UOV and Transformations

Previously

- Storage mapping concepts and example with 1D stencil computation
- ScOP for smith waterman and Pluto

Assignments

–Quiz 2 on RamCT is due by Friday night

–Intermediate reports are due next Wednesday, will not be doing regrades

–HW6 (posted) and HW7 (will be posted within a week) due April 5th

Today

- Idea for LCPC paper based on semester projects
- The Universal Occupancy Vector
- Transformation review: fusion, fission, skewing
- Algorithms needed for automation

LCPC Paper idea

Some LCPC history

Basic outline of the paper

- Survey of the current polyhedral model power
- Case studies that evaluate the current tools based on…
  - Learning curve
  - Ease of use: documentation and robust error messages
  - Amount of tweaking needed to approach “best” possible performance
  - Resulting performance
  - Implementation limitations: what algorithms are not available
- Future research directions

Student semester long projects

Case studies

- Greg: LMie computes the scattering properties for polydisperse homogeneous spherical particles using the Mie solution, POCC
- Jared: Wavelets have become increasingly important in e cient video and image encoding. Pluto
- Glenn: genetic algorithm, POET
- Lixing: embedded applications benchmark, POET
- Matt: nearest neighbor algorithm for data mining, omega
- Brendan: libquantum Shor's algorithm for integer factorization, AlphaZ
- Nirmal: polyhedral benchmark suite, AlphaZ and PLUTO
- Ryan: support vector machine based learning algorithm, omega
- Steve: shortest path, AlphaZ
- Wenxiang: WalLS for solving SAT problems, POCC

Schedule-Independent Storage Mapping for Loops

Michelle Mills Strout
Larry Carter, Jeanne Ferrante, Beth Simon

UCSD - High Performance Computation Lab
http://www.cs.ucsd.edu/groups/hpcl/hpcl.html

Space vs. Flexibility

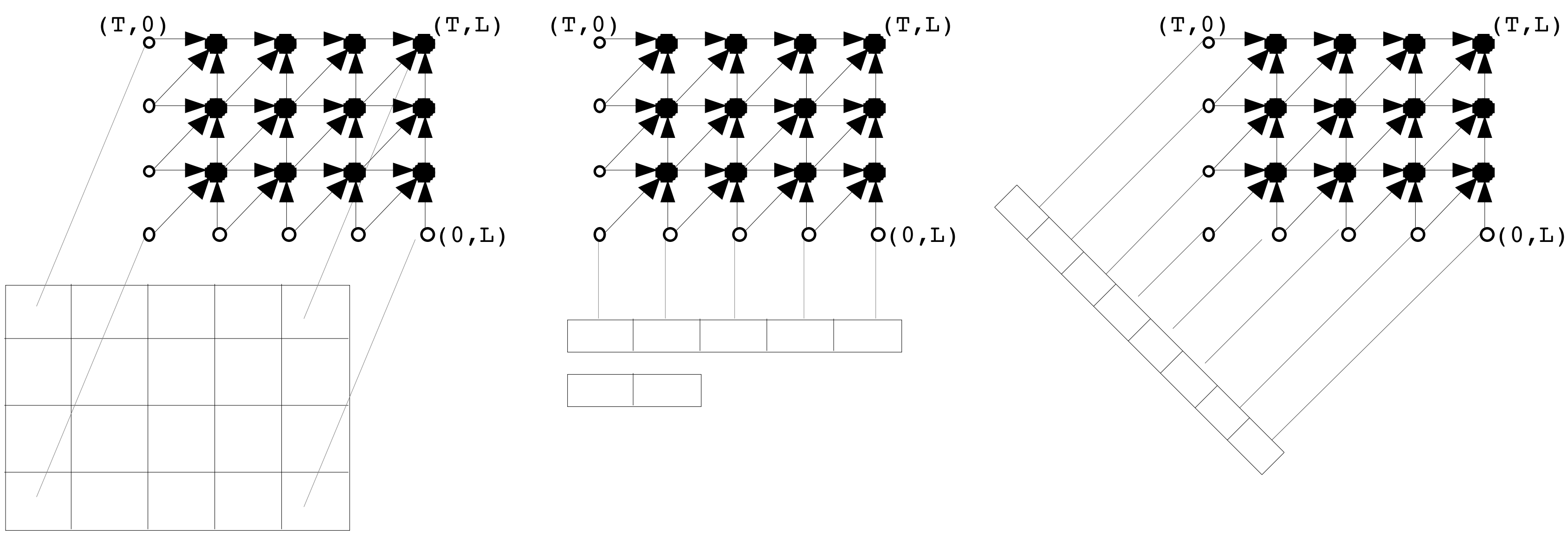
Space - storage necessary to execute a loop

Flexibility - ability to execute loop with any legal schedule

Maximal Flexibility
(Array Expansion)

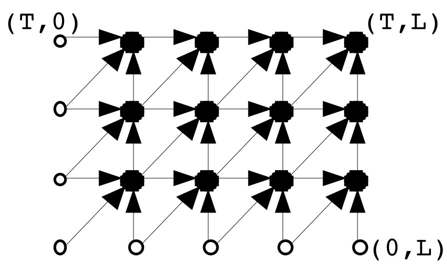
Minimal Storage
(Storage Optimization)

Best of Both

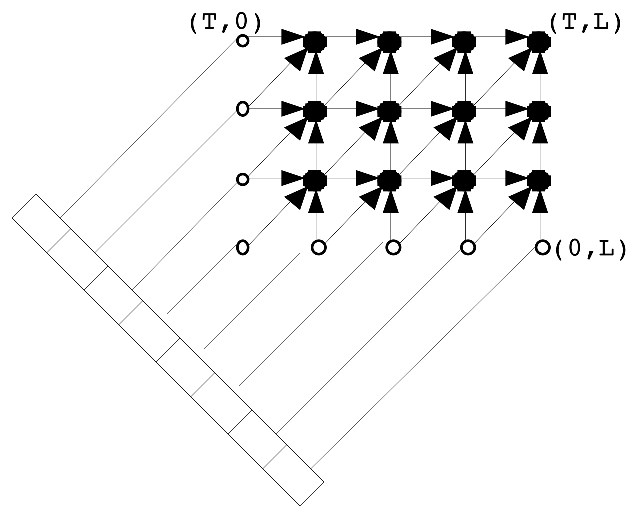


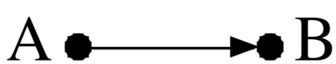
Problem

Iteration Space Graph (ISG)



Storage Mapping





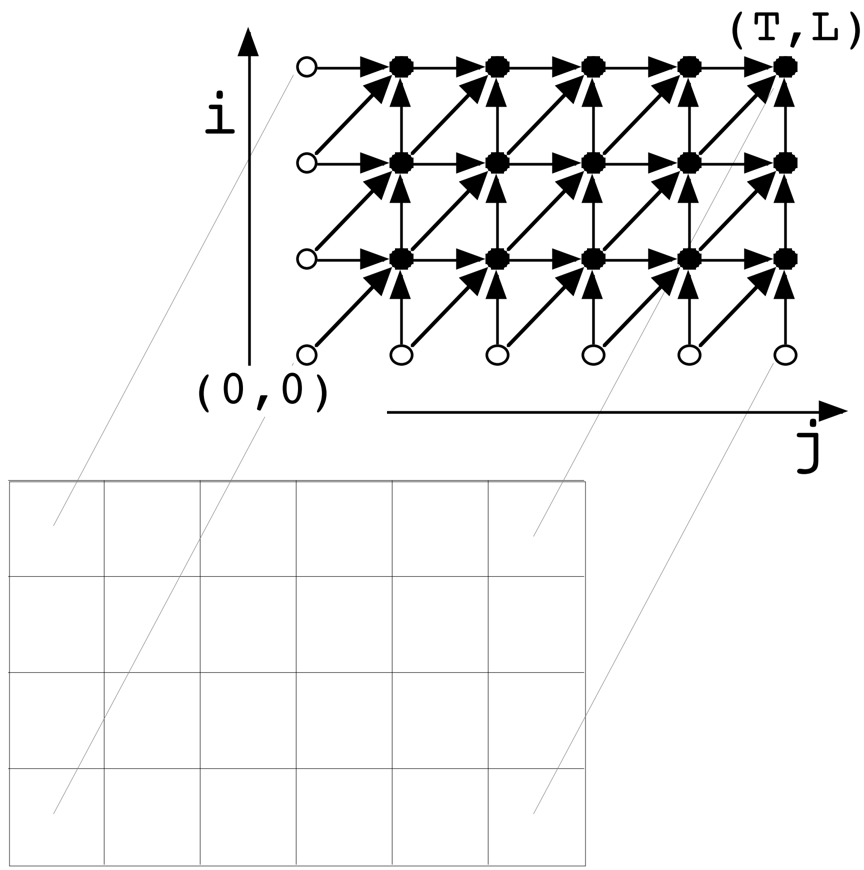
Data Dependence - B uses value produced by A

Universal Occupancy Vector (UOV) - B can reus
A's storage in any legal schedule

Outline

- Example
- UOV Selection
- Minimizing Storage
- Experimental Results

Array Expanded Version



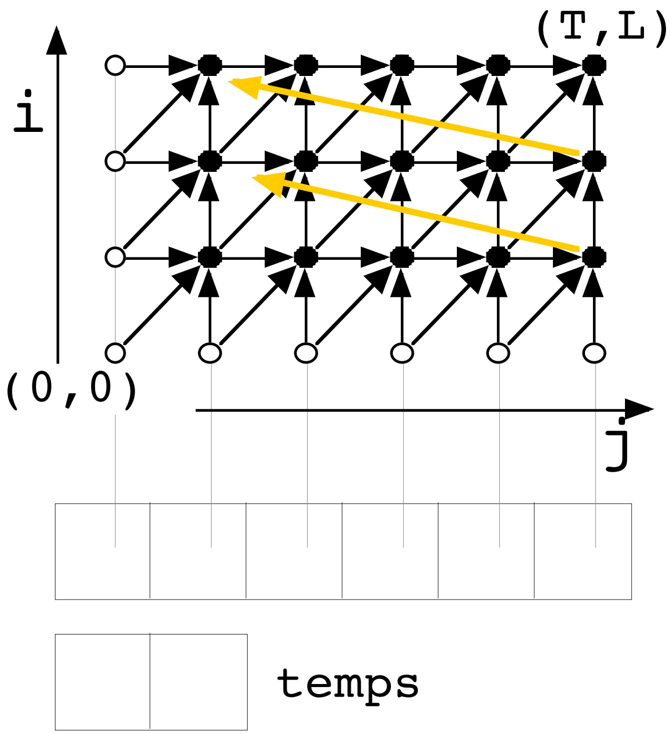
Loop:

```
for i=1 to T
  for j=1 to L
    A[i,j] = f(A[i-1,j],
               A[i,j-1],
               A[i-1,j-1])
```

Observations:

- <u>Any</u> legal schedule allowed
- Storage requirements: <u>TxL</u>

Storage Optimized Version



Loop:

```
for i=1 to T
   for j=1 to L
      temp1 = A[j]
      A[j] = f(A([j],
                A[j-1],temp2)
      temp2 = temp1
```

Observations:

- Only one legal schedule
- Storage requirements:
- L (+ 2 temps)

storage-related dependences
(not all shown)

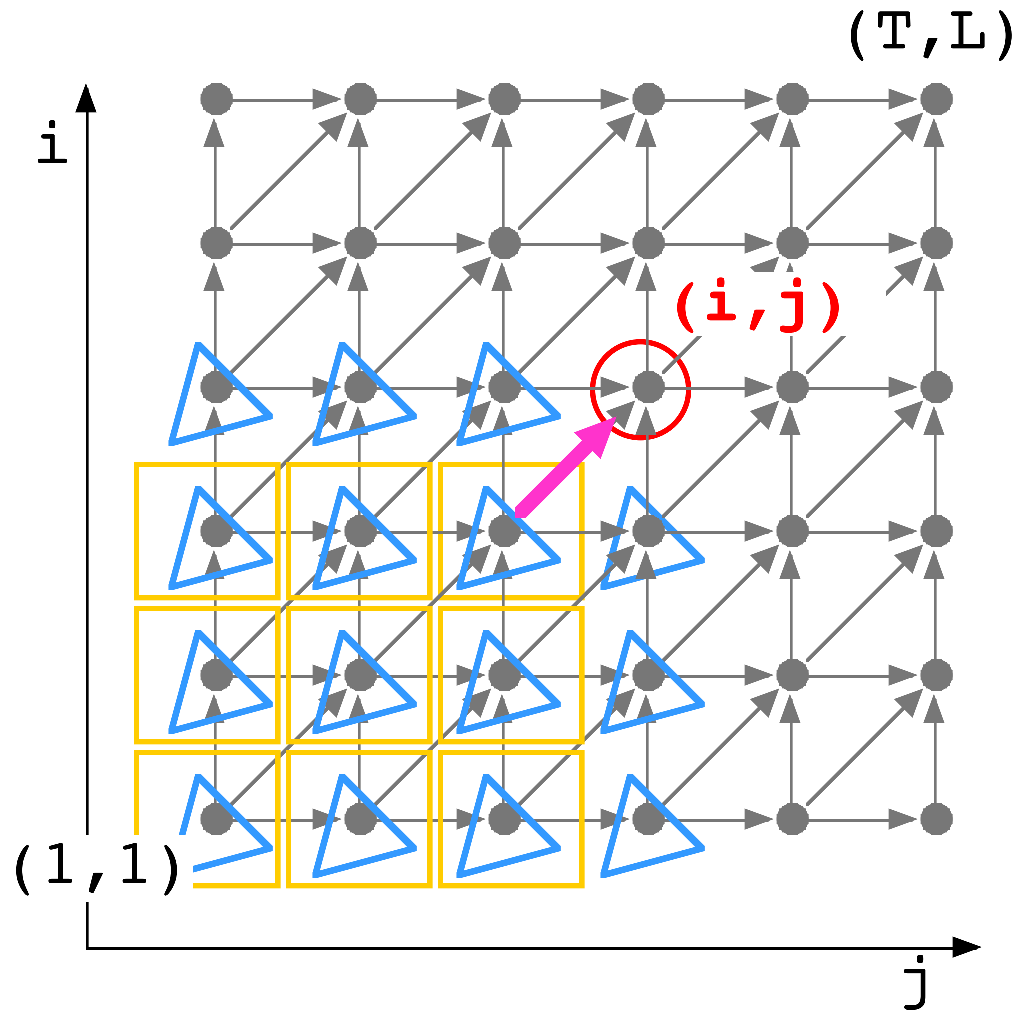
UOV Selection

DONE set: must execute before (i,j)

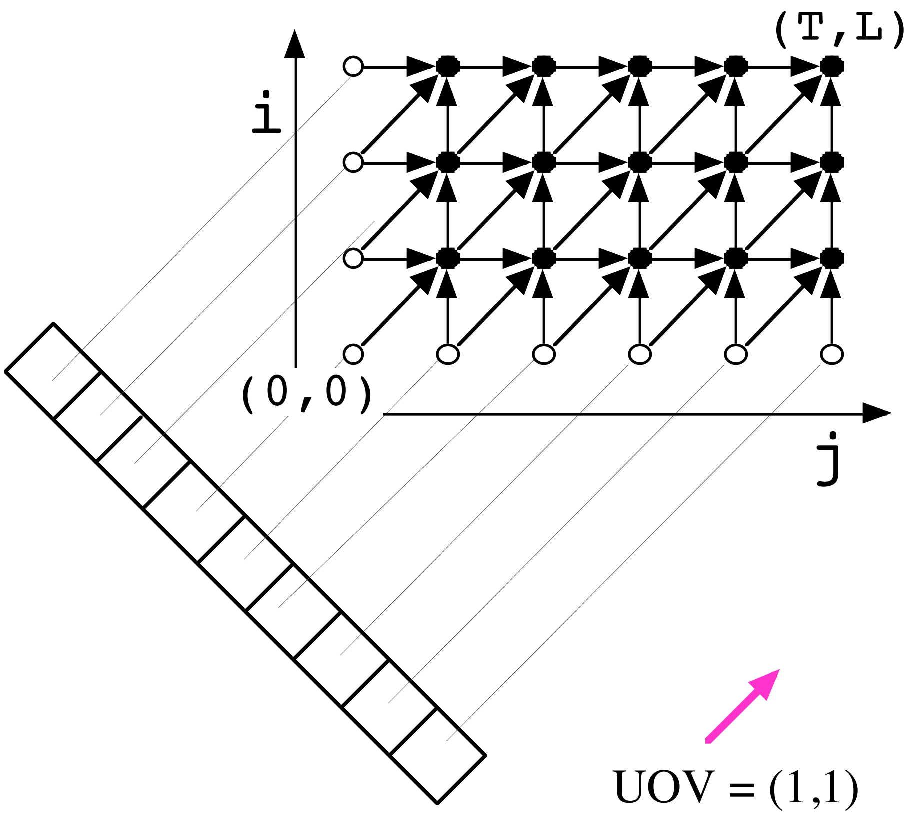
DEAD set: all dependences must lead to DONE set or (i,j)

UOV = (i,j) - (k,l) where (k,l)∈DEAD set

Note: Sum of stencil dependences is always a UOV



OV-Mapped Version



Loop:

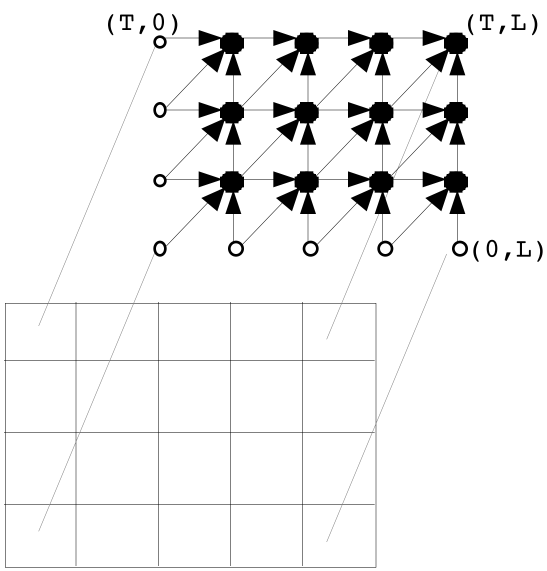
```
for i=1 to T
  for j=1 to L
    A[L-i+j] =
      f(A[L-(i-1)+j],
        A[L-i+(j-1)],
        A[L-(i-1)+(j-1)])
```

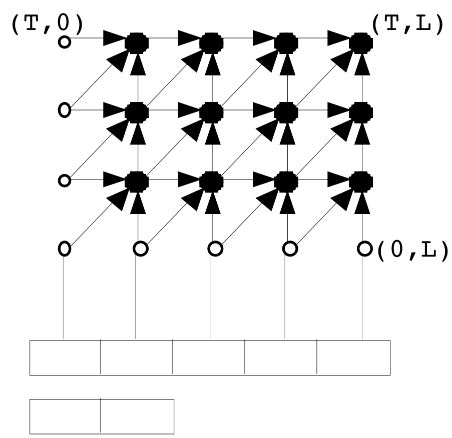
Observations:

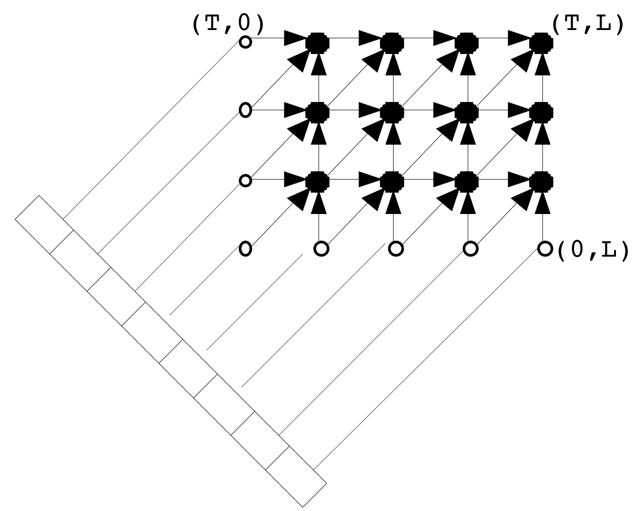
- Any legal schedule allowed
- Storage requirements: T+L+1

Example Summary

	Array Expansion	Storage Optimized	UOV-Mapped
Storage	TxL	L+2	T+L+1
Schedule	All	Only One	All





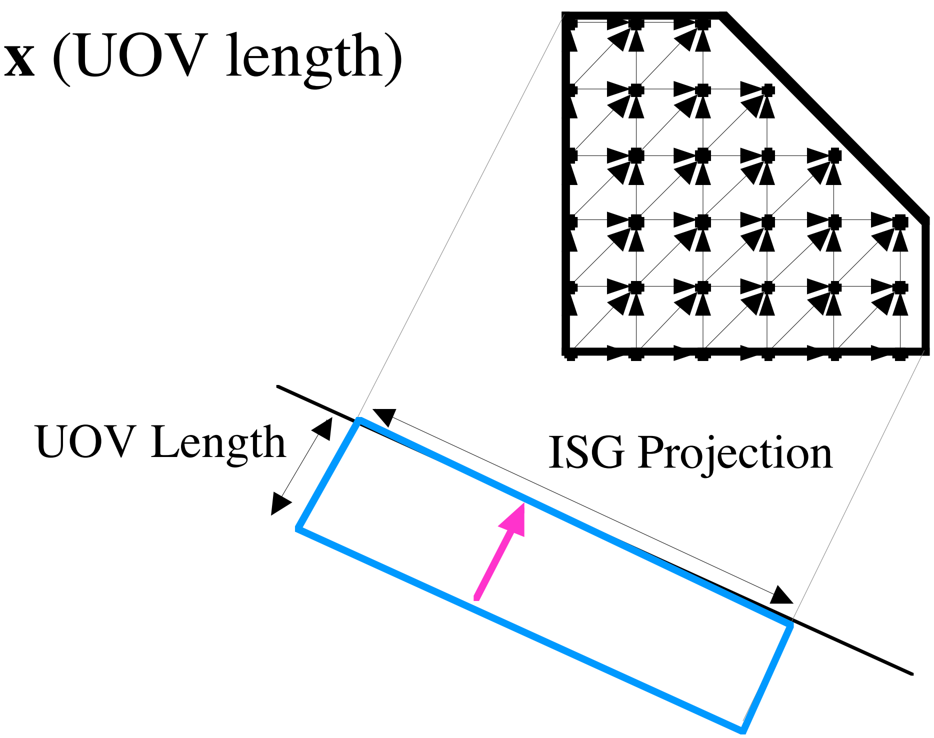


Minimizing Storage

Goal: Select UOV with minimum storage requirements

Fixed Size ISG
- Minimize (ISG Projection) **x** (UOV length)

Unknown Size ISG
- Minimize UOV length

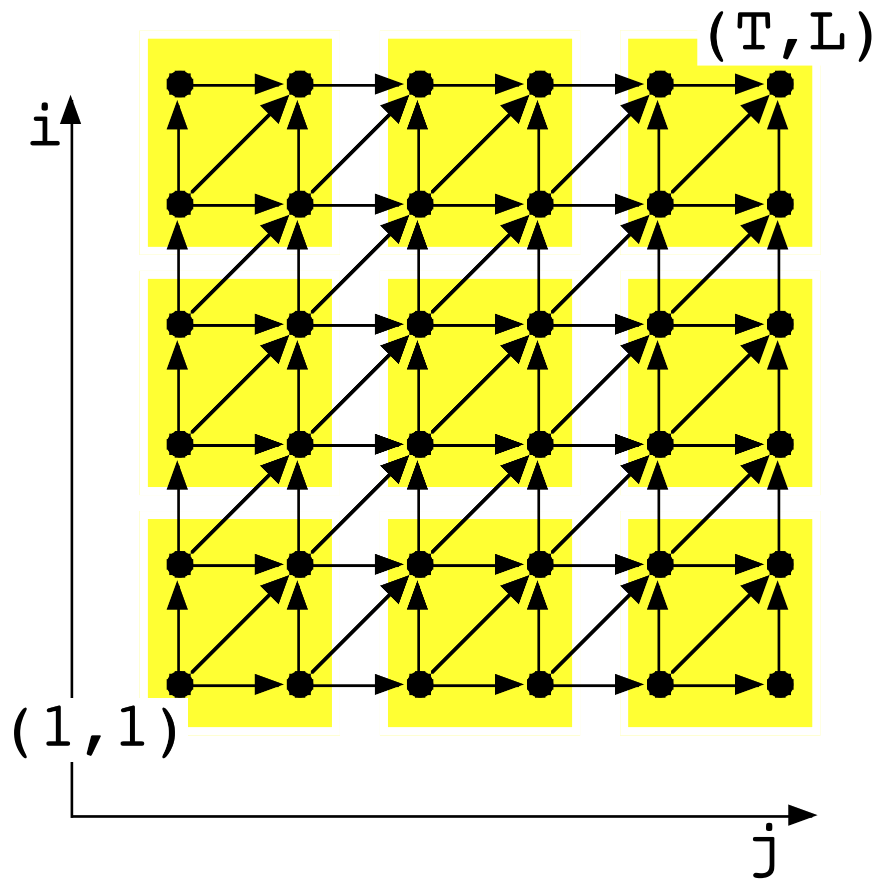


Tiling

Partition ISG into tiles.
Execute tiles atomically.

Why Tile?

- Data locality
- Coarse-grain parallelism
- Enhanced ILP

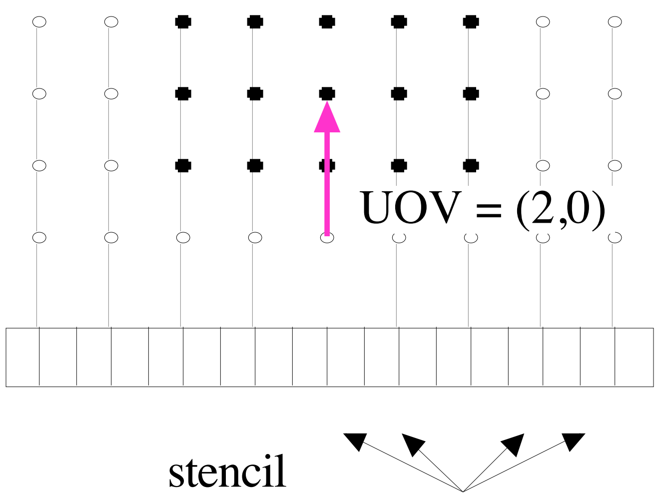


Experiments

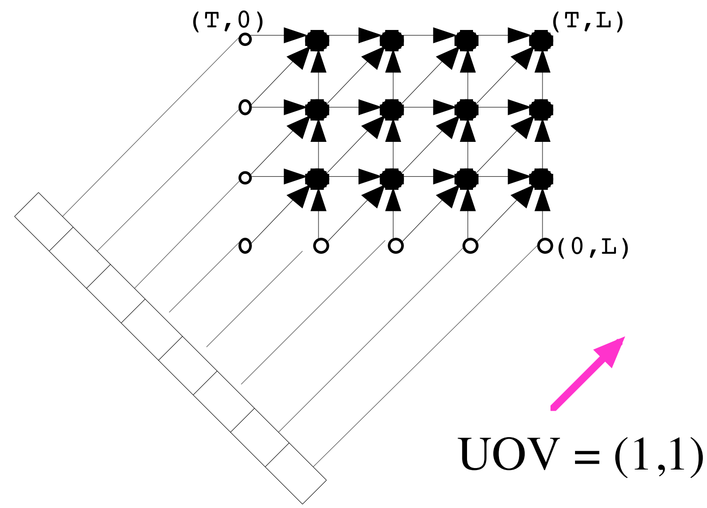
Results Show

- Minimal overhead
- Performance scales with problem size

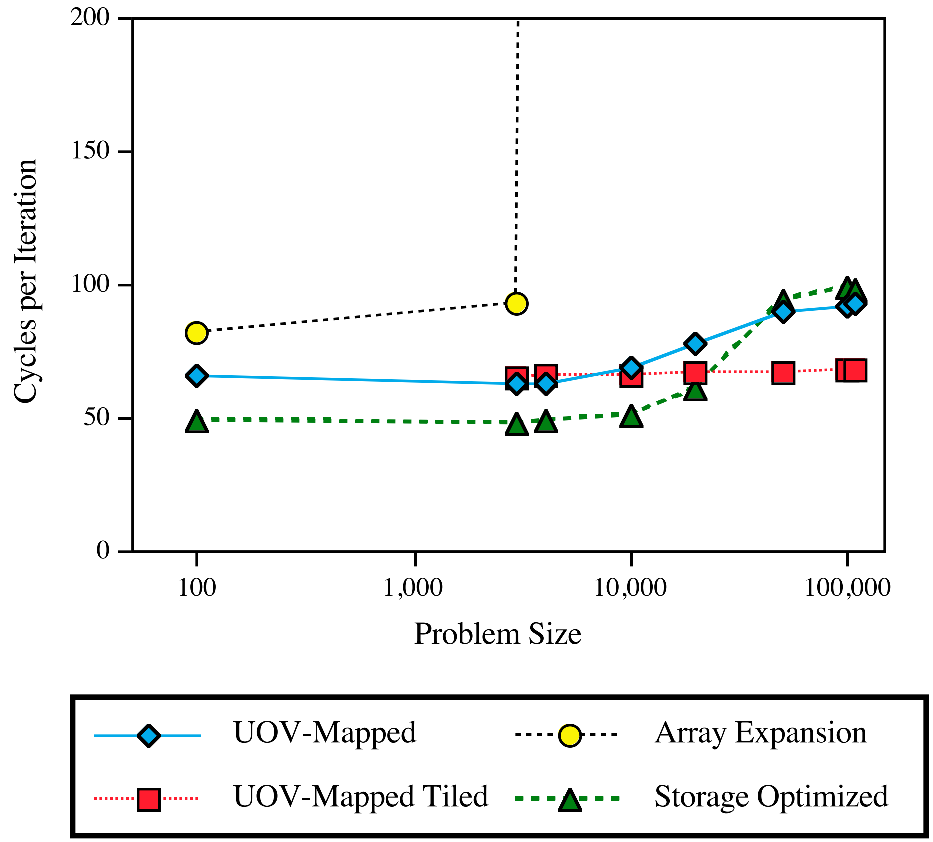
Simple Code



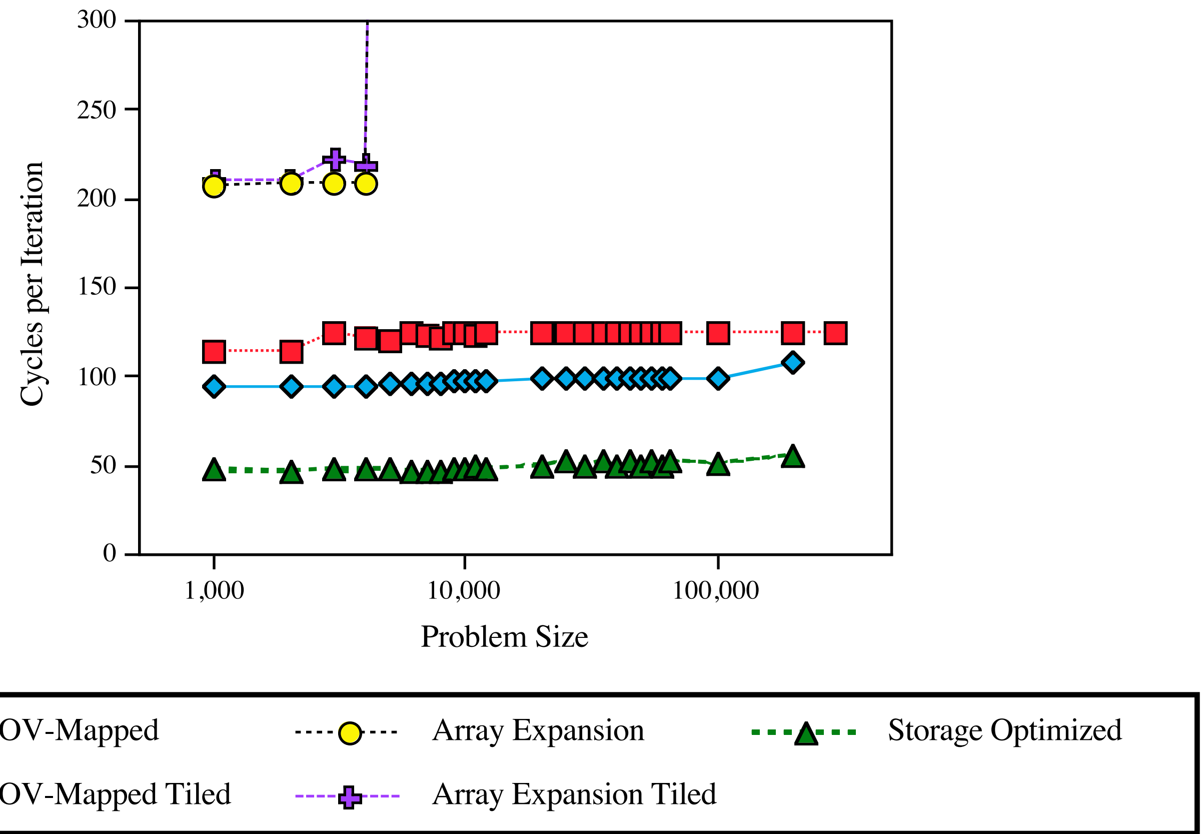
Protein String Matching



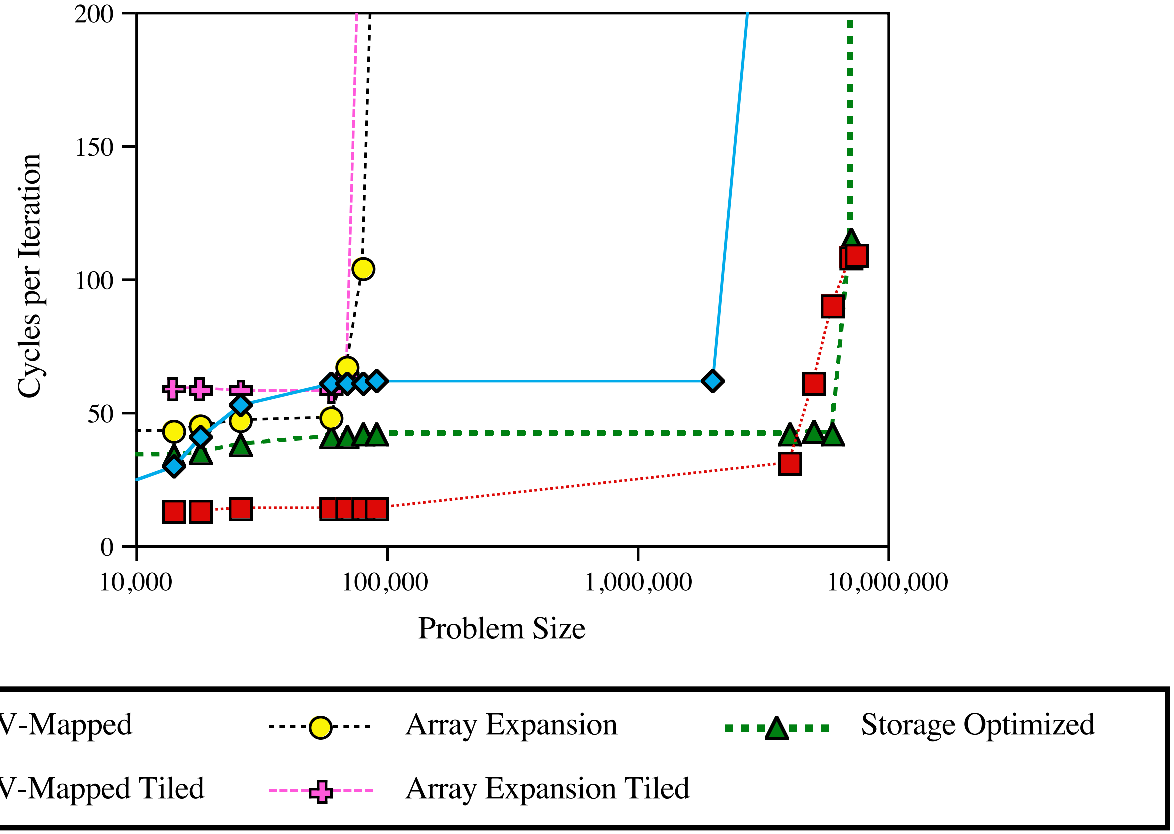
Protein String Matching
Pentium Pro



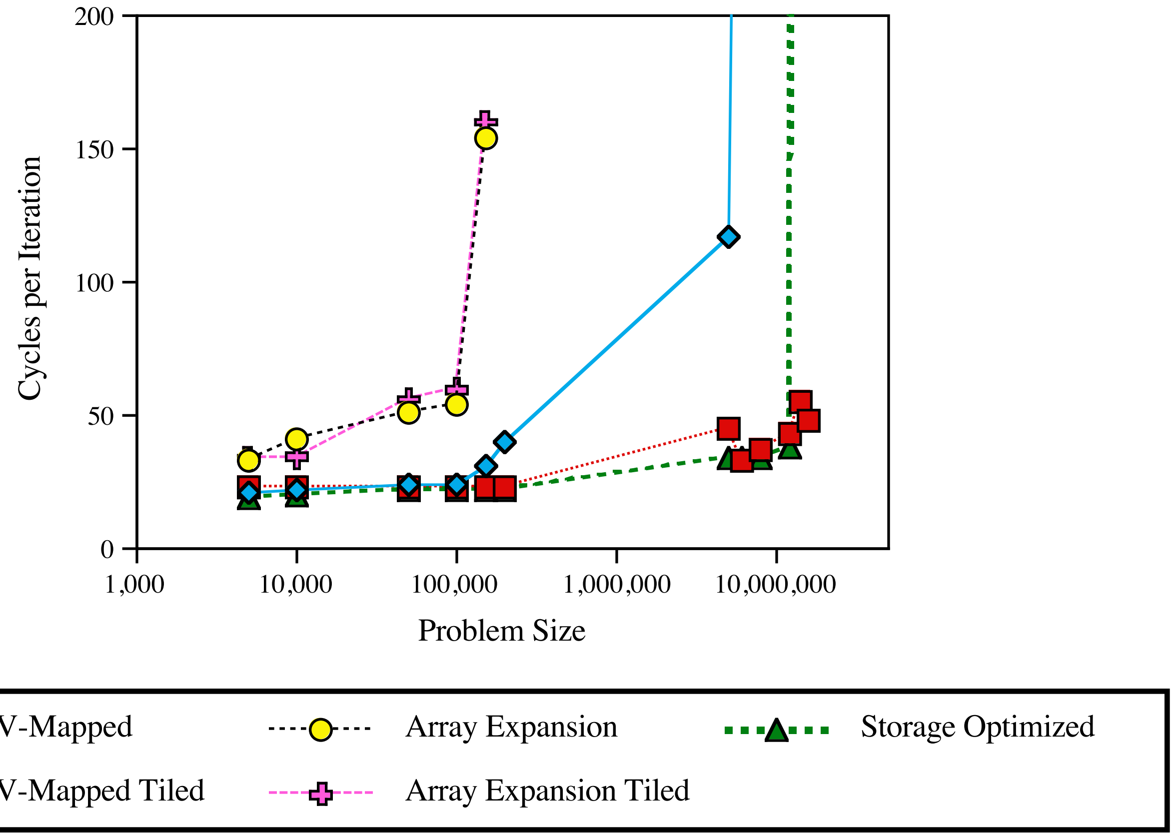
Protein String Matching
Alpha 21164



Simple Code
Pentium Pro



Simple Code
Alpha 21164



Read the Paper!

- Formalizes Occupancy Vector
- UOV recognition is NP-Complete
- Branch and bound algorithm to find UOV
- Code generate for UOV-mapped 2D loop
- More experimental results

Universal Occupancy Vector (UOV)

<First see the UOV slides>

Occupancy vector indicates iterations that share storage

- $\vec{ov} = \vec{q} - \vec{p}$, then iterations q and p share storage
- Pick a storage mapping by ensuring that $\vec{mv} \cdot \vec{ov} = 0$ in

$$SM_{ov}(\vec{q}) = \vec{mv} \cdot \vec{q} + shift + modterm$$

Determining if a storage mapping is legal

- For any schedule, universal occupancy vector
  - Let $V = \{\vec{v_1}, ..., \vec{v_m}\}$ be the set of value/exact flow dependences
  - A universal occupancy vector $\vec{ov}$ must satisfy the following set of equations with all a_{ij} being integers such that $a_{ii} > 0, a_{ij} \geq 0$

$$\begin{aligned} \vec{ov} &= a_{11}\vec{v_1} + ... + a_{1m}\vec{v_m} \\ &... \\ \vec{ov} &= a_{m1}\vec{v_1} + ... + a_{mm}\vec{v_m} \end{aligned}$$

- When a schedule is specified the introduced output dependence needs to be a subset of the current memory-based dependences

UOV 3/22/12

$V = \{\vec{v}_1, \ldots, \vec{v}_m\}$ set of flow deps

$\vec{y}, \vec{x}, \vec{q}, \vec{p}$ are iteration points

$$DONE(V, \vec{x}) = \{\vec{y} \mid \exists a_j \geq 0, \vec{y} + \sum_{j=1}^{m} a_j \vec{v}_j = \vec{x}\}$$

$$DEAD(V, \vec{q}) = \{\vec{p} \mid \forall \vec{v}_i \in V, \vec{p} + \vec{v}_i \in DONE(V, \vec{q})\}$$

$$UOV(V) = \{\vec{q} - \vec{p} \mid \vec{p} \in DEAD(V, \vec{q})\}$$

$$= \{\vec{q} - \vec{p} \mid \forall \vec{v}_i \in V, \underline{\vec{p} + \vec{v}_i} \in DONE(V, \vec{q})\}$$

$$= \{ \quad \mid \forall \vec{v}_i \in V, \exists a_{ij} \geq 0, \vec{p} + \vec{v}_i + \sum_{j=1}^{m} a_{ij} \vec{v}_j = \vec{q}\}$$

For each $\vec{ov}$ in UOV(v) we have integers

a_{ij} st

$$\vec{ov} = \vec{v}_i + \sum_{j=1}^{m} a_{ij} \vec{v}_j$$

$$\vec{ov} = \vec{v}_m + \sum_{j=1}^{m} a_{mj} \vec{v}_j$$

Fission Example

Example

```
do i = 1, n
  a[i] = a[i] + c
  x[i+1] = x[i]*7 + x[i+1] + a[i]
end do
```

In Alphabets

```
affine fission {N | N>1}
given
  int c;
  int ain {i|1<=i<=N};
  int xin {i|1<=i<=N+1};
returns
  int a {i|1<=i<=N};
  int x {i|2<=i<=N};
through
    x[i] = case
      {| i==2} : xin[i];
      {| i> 2} : x[i-1]*7 + xin[i] + ain[i-1];
    esac;
    a[i] = ain[i] + c;
.
```

AlphaZ Compiler Script

```
# Both statements in the same loop
setSpaceTimeMap(prog, system, "a", "(i->i,0)");
setSpaceTimeMap(prog, system, "x", "(i->i+1,1)");
setDimensionType(prog, system, "a,x", 0, "S");
setDimensionType(prog, system, "a,x", 1, "O");

# Fission
#setSpaceTimeMap(prog, system, "a", "(i->0,i)");
#setSpaceTimeMap(prog, system, "x", "(i->1,i)");
#setDimensionType(prog, system, "a,x", 0, "O");
#setDimensionType(prog, system, "a,x", 1, "S");
```

Stencil 1D Computation Example (Skewing)

Example

```
// assume u[i] initialized to some values
for (s=1; s<T; s+=2) {
  for (i=1; i<(N-1); i++) {
    tmp[i] = 1/3 * (u[i-1] + u[i] + u[i+1]); // S1
  }
 for (j=1; j<(N-1); j++) {
    u[i] = 1/3 * (tmp[j-1] + tmp[j] + tmp[j+1]); // S2
  }
}
```

In Alphabets

```
# Alphabets code given on progress page as stencil1D.ab.

# Skewing the i loop.
setSpaceTimeMap(prog, system, "temp", "(s,i->s,i+s)");
setSpaceTimeMap(prog, system, "U", "(i->T+1,i+T+1)");
setDimensionType(prog, system, "temp,U", 0, "S");
setDimensionType(prog, system, "temp,U", 1, "P");
```

Algorithms needed for automation

Operations on sets and relations

- Union iteration space sets
- Union relations that represent dependences
- Apply a relation to a set to model transforming a loop and to check transformation legality
- Compose two relations to model composing transformations
- Is one relation a subset of another relation for checking the legality of occupancy vectors

Scheduling

- Determine an efficient and legal schedule
- Determine which loops should be parallel

Storage Mapping

- If not using UOV, then need to do this in coordination with the scheduling

Code Generation

- Given a schedule and which loops to parallelize and/or tile, generate efficient code
- Code generation for parameterized tiles

Next Time

Lecture

- Operations on polyhedral sets and relations

Schedule

- Quiz 2 due March 23rd
- Project intermediate report due March 28th
- April 3rd will be a lab day during class, Manaf will help people with AlphaZ and Pluto. Distance students can email questions or use the discussion board.
- HW6 and HW7 will BOTH be due April 4th