Workshop on Essential Abstractions in GCC

Introduction to Parallelization and Vectorization

GCC Resource Center
(www.cse.iitb.ac.in/grc)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

3 July 2011

Outline

- Transformation for parallel and vector execution
- Data dependence

3 July 2011 intro-par-vect: Outline 1/28

Outline

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

The Scope of this Tutorial

- What this tutorial does not address
  - Algorithms used for parallelization and vectorization
  - Code or data structures of the parallelization and vectorization pass of GCC
  - Machine level issues related to parallelization and vectorization
- What this tutorial addresses

  Basics of Discovering Parallelism using GCC

3 July 2011 intro-par-vect: Outline 2/28

The Scope of this Tutorial

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Part 1

Transformations for Parallel and Vector Execution

Notes

A Taxonomy of Parallel Computation

	Single Program	Multiple Programs
Single Data	SPSD	MPSD
Multiple Data	SPMD	MPMD

3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 3/28

A Taxonomy of Parallel Computation

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 3/28

A Taxonomy of Parallel Computation

	Single Program		Multiple Programs
	Single Instruction	Multiple Instructions	
Single Data	SISD	MISD	MPSD
Multiple Data	SIMD	MIMD	MPMD

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 3/28

A Taxonomy of Parallel Computation

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

A Taxonomy of Parallel Computation

	Single Program		Multiple Programs
	Single Instruction	Multiple Instructions	
Single Data	SISD	?	?
Multiple Data	SIMD	MIMD	MPMD

Redundant computation for validation of intermediate steps

A Taxonomy of Parallel Computation

Notes

A Taxonomy of Parallel Computation

	Single Program		Multiple Programs
	Single Instruction	Multiple Instructions	
Single Data	SISD	MISD	MPSD
Multiple Data	SIMD	MIMD	MPMD

Transformations performed by a compiler

A Taxonomy of Parallel Computation

Notes

Vectorization: SISD ⇒ SIMD

- Parallelism in executing operation on shorter operands (8-bit, 16-bit, 32-bit operands)
- Existing 32 or 64-bit arithmetic units used to perform multiple operations in parallel
  A 64 bit word ≡ a vector of 2×(32 bits), 4×(16 bits), or 8×(8 bits)

Vectorization: SISD ⇒ SIMD

Notes

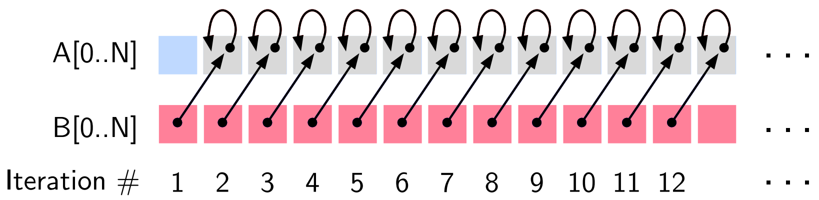
Example 1

Vectorization	(SISD ⇒ SIMD)	: Yes
Parallelization	(SISD ⇒ MIMD)	: Yes

Original Code

```
int A[N], B[N], i;
for (i=1; i<N; i++)
  A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location



Example 1

Notes

Example 1

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : Yes

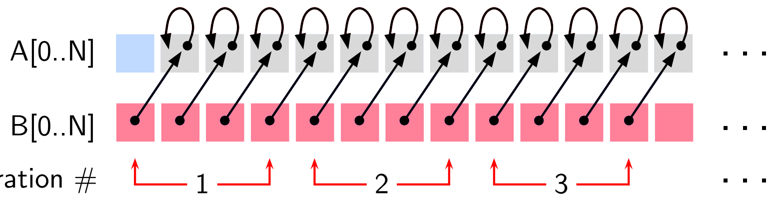
Vectorization Factor

Original Code

```
int A[N], B[N], i;
for (i=1; i<N; i++)
  A[i] = A[i] + B[i-1];
```

Vectorized Code

```
int A[N], B[N], i;
for (i=1; i<N; i=i+4)
  A[i:i+3] = A[i:i+3] +
B[i-1:i+2];
```



Example 1

Notes

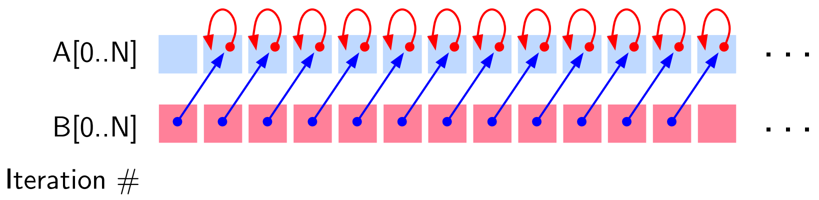
Example 1

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : Yes

Original Code

```
int A[N], B[N], i;
for (i=1; i<N; i++)
  A[i] = A[i] + B[i-1];
```

Observe reads and writes into a given location



Example 1

Notes

Example 1

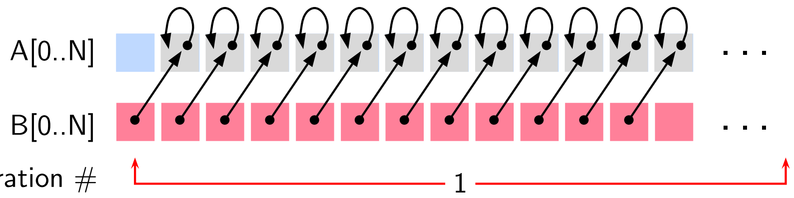
Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : Yes

Original Code

```
int A[N], B[N], i;
for (i=1; i<N; i++)
  A[i] = A[i] + B[i-1];
```

Parallelized Code

```
int A[N], B[N], i;
foreach (i=1; i<N; )
  A[i] = A[i] + B[i-1];
```



3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 6/28

Example 1

Notes

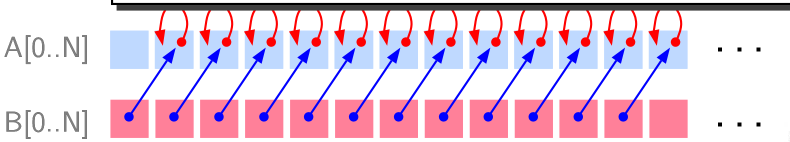
Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Example 1: The Moral of the Story

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : Yes

When the same location is accessed across different iterations, the order of reads and writes must be preserved

Nature of accesses in our example		
Iteration i	Iteration $i+k$	Observation
Read	Write	No
Write	Read	No
Write	Write	No
Read	Read	Does not matter



3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 7/28

Example 1: The Moral of the Story

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

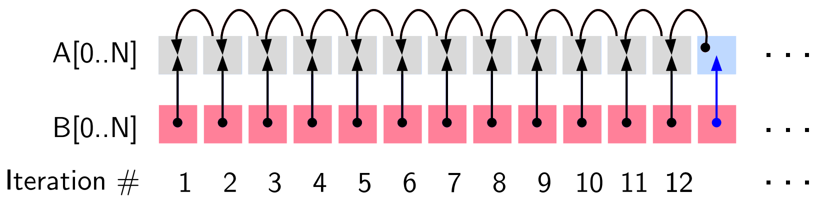
Example 2

Vectorization	(SISD ⇒ SIMD)	: Yes
Parallelization	(SISD ⇒ MIMD)	: No

Original Code

```
int A[N], B[N], i;
for (i=0; i<N; i++)
  A[i] = A[i+1] + B[i];
```

Observe reads and writes into a given location



3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 8/28

Example 2

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 8/28

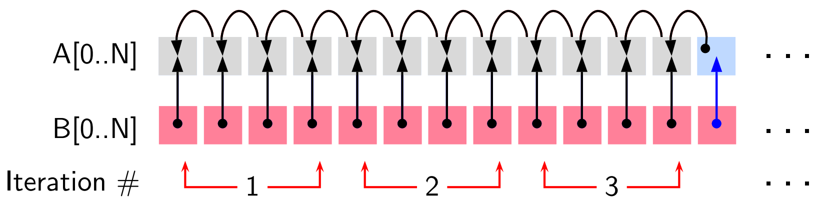
Example 2

Vectorization	(SISD ⇒ SIMD)	: Yes
Parallelization	(SISD ⇒ MIMD)	: No

Original Code

```
int A[N], B[N], i;
for (i=0; i<N; i++)
  A[i] = A[i+1] + B[i];
```

- Vector instruction is synchronized: All reads before writes in a given instruction
- Read-writes across multiple instructions executing in parallel may not be synchronized



Essential Abstractions in GCC GCC Resource Center, IIT Bombay

3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 8/28

Example 2

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

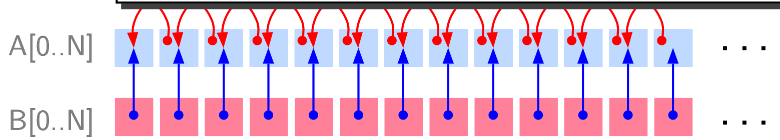
Example 2: The Moral of the Story

Vectorization (SISD ⇒ SIMD) : Yes
Parallelization (SISD ⇒ MIMD) : **No**

int A[
for (i
A[i]

When the same location is accessed across different iterations, the order of reads and writes must be preserved

Nature of accesses in our example		
Iteration i	Iteration $i + k$	Observation
Read	Write	Yes
Write	Read	No
Write	Write	No
Read	Read	Does not matter



Example 2: The Moral of the Story

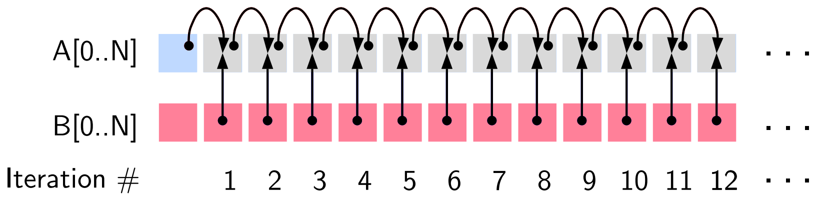
Notes

Example 3

Vectorization (SISD ⇒ SIMD) : No
Parallelization (SISD ⇒ MIMD) : No

```
int A[N], B[N], i;
for (i=0; i<N; i++)
  A[i+1] = A[i] + B[i+1];
```

Observe reads and writes into a given location



Example 3

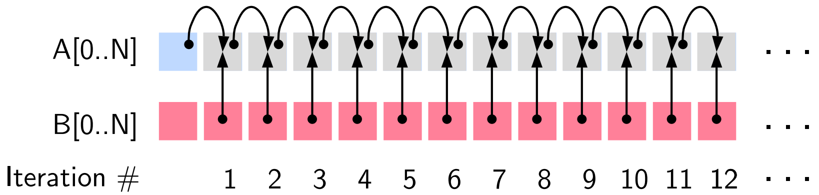
Notes

Example 3

Vectorization (SISD ⇒ SIMD) : **No**
Parallelization (SISD ⇒ MIMD) : **No**

```
int A[N], B
for (i=0; i
  A[i+1] =
```

Nature of accesses in our example		
Iteration i	Iteration $i+k$	Observation
Read	Write	No
Write	Read	Yes
Write	Write	No
Read	Read	Does not matter



3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 10/28

Example 3

Notes

Example 4

Vectorization (SISD ⇒ SIMD) : No
Parallelization (SISD ⇒ MIMD) : Yes

- This case is not possible
- Vectorization is a limited granularity parallelization
- If parallelization is possible then vectorization is trivially possible
- Blank line to increase height
- Blank line to increase height

3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 11/28

Example 4

Notes

Data Dependence

Let statements S_i and S_j access memory location m at time instants t and $t + k$

Access in S_i	Access in S_j	Dependence	Notation
Read m	Write m	Anti (or Pseudo)	$S_i\ \bar{\delta}\ S_j$
Write m	Read m	Flow (or True)	$S_i\ \delta\ S_j$
Write m	Write m	Output (or Pseudo)	$S_i\ \delta^O\ S_j$
Read m	Read m	Does not matter	

- Pseudo dependences may be eliminated by some transformations
- True dependence prohibits parallel execution of S_i and S_j

3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 12/28

Data Dependence

Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Consider dependence between statements S_i and S_j in a loop

- Loop independent dependence. t and $t + k$ occur in the same iteration of a loop
  - S_i and S_j must be executed sequentially
  - Different iterations of the loop can be parallelized
- Loop carried dependence. t and $t + k$ occur in the different iterations of a loop
  - Within an iteration, S_i and S_j can be executed in parallel
  - Different iterations of the loop must be executed sequentially
- S_i and S_j may have both loop carried and loop independent dependences

3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 13/28

Notes

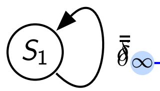
Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Dependence in Example 1

- Program

```
int A[N], B[N], i;
for (i=1; i<N; i++)
    A[i] = A[i] + B[i-1];   /* S1 */
```

- Dependence graph



Dependence in the same iteration

- No loop carried dependence
  Both vectorization and parallelization are possible

Dependence in Example 1

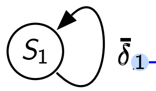
Notes

Dependence in Example 2

- Program

```
int A[N], B[N], i;
  for (i=0; i<N; i++)
    A[i] = A[i+1] + B[i];  /* S1 */
```

- Dependence graph



Dependence due to the outermost loop

- Loop carried anti-dependence
  Parallelization is not possible
  Vectorization is possible since all reads are done before all writes

Dependence in Example 2

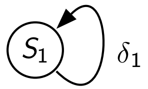
Notes

Dependence in Example 3

- Program

```
int A[N], B[N], i;
  for (i=0; i<N; i++)
    A[i+1] = A[i] + B[i+1];   /* S1 */
```

- Dependence graph



- Loop carried flow-dependence
  Neither parallelization not vectorization is possible

Dependence in Example 3

Notes

Iteration Vectors and Index Vectors: Example 1

```
for (i=0, i<4; i++)
  for (j=0; j<4; j++)
  {
     a[i+1][j] = a[i][j] + 2;
  }
```

Iteration Vector	Index Vector LHS	Index Vector RHS
0, 0	1, 0	0, 0
0, 1	1, 1	0, 1
0, 2	1, 2	0, 2
0, 3	1, 3	0, 3
1, 0	2, 0	1, 0
1, 1	2, 1	1, 1
1, 2	2, 2	1, 2
1, 3	2, 3	1, 3
2, 0	3, 0	2, 0
2, 1	3, 1	2, 1
2, 2	3, 2	2, 2
2, 3	3, 3	2, 3
3, 0	4, 0	3, 0
3, 1	4, 1	3, 1
3, 2	4, 2	3, 2
3, 3	4, 3	3, 3

Loop carried dependence exists if

- there are two distinct iteration vectors such that
- the index vectors of LHS and RHS are identical

Conclusion: Dependence exists

Iteration Vectors and Index Vectors: Example 1

Notes

Iteration Vectors and Index Vectors: Example 2

```
for (i=0, i<4; i++)
  for (j=0; j<4; j++)
  {
     a[i][j] = a[i][j] + 2;
  }
```

Loop carried dependence exists if

- there are two distinct iteration vectors such that
- the index vectors of LHS and RHS are identical

Conclusion: No dependence

Iteration Vector	Index Vector LHS	Index Vector RHS
0, 0	0, 0	0, 0
0, 1	0, 1	0, 1
0, 2	0, 2	0, 2
0, 3	0, 3	0, 3
1, 0	1, 0	1, 0
1, 1	1, 1	1, 1
1, 2	1, 2	1, 2
1, 3	1, 3	1, 3
2, 0	2, 0	2, 0
2, 1	2, 1	2, 1
2, 2	2, 2	2, 2
2, 3	2, 3	2, 3
3, 0	3, 0	3, 0
3, 1	3, 1	3, 1
3, 2	3, 2	3, 2
3, 3	3, 3	3, 3

Iteration Vectors and Index Vectors: Example 2

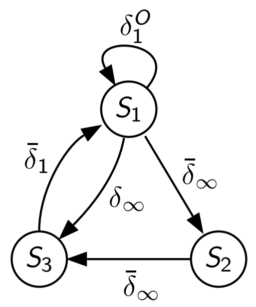
Notes

Example 4: Dependence

Program to swap arrays

```
for (i=0; i<N; i++)
{
    T = A[i];        /* S1 */
    A[i] = B[i];     /* S2 */
    B[i] = T;        /* S3 */
}
```

Dependence Graph



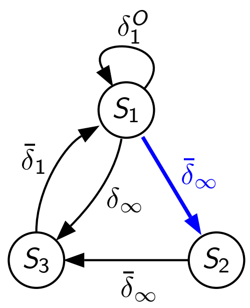
Example 4: Dependence

Notes

Example 4: Dependence

Program to swap arrays	Dependence Graph

```
for (i=0; i<N; i++)
{
    T = A[i];          /* S1 */
    A[i] = B[i];       /* S2 */
    B[i] = T;          /* S3 */
}
```



Loop independent anti dependence due to `A[i]`

3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 19/28

Example 4: Dependence

Notes

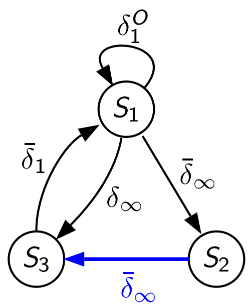
Essential Abstractions in GCC GCC Resource Center, IIT Bombay

3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 19/28

Example 4: Dependence

Program to swap arrays	Dependence Graph

```
for (i=0; i<N; i++)
{
    T = A[i];          /* S1 */
    A[i] = B[i];       /* S2 */
    B[i] = T;          /* S3 */
}
```



Loop independent anti dependence due to `B[i]`

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 19/28

Example 4: Dependence

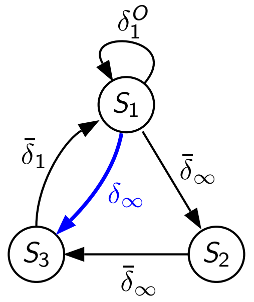
Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Example 4: Dependence

Program to swap arrays	Dependence Graph

```
for (i=0; i<N; i++)
{
    T = A[i];          /* S1 */
    A[i] = B[i];       /* S2 */
    B[i] = T;          /* S3 */
}
```



Loop independent flow dependence due to T

3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 19/28

Example 4: Dependence

Notes

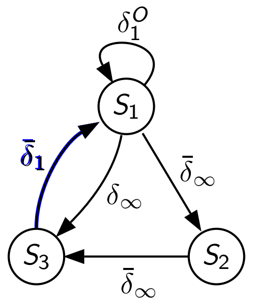
Essential Abstractions in GCC GCC Resource Center, IIT Bombay

3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 19/28

Example 4: Dependence

Program to swap arrays	Dependence Graph

```
for (i=0; i<N; i++)
{
    T = A[i];          /* S1 */
    A[i] = B[i];       /* S2 */
    B[i] = T;          /* S3 */
}
```



Loop carried anti dependence due to T

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

3 July 2011 intro-par-vect: Introduction to Parallelization and Vectorization 19/28

Example 4: Dependence

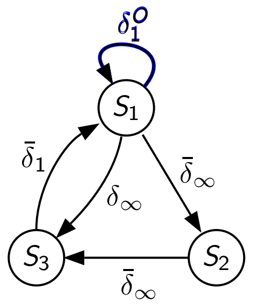
Notes

Essential Abstractions in GCC GCC Resource Center, IIT Bombay

Example 4: Dependence

Program to swap arrays	Dependence Graph

```
for (i=0; i<N; i++)
{
    T = A[i];         /* S1 */
    A[i] = B[i];      /* S2 */
    B[i] = T;         /* S3 */
}
```



Loop carried output dependence due to T

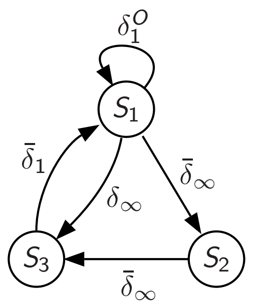
Example 4: Dependence

Notes

Example 4: Dependence

Program to swap arrays | Dependence Graph

for (i=0; i<N; i++)
{
    T = A[i];   /* S1 */
    A[i] = B[i];   /* S2 */
    B[i] = T;   /* S3 */
}



Example 4: Dependence

Notes

Tutorial Problem for Discovering Dependence

Draw the dependence graph for the following program
(Earlier program modified to swap 2-dimensional arrays)

```
for (i=0; i<N; i++)
{
    for (j=0; j<N; j++)
    {   T = A[i][j];       /* S1 */
        A[i][j] = B[i][j]; /* S2 */
        B[i][j] = T;       /* S3 */
    }
}
```


Tutorial Problem for Discovering Dependence

Notes

Data Dependence Theorem

There exists a dependence from statement S_1 to statement S_2 in common nest of loops if and only if there exist two iteration vectors **i** and **j** for the nest, such that

1. $\mathbf{i} < \mathbf{j}$ or $\mathbf{i} = \mathbf{j}$ and there exists a path from S_1 to S_2 in the body of the loop,
2. statement S_1 accesses memory location M on iteration **i** and statement S_2 accesses location M on iteration **j**, and
3. one of these accesses is a write access.

Data Dependence Theorem

Notes

Anti Dependence and Vectorization

Read lexicographically precedes Write

```
int A[N], B[N], C[N], i;
for (i=0; i<N; i++) {
   C[i] = A[i+2];
   A[i] = B[i];
}
```

→

```
int A[N], B[N], C[N], i;
for (i=0; i<N; i=i+4) {
   C[i:i+3] = A[i+2:i+5];
   A[i:i+3] = B[i:i+3];
}
```


Anti Dependence and Vectorization

Notes

Anti Dependence and Vectorization

Write lexicographically precedes Read

```
int A[N], B[N], C[N], i;
for (i=0; i<N; i++) {
   A[i] = B[i];
   C[i] = A[i+2];
}
```

→

```
int A[N], B[N], C[N], i;
for (i=0; i<N; i++) {
   C[i] = A[i+2];
   A[i] = B[i];
}
```

↓

```
int A[N], B[N], C[N], i;
for (i=0; i<N; i=i+4) {
   C[i:i+3] = A[i+2:i+5];
   A[i:i+3] = B[i:i+3];
}
```


Anti Dependence and Vectorization

Notes

True Dependence and Vectorization

Write lexicographically precedes Read

```
int A[N], B[N], C[N], i;
for (i=0; i<N; i++) {
   A[i+2] = C[i];
   B[i] = A[i];
}
```

→

```
int A[N], B[N], C[N], i;
for (i=0; i<N; i=i+4) {
   A[i+2:i+5] = C[i:i+3];
   B[i:i+3] = A[i:i+3];
}
```


True Dependence and Vectorization

Notes

Conjunction of Dependences and Vectorization

Anti Dependence and True Dependence

```
int A[N], i;
for (i=0; i<N; i++) {
   A[i] = A[i+2];
}
```

→

```
int A[N], i, temp;
for (i=0; i<N; i++) {
   temp = A[i+2];
   A[i] = temp;
}
```

↓

```
int A[N], T[N], i;
for (i=0; i<N; i++) {
   T[i] = A[i+2];
   A[i] = T[i];
}
```

→

```
int A[N], T[N], i;
for (i=0; i<N; i=i+4) {
   T[i:i+3] = A[i+2:i+5];
   A[i:i+3] = T[i:i+3];
}
```


Conjunction of Dependences and Vectorization

Notes

Conjunction of Dependences and Vectorization

True Dependence and Anti Dependence

```
int A[N], B[N], i;
for (i=0; i<N; i++) {
   A[i] = B[i];
   B[i+2] = A[i+1];
}
```

```
int A[N], B[N], i;
for (i=0; i<N; i++) {
   B[i+2] = A[i+1];
   A[i] = B[i];
}
```

```
int A[N], B[N], i;
for (i=0; i<N; i=i+4) {
   B[i+2:i+5] = A[i+1:i+4];
   A[i:i+3] = B[i:i+3];
}
```


Conjunction of Dependences and Vectorization

Notes

Cyclic Dependency and Vectorization

Cyclic True Dependence

```
int A[N], B[N], i;
for (i=0; i<N; i++) {
   B[i+2] = A[i];
   A[i+1] = B[i];
}
```

Cyclic Anti Dependence

```
int A[N], B[N], i;
for (i=0; i<N; i++) {
   B[i] = A[i+1];
   A[i] = B[i+2];
}
```

Rescheduling of statements will not break the cyclic dependency - cannot vectorize

Cyclic Dependency and Vectorization

Notes

Last but not the least . . .

Thank You!