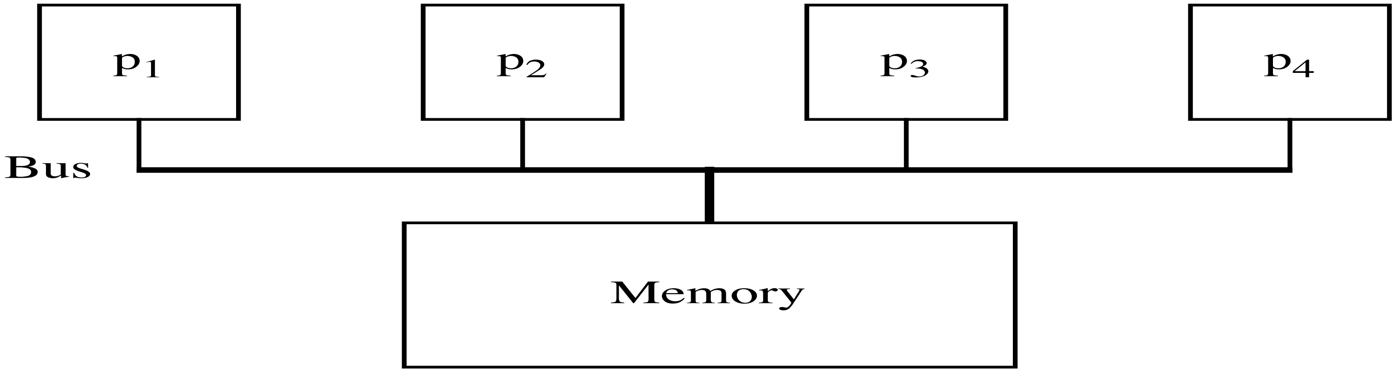
Exploring Parallelism At Different Levels

Balanced composition and customization of optimizations

Exploring Parallelism

Focus on Parallelism at different granularities

- On shared memory symmetric multiprocessors
  - The processors can run separate processes/threads
  - Starting processes and process synchronizations are expensive
  - Shared memory accesses can cause slowdowns
  - Processors have private caches and internal parallelism



Means Of Parallelism

- Data/Loop parallelism: single instruction stream
  - Threads operating concurrently on different data
  - E.g., OpenMP parallel for, CUDA/OpenCL kernels, vector operations…
- Task parallelism: explicit multi-tasking
  - Explicitly create/manage parallel threads or tasks, e.g., through pthreads, TBB, Cilk, …
  - Different threads communicate with each other via common patterns of data sharing, e.g., task queues
- Here we focus on data parallelism over loops
  - Loop parallelization: parallel do; Recognition of reduction; Privatization of variables; pipelining
  - Loop selection, skewing, and interchange
  - Loop fusion (vs. loop fission/distribution)

Outline

- Exploring parallelism at different levels
  - Loop parallelization at different granularities
    - OpenMP parallel for
    - SIMD vectorization
    - Pipelined parallelism

- composition of optimizations
    - Balancing degree of parallelism, cost of synchronization, memory performance, and CPU efficiency

Loop Parallelization

- **It is valid to convert a sequential loop to a parallel loop if the loop carries no dependence.**

- It is safe to evaluate different iterations of I in parallel

```
DO I=1,N
     X(I) = X(I) + C
ENDDO
```

- However, the same is not true for the following loop

```
DO I=1,N
     X(I+1) = X(I) + C
ENDDO
```

Here values computed in one iteration are used in the next

Recognition of Reductions

- Reducing an array of values into a single value
  - Sum, min/max, count reductions

```
S = 0.0
DO I = 1, N
        S = S + A(I)
ENDDO
```

Not directly parallelizable

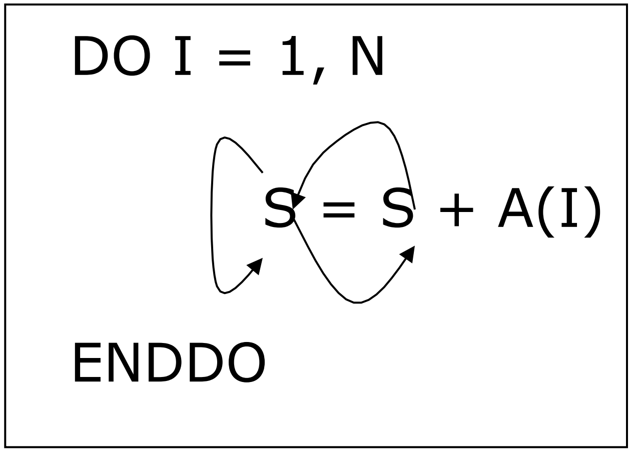
- Assuming commutativity and associativity

```
S = 0.0
DO k = 1, 4
  SUM(k) = 0.0
ENDDO
DO I = 1, N, 4
   SUM(1:3) = SUM(1:3) + A(I:I+3)
ENDDO
DO k = 1, 4
  S = S + SUM(k)
ENDDO
```

Can use vector registers to operate in parallel

Recognition of Reductions

- Reduction recognized by
  - Presence of self true, output and anti dependences
  - Absence of other true dependences



```
DO I = 1, N
        S = S + A(I)
ENDDO
```

```
DO I = 1, N
   S = S + A(I)
   T(I) = S
ENDDO
```

Privatization of Variables

- A variable x in a loop L is privatizable if it is defined before used along every path from the loop entry

```
    DO I = 1,N
S1      T = A(I)
S2      A(I) = B(I)
S3      B(I) = T
    ENDDO
```

```
    PARALLEL DO I = 1,N
      PRIVATE t
S1    t = A(I)
S2    A(I) = B(I)
S3    B(I) = t
    ENDDO
```

- Private and reduction variables must be identified correctly for loop parallelization to be correct
  - To ensure no dependences (synchronizations) among threads

```
#pragma omp for private(j)
for (i=0; i <N; i++) {
   for (j = 0; j < N; j++) {
           X[i][j] = X[i][j] + C;
}
```

Multi-level Loop Parallelism

- Coarse-grained parallelism
  - Create multiple threads on different CPU cores

```
#pragma omp parallel for
for (i=0; i <N; i++) {
        X[i] = X[i] + C;
}
```

- Fine-grained parallelism
  - Internal parallelism within each CPU core (e.g., SIMD vectorization)

```
vec_splat(C,r1)
for (i=0; i<N; i = i + 4){
  vec_mov_mr(X+i,r2)
  vec_add_rr(r1,r2)
  vec_mov_rm(r2,X+i)
}
```

Loop Strip Mining

- Converts available parallelism into a form more suitable for the hardware

```
DO I = 1, N
   A(I) = A(I) + B(I)
ENDDO

k = CEIL (N / P)
PARALLEL DO I = 1, N, k
     DO i = I, MIN(I + k-1, N)
       A(i) = A(i) + B(i)
     ENDDO
END PARALLEL DO
```

Loop Selection

- Consider:

```
    DO I = 1, N
      DO J = 1, M
S         A(I+1,J+1) = A(I,J) + A(I+1,J)
      ENDDO
    ENDDO
```

  - Direction matrix: $\begin{pmatrix} < & < \\ = & < \end{pmatrix}$
- Interchanging the loops can lead to:

```
DO J = 1, M
    A(2:N+1,J+1) = A(1:N,J) + A(2:N+1,J)
ENDDO
```

- Which loop to shift?
  - Select a parallel loop at outermost for coarse-grained parallelism
  - Select a parallel loop (with continuous memory access) at the innermost level for fine-grained parallelism

Loop Interchange

- Move parallel loops to outermost level
  - In a perfect nest of loops, a particular loop can be parallelized at the outermost level if and only if the column of the direction matrix for that nest contain only ‘= ‘ entries
- Example

```
DO I = 1, N
   DO J = 1, N
         A(I+1, J) = A(I, J) + B(I, J)
   ENDDO
ENDDO
```

  - OK for vectorization
  - Problematic for coarse-grained parallelization
    - Should the J loop be moved outside ?

Loop Selection

- Generate most parallelism with adequate granularity
  - Key is to select proper loops to run in parallel
  - Optimality is a NP-complete problem
- Informal parallel code generation strategy
  - Select parallel loops and move them to the outermost position
  - Select a sequential loop to move outside and enable internal parallelism

```
DO I = 2, N+1
   DO J = 2, M+1
       parallel DO K = 1, L
          A(I, J, K+1) = A(I,J-1,K)+A(I-1,J,K+2)+A(I-1,J,K)
       ENDDO
   ENDDO
ENDDO
```

$$\begin{pmatrix} = & < & < \\ < & = & > \\ < & = & = \end{pmatrix}$$

Loop Skewing

```
DO I = 2, N+1
   DO J = 2, M+1
      DO K = 1, L
         A(I, J, K) = A(I,J-1,K) + A(I-1, J, K)
         B(I, J, K+1) = B(I, J, K) + A(I, J, K)
      ENDDO
   ENDDO
ENDDO
```

$$\begin{pmatrix} = & < & = \\ < & = & = \\ = & = & < \\ = & = & = \end{pmatrix}$$

- Skewed using k=K+I+J:

```
DO I = 2, N+1
   DO J = 2, M+1
      DO k = I+J+1, I+J+L
         A(I, J, k-I-J) = A(I, J-1, k-I-J) + A(I-1, J, k-I-J)
         B(I, J, k-I-J+1) = B(I, J, k-I-J) + A(I, J, k-I-J)
      ENDDO
   ENDDO
ENDDO
```

$$\begin{pmatrix} = & < & < \\ < & = & < \\ = & = & < \\ = & = & = \end{pmatrix}$$

Loop Skewing + Interchange

```
DO k = 5, N+M+1
    PARALLEL DO I = MAX(2, k-M-L-1), MIN(N+1, k-L-2)
        PARALLEL DO J = MAX(2, k-I-L), MIN(M+1, k-I-1)
            A(I, J, k-I-J) = A(I, J-1, k-I-J) + A(I-1, J, k-I-J)
            B(I, J, k-I-J+1) = B(I, J, k-I-J) + A(I, J, k-I-J)
        ENDDO
    ENDDO
ENDDO
```

- Selection Heuristics
  - Parallelize outermost loop if possible
  - Make at most one outer loop sequential to enable inner parallelism
  - If both fails, try skewing
  - If skewing fails, try minimize the number of outside sequential loops

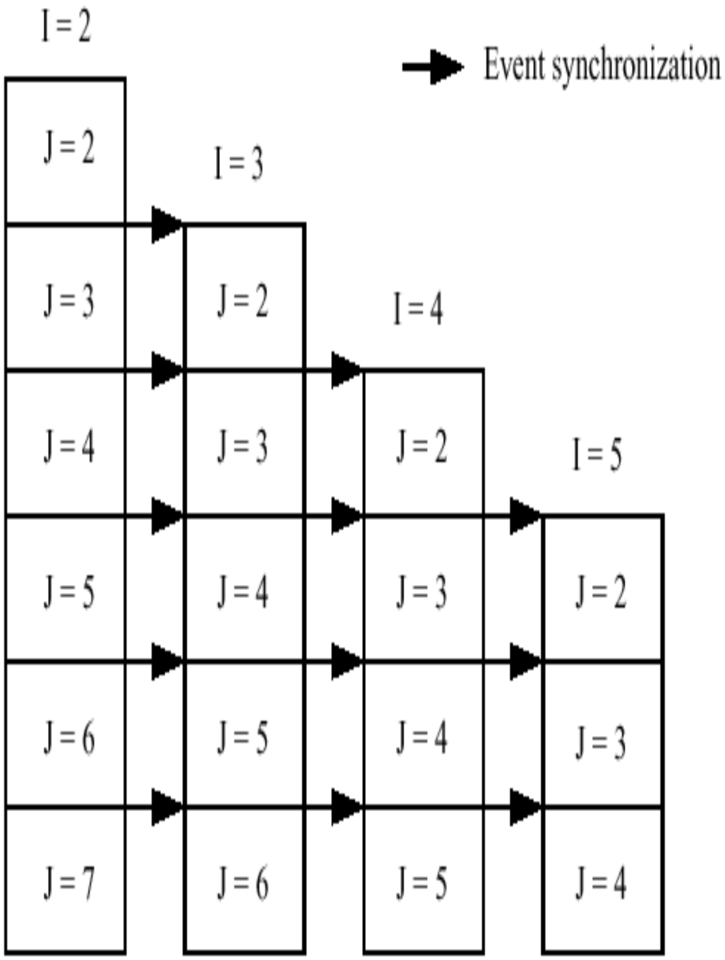
Pipelined Parallelism For Stencils

- Useful where complete parallelization is not available
  - Fortran command DOACROSS

```
DO I = 2, N-1
   DO J = 2, N-1
      A(I, J) = .25 * (A(I-1,J)+A(I,J-1) +A(I+1,J)+A(I,J+1))
   ENDDO
ENDDO
```

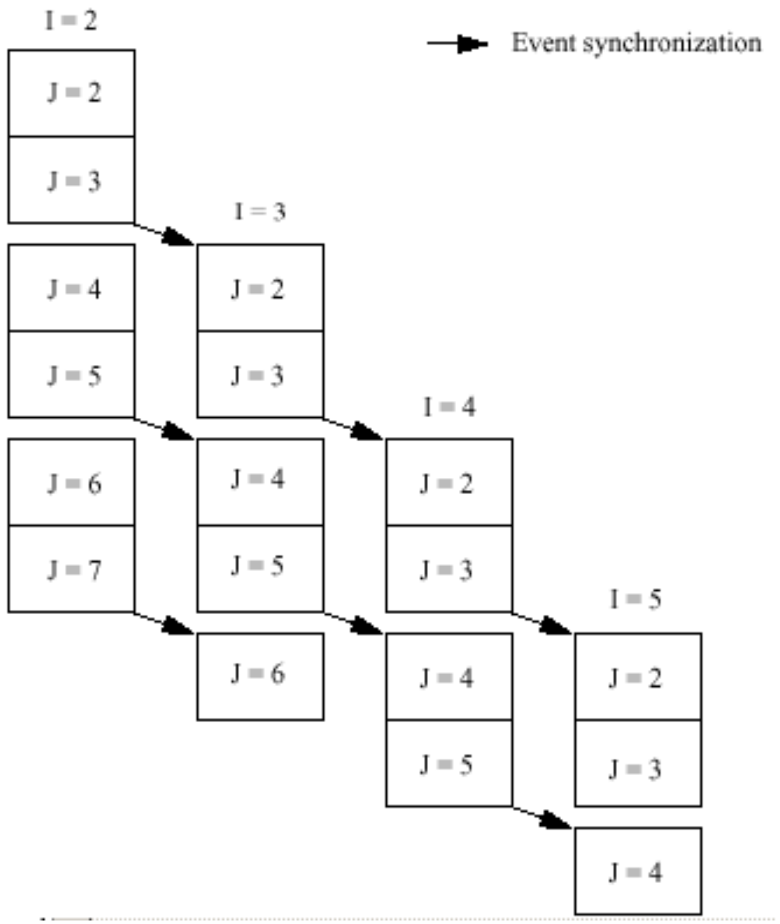
  - Pipelined Parallelism

```
DOACROSS I = 2, N-1
   POST (EV(1))
   DO J = 2, N-1
      WAIT(EV(J-1))
      A(I, J) = .25 * (A(I-1,J) + A(I,J-1)+ A(I+1,J) + A(I,J+1))
      POST (EV(J))
   ENDDO
ENDDO
```



Reducing Synchronization Cost

```
DOACROSS I = 2, N-1
    POST (E(1))
    K = 0
    DO J = 2, N-1, 2
        K = K+1
        WAIT(EV(K))
        DO j = J, MAX(J+1, N-1)
            A(I, J) = .25*(A(I-1,J) +
    A(I,J-1) + A(I+1,J) + A(I,J+1)
        ENDDO
        POST (EV(K+1))
    ENDDO
ENDDO
```

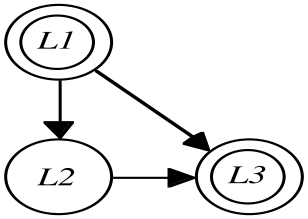


Loop Distribution and Fusion

- Loop distribution eliminates carried dependences by separating them across different loops
  - Good only for fine-grained parallelism
- Coarse-grained parallelism requires sufficiently large parallel loop bodies
  - Solution: fuse parallel loops together after distribution
  - Loop strip-mining can also be used to reduce communication
- Loop fusion is often applied after loop distribution
  - Regrouping of the loops by the compiler

Loop Fusion

- Transformation: opposite of loop distribution
  - Combine a sequence of loops into a single loop
  - Iterations of the original loops now intermixed with each other
- Safety: cannot have fusion-preventing dependences
  - Cannot bypass statements with dependences both from and to the fused loops
  - Loop-independent dependences cannot become backward carried after fusion



Fusing L1 with L3 violates the ordering constraint.

```
    DO I = 1,N
S1      A(I) = B(I)+C
    ENDDO
    DO I = 1,N
S2      D(I) = A(I+1)+E
    ENDDO
```

```
    DO I = 1,N
S1      A(I) = B(I)+C
S2      D(I) = A(I+1)+E
    ENDDO
```

Loop Fusion Profitability

- Parallel loops should generally not be merged with sequential loops.
    - A dependence is parallelism-inhibiting if it is carried by the fused loop
    - The carried dependence may be realigned via Loop alignment
- What if the loops to be fused have different lower and upper bounds?
    - Loop alignment, peeling, and index-set splitting

```
        DO I = 1,N
S1          A(I+1) = B(I) + C
        ENDDO
        DO I = 1,N
S2          D(I) = A(I) + E
        ENDDO
```

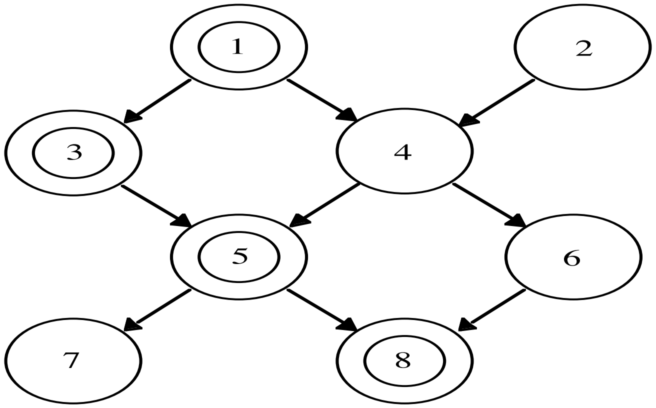
```
        DO I = 1,N
S1          A(I+1) = B(I) + C
S2          D(I) = A(I) + E
        ENDDO
```

The Typed Fusion Algorithm

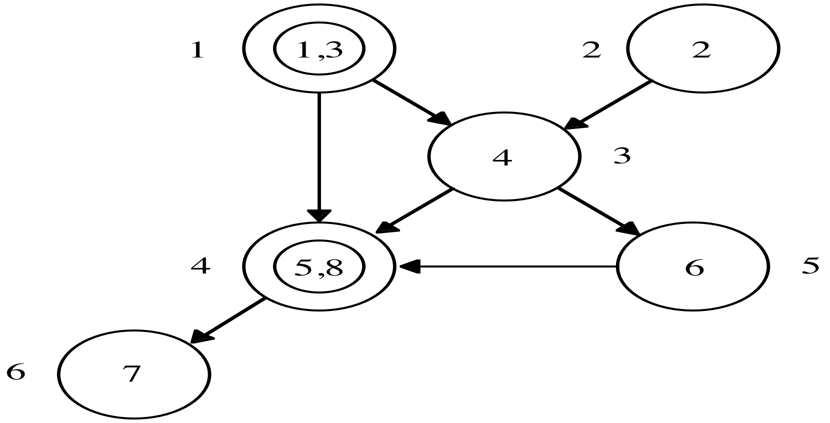
- Input: loop dependence graph (V,E)
- Output: a new graph where loops to be fused are merged into single nodes
- Algorithm
  - Classify loops into two types: parallel and sequential
  - Gather all dependences that inhibit fusion --- call them bad edges
  - Merge nodes of V subject to the following constraints
    - Bad Edge Constraint: nodes joined by a bad edge cannot be fused.
    - Ordering Constraint: nodes joined by path containing non-parallel vertex should not be fused

Typed Fusion Example

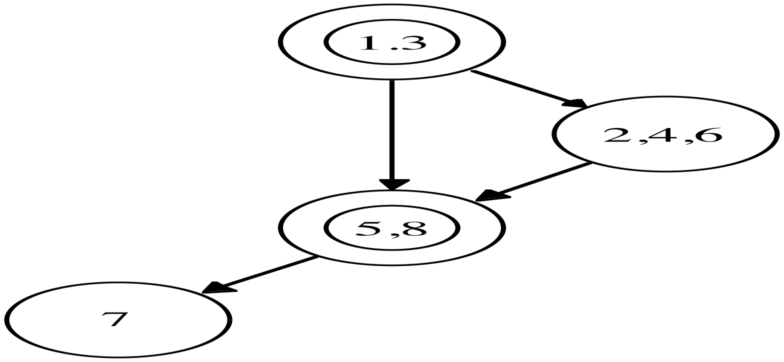
Original loop graph



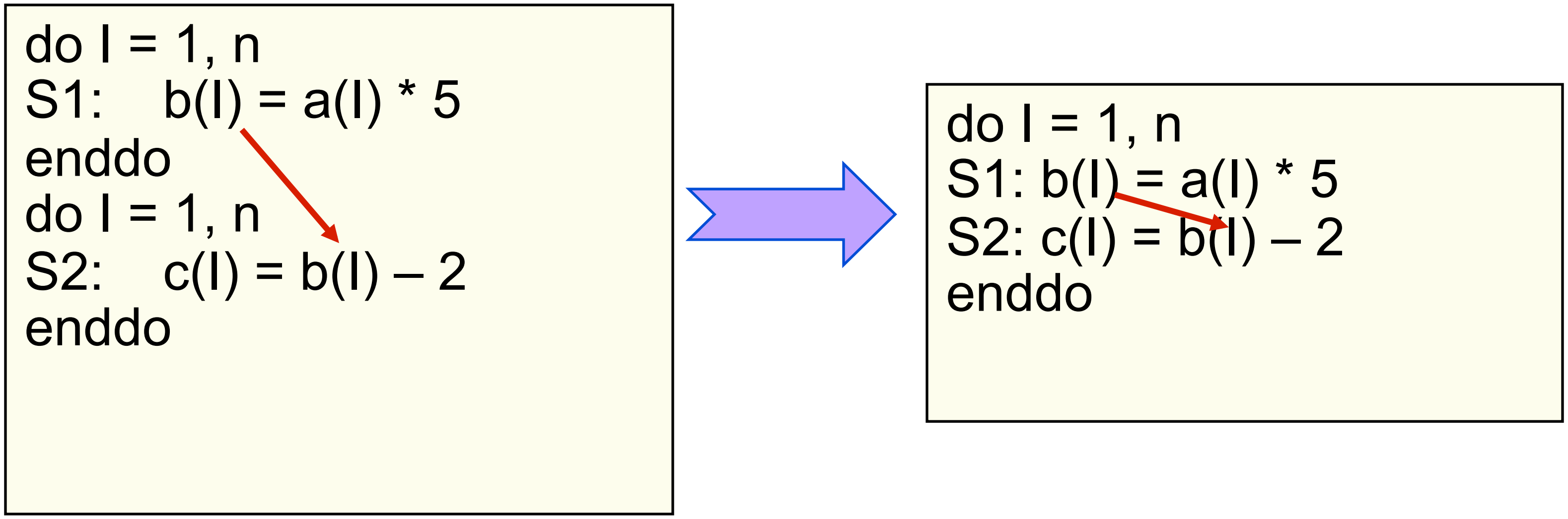
After fusing parallel loops



After fusing sequential loops



Loop Fusion/Fission For Locality



```
do I = 1, n
S1:    b(I) = a(I) * 5
enddo
do I = 1, n
S2:    c(I) = b(I) – 2
enddo
```

```
do I = 1, n
S1: b(I) = a(I) * 5
S2: c(I) = b(I) – 2
enddo
```

Putting It All Together

- Good Part
  - Many transformations imply more choices to exploit parallelism
- Bad Part
  - Choosing the right transformation
  - How to automate transformation selection?
  - Interference between transformations
- Effective optimization must
  - Take a global view of transformed code
  - Know the architecture of the target machine

- **Example of Interference**

```
DO I = 1, N
   DO J = 1, M
     S(I) = S(I) + A(I,J)
   ENDDO
ENDDO
```

Sum Reduction gives..

```
Parallel DO I = 1, N
   S(I) = S(I) + SUM(A(I,1:M))
ENDDO
```

Loop Interchange gives..

```
DO J = 1, N
   S(1:N) = S(1:N) + A(1:N,J)
ENDDO
```