CS536

Loop Optimizations

A Sahu

CSE, IIT Guwahati

Outline

- Basic Loop Optimization
- Basic Data Flow Analysis

Loop Optimization

http://www2.eecs.berkeley.edu/Pubs/TechRpts/1993/CSD-93-781.pdf

Bacon et al . Compiler Transformations for High-Performance Computing, ACM Computing Survey

Loop Optimization Example

- Basic LO
  - Loop invariant code removal
  - Induction variable strength reduction
  - Induction variable reduction
- Advance LO
  - Loop Interchange
  - Loop Splitting: Peeling Special Case
  - Loop Fusion/Jamming
  - Loop Fission/Distribution
  - Loop Unrolling

Loop Fusion

```
for (i = 0; i < 300; i++)
        a[i] = a[i] + 3;
for (i = 0; i < 300; i++)
        b[i] = b[i] + 4;
```

```
for (i = 0; i < 300; i++) {
        a[i] = a[i] + 3;
        b[i] = b[i] + 4;
}
```

Reduces branches

Improve parallelism

Create bigger basic block

Loop Fission/Split

```
for (i = 0; i < 1000; i++) {
  if(i%2==0)
          a[i] = a[i] + 10;
   else  a[i]= a[i] + 20;
}
```

```
for (i = 0; i < 1000; i=i+2)
          a[i]=a[i]+10;
for (i = 1; i < 1000; i=i+2)
           a[i]=a[i]+20;
```

Reduces branches (of if/else)
Both loop in total do for 1000
Improve parallelism

Loop Peeling

```
int p = 100;
for (int i=0; i<100; ++i) {
        y[i] = x[i] + x[p];
        p = i;
}
```

```
y[0] = x[0] + x[100];
for (int i=1; i<100; ++i) {
        y[i] = x[i] + x[i-1];
}
```

p = 100 only for the first iteration, and
for all other iterations, p = i - 1

Loop unrolling

```
for (x = 0; x < 100; x++) {
    A[x]=x*2+5;
}
```

```
for (x = 0; x < 100; x += 4 ) {
        A[x]=x*2+5;
        A[x+1]=(x+1)*2+5;
        A[x+2]=(x+2)*2+5;
        A[x+3]=(x+3)*2+5;
}
```

It improve parallelization

Increase size of the BB

Loop unrolling

```
for (x = 0; x < 100; x++) {
        process(x);
}
```

```
for (x = 0; x < 100; x += 4 ) {
        process(x);
        process (x + 1);
        process (x + 2);
        process (x + 3);
}
```

It improve parallelization

Software Pipelining

```
for (x = 0; x < 100; x++)
{         y=P1(x);
          z=P2(y);
}
```

```
y=P1(0);
for (x = 1; x < 99; x += 4 ) {
        y=P1(x); || ; z=P2(y);
        }
z=P2(99);
```

Depended work

0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7

Serial

0 1 2 3 4 5 6 7

0 1 2 3 4 5 6 7

Pipelined Parallelism

Seven Primitive Transformations

- Loop Fusion
- Loop Fission
- Re-Indexing
- Scaling
- Reversal
- Permutation
- Skewing

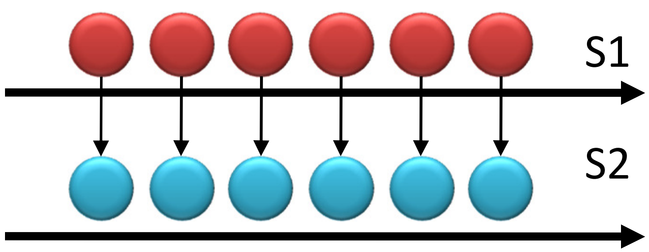
Transformations: Loop Fusion

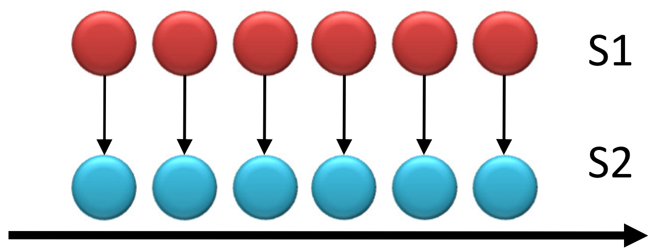
```
for(i=1;i<=N;i++)
   Y[i]=Z[i]; //s1
for(j=1;j<=N;j++)
   X[i]=Y[i]; // s2
```

Fusion

```
for(p=1;p<=N;p++){
   Y[p]=Z[p]; //S1
   X[p]=Y[p]; //S2
}
```

s1: p=i
s2: p=j





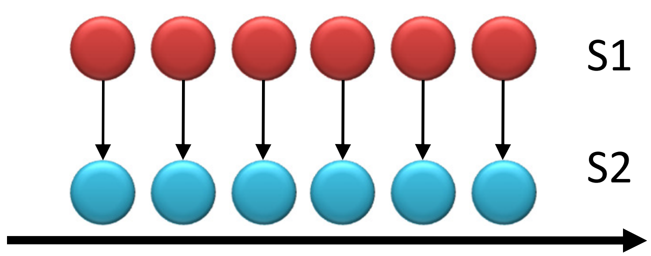
Transformations: Loop fusion

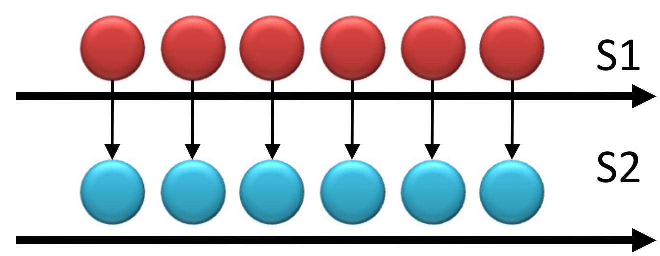
```
for(p=1;p<=N;p++){
    Y[p]=Z[p]; //S1
    X[p]=Y[p]; //S2
}
```

Fission

```
for(i=1;i<=N;i++)
   Y[i]=Z[i]; //s1
for(j=1;j<=N;j++)
   X[i]=Y[i]; // s2
```

s1: i=p
s2: j=p





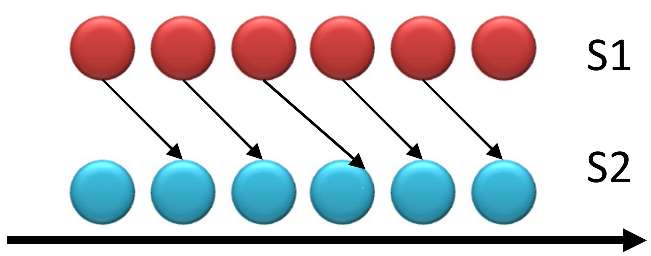
Transformations: Re-Indexing

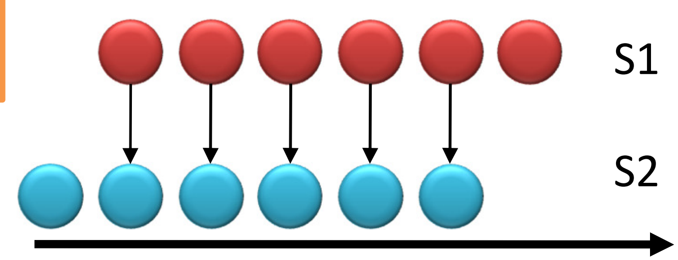
```
for(i=1;i<=N;i++){
   Y[i]=Z[i]; //S1
   X[i]=Y[i-1]; //S2
}
```

Re-Indexing

```
If (N>=1) X[1]=Y[0];
for(p=1;p<N;p++) {
   Y[p]=Z[p]; //s1
   X[p+1]=Y[p]; // s2
}
If(N>=1) Y[N]=Z[N];
```

s1: p=i
s2: p=i-1





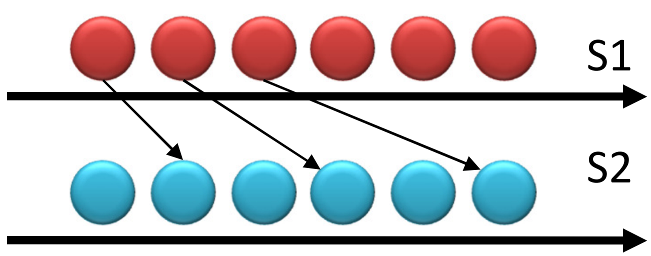
Transformations: Scaling

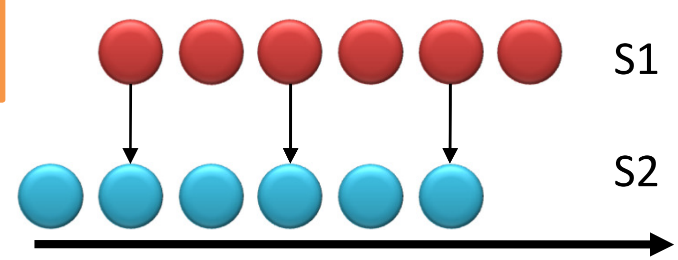
```
for(i=1;i<=N;i++){
   Y[2*i]=Z[2*i]; //S1
}
For(j=1;j<=N;j++){
   X[j]=Y[j]; //S2
}
```

Scaling

```
for(p=1;p<=2*N;p++) {
  if (p%2==0)
      Y[p]=Z[p]; //s1
  X[p]=Y[p]; // s2
}
```

s1: p=2*i
s2: p=j





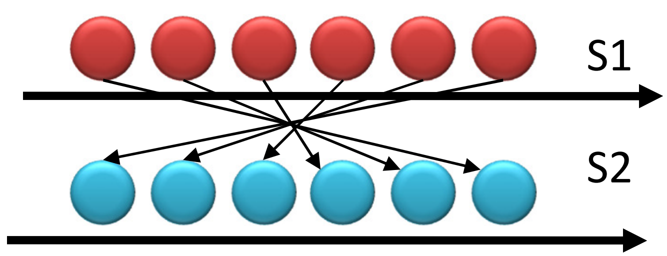
Transformations: Reversal

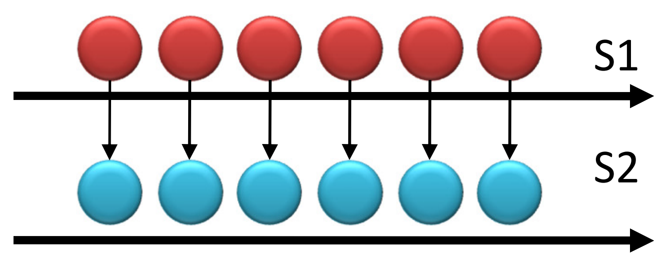
```
for(i=1;i<=N;i++){
   Y[N-i]=Z[i]; //S1
}
For(j=1;j<=N;j++){
   X[j]=Y[j]; //S2
}
```

Reversal

```
for(p=1;p<=N;p++) {
  Y[p]=Z[N-p];
  X[p]=Y[p]
}
```

s1: p=N-i
s2: p=j



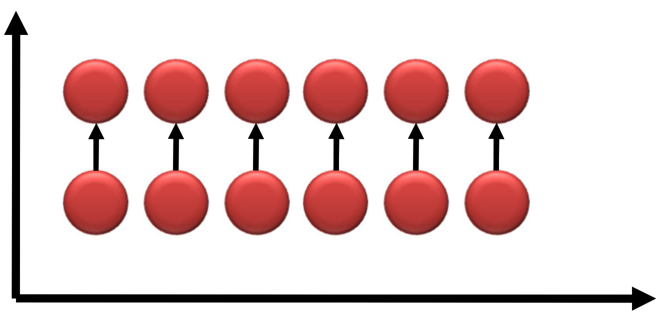


Transformations:Permutation

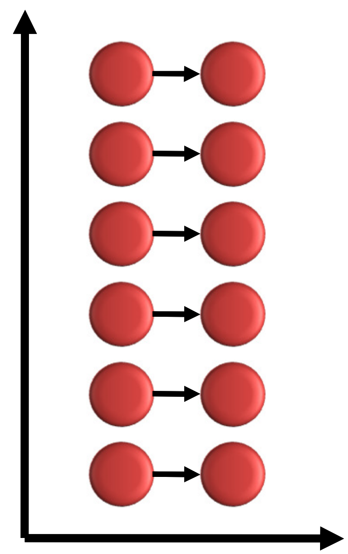
```
for(i=1;i<=N;i++){
   for(j=1;j<=M;j++){
      Z[i][j]=Z[i-1][j];
   }
}
```

Permutation

```
for(p=1;p<=M;p++) {
   for(q=1;q<=N;q++) {
      Z[q][p]=Z[q-1][p];
   }
}
```



$$\begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$$



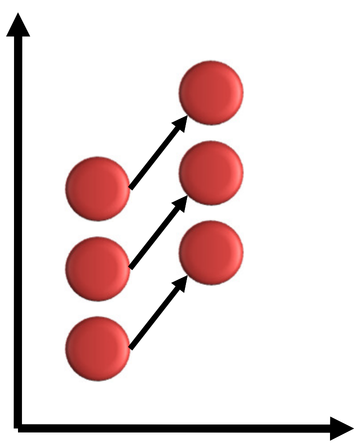
//confusion in class, it was Z[p][q] instead of Z[q][p]

Transformations:Skewing

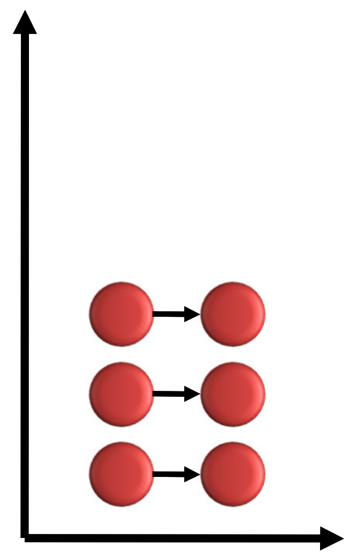
```
for(i=1;i<=N+M-1;i++){
   for(j=max(1,i+M) ;
     j<=min(i,M);j++){
       Z[i][j]=Z[i-1][j-1];
   }
}
```

Permutation

```
for(p=1;p<=M;p++) {
   for(q=1;q<=N;q++) {
      Z[p][q-p]
              =Z[p-1][q-p-1];
   }
}
```



$$\begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I \\ j \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$



Loop Optimizations

- Most important set of optimizations
  - Programs are likely to spend more time in loops
- Presumption: Loop has been identified
- Optimizations:
  - Loop invariant code removal
  - Induction variable strength reduction
  - Induction variable reduction

Loops in Flow Graph

- **Dominators**: A node *d* of a flow graph *G* dominates a node *n*, if every path in *G* from the initial node to *n* goes through *d*.

  Represented as: ***d dom n***

- Corollaries:
  - Every node dominates itself.
  - The initial node dominates all nodes in *G.*
  - The entry node of a loop dominates all nodes in the loop.