Lecture 25: Advanced Data Prefetching Techniques

Prefetching and data prefetching overview, Stride prefetching, Markov prefetching, precomputation-based prefetching

Zhao Zhang, CPRE 585 Fall 2003

Where Are Solutions?

1. Reducing miss rates
   - Larger block size
   - Larger cache size
   - Higher associativity
   - Victim caches
   - Way prediction and Pseudoassociativity
   - Compiler optimization

2. Reducing miss penalty
   - Multilevel caches
   - Critical word first
   - Read miss first
   - Merging write buffers

3. Reducing miss penalty or miss rates via parallelism
   - Non-blocking caches
   - Hardware prefetching
   - Compiler prefetching

4. Reducing cache hit time
   - Small and simple caches
   - Avoiding address translation
   - Pipelined cache access
   - Trace caches

Prefetching Evaluation

Prefetch: Predict future accesses and fetch data before they are demanded

Accuracy: How many prefetched items are really needed?
   - False prefetching: fetched wrong data
   - Cache pollution: replace “good” data with “bad” data

Coverage: How many cache misses are removed?

Timeliness: Does the data return before they are demanded?

Other considerations: complexity and cost

Prefetching Targets

Instruction prefetching
   - Stream buffer is very useful

Data prefetching
   - More complicated because of the diversities in data access pattern
   - Prefetching for dynamic data (hashing, heap, sparse array, etc.)
     - Usually with irregular access patterns
   - Linked-list prefetching (Pointer chasing)
     - A special type of data prefetching for data in linked-list

Memory Wall

Consider memory latency of 1000 processor cycles or a few thousands of instructions ...

Where Are Solutions?

1. Reducing miss rates
   - Larger block size
   - Larger cache size
   - Higher associativity
   - Victim caches
   - Way prediction and Pseudoassociativity
   - Compiler optimization

2. Reducing miss penalty
   - Multilevel caches
   - Critical word first
   - Read miss first
   - Merging write buffers

3. Reducing miss penalty or miss rates via parallelism
   - Non-blocking caches
   - Hardware prefetching
   - Compiler prefetching

4. Reducing cache hit time
   - Small and simple caches
   - Avoiding address translation
   - Pipelined cache access
   - Trace caches

Increase cache size?

More levels of memory hierarchy?
   - Itanium: 2-4MB L3 cache
   - IBM Power4: 32MB eDRAM cache

Large caches are still very useful but may not help fully address the issue.
Prefetching Implementations

- Sequential and stride prefetching
  - Tagged prefetching
  - Simple stream buffer
  - Stride prefetching
- Correlation-based prefetching
  - Markov prefetching
  - Dead-block correlating prefetching
- Precomputation-based
  - Keep running programs on cache misses; or
  - Use separate hardware for prefetching; or
  - Use compiler-generated threads on multi-threaded processors
- Other considerations
  - Predict on miss addresses or reference address?
  - Prefetch into cache or a temp. buffer?
  - Demand-based or decoupled prefetching?

Recall Stream Buffer Diagram

- Data tags
- Direct mapped cache
- Stream buffer

Stride Prefetching

Limits of streaming buffer:
- Program may access data in either direction; i.e. how about
  - for (i=N-1; i >= 0; i --)
- Data may be accessed in strides, i.e.
  - for (i = 0; i < N; i ++)
  - for (j = 0; j < N; j ++)
  - sum[i] += X[i][j];

Stride Prefetching Example

```plaintext
float a[100][100], b[100][100], c[100][100];
...
for (i = 0; i < 100; i++)
for (j = 0; j < 100; j++)
for (k = 0; k < 100; k++)
a[i][j] += b[i][k] * c[k][j];
```

### Markov Prefetching

Target irregular mem access pattern

```
Miss addresses:
A B C D E A C F E A A B C D E A B C D C
```

Markov model

- Misses
- Predicted addresses

Joseph and Grunwald, ISCA 1997
Markov Prefetching Performance

From left to right: number of addresses in table

Markov Prefetching Performance

From left to right: stream, stride, correlation (Pomerene and Puzak), Markov, stream+stride+Markov serial, stream+stride+Markov parallel

Predictor-directed Stream Buffer

Cons of existing approaches:
- Stride prefetching (using updated stream buffer): Only useful for strid access; being interfered by non-stride accesses
- Markov prefetching: Working for general access patterns but requiring large history storage (megabytes)
- PSB: Combining the two methods
  - To improve coverage of stream buffer; and
  - Keep the required storage low (several kilobytes)
  - Sair et al., MICRO 2000

Precomputation-based Prefetching

Potential problems of stream buffer or Markov prefetching:
- Low accuracy => high memory bandwidth waste
- Another approach: use some computation resource for prefetching, because computation is increasingly cheaper
- Speculative execution for prefetching
  - No architectural changes
  - Not limited by hardware
  - With high accuracy and good coverage
  - for (i=0; i<10; i++)
  - for (j=0; j<100; j++)
  - if (a[i][j]!=b[i][j])
  - data[i][j]++;
  - Loop:
  - I1 load r1=[r2]
  - I2 add r3=r3+1
  - I3 add r6=r3-100
  - I4 add r7=r7+8
  - I5 add r8=r8+4
  - I6 load r9=[r1]
  - I7 add r5=r5+1
  - I8 store [r1]=r5
  - I9 blt r6, lop
  - Collins et al, MICRO 2001

Prefetching by Dynamically Building Data-dependence Graph

Annavaram et al., “Data prefetching by dependence graph precomputation”, ISCA 2001
- Needs external help to identify problematic loads
- Builds dependence graph in reverse order
- Uses separate prefetching engine

![Diagram of prefetching by dynamically building data-dependence graph]

Updated inst fetch queue

DG generator

DG Buffer

EXE Engine

prefetching
Using "Future" Threads for Prefetching
Balasubramonian et al. “Dynamically allocating processor resources between nearby and distant ILP,” ISCA 2001
- OOO processors stall on cache misses to DRAM because of exhausting some resources (IQ or ROB or registers)
- Why not keep the program run during the stall time for prefetching?
- Then, must reserve resources for "future" thread
- Future thread continues the execution for prefetching
- Using the existing OOO pipeline and FUs for execution
- May release registers or ROB speculatively, thus can examine a much larger instruction window
- Still accurate in producing reference addresses

Precomputation with SMT Supporting Speculative Threads
- Precomputation is done by an explicit speculative thread (p-thread)
- The code of p-threads may be constructed by compiler or hardware
- Main thread execution spawns p-threads on triggers (e.g. when an PC is encountered)
  - Main thread some register values and initial PC for p-thread
  - P-thread may trigger another p-thread for further prefetching
For more compiler issues, see Luk, “Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors”, ISCA 2001

Summary of Advanced Prefetching
- Being actively studied because of the increasing CPU-memory speed gap
- Improving cache performance beyond the limit of cache size
- Precomputation may be limited in prefetching distance (how good is the timeliness?)
- Note there is no perfect cache/prefetching solution, e.g.
  ```
  while (1) {
    myload (addr);
    addr = myrandom() + addr;
  }
  ```
- How to design complexity-effective memory systems for future processors?