Loop Scheduling and Software Pipelining
Reading List

- Slides: Topic 7 and 7a
- Other papers as assigned in class
  or homework:
ABET Outcome

• Ability to apply knowledge of basic code generation techniques, e.g. Loop scheduling e.g. software pipelining techniques to solve code generation problems.
• An ability to identify, formulate and solve loops scheduling problems using software pipelining techniques.
• Ability to analyze the basic algorithms on the above techniques and conduct experiments to show their effectiveness.
• Ability to use a modern compiler development platform and tools for the practice of above.
• A Knowledge on contemporary issues on this topic.
Outline

- Brief overview
- Problem formulation of the modulo scheduling problem
- Solution methods
- Summary
General Compiler Framework

- Good IPO
- Good LNO
- Good global optimization
- Good integration of IPO/LNO/OPT
- Smooth information passing between FE and CG
- Complete and flexible support of \textit{inner-loop} scheduling (SWP), instruction scheduling and register allocation
Questions?

- How to formulate the loop scheduling problem?
- How to model it?
- How to solve it?
Questions (cont’d)

• Instruction scheduling for code without loops – a review
• Dependence graphs may become cyclic! So, “critical path length” is less obvious!
• Is it becoming harder! (?)
• What new insights are required to formulate and solve it?
Challenges of Loop Scheduling

A DDG With Cycles

Right strategy?
Observations

- Execution of “good loops” tend to be “regular” and “repetitive” - a “pattern” may appear
- This gives “cyclic scheduling” problem a new twist!
- How to efficiently derive a “pattern”?
Problem Formulation (I)

Given a weighted dependence graph, derive a schedule which is “time-optimal” under a machine model $M$.

**Def:** A schedule $S$ of a loop $L$ is time-optimal if among all “legal” schedules of $L$, no other schedule that is faster than $S$.

**Note:** There may be more than one time-optimal schedule.
A Short Tour on Data

Dependence Graphs for Loops
Basic Concept and Motivation

- Data dependence between 2 accesses
  - The same memory location
  - Exist an execution path between them
  - One of them is a write

- Three types of data dependence
- Dependence graphs
- Things are not simple when dealing with loops
Types of Data Dependence

- Flow dependence
- Anti-dependence
- Output dependence
Data Dependence

Example 1:

S1: A = 0
S2: B = A
S3: C = A + D
S4: D = 2

\[ S_x \delta S_y \Rightarrow S_y \text{ depends on } S_x \]
Data Dependence

Example 2:

S1: A = 0
S2: B = A
S3: A = B + 1
S4: C = A

\( S_1 \xrightarrow{\delta^0} S_3 \) : Output-dep
\( S_2 \xrightarrow{\delta^1} S_3 \) : anti-dep
Should we consider input dependence?

\[ = X \]

Is the reading of the same \( X \) important?

Well, it may be!
(If we intend to group the 2 reads together for cache optimization!)
Subscript Variables

- Extension of def-use chains to employ a more precise treatment of arrays, especially in iterative loops.

```
DO I = 1, N
    A(I + 1) = X(I + 1) + B(I)
    X(I) = A(I) * 5
ENDDO
```
Dependence Graph

Applications
- register allocation
- instruction scheduling
- loop scheduling
- vectorization
- parallelization
- memory hierarchy optimization

Con’d
Data Dependence in Loops

An Example

Find the dependence relations due to the array X in the program below:

(1) for I = 2 to 9 do
(2) X[I] = Y[I] + Z[I]
(3) A[I] = X[I-1] + 1
(4) end for

Solution

To find the data dependence relations in a simple loop, we can unroll the loop and see which statement instances depend on which others:

<table>
<thead>
<tr>
<th></th>
<th>I = 2</th>
<th>I = 3</th>
<th>I = 4</th>
</tr>
</thead>
</table>
In our example, there is a loop-carried, lexically forward flow
dependence relation.

Data dependence graph for statements in a loop.

- Loop-carried vs loop-independent
- Lexical-forward vs lexical backward
An Example

for i = 0 to N - 1 do
a: a[i] = a[i - 1] + R[i];
b: b[i] = a[i] + c[i - 1];
c: c[i] = b[i] + 1;
end;

Note: We use a token here to represent a flow dependence of distance = 1

<table>
<thead>
<tr>
<th>time</th>
<th>i = 0</th>
<th>i = 1</th>
<th>i = 2</th>
<th>i = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>c</td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

So, iteration interval II = 2

Assume each operation takes 1 cycle and there is only one addition unit!
Software Pipeline: Concept

Software Pipeline is a technique that reduces the execution time of important loops by interweaving operations from many iterations to optimize the use of resources.
The Structure of the SWP code

% prologue
a[0] = a[-1] + R[0];

% pattern
for i = 0 to N-2 do
    b[i] = a[i] + c[i -1];
    a[i + 1] = a[i] + R[i + 1];
    c[i] = b[i] + 1;
end;

% epilogue
b[N - 1] = a[N - 1] + c[N - 2];
c[N - 1] = b[N - 1] + 1
Software Pipeline (Cont’d)

What limits the speed of a loop?
• **Data dependencies**: recurrence initiation interval (rec_mii)
• **Processor resources**: resource initiation interval (res_mii)
Previous Approaches

- **Approach I** (Operational):
  “Emulate” the loop execution under the machine model and a “pattern” will eventually occur

  [AikenNic88, EbciogluNic89, GaoEtAl91]

- **Approach II** (Periodic scheduling):
  Specify the scheduling problem into a periodical scheduling problem and find optimal solution

  [Lam88, RauEtAl81, GovindAltmanGao94]
Periodic Schedule
(Modulo Scheduling)

The time (cycle) when the I-th instance of the operation v is scheduled:

\[ t(i, v) = T \times i + A[v] \quad \text{where } T = II \]

so \[ t(i + 1, v) - t(i, v) = T^*(i + 1) - T^*(i) = T \]

For our example:

\[ t(i, v) = 2i + A[v] \]

where \[ A(a) = 1 \]
\[ A(b) = 0 \]
\[ A(c) = 1 \]

**Question:** Is this an optimal schedule?
Yes, the schedule
\[ t(i, v) = 2i + A[v] \]
is time-optimal!

With \( I = 2 \)
Restate the problem

Given a DDG of a loop L, how to determine the fastest computation rate of L -- also called minimum initiation interval (MII) ?

Hint: Consider the “Critical Cycles” as well as critical resource usage in L
Recurrence MII -- RecMII

• An Example
An Example (Revisit) : RecMII

for i = 0 to N - 1 do
  a: \( a[i] = a[i - 1] + R[i]; \)
  b: \( b[i] = a[i] + c[i - 1]; \)
  c: \( c[i] = b[i] + 1; \)
end;

Assume each operation takes 1 cycle and there is only one addition unit!
Maximum Computation Rate

**Theorem:** The maximum computation rate of a loop is bounded by the following ratio

$$r_{opt}(C) = \min\{ \frac{Dc}{Wc} \}$$

where $C$ is a dependence cycle, $Dc$ is the total dependence distance along $C$ and $Wc$ is total execution time of $C$. i.e.

$$Dc = \sum_c d_i \quad (d_i \text{ is the dependence distance along the edge } i \text{ in } C)$$

$$Wc = \sum_c w_i \quad (w_i \text{ is the edge weight along the edge } i \text{ in } C)$$

Hint: now one must think about deadlines.
RecMII (Contn’d)

And the optimal period

\[ MII = T_{opt} = \frac{1}{r_{opt}} \]

**Def:** A cycle is critical if the period of the cycle equal to MII

*(We should write it as RecMII)*

**Note:** a loop may have multiple critical cycles!
An Example (Revisit) : RecMII

for i = 0 to N - 1 do
  a:    a[i] = a[i-1] + R[i];
  b:    b[i] = a[i] + c[i-1];
  c:    c[i] = b[i] + 1;
end;

Assume each operation takes 1 cycle and there is only one addition unit!
How About Machine Resource and Register Constraints?

- Numbers and types of FUs, etc, (ResMII)
- Number of registers
Software Pipelining

- Review of previous work
  \[\text{[RauFisher93]} \text{[Rau94]}\]

- Minimum initiation interval
  \[\text{[MII]}\]

\[\text{MII} = \text{max} \{\text{RecMII}, \text{ResMII}\}\]

where

\text{RecMII: determined by “critical” recurrence cycles in DDG}

\text{ResMII: determined by resource constraints (#s and types of FUs)}
Consider a simple example of n nodes and 1 FU -- what is ResMII?

The Resource Constraint MII (ResMII)

- Can be calculated by totaling, for each resource, the usage requirement imposed by one iteration of the loop
- can be done by bin-packing a resource reservation table (expensive)
- usually derive a lower-bound is enough as a starting point to begin the search process
- More price with complex hardware pipelines?
Example 1: Reservation Table - An Example

(a) a Pipeline M

(b) The Reservation Table of M

<table>
<thead>
<tr>
<th>Stage</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Quiz?

• What is the RT of a fully pipelined adder with 3 stages?
• How about an unpipelined adder?
• How to compute ResMII for each case?
Modulo Reservation Table

• The modulo reservation table only has length = II
• Each entry in the table records a sequence of reservations corresponds a sequence of slots every II cycles
• Hints: think about your weekly calendar
Modulo Scheduling

Choose Next Operation X

try to place x in a time slot i in II

succeed

Remove from L is L =<>?

No

A legal schedule is found

failed

II = II + 1

MII
Heuristic Method for Modulo Scheduling

Why a simple variant list scheduling may not work?

Hint: consider the deadline constraints of operations in a cycle.
Counter Example I:  List Scheduling May Fail!

(a)

(b) MII = RecMII = 4

Note: if simple list scheduling is used
B cannot be scheduled due to the
deadline set by scheduling C: deadlock!

(c)
MII = RecMII = 4

Note: we cannot fire C as early as possible!
Example I (Cont’d)

In previous figure,

We show an example demonstrating the problem with greedy scheduling in the presence of recurrences.

(a) The data dependence graph with a cycle.
(b) The resulting partial schedule when C is scheduled greedily. B cannot be scheduled.
(c) The resulting valid schedule when C is not scheduled greedily – delayed to two cycles later.
Example 2: Problems with Greedy schedule due to ResMII

A: non-pipelined adders
M: non-pipelined multipliers

(a) DDG
(b) A greedy schedule which cannot schedule A4 with ResMII
(c) a non-greedy schedule which achieves ResMII
Example 2 (Cont’d)

In previous figure,

We show an example demonstrating the problem with greedy scheduling in the presence of complex reservation tables.

(a) The data dependence graph without cycle.

(b) The resulting partial schedule when A1, M6, C2, and A3 are scheduled greedily. A4 cannot be scheduled.

(c) The resulting valid schedule when A3 is scheduled one cycle later.
Example 3: infeasibility of MII

Note: ResMII = 2. But is there a legal schedule under II = 2?

Note: The presence of complex reservation tables
Example 3 (cont’d)

Adder   Mult     Bus
A1       M2        M2

Adder   Mult     Bus
A1       M2        A1
MA3       MA3      A1
MA3       M2

MII = 2
II = 2

(b) cannot fit MA3
under II=2

MII = 2
II = 3

(c) A feasible schedule
for II=3

Note: It is possible that there is no valid schedule at MII!
Infeasibility of MII

The previous slide shows an example demonstrating the infeasibility of the MII in the presence of complex reservation tables. (a) The three operations and their reservation tables. (b) The MRT corresponding to the dead-end partial schedule for an II of 2 after A1 and M2 have been scheduled. (c) The MRT corresponding to a valid schedule for an II of 3.
Example 4: Infeasibility of MII

Assume: fully pipelined adders

Note: It is possible that there is no valid schedule at MII! And, the reservation table here is simple!
Example 4 (cont’d)

The previous slides shows an example demonstrating the infeasibility of the MII due to the interaction between the recurrence constraints and the resource usage constraints.
(a) The data dependence graph.
(b) The MRT corresponding to the dead-end partial schedule for an II of 4 after A1 and A2 have been scheduled.
How to derive a best feasible schedule?

- It is possible do so via exhaustive search.
- But, it is expensive!
A Taxonomy of Software Pipelining

Operational Approach

Software Pipelining

Periodic Scheduling (Modulo Scheduling)

Heuristic (Aiken88, AikenNic88, Ebciohlglu89, etc.)

Formal Model \((GaoWonNin91)\)

In-Exact Method (Heuristic)
\((RauGla81, Lam88, RauEtAl92, Huff93, DehnertTow93, Rau94, WanEis93, StouchininEtAl00)\)

Basic Formulation \((DongenGao92)\)

Exact Method

"Showdown" \((RuttenbergGao StouchininWoody96)\)

Register Optimal \((Ninggao91, NingGao93, Ning93)\)

Resource Constrained \((GovindALTGao94)\)

Resource & Register \((GovindALTGao95, Altman95, PLDI95)\)

ILP based

Exhaustive Search \((EuroPar96)\)

FSA Co-Scheduling formulation \((GovindAltmanGao96)\)

FSA Construction Method \((GovindAltmanGao98)\)

FSA Heuristic/optimization \((ZhangGovindRyanGao99)\)

Theory of Co-Scheduling \((GovindAltmanGao00)\)

FSA Based Method
(Model hardware pipeline with sharing and hazards)
Advanced Topics

- Consider register constraints
- Realistic pipeline architecture constraints (with structural hazards)
- Loop body with conditionals
- Multi-Dimensional Loops