Data Prefetch and Software Pipelining
Agenda

- *Data Prefetch*
- Software Pipelining
Why Data Prefetching

- Increasing Processor – Memory “distance”
- Caches do work !!! … IF …
  - Data set cache-able, accesses local (in space/time)
- Else ? …
Data Prefetching

What is it?

- Request for a future data need is initiated
- Useful execution continues during access
- Data moves from slow/far memory to fast/near cache
- Data ready in cache when needed (load/store)
Data Prefetching

- When can it be used?
  - Future data needs are (somewhat) predictable

- How is it implemented?
  - In hardware: history based prediction of future access
  - In software: compiler inserted prefetch instructions
Software Data Prefetching

- Compiler scheduled prefetches
- Moves entire cache lines (not just datum)
  - Spatial locality assumed – often the case
- Typically a non-faulting access
  - Compiler free to speculate prefetch address
- Hardware not obligated to obey
  - A performance enhancement, no functional impact
  - Loads/store may be preferentially treated
Software Data Prefetching

Use

- Mostly in Scientific codes
  - Vectorizable loops accessing arrays deterministically
    - Data access pattern is predictable
    - Prefetch scheduling easy (far in time, near in code)
  - Large working data sets consumed
    - Even large caches unable to capture access locality

- Sometimes in Integer codes
  - Loops with pointer de-references
Selective Data Prefetch

do j = 1, n
  do i = 1, m
    A(i,j) = B(1,i) + B(1,i+1)
  enddo
enddo

E.g. A(i,j) has spatial locality, therefore only one prefetch is required for every cache line.
Formal Definitions

- **Temporal locality** occurs when a given reference reuses exactly the same data location.
- **Spatial locality** occurs when a given reference accesses different data locations that fall within the same cache line.
- **Group locality** occurs when different references access the same cache line.
Prefetch Predicates

- If an access has spatial locality, only the first access to the same cache line will incur a miss.
- For temporal locality, only the first access will incur a cache miss.
- If an access has group locality, only the leading reference incurs cache miss.
- If an access has no locality, it will miss in every iteration.
Example Code with Prefetches

do j = 1, n
  do i = 1, m
    \[ A(i,j) = B(1,i) + B(1,i+1) \]
    if (i\&7 == 0)
      prefetch \( A(i+k,j) \)
    if (j == 1)
      prefetch \( B(1,i+t) \)
  enddo
enddo

Assumed CLS = 64 bytes and data size = 8 bytes

k and t are prefetch distance values
Spreading of Prefetches

- If there is more than one reference that has spatial locality within the same loop nest, spread these prefetches across the 8-iteration window.

- Reduces the stress on the memory subsystem by minimizing the number of outstanding prefetches.
Example Code with Spreading

\[ \text{do } j = 1, n \text{ do } i = 1, m \]
\[ C(i,j) = D(i-1,j) + D(i+1,j) \]
\[ \text{if } (i \text{ and } i,7) == 0 \]
\[ \quad \text{prefetch } (C(i+k,j)) \]
\[ \text{if } (i \text{ and } i,7) == 1 \]
\[ \quad \text{prefetch } (D(i+k+1,j)) \]
\[ \text{ enddo } \]
\[ \text{ enddo } \]

Assumed CLS = 64 bytes and data size = 8 bytes

\( k \) is the prefetch distance value
Prefetch Strategy - Conditional

Example loop

L:
Load A(I)
Load B(I)
...
I = I + 1
Br L, if I<n

Conditional Prefetching

L:
Load A(I)
Load B(I)
Cmp pA=(I mod 8 == 0)
if(pA) prefetch
A(I+X)
Cmp pB=(I mod 8 == 1)
If(pB) prefetch
B(I+X)
...
I = I + 1
Br L, if I<n

Code for condition generation
Prefetches occupy issue slots
Prefetch Strategy - Unroll

Example loop

\[
\begin{align*}
\text{L:} & \\
\text{Load } A(I) & \\
\text{Load } B(I) & \\
\ldots & \\
I = I + 1 & \\
\text{Br L, if } I < n
\end{align*}
\]

Unrolled

\[
\begin{align*}
\text{Unr Loop:} & \\
prefetch A(I+X) \quad & \\
\text{load } A(I) & \\
\text{load } B(I) & \\
\ldots & \\
prefetch B(I+X) \quad & \\
\text{load } A(I+1) & \\
\text{load } B(I+1) & \\
\ldots & \\
prefetch C(I+X) \quad & \\
\text{load } A(I+2) & \\
\text{load } B(I+2) & \\
\ldots & \\
prefetch D(I+X) \quad & \\
\text{load } A(I+3) & \\
\text{load } B(I+3) & \\
\ldots & \\
prefetch E(I+X) \quad & \\
\text{load } A(I+4) & \\
\text{load } B(I+4) & \\
\ldots & \\
\text{load } A(I+5) & \\
\text{load } B(I+5) & \\
\ldots & \\
\text{load } A(I+6) & \\
\text{load } B(I+6) & \\
\ldots & \\
\text{load } A(I+7) & \\
\text{load } B(I+7) & \\
\ldots & \\
I = I + 8 & \\
\text{Br Unr Loop, if } I < n
\end{align*}
\]

↓Code bloat (>8X)

↓Remainder loop
Software Data Prefetching

Cost

- Requires memory instruction resources
  - A prefetch instruction for each access stream
- Issues every iteration, but needed less often
  - If branched around, inefficient execution results
  - If conditionally executed, more instruction overhead results
  - If loop is unrolled, code bloat results
Software Data Prefetching

Cost

- Redundant prefetches get in the way
  - Resources consumed until prefetches discarded!
- Non redundant need careful scheduling
  - Resources overwhelmed when many issued & miss
Desirable Characteristics

- Uses minimal instruction resources
  - One prefetch instruction for multiple streams
- Minimizes redundant prefetches
  - No code bloat, no prefetch branches
- Issues prefetches spaced in time
  - Machine resources utilized evenly
- Solution: rotating register prefetch if there is HW support.
Rotating Registers

- Register rotation provides an automatic renaming mechanism.
- Instructions contain a “virtual” register number.

| Iteration 1 | r32 r33 r34 r35 r36 r37 r38 r39 |
| Iteration 2 | r33 r34 r35 r36 r37 r38 r39 r32 |
| Iteration 3 | r34 r35 r36 r37 r38 r39 r32 r33 |
Rotating Reg Prefetch Illustrated

Example loop

```
Orig_loop:
  Load A(I)
  Load B(I)
  Load C(I)
  Load D(I)
  Load E(I)
  ... 
  I = I + 1
  Br Orig_loop, if I<n
```

Rotating Register Prefetching

```
r33 = address of E(1+X)
r34 = address of D(1+X)
r35 = address of C(1+X)
r36 = address of B(1+X)
r37 = address of A(1+X)
Method1Loop:
  prefetch [r37]
  r32 = r37 + INCR
  ... 
  load A(I)
  load B(I)
  load C(I)
  load D(I)
  load E(I)
  ... 
  I = I + 1
  Br Method1Loop, if I<n
```

↑Single prefetch inst
↑No loop unrolling
↑At most 1 miss/iter
Measurements – SPECfp2000

Performance Gain over No prefetching

-20 0 20 40 60 80 100 120 140 160

wupwise swim mgrid applu mesa galgel art equake facerec ammp lucas Fma3d sixtrack apsi Geomea

#REF! #REF! Data Prefetch
Agenda

- Data Prefetch
- *Software Pipelining*
Software Pipelining

- Obtain parallelism by executing iterations of a loop in an overlapping way.
- We’ll focus on simplest case: the do-all loop, where iterations are independent.
- **Goal**: Initiate iterations as frequently as possible.
- **Limitation**: Use same schedule and delay for each iteration.
Machine Model

- Timing parameters: LD = 2, others = 1 clock.
- Machine can execute one LD or ST and one arithmetic operation (including branch) at any one clock.
  - I.e., we’re back to one ALU resource and one MEM resource.
Example

\begin{verbatim}
for (i=0; i<N; i++)
    B[i] = A[i];

\textbf{r9} holds 4N; \textbf{r8} holds 4*i.

\textbf{L:} \texttt{LD r1, a(r8)}
\begin{verbatim}
nop
\texttt{ST b(r8), r1}
\texttt{ADD r8, r8, #4}
\texttt{BLT r8, r9, L}
\end{verbatim}
\end{verbatim}

Notice: data dependences force this schedule. No parallelism is possible.
Let’s Run 2 Iterations in Parallel

- Focus on operations; worry about registers later.

```
LD
nop  LD
ST  nop
ADD  ST
BLT  ADD
BLT
```

Oops --- violates ALU resource constraint.
Introduce a NOP

LD
nop
ST
ADD
nop
BLT

LD
nop
ST
ADD
nop
BLT

Add a third iteration.
Several resource conflicts arise.
Is It Possible to Have an Iteration Start at Every Clock?

- **Hint:** No.
- **Why?**
- An iteration injects 2 MEM and 2 ALU resource requirements.
  - If injected every clock, the machine cannot possibly satisfy all requests.
- **Minimum delay = 2.**
A Schedule With Delay

Initialization

Identical iterations of the loop

Coda
Assigning Registers

- We don’t need an infinite number of registers.
- We can reuse registers for iterations that do not overlap in time.
- But we can’t just use the same old registers for every iteration.
Assigning Registers --- (2)

- The inner loop may have to involve more than one copy of the smallest repeating pattern.
  - Enough so that registers may be reused at each iteration of the expanded inner loop.
- Our example: 3 iterations coexist, so we need 3 sets of registers and 3 copies of the pattern.
Example: Assigning Registers

- Our original loop used registers:
  - r9 to hold a constant 4N.
  - r8 to count iterations and index the arrays.
  - r1 to copy a[i] into b[i].

- The expanded loop needs:
  - r9 holds 12N.
  - r6, r7, r8 to count iterations and index.
  - r1, r2, r3 to copy certain array elements.
### The Loop Body

To break the loop early

<table>
<thead>
<tr>
<th>Iteration $i$</th>
<th>Iteration $i + 1$</th>
<th>Iteration $i + 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>L:</td>
<td>L’</td>
<td>L”</td>
</tr>
<tr>
<td>ADD r8, r8, #12</td>
<td>nop</td>
<td>LD r3, a(r6)</td>
</tr>
<tr>
<td>BGE r8, r9, L’</td>
<td>ST b(r7), r2</td>
<td>nop</td>
</tr>
<tr>
<td>LD r1, a(r8)</td>
<td>ADD r7, r7, #12</td>
<td>LD r3, a(r6)</td>
</tr>
<tr>
<td>nop</td>
<td>BGE r7, r9, L”</td>
<td>nop</td>
</tr>
<tr>
<td>nop</td>
<td>LD r2, a(r7)</td>
<td>nop</td>
</tr>
<tr>
<td>ST b(r8), r1</td>
<td>nop</td>
<td>BLT r6, r9, L</td>
</tr>
</tbody>
</table>

> Each register handles every third element of the arrays.

$L’$ and $L”$ are places for appropriate codas.

---

Stanford University

CS243 Winter 2006
Cyclic Data-Dependence Graphs

- We assumed that data at an iteration depends only on data computed at the same iteration.
  - Not even true for our example.
    - r8 computed from its previous iteration.
    - But it doesn’t matter in this example.
- **Fixup**: edge labels have two components: (iteration change, delay).
Example: Cyclic D-D Graph

(A) LD r1,a(r8)

(B) ST b(r8),r1

(C) ADD r8,r8,#4

(D) BLT r8,r9,L

(A) must wait at least one clock after the (C) from the previous iteration.

(C) must wait at least one clock after the (B) from the same iteration.
Matrix of Delays

- Let $T$ be the delay between the start times of one iteration and the next.
- Replace edge label $<i,j>$ by delay $j-iT$.
- Compute, for each pair of nodes $n$ and $m$ the total delay along the longest acyclic path from $n$ to $m$.
- Gives upper and lower bounds relating the times to schedule $n$ and $m$. 

**Example: Delay Matrix**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1-T</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Edges

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1-T</td>
<td></td>
<td>3-T</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Acyclic Transitive Closure

\[
S(B) \geq S(A) + 2
\]

\[
S(A) \geq S(B) + 2 - T
\]

\[
S(B) - 2 \geq S(A) \geq S(B) + 2 - T
\]

**Note:** Implies \( T \geq 4 \) (because only one register used for loop-counting). If \( T=4 \), then A (LD) must be 2 clocks before B (ST). If \( T=5 \), A can be 2-3 clocks before B.

Stanford University
Iterative Modulo Scheduling

- Compute the lower bounds ($M_{II}$) on the delay between the start times of one iteration and the next (*initiation interval, aka II*)
  - due to resources
  - due to recurrences
- Try to find a schedule for $II = M_{II}$
- If no schedule can be found, try a larger $II$. 
Summary

References for compiler data prefetch:

- Todd Mowry, Monica Lam, Anoop Gupta, “Design and evaluation of a compiler algorithm for prefetching”, in ASPLOS’92,
  http://citeseer.ist.psu.edu/mowry92design.html.

- Gautam Doshi, Rakesh Krishnaiyer, Kalyan Muthukumar, “Optimizing Software Data Prefetches with Rotating Registers”, in PACT’01,
  http://citeseer.ist.psu.edu/670603.html.