#### 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

intro-par-vect: Outline

**Outline** 

- Transformation for parallel and vector execution
- Data dependence

3 July 2011

1/28

- 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



2/28

3 July 2011

#### Part 1

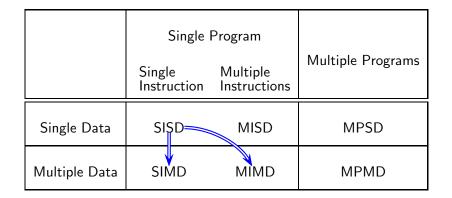
# Transformations for Parallel and Vector Execution

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

	Single Program		6
	Single Instruction	Multiple Instructions	Multiple Programs
Single Data	SISD	MISD	MPSD
Multiple Data	SIMD	MIMD	MPMD

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

Redundant computation for validation of intermediate steps



Transformations performed by a compiler

#### **Vectorization:** SISD ⇒ SIMD

(8-bit, 16-bit, 32-bit operands)Existing 32 or 64-bit arithmetic units used to perform multiple

Parallelism in executing operation on shorter operands

• Existing 32 or 64-bit arithmetic units used to perform multiple operations in parallel A 64 bit word  $\equiv$  a vector of 2×(32 bits), 4×(16 bits), or 8×(8 bits)

5/28

## Example 1

```
 \begin{array}{ll} \text{Vectorization} & (\mathsf{SISD} \Rightarrow \mathsf{SIMD}) & : \; \mathsf{Yes} \\ \mathsf{Parallelization} & (\mathsf{SISD} \Rightarrow \mathsf{MIMD}) & : \; \mathsf{Yes} \\ \end{array}
```

Original Code

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

Example 1

3 July 2011

5/28

Vectorization  $(SISD \Rightarrow SIMD)$ : Yes  $(SISD \Rightarrow MIMD)$ : Yes Parallelization

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

A[0..N]























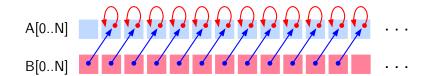
#### Example 1

Vectorization  $(SISD \Rightarrow SIMD)$ : Yes  $(SISD \Rightarrow MIMD)$ : Yes Parallelization

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



5/28

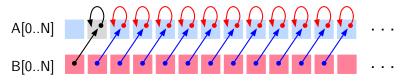
#### Example 1

Vectorization  $(SISD \Rightarrow SIMD)$ : Yes  $(SISD \Rightarrow MIMD)$ : Yes Parallelization

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



Iteration #

3 July 2011

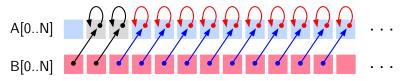
### Example 1

Vectorization  $(SISD \Rightarrow SIMD)$ : Yes  $(SISD \Rightarrow MIMD)$ : Yes Parallelization

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



Iteration # 1 2

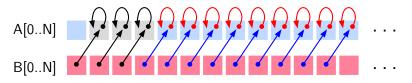
#### Example 1

Vectorization  $(SISD \Rightarrow SIMD)$ : Yes  $(SISD \Rightarrow MIMD)$ : Yes Parallelization

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



Iteration # 1 2 3

5/28

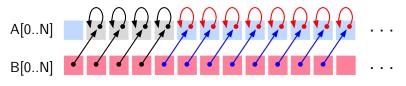
#### Example 1

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  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



Iteration # 1 2 3 4

3 July 2011

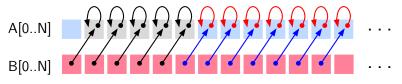
#### Example 1

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  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



Iteration # 1 2 3 4

3 July 2011

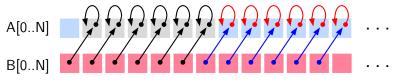
#### Example 1

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  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



Iteration # 1 2 3 4 5

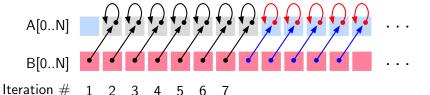
#### Example 1

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  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



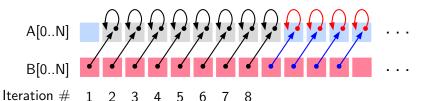
**Essential Abstractions in GCC** 

#### Example 1

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  MIMD) : Yes

Original Code

Observe reads and writes into a given location



Essential Abstractions in GCC

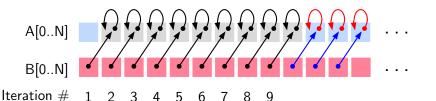
#### Example 1

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  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



**Essential Abstractions in GCC** 

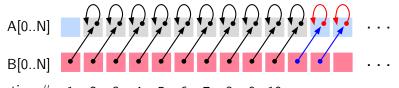
#### Example 1

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  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



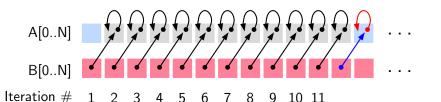
Iteration # 1 2 3 4 5 6 7 8 9 10

#### Example 1

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  MIMD) : Yes

Original Code

Observe reads and writes into a given location

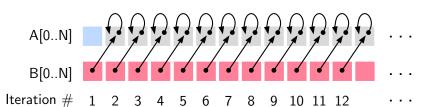


#### Example 1

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  MIMD) : Yes

Original Code

Observe reads and writes into a given location



#### Example 1

Vectorization  $(SISD \Rightarrow SIMD)$ : Yes  $(SISD \Rightarrow MIMD)$ : Yes Parallelization

Vectorization **Factor** 

5/28

Original Code

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

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];

Vectorized Code

A[0..N]B[0..N]

Iteration #

#### Example 1

Vectorization  $(SISD \Rightarrow SIMD)$ : Yes  $(SISD \Rightarrow MIMD)$ : Yes Parallelization

Vectorization **Factor** Vectorized Code

5/28

Original Code

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

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];

A[i] = A[i] + B[i-1];

A[0..N]B[0..N]

Iteration #

#### Example 1

Vectorization  $(SISD \Rightarrow SIMD)$ : Yes  $(SISD \Rightarrow MIMD)$ : Yes Vectorization Parallelization **Factor** Original Code Vectorized Code int A[N], B[N], i; int A[N], B[N], i; for (i=1: i<N: i=i+(4))for (i=1: i<N: i++) A[i:i+3] = A[i:i+3] +A[i] = A[i] + B[i-1];B[i-1:i+2]; A[0..N]B[0..N] 1 — 1 — 2 – Iteration #

#### Example 1

Vectorization  $(SISD \Rightarrow SIMD)$ : Yes  $(SISD \Rightarrow MIMD)$ : Yes Parallelization

Vectorization **Factor** Vectorized Code

5/28

Original Code

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

B[i-1:i+2];

A[i:i+3] = A[i:i+3] +

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

Iteration #

A[0..N]

6/28

 $\begin{array}{ll} \text{Vectorization} & (\text{SISD} \Rightarrow \text{SIMD}) & : \text{ Yes} \\ \text{Parallelization} & (\text{SISD} \Rightarrow \text{MIMD}) & : \text{ Yes} \\ \end{array}$ 

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

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  MIMD) : Yes

Original Code

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

Observe reads and writes into a given location



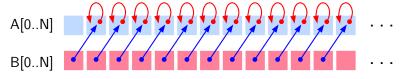
#### Example 1

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  MIMD) : Yes

Original Code

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

Observe reads and writes into a given location



Iteration #

3 July 2011

#### Example 1

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  MIMD) : Yes

Original Code

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

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

Parallelized Code

```
A[0..N]

B[0..N]

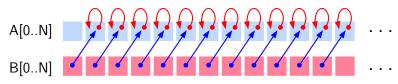
Iteration #
```

# **Example 1: The Moral of the Story**

Vectorization  $(SISD \Rightarrow SIMD)$ : Yes  $(SISD \Rightarrow MIMD)$ : Yes Parallelization

```
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

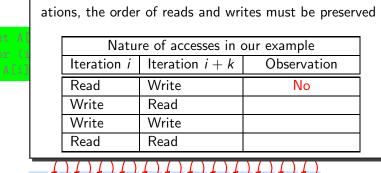


Vectorization  $(SISD \Rightarrow SIMD)$  : Yes  $(SISD \Rightarrow MIMD)$  : Yes Parallelization

> 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 + kObservation Read Write Write Read Write Write Read Read A[0..N]

#### **Example 1: The Moral of the Story**

Vectorization  $(SISD \Rightarrow SIMD)$  : Yes  $(SISD \Rightarrow MIMD)$  : Yes Parallelization



When the same location is accessed across different iter-

A[0..N]

### **Example 1: The Moral of the Story**

Vectorization  $(SISD \Rightarrow SIMD)$  : Yes  $(SISD \Rightarrow MIMD)$  : Yes Parallelization

> ations, the order of reads and writes must be preserved Nature of accesses in our example Iteration i Iteration i + kObservation Read Write No Write Read No Write Write Read Read

When the same location is accessed across different iter-

A[0..N]

# **Example 1: The Moral of the Story**

Vectorization  $(SISD \Rightarrow SIMD)$  : Yes  $(SISD \Rightarrow MIMD)$  : Yes Parallelization

> ations, the order of reads and writes must be preserved Nature of accesses in our example Iteration i Iteration i + kObservation Read Write No Write Read Nο Write Write No Read Read

When the same location is accessed across different iter-

A[0..N]

B[0..N]

# **Example 1: The Moral of the Story**

Read

Vectorization  $(SISD \Rightarrow SIMD)$  : Yes  $(SISD \Rightarrow MIMD)$  : Yes Parallelization

Read

ations, the order of reads and writes must be preserved Nature of accesses in our example Iteration i Iteration i + kObservation Write No Read Write Read Nο Write Write No

When the same location is accessed across different iter-

A[0..N]

B[0..N]

Does not matter

#### Example 2

Vectorization  $(SISD \Rightarrow SIMD)$ : Yes  $(SISD \Rightarrow MIMD)$ : No Parallelization

Original Code

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

8/28

### Example 2

 $\begin{array}{ll} \text{Vectorization} & (\mathsf{SISD} \Rightarrow \mathsf{SIMD}) & : \; \mathsf{Yes} \\ \mathsf{Parallelization} & (\mathsf{SISD} \Rightarrow \mathsf{MIMD}) & : \; \mathsf{No} \\ \end{array}$ 

Original Code

Observe reads and writes into a given location

A[0..N] ...

8/28

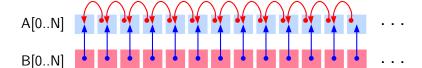
# Example 2

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  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



Iteration #

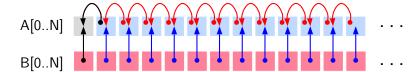
# Example 2

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  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



Iteration #

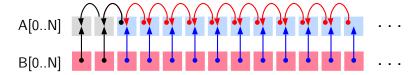
# Example 2

 $\begin{array}{lll} \mbox{Vectorization} & (\mbox{SISD} \Rightarrow \mbox{SIMD}) & : \mbox{Yes} \\ \mbox{Parallelization} & (\mbox{SISD} \Rightarrow \mbox{MIMD}) & : \mbox{No} \\ \end{array}$ 

#### 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



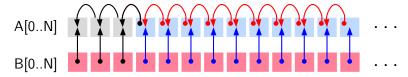
Iteration # 1

### Example 2

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  MIMD) : No

#### Original Code

Observe reads and writes into a given location



Iteration # 1 2

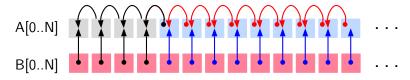
# Example 2

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  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



Iteration # 1 2 3

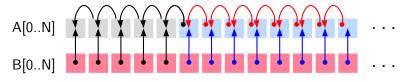
# Example 2

Vectorization  $(SISD \Rightarrow SIMD)$ : Yes  $(SISD \Rightarrow MIMD)$ : No Parallelization

#### 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



Iteration #

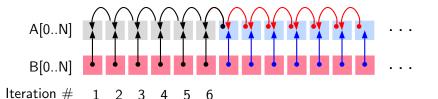
3 July 2011

### Example 2

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  MIMD) : No

Original Code

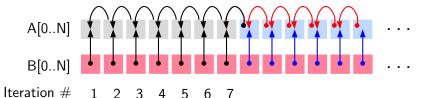
```
int A[N], B[N], i;
for (i=0; i<N; i++)
A[i] = A[i+1] + B[i];
```



#### Example 2

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  MIMD) : No

Original Code

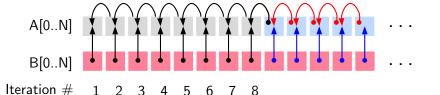


#### Example 2

Vectorization  $(SISD \Rightarrow SIMD)$ : Yes  $(SISD \Rightarrow MIMD)$ : No Parallelization

#### Original Code

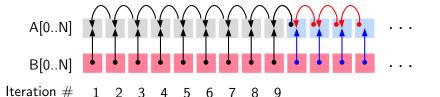
```
int A[N], B[N], i;
for (i=0; i<N; i++)
  A[i] = A[i+1] + B[i];
```



#### Example 2

Vectorization  $(SISD \Rightarrow SIMD)$ : Yes  $(SISD \Rightarrow MIMD)$ : No Parallelization

Original Code

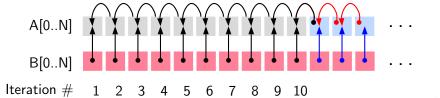


# Example 2

Vectorization  $(SISD \Rightarrow SIMD)$ : Yes  $(SISD \Rightarrow MIMD)$ : No Parallelization

Original Code

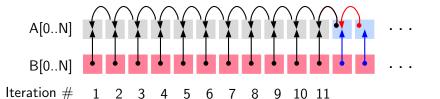
```
int A[N], B[N], i;
for (i=0; i<N; i++)
  A[i] = A[i+1] + B[i];
```



#### Example 2

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  MIMD) : No

Original Code

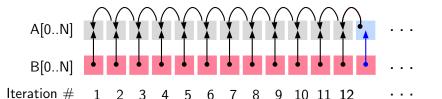


### Example 2

 $\begin{array}{lll} \mbox{Vectorization} & (\mbox{SISD} \Rightarrow \mbox{SIMD}) & : \mbox{Yes} \\ \mbox{Parallelization} & (\mbox{SISD} \Rightarrow \mbox{MIMD}) & : \mbox{No} \\ \end{array}$ 

#### Original Code

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

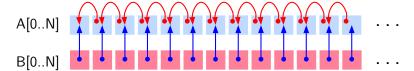


# Example 2

 $\begin{array}{lll} \mbox{Vectorization} & (\mbox{SISD} \Rightarrow \mbox{SIMD}) & : \mbox{ Yes} \\ \mbox{Parallelization} & (\mbox{SISD} \Rightarrow \mbox{MIMD}) & : \mbox{ No} \\ \end{array}$ 

#### Original Code

 Vector instruction is synchronized: All reads before writes in a given instruction



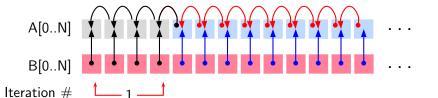
Iteration #

# Example 2

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  MIMD) : No

#### Original Code

 Vector instruction is synchronized: All reads before writes in a given instruction

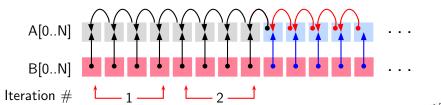


# Example 2

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  MIMD) : No

#### Original Code

 Vector instruction is synchronized: All reads before writes in a given instruction



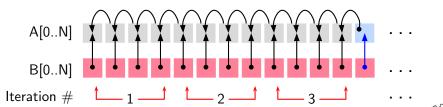
### Example 2

Vectorization  $(SISD \Rightarrow SIMD)$ : Yes  $(SISD \Rightarrow MIMD)$ : No Parallelization

#### Original Code

reads before writes in a given instruction A[i] = A[i+1] + B[i]:

Vector instruction is synchronized:



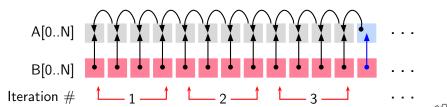
### Example 2

Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  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



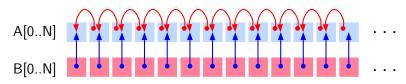
3 July 2011

# **Example 2: The Moral of the Story**

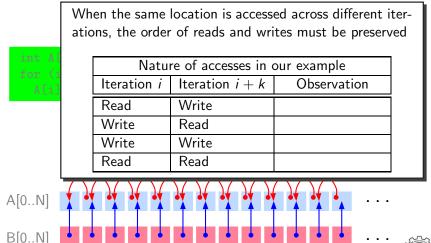
Vectorization  $(SISD \Rightarrow SIMD)$ : Yes  $(SISD \Rightarrow MIMD)$ : No Parallelization

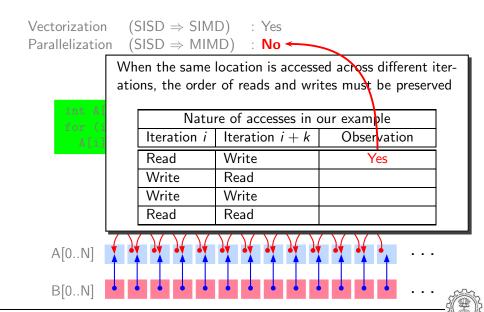
Original Code

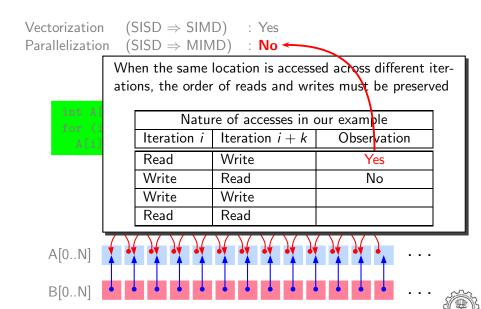
```
int A[N], B[N], i;
for (i=0; i<N; i++)
  A[i] = A[i+1] + B[i];
```

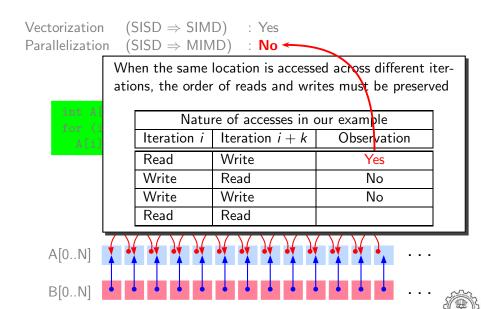


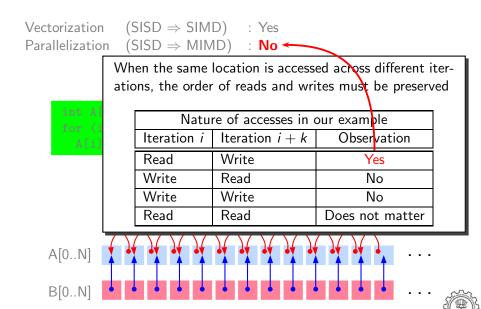
Vectorization (SISD  $\Rightarrow$  SIMD) : Yes Parallelization (SISD  $\Rightarrow$  MIMD) : No











10/28

Example 3

 $\begin{array}{lll} \mbox{Vectorization} & (\mbox{SISD} \Rightarrow \mbox{SIMD}) & : \mbox{ No} \\ \mbox{Parallelization} & (\mbox{SISD} \Rightarrow \mbox{MIMD}) & : \mbox{ No} \\ \end{array}$ 

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

# Example 3

Vectorization  $(SISD \Rightarrow SIMD)$ : No  $(SISD \Rightarrow MIMD)$ : No Parallelization

```
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

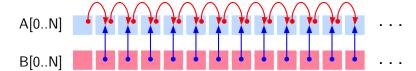
A[0..N]B[0..N]

# Example 3

 $\begin{array}{lll} \mbox{Vectorization} & (\mbox{SISD} \Rightarrow \mbox{SIMD}) & : \mbox{No} \\ \mbox{Parallelization} & (\mbox{SISD} \Rightarrow \mbox{MIMD}) & : \mbox{No} \\ \end{array}$ 

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

Observe reads and writes into a given location



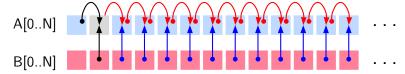
Iteration #

# Example 3

 $\begin{array}{lll} \mbox{Vectorization} & (\mbox{SISD} \Rightarrow \mbox{SIMD}) & : \mbox{No} \\ \mbox{Parallelization} & (\mbox{SISD} \Rightarrow \mbox{MIMD}) & : \mbox{No} \\ \end{array}$ 

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

Observe reads and writes into a given location



Iteration #

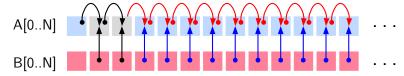
.

# Example 3

 $\begin{array}{lll} \mbox{Vectorization} & (\mbox{SISD} \Rightarrow \mbox{SIMD}) & : \mbox{No} \\ \mbox{Parallelization} & (\mbox{SISD} \Rightarrow \mbox{MIMD}) & : \mbox{No} \\ \end{array}$ 

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

Observe reads and writes into a given location



Iteration #

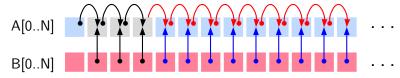
2

# Example 3

 $\begin{array}{lll} \mbox{Vectorization} & \mbox{(SISD} \Rightarrow \mbox{SIMD}) & : \mbox{ No} \\ \mbox{Parallelization} & \mbox{(SISD} \Rightarrow \mbox{MIMD}) & : \mbox{ No} \\ \end{array}$ 

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

Observe reads and writes into a given location



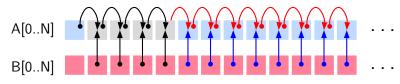
Iteration # 1 2 3

# Example 3

 $\begin{array}{lll} \mbox{Vectorization} & (\mbox{SISD} \Rightarrow \mbox{SIMD}) & : \mbox{ No} \\ \mbox{Parallelization} & (\mbox{SISD} \Rightarrow \mbox{MIMD}) & : \mbox{ No} \\ \end{array}$ 

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

Observe reads and writes into a given location

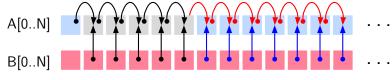


# Example 3

Vectorization (SISD  $\Rightarrow$  SIMD) : No Parallelization (SISD  $\Rightarrow$  MIMD) : No

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

Observe reads and writes into a given location



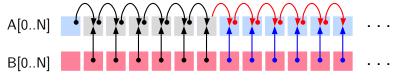
Iteration # 1 2 3 4 5

# Example 3

Vectorization (SISD  $\Rightarrow$  SIMD) : No Parallelization (SISD  $\Rightarrow$  MIMD) : No

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

Observe reads and writes into a given location



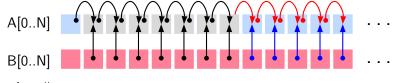
Iteration # 1 2 3 4 5 6

# Example 3

Vectorization (SISD  $\Rightarrow$  SIMD) : No Parallelization (SISD  $\Rightarrow$  MIMD) : No

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

Observe reads and writes into a given location



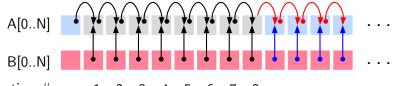
Iteration # 1 2 3 4 5 6

# Example 3

Vectorization (SISD  $\Rightarrow$  SIMD) : No Parallelization (SISD  $\Rightarrow$  MIMD) : No

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

Observe reads and writes into a given location

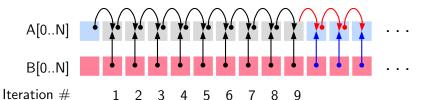


# Example 3

Vectorization (SISD  $\Rightarrow$  SIMD) : No Parallelization (SISD  $\Rightarrow$  MIMD) : No

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

Observe reads and writes into a given location



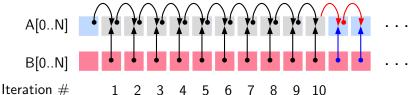
Essential Abstractions in GCC

# Example 3

Vectorization (SISD  $\Rightarrow$  SIMD) : No Parallelization (SISD  $\Rightarrow$  MIMD) : No

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

Observe reads and writes into a given location

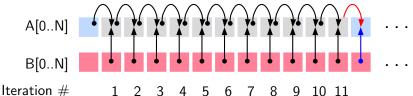


# Example 3

Vectorization (SISD  $\Rightarrow$  SIMD) : No Parallelization (SISD  $\Rightarrow$  MIMD) : No

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

Observe reads and writes into a given location

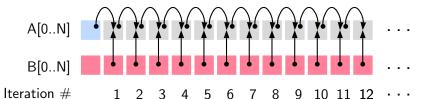


# Example 3

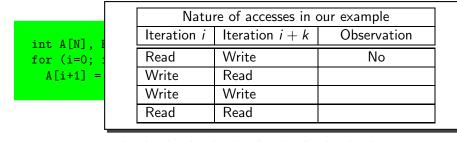
 $\begin{array}{lll} \mbox{Vectorization} & (\mbox{SISD} \Rightarrow \mbox{SIMD}) & : \mbox{No} \\ \mbox{Parallelization} & (\mbox{SISD} \Rightarrow \mbox{MIMD}) & : \mbox{No} \\ \end{array}$ 

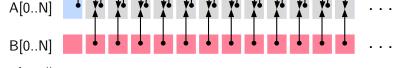
```
int A[N], B[N], i;
for (i=0; i<N; i++)
   A[i+1] = A[i] + B[i+1];</pre>
```

Observe reads and writes into a given location

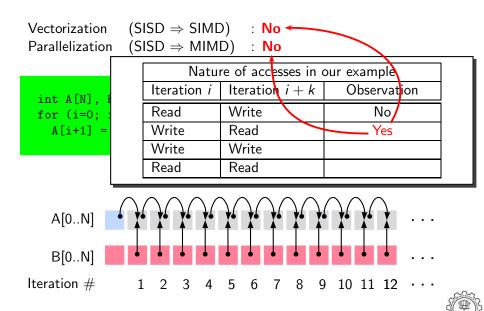


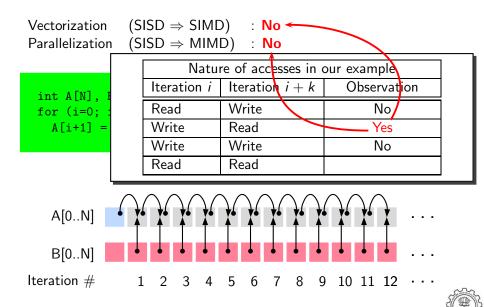
Vectorization (SISD  $\Rightarrow$  SIMD) : No Parallelization (SISD  $\Rightarrow$  MIMD) : No

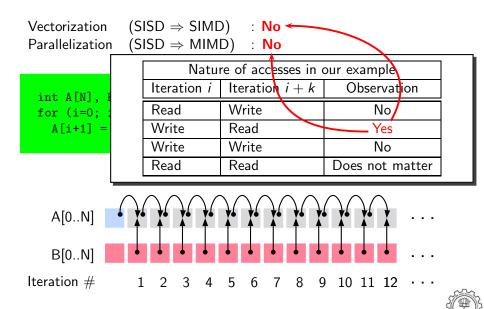




Iteration # 1 2 3 4 5 6 7 8 9 10 11 12 · · ·







Vectorization

3 July 2011

intro-par-vect: Introduction to Parallelization and Vectorization

: No

11/28

 $(SISD \Rightarrow SIMD)$  $(SISD \Rightarrow MIMD)$ : Yes Parallelization

GCC Resource Center, IIT Bombay

GCC Resource Center, IIT Bombay

3 July 2011

 $(SISD \Rightarrow SIMD)$  $(SISD \Rightarrow MIMD)$  : Yes Parallelization

intro-par-vect: Introduction to Parallelization and Vectorization

Example 4

: No

This case is not possible

11/28

Vectorization

3 July 2011

11/28

# Example 4

: No

 $(SISD \Rightarrow SIMD)$ 

Parallelization  $(SISD \Rightarrow MIMD)$  : Yes

- This case is not possible
- Vectorization is a limited granularity parallelization

Vectorization (SISD  $\Rightarrow$  SIMD) : No Parallelization (SISD  $\Rightarrow$  MIMD) : Yes

- This case is not possible
- Vectorization is a limited granularity parallelization
- If parallelization is possible then vectorization is trivially possible

Access in  $S_i$  Access in  $S_i$  Dependence

#### Data Dependence

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

Read <i>m</i>	Write m	Anti (or Pseudo)	$S_i \ \bar{\delta} \ S_j$
Write m	Read <i>m</i>	Flow (or True)	$S_i \delta S_j$
Write m	Write m	Output (or Pseudo)	$S_i \delta^O S_j$
Read <i>m</i>	Read <i>m</i>	Does not matter	

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

Notation

## 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
  - $\triangleright$   $S_i$  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
  - $\blacktriangleright$  Within an iteration,  $S_i$  and  $S_j$  can be executed in parallel
  - ▶ Different iterations of the loop must be executed sequentially
- S<sub>i</sub> and S<sub>j</sub> may have both loop carried and loop independent dependences

## Dependence in Example 1

Program

Dependence graph



No loop carried dependence
 Both vectorization and parallelization are possible

### Dependence in Example 1

Program

• Dependence graph

Dependence in the same iteration



No loop carried dependence
 Both vectorization and parallelization are possible

## Dependence in Example 2

Program

Dependence graph



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

### Dependence in Example 2

Program

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 3

Program

Dependence graph



Loop carried flow-dependence
 Neither parallelization not vectorization is possible

17/28

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

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

LHS

Iteration

Vector

3.3

3.3

Index Vector

RHS

LHS

3 July 2011

# 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;
}</pre>

Loop carried dependence exists if

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

```
0.0
          1,0
                  0.0
0, 1
          1, 1
                  0, 1
0, 2
          1, 2
                  0, 2
0,3
          1,3
                  0, 3
         2, 0
1,0
                  1,0
         2, 1
1, 1
                  1, 1
         2, 2
1, 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
                  2,3
         3, 3
3,0
         4,0
                  3,0
3, 1
         4, 1
                  3, 1
```

Iteration

Vector

3, 2

3, 3

3, 2

3.3

4, 2

LHS

1,0

1, 1

1, 2

1,3

2, 0

2, 1

2, 2

2,3

Index Vector

RHS

0.0

0, 1

0, 2

0, 3

1,0

1, 1

1, 2

1,3

3, 2

3.3

3 July 2011

# 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;
  }
```

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

2,0 3, 0 2,0 2, 1 3, 1 2, 12, 2 3, 2 2, 2 2,3 2,3 3, 3 3,0 4,0 3,0 3, 1 4, 13, 1

3, 2

3, 3

Iteration

Vector

0.0

0, 1

0, 2

0, 3

1,0

1, 1

1, 2

1,3

4, 2

LHS

1,0

1, 1

Index Vector

RHS

0.0

0, 1

Iteration

Vector

0.0

0, 1

3 July 2011

# 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;
  }
```

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

0, 21, 2 0, 20, 31,3 0, 32,0 1,0 1,0 1, 12, 1 1, 12, 2 1, 2 1, 2 1,3 2,3 1,3 2,0 3, 0 2,0 2, 1 3, 1 2, 12, 2 3, 2 2, 2 2,3 2,3 3, 3 3,0 4,0 3,0 3, 1 4, 13, 1 3, 2 4, 2 3, 2 3, 3 4, 3 3.3

LHS

1,0

1, 1

1, 2

Index Vector

RHS

0.0

0, 1

0, 2

3 July 2011

# 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;
  }
```

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

1, 11, 2 1,3 2,0 2, 1 2, 2 2,3

3,0

3, 1

3, 2

3, 3

Iteration

Vector

0.0

0, 1

0, 2

0, 3

1,0

1,3 0, 32, 01,0 2, 1 1, 12, 2 1, 2 2,3 1,3 3, 0 2,0 3, 1 2, 13, 2 2, 2 2,3 3, 3 4,0 3,0

3, 1

3, 2

3,3

4, 1

4, 2

}

for (i=0, i<4; i++) for (j=0; j<4; j++)

```
a[i][j] = a[i][j] + 2;
```

S RHS
0,0
0,1
2 0,2
3 0,3
1,0
1,1
2 1,2
3 1,3
2,0
1 2,1
2 2,2
3 2,3
3,0
1 3,1

3, 3

Iteration Index Vector

3, 1 3, 2

3.3

3, 2

3.3

Index Vector

RHS

LHS

3 July 2011

## Iteration Vectors and Index Vectors: Example 2 Iteration

```
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

```
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
```

Vector

3, 2

3.3

3, 2

3,3

3, 2

Index Vector

RHS

0.0

0, 1

0, 2

0, 3

1,0

1, 1

1, 2

1,3

3.3

LHS

0.0

0, 1

0, 2

0,3

1,0

1, 1

1, 2

1,3

# **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
- are identical

Conclusion: No dependence

2,0 2, 02,0 2, 1 2, 12, 12, 2 2, 22, 2 2,3 2,3 2,3 the index vectors of LHS and RHS 3,0 3,0 3,0 3, 1 3, 1 3, 1 3, 2 3, 2 3, 2

3, 3

Iteration

Vector

0.0

0, 1

0, 2

0,3

1,0

1, 1

1, 2

1,3

```
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 */
}
```

```
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]

Loop independent anti dependence due to B[i]

```
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

```
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

```
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 */
}

\delta_0
\delta_1
\delta_0
\delta_1
\delta_0
\delta_0
```

Loop carried output dependence due to T

# **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 */
}
```

# **Tutorial Problem for Discovering Dependence**

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

### ·

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  $\mathbf{i}$  and  $\mathbf{j}$  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  $\mathbf{i}$  and statement  $S_2$  accesses location M on iteration  $\mathbf{i}$ , and
- 3. one of these accesses is a write access.

# 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];
}</pre>
```

# 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];
}</pre>
```

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];
}</pre>
```

23/28

# 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];
}</pre>
```

# 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];
}</pre>
```

# Write lexicographically precedes Read

**True Dependence and Vectorization** 

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

# 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];
}
</pre>
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];
}
```

Anti Dependence and True Dependence

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

### 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;
}</pre>
```

25/28

3 July 2011

### Anti Dependence and True Dependence

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

### Anti Dependence and True Dependence

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

### 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];
}</pre>
```

# 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];
}</pre>
```

### True Dependence and Anti Dependence

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

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];
}</pre>
```

27/28

```
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];
```

Cyclic True Dependence

3 July 2011

Cyclic Anti Dependence

27/28

# Cyclic Dependency and Vectorization

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

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

Rescheduling of statements will not break the cyclic dependency - cannot vectorize  $\mbox{\ }$ 

# Thank You!