Module 13: INTRODUCTION TO COMPILERS FOR HIGH PERFORMANCE COMPUTERS

Lecture 26: SIMD Architecture

The Lecture Contains:

Previous Next

- Program has high synchronization cost: n2 forks and synchronizations.
- Different processors may access c[i,j] and c[i,j+1] which may be on the same cache line.

Execute Outer Loop in Parallel

```
doall i := 1 to n do
    for k := 1 to n do
        for j := 1 to n do
            c[i,j] = c[i,j] + a[i,k] * b[k,j]
        endfor
    endfor
endall
```

- Only one fork and synchronization
- Each task is large grain operation
- Each task fetches whole array B

SIMD Architecture

- Single front end issuing instructions
- Back ends or PEs execute each instruction
- Code divided into scalar and parallel code
- Scalar code executes on front end
- Each PE can access local memory directly
- Messages must be used to access values from another PE’s memory
- Assume each PE has one row of each matrix
- Variables 1A, 1B and 1C contain rows of A, B and C on each PE
- Fetch operation fetches 1B[j] from PE
- B*kj* is front end variable

```
for j := 1 to n do
    PE(0:n-1):ctmp = 0
    for k := 1 to n do
        Bkj = fetch(k-1)(1B[j])
        PE(0:n-1):ctmp = ctmp + 1A(k) * Bkj
    endfor
    PE(0:n-1):1C[j] = ctmp
endfor
```

Previous Next

Message Passing Architecture

```
for j := 1 to n do
    ctmp = 0
    for k := 1 to n do
        if (k-1 = Pindex)
        then
            Bkj = 1B[j]
            broadcast(Bkj)
        else
            receive(Bkj)
        endif
        ctmp = ctmp + 1A(k) * Bkj
    endfor
    1C[j] = ctmp
endfor
```

Minimize Broadcast

```
for k := 1 to n do
    if (k-1 = Pindex)
    then
        Bk[1 : n] = 1B[1:n]
        broadcast(Bk[1 : n])
    else
        receive(Bk[1 : n])
    endif
    for j = 1 to n do
        1C[j] = 1C[j] + 1A(k) * Bk[j]
    endfor
endfor
```

Previous Next

SPMD Code After Strip Mining

```
for k := 1 to n do
if (k-1 = Pindex)
then
Bk[1 : n] = 1B[1:n]
broadcast(Bk[1 : n])
else
receive(Bk[1 : n])
endif
for j = 1 to n do
for i = 1 to nrows do
1C[i,j] = 1C[i,j] + 1A(i,k) * Bk[j]
endfor
endfor
endfor
```

Sequential and Parallel Loops

Sequential Loop: The second iteration does not start until the first iteration is complete.
Forall Loop: This is a parallel loop corresponding to an array assignment. Each of the statements is executed completely for all the values of the index variable before the next statement is started.

```
Forall i= 1, n
S1                                    S1(1:n)
S2             is equivalent to      S2(1:n)
⋮                                     ⋮
Sm                                    Sm(1:n)
Endall
```

Dopar Loop: This is a parallel loop corresponding to parallel processors. Each iteration of the loop is executed in parallel by a different processor.

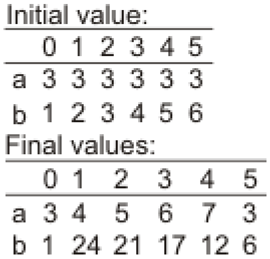
- The code within each iteration executes sequentially
- Initial state seen by each processor is same as the state before the loop
- Any variable update done by a processor can not be seen by any other processor
- If two iteration change the same variable, the result is non-deterministic merge

Previous Next

Dosingle Loop

- Represents single assignment statement
- Each variable assigned must be indexed by all loop index variables
- Each element must be assigned only once
- A statement in a dosingle sees all the updates
    - From previous or subsequent iterations
    - From previous or subsequent statements

```
dosingle i = 1 to 4 do
a[i] = a[i-1] + 1
b[i] = b[i+1] + a[i-1]
enddo
```



Initial value:

	0	1	2	3	4	5
a	3	3	3	3	3	3
b	1	2	3	4	5	6

Final values:

	0	1	2	3	4	5
a	3	4	5	6	7	3
b	1	24	21	17	12	6

Summary

- Re-structure code to exploit pipeline and caches.
- Convert sequential loops involving scalars to vector operation.
- Convert sequential loops into parallel loops.
- Partition data for data parallel model.
- Minimize messages to reduce overheads.

Control Flow Analysis

- Shows hierarchical flow of control
- Source control flow is not available in MIR or LIR
- Loops may be constructed of ifs and gotos

Previous Next

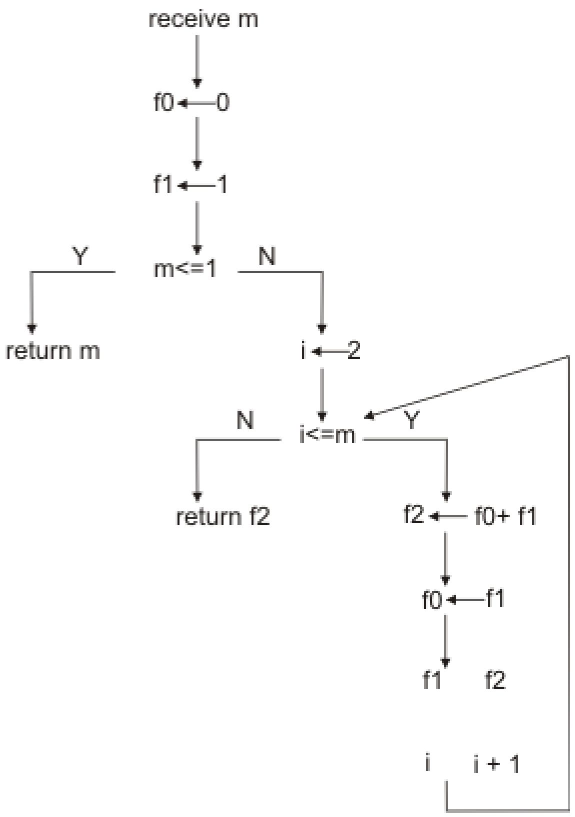
Example 1

```
unsigned int fib(m)
unsigned intm;
{
unsigned int f0 = 1, f1 = 1, f2, i;
if(m <= 1){
returnm;
}
else{
for(i = 2; i <= m; i + +){
f2 = f0 + f1;
f0 = f1;
f1 = f2;
}
return f2;
}
}
```

Example 1 Continued...

```
receive m
f0 ← 0
f1 ← 1
if m <= 1 got L3
i ← 2
L1: if i <= m goto L2
return f2
L2: f2 ← f0 + f1
f0 ← f1
f1 ← f2
i ← i + 1
goto L1
L3: return m
```

Previous Next



Basic Blocks

Useful for collecting information for optimization

Basic Block : Sequence of consecutive statements where flow of control enters at the beginning and leaves at the end. No branching permitted at an intermediate statement.

Partition Into Basic Blocks

Input : A sequence of three address statements

Output : A list of basic blocks

1. Mark leaders
   a. First statement is a leader
   b. Any statement which is target of a goto is a leader
   c. Any statement that follows a goto statement is leader
2. For each leader all the statement following it up to the next leader or the end of the program make a basic block.

Previous Next

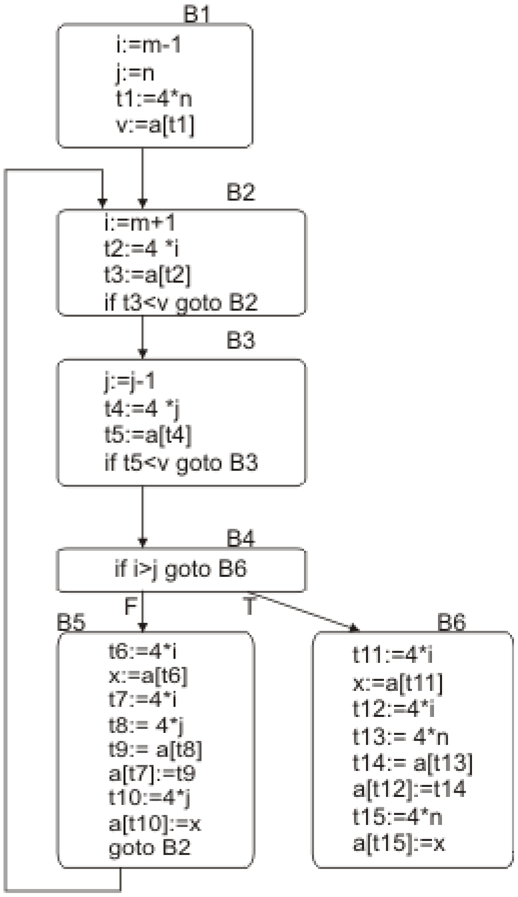
```
(1) i := m - 1
(2) j := n
(3) t1 := 4* n
(4) v := a[t1]
(5) i := i + 1
(6) t2 := 4 * i
(7) t3 := a[t2]
(8) if t3 < v goto (5)
(9) j := j - 1
(10) t4 := 4 * j
(11) t5 := a[t4]
(12) if t5 > v goto (9)
(13) if i >= j goto (23)
(14) t6 := 4 * i
(15) x := a[t6]
(16) t7 := 4 * i
(17) t8 := 4 * j
(18) t9 := a[t8]
(19) a[t7] := t9
(20) t10 := 4 * j
(21) a[t10] := x
(22) goto (5)
(23) t11 := 4 * i
(24) x := a[t11]
(25) t12 := 4 * i
(26) t13 := 4 * n
(27) t14 := a[t13]
(28) a[t12] := t14
(29) t15 := 4 * n
(30) a[t15] := x
```

Flow Graph

Add flow of control information to basic blocks. The directed graph is called flow graph. The nodes of the flow graph are basic blocks.

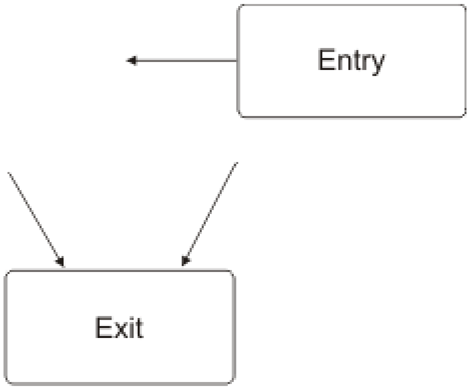
One node is initial. There is a directed edge from block *B1* to block *B2* if *B2* follows *B1* in execution order.

- Jump from *B1* to *B2*
- *B2* follows *B1* in the order of the program



Previous Next

- Introduce two special nodes entry and exit
- Control always enters through the entry node
- Control always leaves through the exit node



Loops in Flow Graphs

A loop has a single entry point and the components of a loop are strongly connected in a CFG.

Dominators

A node d dominates a node n if every path from entry to n passes through d. Every node dominates itself.
Dominance is a reflexive partial order.

- Reflexive: A dom a a
- Antisymmetry: A dom b and b dom a a=b
- Transitive: A dom b and b dom c a dom c

Previous Next