SOFTWARE PIPELINING

- for loops, also with loop-carried dependences
  - basic idea: overlap instructions from different iterations to increase instruction-level parallelism
  - profitable especially for long pipeline delays, multiple units may be combined with loop unrolling
  - high register need
  - optimal solution: NP-complete (as usual…)

Towards Software Pipelining

- code block (unrolling factor – 1) \times body size
- longer basic blocks for local optimizations

Recall Loop Unrolling

- unroll basic blocks for local optimizations
- for loops, also with loop-carried dependences
Towards Software Pipelining

Goal:
Combine operations from different subsequent iterations to form a new kernel (pattern, stable state) such that no data dependences are violated and no resource conflicts occur.

The length of the kernel is called the initiation interval (II):

\[ II = \text{# clock cycles between issuing the first instructions of two subsequent kernel iterations} \]

Total execution time:

\[ \text{Total execution time} = \text{length (prologue)} + \text{length (epilogue)} + II \]

Two operations \( A, B \) are in conflict if they require the same resource at the same time.

Prerequisite: Resource reservation table

Two operations \( A, B \) in conflict:

- Need to check for resource conflicts during scheduling

Prerequisite: Classification of data dependences

Notation:

- \( a_i \): instance of instruction \( a \) in iteration \( i \)
- \( d \): dependence distance: 0 for loop-independent, \( > 0 \) for loop-carried dep.
- \( l \): minimum delay time (in cycles until dependent operation can be issued)

Dependence difference (\( d \)): \( d = d_i - d_j \)

Dependence from \( a \) to \( b \) at iteration \( i \):

\[ @a_i \rightarrow b_i \]

Total execution time:

\[ T = \sum_{i=0}^{N-1} (l_i + d_i) \]

Optimum:

\[ \text{Minimum } II. \]

Two categories of data dependences:

- Datapath:
  - Dataflow control
  - Parallelism
- Controlpath:
  - Control flow

Prerequisite: Classification of data dependences

Two categories of algorithms:

- Software pipelining
- Microcode generation

Example: Preceded add, pipelined multiply

Resource reservation table

Use to check for resource conflicts during scheduling

Here, add cannot follow a mul by 2 cycles.
Resources:

Loower bounds for the minimum initiation interval $M_{II}$:

$$M_{II} = \text{number of instructions in the original loop body}$$

$$P = \text{number of processors/functional units}$$

$$L_{\text{min}} = \text{maximum over all resources}$$

### Lower bound on $M_{II}$

Assume: $4$ units, fully pipelined

$M = 2$

$P = 4$

$J = 4$

$M = 2$

### Uniform Resource Constraints

#### Computing $T_{\text{min}}$

The minimum initiation interval is relatively simple.

### Module Resource Reservation Table

- $M_{II}$ is relatively simple.

### References

- Lam [88]
- Zaky [89]
- Floyd-Warshall
- Allman et al. [95]
- Lam [88]

### Computing $M_{II}$

- Determining $M_{II}$ is relatively simple.
- Computing the minimum initiation interval $M_{II}$.
- Determine Recurrence constraints:
  - (multiplication over ratios for each resource type)
  - (lower bound on $M_{II}$)
  - (max) Res $= 2$
  - (cycle lengths) $= 2$

### Linear Programming

- Composing distance matrices for a path algebra.
- Loop algebra.
- Floyd-Warshall.
- Examine all cycles.
- Determine $M_{II}$.

### Scheduling the Kernel

- Apply some placement heuristic, e.g.: as early as possible.
- Apply some placement heuristic, e.g.: list scheduling. (A B C D E F G)
- Mark occupied slots in all iterations...
- If not possible, increase $II$ and try again.

### Scheduling the Kernel (2)

- Modulo resource reservation table.
- If there are recurrences in the dependence graph:
  - Find SCC's
  - Find cycle in SCC with longest accumulated distance
  - Schedule each SCC individually
  - Compose distance matrices for all paths

### Remarks

- More about this in the survey paper by Allen et al. '95.
- Linear programming.
- Composing distance matrices for a path algebra.
- Loop algebra.

### Computing the Minimum Initiation Interval

- Disadvantages: separate schedules of SCCs may not fit well together.
- Similarities: collapse if they are statements.
- Apply list scheduling to resulting acyclic graph.
- Include cycles in SCC individually.
- Induce cycles in SCC with longest accumulated distance.
- Find SSCs.

### Resource Constraints

- $R_{\text{min}}(I/I) = \text{ceil}(T_{\text{min}})$
Register assignment

Problem:
advanced by window size, modulo register file size, register renaming / modulo variable expansion

Software methods
Hardware support

+ Higher resource utilization than loop unrolling
+ Modular code size increase
+ Modifiable by compiler transformation at target level

Advantages over loop unrolling: e.g. in OLE backend CG for IA-64

Regardles software pipelining is NP-complete

Further reading on resource-constrained software pipelining:

Optimal software pipelining is NP-complete

Loop unrolling may provide additional efficiency

Software pipelining for loops with conditional branches [Stoelday, Lee '96]

May use predication to save prologue and epilogue code [Rau et al., '92]

Extensions


Furthermore, standard compiler transformations at target level can be combined with unrolling:

- Higher resource utilization than loop unrolling
- Modular code size increase