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



July 2010

July 2010

Par-Vect Intro: Outline

1/1

• Transformation for parallel and vector execution

Data dependence

GCC Resource Center, IIT Bombay

The Scope of this Tutorial

Par-Vect Intro: Outline

- Algorithms used for parallelization and vectoriation
- Machine level issues related to parallelization and vectoriation
- What this tutorial addresses.

Basics of Discovering Parallelism using GCC



2/1

July 2010

Part 1

Transformations for Parallel and Vector Execution

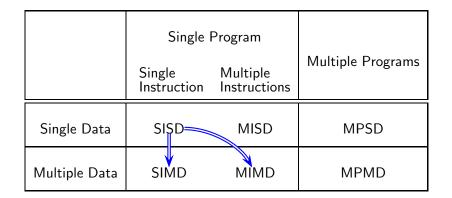
	Single Program	Multiple Programs
Single Data	SPSD	MPSD
Multiple Data	SPMD	MPMD

	Single Program		
	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

July 2010

Vectorization: $SISD \Rightarrow SIMD$

Par-Vect Intro: Transformations for Parallel and Vector Execution

- Parallelism in executing operation on shorter operands (8-bit, 16-bit, 32-bit operands)

July 2010

5/1

Par-Vect Intro: Transformations for Parallel and Vector Execution

Example 1

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

Original Code

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

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

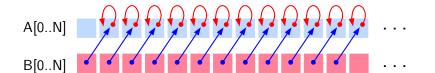
A[0..N] ...

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

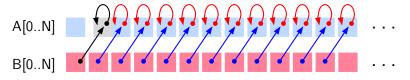


Vectorization $(SISD \Rightarrow SIMD)$: Yes $(SISD \Rightarrow MIMD)$: Yes Parallelization

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



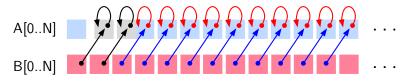
Iteration #

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

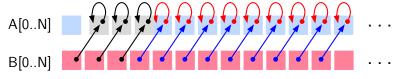


Iteration # 1

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

Original Code

Observe reads and writes into a given location



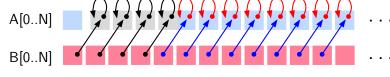
Iteration #

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

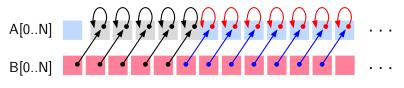


Iteration # 1 2 3

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

Original Code

Observe reads and writes into a given location



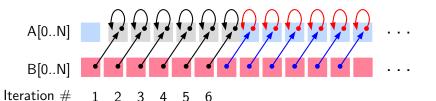
Iteration # 1 2 3 4 !

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

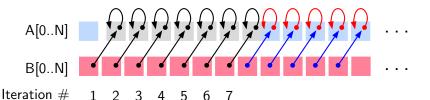


Essential Abstractions in GCC

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

Original Code

Observe reads and writes into a given location

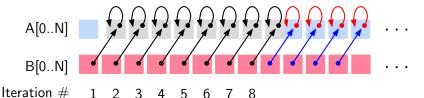


Essential Abstractions in GCC

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

Original Code

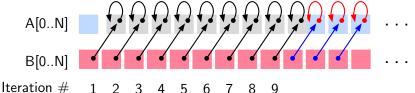
int A[N], B[N], i; for (i=1; i<N; i++) A[i] = A[i] + B[i-1];



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

Original Code

Observe reads and writes into a given location

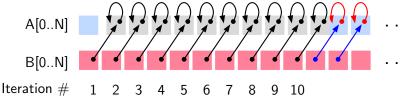


Essential Abstractions in GCC

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

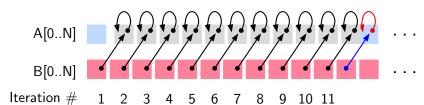
Original Code

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



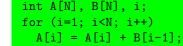
Vectorization (SISD \Rightarrow SIMD) : Yes Parallelization (SISD \Rightarrow MIMD) : Yes

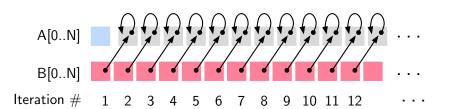
Original Code



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

Original Code





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

Vectorization **Factor**

5/1

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 #

Vectorized Code

Example 1

Vectorization(SISD \Rightarrow SIMD): YesParallelization(SISD \Rightarrow 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];

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];</pre>

A[0..N]
B[0..N]
Iteration # 1

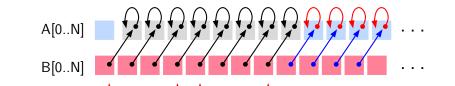
Vectorization (SISD \Rightarrow SIMD) : Yes
Parallelization (SISD \Rightarrow MIMD) : Yes

Original Code

Vectorized Code

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

for (i=1; i<N; i=i+4)



Iteration #

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

A[i:i+3] = A[i:i+3] + B[i-1:i+2];

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

Iteration #

July 2010

6/1

illipic .

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

Original Code

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

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

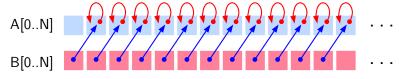
A[0..N] ...

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 #

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

Original Code

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

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

Parallelized Code

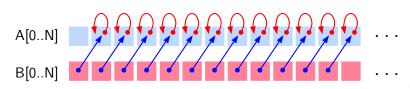
```
A[0..N] ...
B[0..N] ...
```

Example 1: The Moral of the Story

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

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

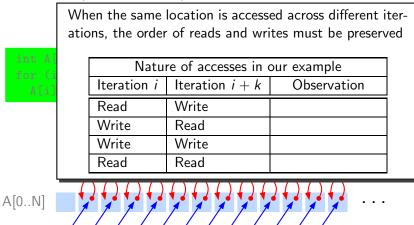
Observe reads and writes into a given location



7/1

Example 1: The Moral of the Story

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



7/1

Example 1: The Moral of the Story

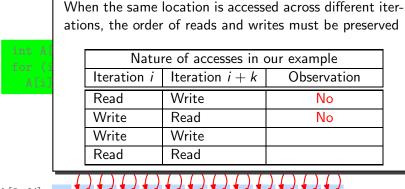
Vectorization $(SISD \Rightarrow SIMD)$: Yes Parallelization (SISD \Rightarrow MIMD) : Yes

	When the same location is accessed across different ations, the order of reads and writes must be present						
int A[for (i	INATURE OF ACCESSES IN OUR EXAMPLE						
A[i]		Iteration i	Iteration $i + k$	Observation			
		Read	Write	No			
		Write	Read				
		Write	Write				
		Read	Read				
A[O N]	4	141414	44444)()()()			

7/1

Example 1: The Moral of the Story

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



A[0..N]

B[0..N]

7/1

Example 1: The Moral of the Story

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

	ations, the order of reads and writes must be preserve			
[Nature of accesses in our example		
]		Iteration i	Iteration $i + k$	Observation
		Read	Write	No
		Write	Read	No
		Write	Write	No
		Read	Read	
L				

When the same location is accessed across different iter-

B[0..N]

A[0..N]

Example 1: The Moral of the Story

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

> 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 Read Read Does not matter

When the same location is accessed across different iter-

A[0..N]

B[0..N]

July 2010

8/1

DIE 2

Par-Vect Intro: Transformations for Parallel and Vector Execution

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

Original Code

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

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

Original Code

July 2010

Observe reads and writes into a given location

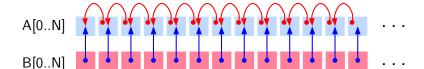
8/1

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



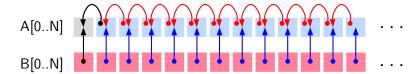
Iteration #

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];</pre>

Observe reads and writes into a given location



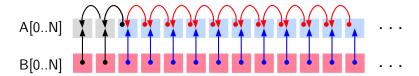
Iteration #

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

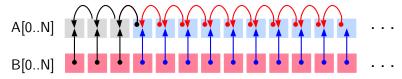


Iteration # 1

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

Original Code

Observe reads and writes into a given location

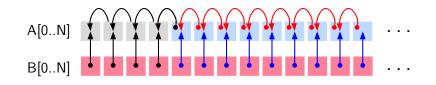


Iteration # 1 2

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

Original Code

Observe reads and writes into a given location



Iteration #

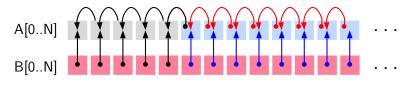
8/1

Example 2

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

Original Code

Observe reads and writes into a given location

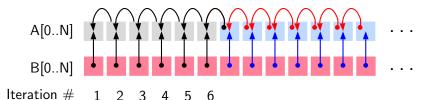


Iteration # 1 2 3 4

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

Original Code

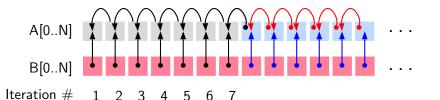
Observe reads and writes into a given location



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

Original Code

Observe reads and writes into a given location



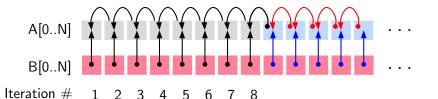
Essential Abstractions in GCC

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];
```

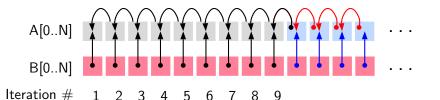
Observe reads and writes into a given location



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

Original Code

Observe reads and writes into a given location

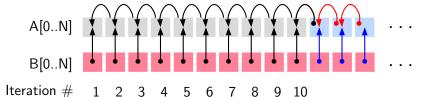


Essential Abstractions in GCC

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

Original Code

Observe reads and writes into a given location



Essential Abstractions in GCC

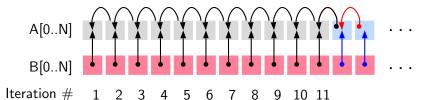
July 2010

8/1

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

Original Code

Observe reads and writes into a given location



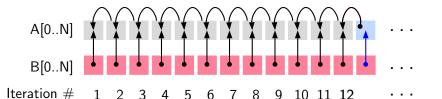
8/1

Example 2

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

Original Code

Observe reads and writes into a given location



July 2010

8/1

Example 2

Par-Vect Intro: Transformations for Parallel and Vector Execution

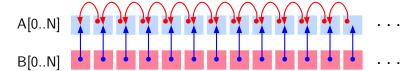
Vectorization (SISD \Rightarrow SIMD) : Yes Parallelization (SISD \Rightarrow MIMD) : No

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



Iteration #

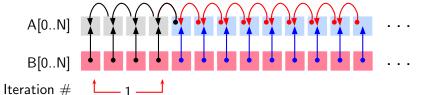
Vectorization $(SISD \Rightarrow SIMD)$: Yes $(SISD \Rightarrow MIMD)$: No Parallelization

Original Code

July 2010

for (i=0; i<N; i++) A[i] = A[i+1] + B[i]: Vector instruction is synchronized: reads before writes in a given instruction

8/1

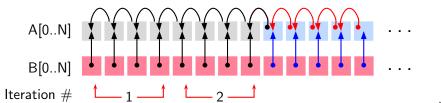


Vectorization (SISD \Rightarrow SIMD) : Yes Parallelization (SISD \Rightarrow MIMD) : No

Original Code

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



8/1

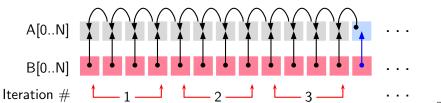
Example 2

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

Original Code

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



July 2010

8/1

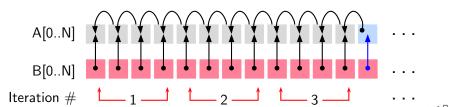
Example 2

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

Original Code

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

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



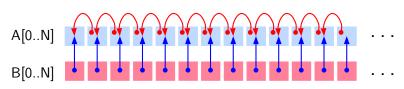
Example 2: The Moral of the Story

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];</pre>
```

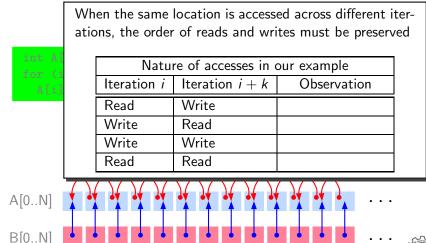
Observe reads and writes into a given location



9/1

Example 2: The Moral of the Story

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

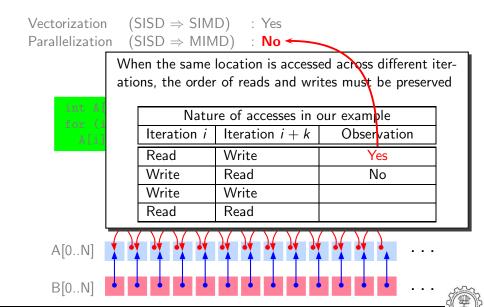


Example 2: The Moral of the Story

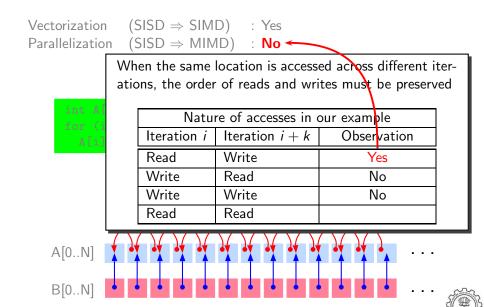
Vectorization $(SISD \Rightarrow SIMD)$: Yes Parallelization $(SISD \Rightarrow MIMD) : No$ When the same location is accessed across different iterations, the order of reads and writes must be preserved Nature of accesses in our example Iteration i Iteration i + kObservation Write Read Yes Write Read Write Write Read Read A[0..N]B[0..N]

9/1

Example 2: The Moral of the Story

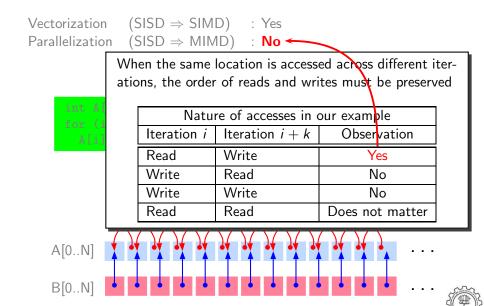


Example 2: The Moral of the Story



9/1

Example 2: The Moral of the Story



July 2010

Vectorization

July 2010

10/1

 $(SISD \Rightarrow SIMD)$

Par-Vect Intro: Transformations for Parallel and Vector Execution

: No

Example 3

 $(SISD \Rightarrow MIMD)$: 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

July 2010

10/1

Example 3

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

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

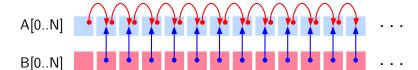
Observe reads and writes into a given location

A[0..N] ...

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

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

Observe reads and writes into a given location

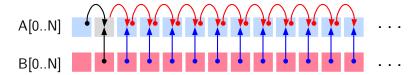


Iteration #

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

```
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

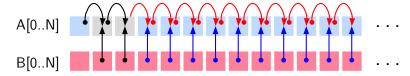


Iteration #

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

```
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

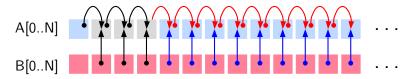


Iteration # 1 2

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

```
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

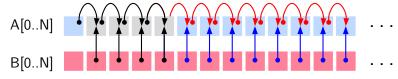


Iteration # 1 2 3

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

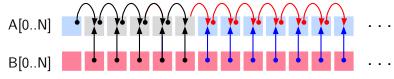


Iteration # 1 2 3 4

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

```
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

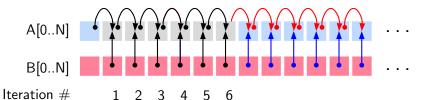


Iteration # 1 2 3 4 5

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

```
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



Essential Abstractions in GCC

GCC Resource Center, IIT Bombay

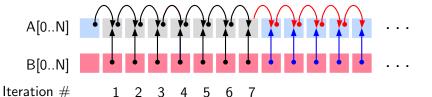
10/1

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

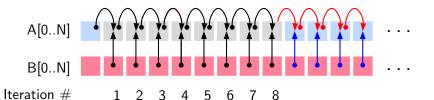


Essential Abstractions in GCC

GCC Resource Center, IIT Bombay

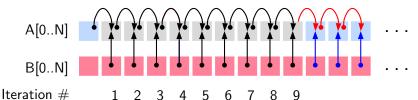
Vectorization (SISD \Rightarrow SIMD) : No Parallelization (SISD \Rightarrow MIMD) : No

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



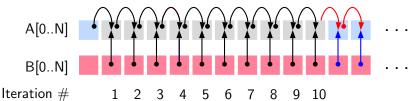
Vectorization (SISD \Rightarrow SIMD) : No Parallelization (SISD \Rightarrow MIMD) : No

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



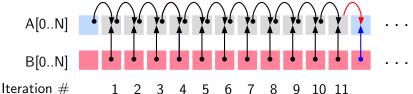
Vectorization (SISD \Rightarrow SIMD) : No Parallelization (SISD \Rightarrow MIMD) : No

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



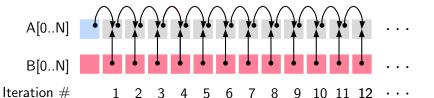
 $\begin{array}{lll} \mbox{Vectorization} & (\mbox{SISD} \Rightarrow \mbox{SIMD}) & : \mbox{No} \\ \mbox{Parallelization} & (\mbox{SISD} \Rightarrow \mbox{MIMD}) & : \mbox{No} \\ \end{array}$

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



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

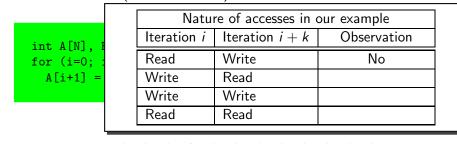
```
int A[N], B[N], i;
for (i=0; i<N; i++)
  A[i+1] = A[i] + B[i+1];</pre>
```

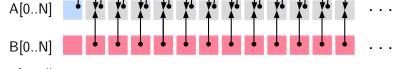


10/1

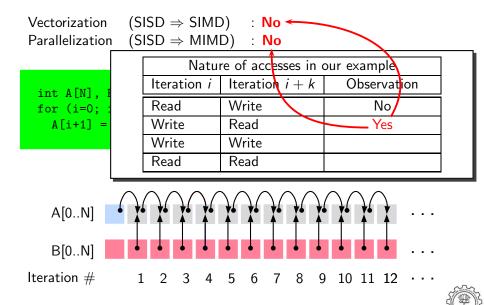
Example 3

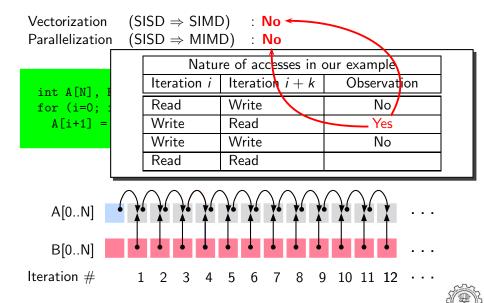
Vectorization $(SISD \Rightarrow SIMD)$: No $(SISD \Rightarrow MIMD)$: No Parallelization

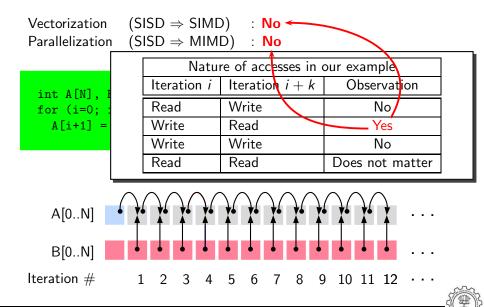




Iteration # 1 2 3 5 10







July 2010

Par-Vect Intro: Transformations for Parallel and Vector Execution

 $(SISD \Rightarrow SIMD)$

 $(SISD \Rightarrow MIMD)$

Par-Vect Intro: Transformations for Parallel and Vector Execution

Vectorization

Parallelization

July 2010

• This case is not possible

July 2010

11/1

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

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

Par-Vect Intro: Transformations for Parallel and Vector Execution

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

Notation

Data Dependence

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

Access in S_i Access in S_i Dependence

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

Loop Carried and Loop Independent Dependences

Consider dependence between statements S_i and S_i in a loop

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

Dependence in Example 1

Program

Dependence graph



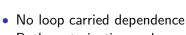
 No loop carried dependence Both vectorization and parallelization are possible

Dependence in Example 1

Program

Dependence graph

Dependence in the same iteration



Both vectorization and parallelization are possible

15/1

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

Dependence Graph 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
     Program to swap arrays
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]

```
Dependence Graph
     Program to swap arrays
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]

Dependence Graph Program to swap arrays 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

```
Dependence Graph
     Program to swap arrays
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

```
Dependence Graph
     Program to swap arrays
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

Dependence Graph 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 Program to swap arrays for (i=0; i<N; i++) T = A[i]; /* S1 */A[i] = B[i]; /* S2 *//* S3 */ B[i] = T;



Dependence Graph Program to swap arrays for (i=0; i<N; i++) S_1 T = A[i]; /* S1 */A[i] = B[i]; /* S2 *//* S3 */ B[i] = T;



Tutorial Problem for Discovering Dependence

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

- Analysis in loop is tricky, as
 - Loops may be nested
 - ▶ Different loop iterations may access same memory location
 - Arrays occur frequently
 - ▶ Far too many array locations to be treated as independent scalars

Data Dependence in Loops

- Analysis in loop is tricky, as
 - Loops may be nested
 - ▶ Different loop iterations may access same memory location
 - Arrays occur frequently
 - ► Far too many array locations to be treated as independent scalars
- Consider array location A [4] [9] in the following program

```
for(i = 0; i <= 5; i ++)
 for(j = 0; j \le 4; j ++)
   A[i+1][3*j] = ...; /* S1 */
        \dots = A[i+3][2*j+1]; /* S2 */
```

Data Dependence in Loops

- Analysis in loop is tricky, as
 - Loops may be nested
 - ▶ Different loop iterations may access same memory location
 - Arrays occur frequently
 - ► Far too many array locations to be treated as independent scalars
- Consider array location A[4][9] in the following program

S2 accesses in iteration (1,4), S1 accesses in iteration (3,3)

Par-Vect Intro: Transformations for Parallel and Vector Execution

July 2010

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

Iteration Index Vector Vector LHS RHS 0.0 1,0 0.0

0, 1

2, 2

2,3

3,0

0, 21, 2 0, 20,3 1,3 0, 32,0 1,0 1,0 2, 1 1, 11, 12, 2 1,2 1, 2 1,3 2,3 1,3 2,0 3,0 2,0 2, 1 3, 1 2, 1

3, 2

3,3

4,0

1, 1

0, 1

2,2

2,3

3,0

Iteration Index Vectors Index Vectors

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

0, 11, 10, 10, 21, 2 0, 20, 31,3 0, 32, 01,0 1,0 2, 1 1, 11, 12, 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 3, 2 4, 2 3, 2

LHS

1,0

RHS

0.0

Vector

0.0

3, 3

3.3

4, 3

Iteration Vectors and Index Vectors: Example 1 Iteration Index Vector

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

3,0 3, 1

Vector

0.0

0, 1

0, 2

0,3

1,0

1, 1

1, 2

1,3

2, 0

2, 1

LHS

1.0

1, 1

1, 2

1,3

2, 0

2, 1

2, 2

2,3

3, 0

3, 1

RHS

0.0

0, 1

0, 2

0, 3

1,0

1, 1

1, 2

1,3

2,0

2, 1

2, 2 3, 2 2, 2 2,3 2,3 3, 3 4.0 3,0 4, 1 3, 1 3, 2 4, 2 3, 2 3, 3 4, 3 3.3

Iteration Index Vector

```
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, 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

3, 1 3, 2

3, 3

4.0

4, 1

4, 2

4.3

LHS

Vector

0.0

2, 1

2, 2

2,3

3,0

3, 1

3, 2

3.3

RHS

0.0

2, 2

2,3

3,0

3, 1

3, 2

3.3

Iteration Vectors and Index Vectors: Example 1 Iteration Index Vector

```
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, 11, 10, 10, 21, 2 0, 20, 31,3 0,3 2, 01,0 1,0 1, 12, 1 1, 11, 2 2, 21, 2 1,3

LHS

1,0

RHS

0.0

Vector

0.0

2, 0

2, 1

2, 2

2,3

3,0

3, 1

3, 2

3, 3

2,3 1,3 3, 0 2,0 3, 1 2, 1 3, 2 2, 22,3 3, 3 4.0 3,0

3, 1

4, 1

4, 2

4, 3

Par-Vect Intro: Transformations for Parallel and Vector Execution

July 2010

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

0, 10, 10, 10, 20, 20, 20,3 0, 30,3 1,0 1,0 1,0 1, 11, 11, 11,2 1, 2 1, 2 1,3 1,3 1,3 2,0 2,0 2,0

2, 1

2, 2

2,3

3,0

3, 1

3, 2

3, 3

LHS

0.0

Index Vector

RHS

0.0

2, 1

2,2

2,3

3,0

Iteration

Vector

0.0

2, 1

2, 2 2,3

3,0

3, 1

3, 2

3, 3

RHS

Vector

```
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

0.0 0.0 0.0 0, 10, 10, 10, 20, 20, 20,3 0,3 0, 3

LHS

1,0 1,0 1,0 1, 11, 11, 11, 2 1, 2 1, 2 1,3 1,3 1,3 2, 0 2, 02,0 2, 1 2, 12, 1 2, 2 2, 22, 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

are identical

RHS

0.0

0, 1

0, 2

0,3

1,0

LHS

0.0

Iteration Vectors and Index Vectors: Example 2 Iteration Index Vector

Vector

0.0

1, 2

1,3

2, 0

2, 1

2, 2

2,3

3,0

3, 1

3, 2

3, 3

```
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

0, 10, 10, 20, 20,3 0,3 1,0 1,0 1, 11, 1

1, 11, 2 1, 2 1,3 1,3 2, 02,0 2, 12, 1 2, 22, 2 2,3 2,3 3,0 3,0

3, 1

3, 1

3, 2

3, 3

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 \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{j} , and
- 3. one of these accesses is a write access.