_j.\vec{s}+\delta_j-\vec{\Gamma}_k.\vec{t}-\delta_k)\geq 1\ \ else\ \ 0,\ \forall\,\vec{s}\bowtie\vec{t}\ \in\ K_i$
$x_{2i}=1\ \ if\ \ (\vec{\Gamma}_j.\vec{s}+\delta_j-\vec{\Gamma}_k.\vec{t}-\delta_k)\leq -1\ \ else\ \ 0,\ \forall\,\vec{s}\bowtie\vec{t}\ \in\ K_i$

Bounds for conflicts associated with statement S_j
Intra-statement : $|\vec{\Gamma}_j.\vec{s}-\vec{\Gamma}_j.\vec{t}|\leq(\vec{u}_j.\vec{P}+w_j)\ \forall\vec{s}\bowtie\vec{t}\in CS_{intra}$
Inter-statement : $|\vec{\Gamma}_j.\vec{s}+\delta_j-\vec{\Gamma}_k.\vec{t}-\delta_k|\leq(\vec{u}'_j.\vec{P}+w'_j)\ \forall\vec{s}\bowtie\vec{t}\in CS_{inter}$

Inter-statement polyhedron associated with S_j must be satisfied only if $\vec{u}'_j=\vec{u}_j$

I. Maximize Conflict Satisfaction
$\eta_{intra}=\sum_{\forall i,\ K_i\in CS_{intra}}(x_{1i}+x_{2i})$

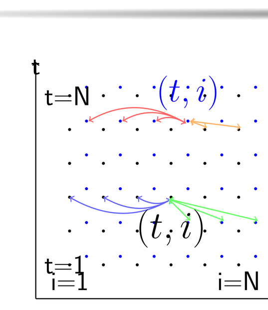
II. Minimize $(\vec{u}_j.\vec{P}+w_j)$ for each statement S_j Affects storage size

III. Maximize Conflict Satisfaction
$\eta_{inter}=\sum_{\forall i,\ K_i\in CS_{inter}}(x_{1i}+x_{2i})$

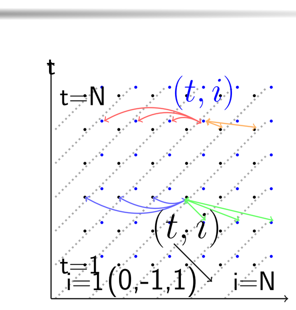
IV. Minimize $(\vec{u}'_j.\vec{P}+w'_j)$ for each statement S_j Affects storage size

Iterate after eliminating satisfied conflicts from the conflict set

Ping-Pong Style Stencil – Example Revisited



(a) Intra and inter-statement conflicts.



(b) $(0,-1,1)$ satisfies all conflicts

- $(\mathbf{0},\mathbf{0},\mathbf{1}),(\mathbf{0},\mathbf{1},\mathbf{0})$ Do not satisfy all conflicts
- $(\mathbf{0},-\mathbf{1},\mathbf{1})$ Satisfies all conflicts creating $\mathbf{N+1}$ 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

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

Statement $S1$ is a redundant copy statement!

Summary

- Unified heuristic for intra-array and inter-array storage reuse
  - array space partitioning to find good storage hyperplanes
- Heuristic driven by a fourfold objective function.
  - greedy conflict satisfaction (impacts the 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.