PREFETCHING

Reading: Data prefetch mechanisms, Steven P. Vanderwiel, David J. Lilja, ACM Computing Surveys, Vol. 32, Issue 2 (June 2000)
• If memory takes a long time, start accessing earlier

Load  L1  L2  DRAM
Data

Prefetch

Load  Data

Total Load-to-Use Latency

May cause resource contention due to extra cache/DRAM activity

Much improved Load-to-Use Latency

Somewhat improved Latency
Prefetching

• Three basic questions
  – What to prefetch?
  – When to prefetch?
  – Where to put?

• Two approaches
  – Software prefetching
    • Compiler or programmer decides
  – Hardware prefetching
    • Hardware decides
Software Prefetching

Reordering can mess up your code

 Hopefully the load miss is serviced by the time we get to the consumer

Using a prefetch instruction (or load to $zero) can help to avoid problems with data dependencies
Software Prefetching

- Two flavors: register prefetch and cache prefetch
- Each flavor can be faulting or non-faulting
  - If address bad, does it create exceptions?
- Faulting register prefetch is binding
  - It is a normal load, address must be OK, uses register
- Not faulting cache prefetch is non-binding
  - If address bad, becomes a NOP
  - Does not affect register state
  - Has more overhead (load still there), ISA change (prefetch instruction), complicates cache (prefetches and loads different)
Prefetching

- Predict future misses and get data into cache
  - If access does happen, we have a hit now
    (or a partial miss, if data is on the way)
  - If access does not happen, *cache pollution*
    (replaced other data with junk we don’t need)
- To avoid pollution, prefetch buffers
  - Pollution a big problem for small caches
  - Have a small separate buffer for prefetches
  - How big?
- Use 2\textsuperscript{nd} level cache as a prefetch buffer.
Review & Outline

• Review:
  – DRAM scheduling
  – Prefetch

• Outline
  – How to insert prefetch requests
  – Software prefetch mechanisms
  – Hardware prefetching algorithms
• **Intrinsics**
  – Programmers can insert “assembly like instructions” in a high level source code.
  – One intrinsic is usually translated into one assembly code

Data prefetch, VanderWiel & Lilja
for (i = 0 ; i < N; i++)
    ip = ip+a[i]*b[i];

No prefetching

for (i = 0 ; i < N; i++)
{
    fetch (&a[i+1]);
    fetch (&b[i+1]);
    ip = ip+a[i]*b[i];
}

Limitations?

Cons: multiple requests for the same cache block

Simple prefetching

No prefetching for a[0], b[0]

Data prefetch, VanderWiel & Lilja
for (i = 0; i < N; i += 4) {
    fetch (&a[i+4]);
    fetch (&b[i+4]);
    ip = ip + a[i] * b[i];
    ip = ip + a[i+1] * b[i+1];
    ip = ip + a[i+2] * b[i+2];
    ip = ip + a[i+3] * b[i+3];
}

• Benefit: one request for each cache block
  – Still missing a[0], b[0]
fetch (&a[0]);
fetch (&b[0]);
for( i =0 ; i < N-4; i+=4) {
    fetch (&a[i+4]);
    fetch (&b[i+4]);
ip = ip+a[i]*b[i];
ip = ip+a[i+1]*b[i+1];
ip = ip+a[i+2]*b[i+2];
ip = ip+a[i+3]*b[i+3];
}
for (; i <n; i++)
ip= ip+a[i]*b[i]
Prefetch Distance

• How early prefetch?
  – One iteration is enough?
  – Memory latency and amount of computation between memory accesses
  – Prefetch distance ($\delta$)

$$\delta = \left\lceil \frac{l}{s} \right\rceil$$

– $l$: the average memory latency (measured in cycle)
– $s$: the estimated cycle time of the shortest possible execution path through one loop iteration

Data prefetch, VanderWiel & Lilja
Example: Prefetch Distance

If L/C = 4
- We must prefetch 4 elements ahead

Problem:
Why? And so?

```c
for( i =0 ; i < N-4; i+=4) {
    fetch (&a[i+16]);
    fetch (&b[i+16]);
    ip = ip+a[i]*b[i];
    ip = ip+a[i+1]*b[i+1];
    ip = ip+a[i+2]*b[i+2];
    ip = ip+a[i+3]*b[i+3];
}
```
for (i=1; i<N; i++) {
    ListNode *p = listHead[i];
    while(p){
        work(p->data);
        p=p->next;
    }
}

If L=3W?
for (i=1; i<N; i++) {
    ListNode *p = listHead[i];
    while(p){
        prefetch ((p->next->next->next->next)
        work(p->data);
        p=p->next;
    }
}
Prefetch Metrics

• # of useful prefetch: # of prefetched block that will be used by demand loads
• **Accuracy** = # of useful prefetch/total # of prefetch
• **Coverage** = # of useful prefetch/total # of cache misses
• **Timeliness**: How timely prefetch cache blocks
Limitations of Software Prefetching

• Compiler or programmer need to insert
  – Usually limit to loops
• Prefetch instruction fetch/execution overhead
• Code expansion
• Static decision: Cache miss behavior needs to be predicted at static time
  – Cache sizes vary machine by machine
  – Today’s processors; cores share caches.

Data prefetch, VanderWiel & Lilja
Please insert prefetch requests

for (i = 0; i < reg->size; i++) {
    if (reg->node[i].state & cond…)
        ....
}

Lee et al. When prefetching works, When it doesn’t and why
// pbeampp.c (mcf)
for (i=2, next=0; i<=B && i<=basket_size; i++) {
    arc = perm[i]->a;
    red_cost = arc->cost - arc->tail->potential + arc->head->potential; ... 
}

Lee et al. When prefetching works, When it doesn’t and why
for (ii = 0; ii < 8196; ii++) {
    PREFETCH(a[ii+D]);
    for (jj = 0; jj < 128; jj++){
        PREFETCH(b[ii+D]);
        if (cond1)
            c[ii] = a[ii]+b[jj];
    }
}

Definition of useful: A block is requested by demand later (anytime)

Accuracy of A:

Accuracy of B:

Cache size: 8KB
D = 100;
each instruction takes 1 cycle (when cache hit)
Data type: 2B, cache block size 8B
Memory latency: fixed 100.
The probability of satisfying cond1 = 65%,
Prefetch accuracy and coverage
Write-allocation policy
for (ii = 0; ii < 8196; ii++) {
    PREFETCH(a[ii+D]);
    for (jj = 0; jj < 128; jj++){
        PREFETCH(b[ii+D]);
        if (cond1)
            c[ii] = a[ii]+b[jj];
    }
}

- How many useless prefetch requests for A.

- The cost of redundant requests are not so high. Why?

- But B?

- 8KB/2B: 4*1024 elements: 1024*4/3 ~ 1.5K → B can fit in the cache
Hardware Prefetching

Depending on prefetch algorithm/miss patterns, prefetcher injects additional memory requests.

Cannot be overly aggressive since prefetches may contend for memory bandwidth, and may pollute the cache (evict other useful cache lines).

Hardware monitors miss traffic to DRAM.
Hardware prefetch schemes

- Hardware prefetch address =
  = \text{func}(\text{demand memory addresses, pc, memory value, old memory address histories, etc..})

Different prefetchers have different algorithms
- looking at only demand memory addresses? : stream, stride
- looking at PC addresses or not
- looking at memory values? Content based prefetching
- old memory address histories? Markov prefetching
Stream/Stride prefetcher

• Miss address streams
  – 1, 2, 3, 4 …
  – Prefetch 5, 6, 7
  – Stream prefetch

• Miss address streams
  – 1, 4, 7, 10,…
  – Prefetch 13, 16, 19,…
  – Stride prefetch
Instruction Prefetching

• Instructions are sequential.
• Easy to predict.
• First implemented
• Next line prefetcher (one block ahead prefetcher)
  – Very simple, if a request for cache line X goes to DRAM, also request X+1
    • FPM DRAM already will have the correct page open for the request for X, so X+1 will likely be available in the row buffer
    • Can optimize by doing Next-Line-Unless-Crossing-A-Page-Boundary prefetching

Data prefetch, VanderWiel & Lilja
Stream Buffer

- Jouppi ‘90
- $K$ prefetched blocks $\rightarrow$ FIFO stream buffer
- As each buffer entry is referenced
  - Move it to cache
  - Prefetch a new block to stream buffer
  - Prefetcher buffer hit! $\rightarrow$ prefetch the next block
- Avoid cache pollution

Data prefetch, VanderWiel & Lilja
• Degree of prefetching
  – For one cache miss, how many do we prefetch?
  – E.g.) addr 0x01: \( \rightarrow \) 0x03, 0x04, 0x05, 0x06

• Prefetch distance
  – How far do we prefetch?
Stream Prefetcher in multi-level cache hierarchy

Different prefetch degree for different memory hierarchy
Initial distance to hide memory latency
**Stride Prefetcher**

Source code level
for ( ii = 0; ii < N; ii++)
Sum += b[ii];

Memory addresses
0x100 0x104 0x108 0x10b ....

Per PC information
Chen-Baer ‘91
Is good to use PC?

- PC information can easily differentiate different address streams
- How soon can we know PC addresses?
Markov Prefetching

- Markov prefetching forms address correlations
  - Joseph and Grunwald (ISCA ‘97)
- Uses global memory addresses as states in the Markov graph
- Correlation Table approximates Markov graph

Miss Address Stream
A B C A B C B C . . .

Markov Graph

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st predict.</td>
<td>B</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>2nd predict.</td>
<td>C</td>
<td>B</td>
<td>A</td>
</tr>
</tbody>
</table>

Slides from Nesbit’s HPCA04 talk
Markov Prefetching

• History based prefetching
  – \(\Rightarrow\) required space for storing history
  – How much space?
  – Is it still good with a large L2 cache?

• What kind of data structures are good for this type?
  – Pointer, link list
Global History Buffer (GHB)

- Unified frame for different prefetching scheme
- Holds miss address history in FIFO order
- Linked lists within GHB connect related addresses
  - Same static load
  - Same global miss address
  - Same global delta
- Linked list walk is short compared with L2 miss latency
- Nesbit and smith ‘04

Slides from Nesbit’s HPCA04 talk
Contented-Directed Prefetching

- Pointer prefetching scheme
- Look at data of memory
- Search for data which might be memory addresses
  – Cooksey et al. ‘02
Content-Directed Prefetching (CDP)

Virtual Address Predictor

Generate Prefetch

Slide from Ebrahimi's HPCA09 talk
Speculative execution for prefetching

- High accuracy and good coverage
- No architectural changes
- Multi-processors
  - SMT (later lecture)

Collins et al. MICRO 2001

for (i=1; i<N; i++) {
    ListNode *p = listHead[i];
    while(p){
        work(p->data);
        p=p->next;
    }
}

original code

for (i=1; i<N; i++) {
    ListNode *p = listHead[i];
    while(p){
        p=p->next;
    }
}

speculative execution code
Review

- **S/W prefetching**
  - Explicit prefetching requests
  - Prefetch distance, avoid requesting the same cache block (loop unrolling)

- **H/W prefetching**
  - Observe cache miss address streams (stream, stride, markov, GHB)
  - Observe data in the load (content-directed prefetching)
  - Pre-execution
Runahead Execution

- A technique to obtain the memory-level parallelism benefits of a large instruction window

- When the oldest instruction is an L2 miss:
  - Checkpoint architectural state and enter runahead mode

- In runahead mode:
  - Instructions are speculatively pre-executed
  - The purpose of pre-execution is to generate prefetches
  - L2-miss dependent instructions are marked INV and dropped

- Runahead mode ends when the original L2 miss returns
  - Checkpoint is restored and normal execution resumes
Runahead Example

Small Window:
- Load 1 Miss
- Load 2 Miss
- Compute
- Stall
- Compute
- Stall

Runahead:
- Load 1 Miss
- Load 2 Miss
- Load 1 Hit
- Load 2 Hit
- Compute
- Runahead
- Compute

Saved Cycles

Runahead Execution Mutlu etal. HPCA 03
Overhead of Prefetching

- Overhead of software prefetching
  - Extra instructions
  - Cache pollution
  - Bandwidth consumption

- Overhead of hardware prefetching
  - Transistors (can we use that space for cache?)
  - Cache pollution
  - Bandwidth consumption

Data prefetch, VanderWiel & Lilja