

## Compiling For High Performance

---

### Issues

- parallelism
- locality

### Classical compilation

- focus on individual operations
- scalar variables
- flow of values
- unstructured code

### High-performance compilation

- focus on aggregate operations (loops)
- array variables
- memory access patterns
- structured code

### Issues

- dependence analysis
- loop transformations

## Forms of Parallelism

---

### Instruction-level parallelism

- for superscalar and VLIW architectures
- examine dependences between statements
- very fine grain parallelism

$A = 1$

$B = C$

### Task-level parallelism

- for multiprocessors
- examine dependences between tasks
- parallelism is not scalable

```
do i = 1,10
```

```
do i = 1,10
```

```
  A(i) = A(i+1)
```

```
  B(i) = B(i+1)
```

### Loop-level parallelism

- for vector machines and multiprocessors
- examine dependences between loop iterations
- parallelism is scalable

```
doall i = 1,10
```

```
  A(i) = B(i+1)
```

## Loop-level Parallelism

---

### Basic approach

- execute loop iterations in parallel
- safe if no loop-carried data dependences (i.e., no accesses to same memory location)

```
do i = 1,10          doall i = 1,10
  A(i) = A(i+1)      A(i) = A(i+10)
```

### Several parallel architectures

- vector processors

```
A[1:10] = B[2:11]
```

- multiprocessors

```
doall i = 1,10
  A(i) = B(i+1)
```

- message-passing machines

```
if (...) send B(1)
if (...) recv B(11)
do i = L,B
  A(i) = B(i+1)
```

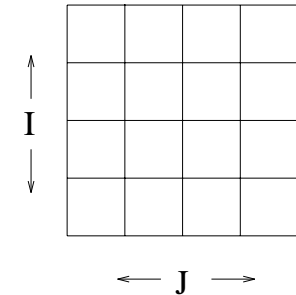
## Which Loops are Parallel?

---

```
do I = 1, N
  do J = 1, N
    S1 A(I,J) = A(I,J-1)
```

```
do I = 1, N
  do J = 1, N
    S2 A(I,J) = A(I-1,J-1)
```

```
do I = 1, N
  do J = 1, N
    S3 B(I,J) = B(I-1,J+1)
```



- a dependence  $D = (d_1, \dots, d_k)$  is *carried* at level  $i$ , if  $d_i$  is the first nonzero element of the distance vector
- a loop  $l_i$  is *parallel*, if  $\nexists$  a dependence  $D_j$  carried at level  $i$

	distance vector
$\forall D_j$	$d_1, \dots, d_{i-1} > 0$
OR	$d_1, \dots, d_i = 0$

## Exposing Parallelism

---

### Scalar analysis

- improve precision of dependence tests
- eliminate unnecessary scalar statements
- based on data-flow analysis

### Solution techniques

- forward propagation  
(expose value of scalar variables)

```
do i = 1, 10      do i = 1, 10
  k = i+1         k = i+1
  A(k) =          A(i+1) =
```

- constant propagation

```
  k = 1          k = 1
do i = 1, 10    do i = 1, 10
  A(i+k) =      A(i+1) =
```

- induction variable recognition

```
  k = 1          k = 1
do i = 1, 10    do i = 1, 10
  k = k+1       A(i+1) =
  A(k) =        k = 11
```

## Vectorization

---

### Vector processors

- operations on vectors of data
- overlap iterations of inner loop

```
doall i = 1, 10      A[1:10] = 1.0
  A(i) = 1.0         B[1:10] = A[1:10]
  B(i) = A(i)
```

- exploits fine-grain parallelism
- expressed in vector languages (APL, Fortran 90)

### Execution model

- single thread of control  
single instruction, multiple data (SIMD)
- load data into vector registers
- efficiently execute pipelined operations

### Issues

- vector length - coalesce loops to reduce overhead
- control flow - convert conditions into explicit data

## Parallelization

---

### Multiprocessors

- multiple independent processors (MIMD)
- assign iterations to different processors

```
doall j = 1,10      do j = L,U
do i = 1,10        do i = 1,10
  A(i,j) = 1.0      A(i,j) = 1.0
```

- exploits coarse-grain parallelism

### Execution model

- fork-join parallelism
- master executes sequential code
- workers (and master) execute parallel code
- master continues after workers finish

### Issues

- granularity - larger computation partitions to reduce overhead
- scheduling - policy for assigning iterations to processors

## Multithreading

---

### High latency event

- I/O
- interprocessor communication
- page miss
- cache miss

### Multiple threads of execution

- switch to new thread after event
- overlaps computation with event
- requires threads, efficient context switch

### Hardware support

- switch sets of registers
- shared caches
- HEP, Tera

### Software support

- uncover parallelism for multiple threads
- reduce context switch overhead