Hare: An Effective Programmable Prefetch Engine for High-Performance Processors

Tien-Fu Chen

Department of Computer Science
National Chung Cheng University
Chiayi, Taiwan, ROC

May 1995

Reducing Latency by On-Chip Support

1. Multi-level on-chip caches
2. Compiler optimization
3. Overlapping between data accesses / with computations
   - Buffering and Pipelining
   - Multithreading
   - Prefetching
     - Software-Directed
     - Hardware-Controlled

Outline

1. Background
2. Hare: A Programmable Prefetch Engine
3. Compile-Time Management
4. Performance Evaluation
5. Conclusion

Prefetching: predict data use

- Software-directed
  - Explicit prefetch instructions
  - Static program analysis
  - Extra prefetch instruction overhead

1. [Porterfield 89]
2. [Gornish et al. 90]
3. [Klaiber and Levy 91]
4. [Mowry and Gupta 92]

- Hardware-controlled
  - Exploit dynamic information
  - Extra hardware complexity

1. Stream buffer [Jouppi 90]
2. Stride Prefetch [Fu and Patel 92]
3. Decode lookahead [Lee et al. 87]
4. Reference prediction [Baer and Chen 91]

An Exemplary Software Prefetching

- Focus on affine array accesses
- Procedures:
  1. Identify misses by locality analysis
     Exploiting *temporal*, *spatial*, *group* reuse
  2. Introduce prefetch predicates
     e.g., $(i \;\; mod \;\; l) == 0$
  3. Reduce overhead by loop splitting and loop unrolling
  4. Schedule prefetches using software pipelining

$$\# \text{ of iterations ahead} = \left\lceil \frac{\delta}{s} \right\rceil$$

- Further improved by locality optimization

Example of instrumented loop

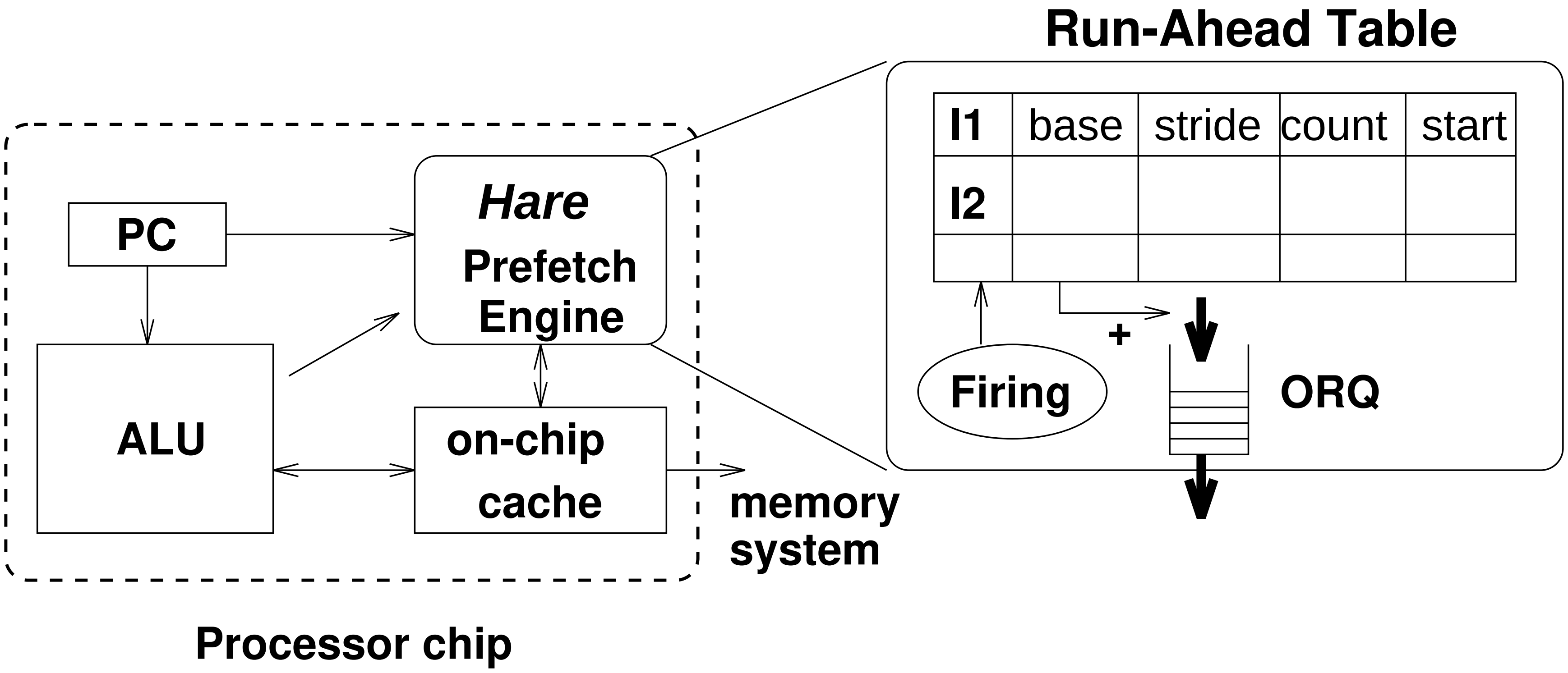
(a) Original code

```
for j = 0 to 100
    for i = 0 to 100
        A[j][i] = B[i][0] + B[i+1][0]
end
```

(b) Instrumented code (inner loop only)

```
prefetch(&A[j][0])
for i = 0 to 5 by 2                      } prologue
    prefetch(&B[i+1][0])
    prefetch(&B[i+2][0])
    prefetch(&A[j][i+1])
end
for i = 0 to 93 by 2                     } main loop
    prefetch(&B[i+7][0])
    prefetch(&B[i+8][0])
    prefetch(&A[j][i+7])
    A[j][i] = B[i][0] + B[i+1][0]
    A[j][i+1] = B[i+1][0] + B[i+2][0]
end
for i = 94 to 100 by 2                   } epilogue
    A[j][i] = B[i][0] + B[i+1][0]
    A[j][i+1] = B[i+1][0] + B[i+2][0]
end
```

Hare: A Programmable Prefetch Engine



```
int X[100],Y[100][100],Z;

for (i=0; i<100; i++)
   X[i] = X[i] + Y[i+1][0];
```

```
# 5   for (i=0; i<100; i++)
    move $4, $0          # init index i
    la   $15, X          # start of X
    la   $16, Y+400
$32:
# 6   X[i] = X[i] + Y[i+1][0];
    lw   $25, 0($15) # X[i]
    lw   $26, 0($16) # Y[i+1][0]
    add  $4, $4, 1
    addu $10,$25,$26# sum up
    sw   $10, 0($15) # store X[i]
    addu $15, $15, 4
    addu $16, $16, 400
    blt  $4, 100, $32
```

```
# 5   for (i=0; i<100; i++)
    move      $4, $0
    la        $15, X
    la        $16, Y+400
    fill_run_ahead I1, X+16, 4, 4, 1
    fill_run_ahead I2, Y+400, 400, 1
$32:
# 6   X[i] = X[i] + Y[i+1][0];
I1: lw        $25, 0($15)
I2: lw        $26, 0($16)
    add       $4, $4, 1
    addu      $10, $25, $26
    sw        $10, 0($15)
    addu      $15, $15, 4
    addu      $16, $16, 400
    blt       $4, 100, $32
```

Compile-Time Management

1. Using **run-ahead** instruction

   `run_ahead <base,stride>,<count,start>`

2. Compiling prefetch predicates

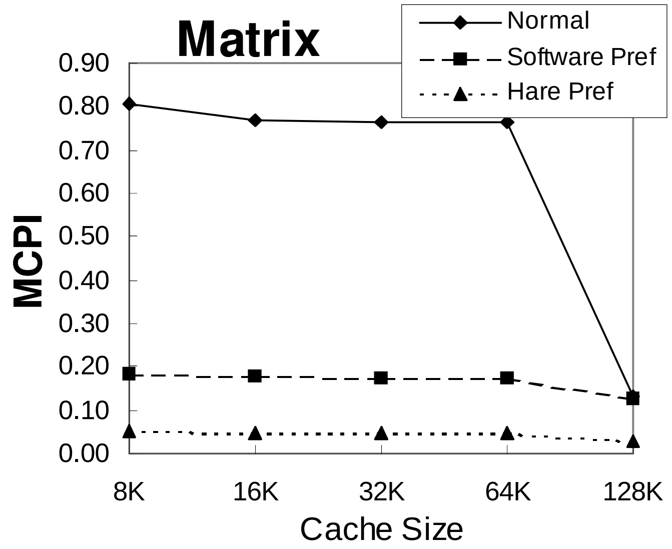
type of locality	miss occurs on iteration	prefetch predicate	`<count, start>`
none	every	True	`<1,1>`
temporal	first	i = 0	`<0,1>`
spatial	every l	$(i \; mod \; l) = s$	*<l,s>*
group	never	False	`<0,0>`

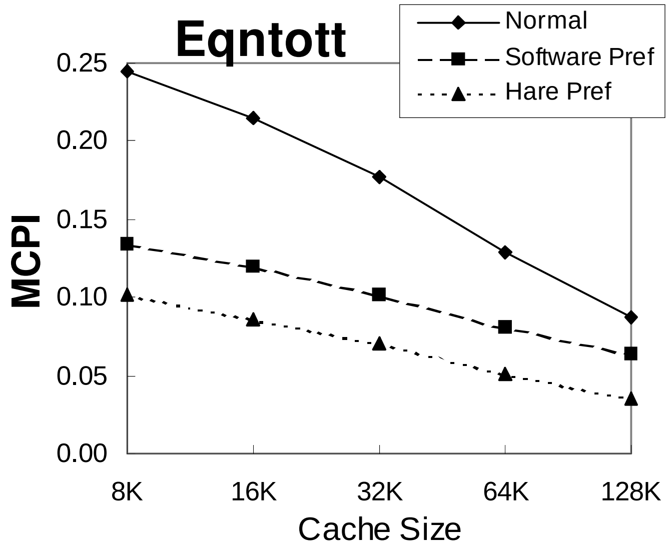
Evaluation Methodology

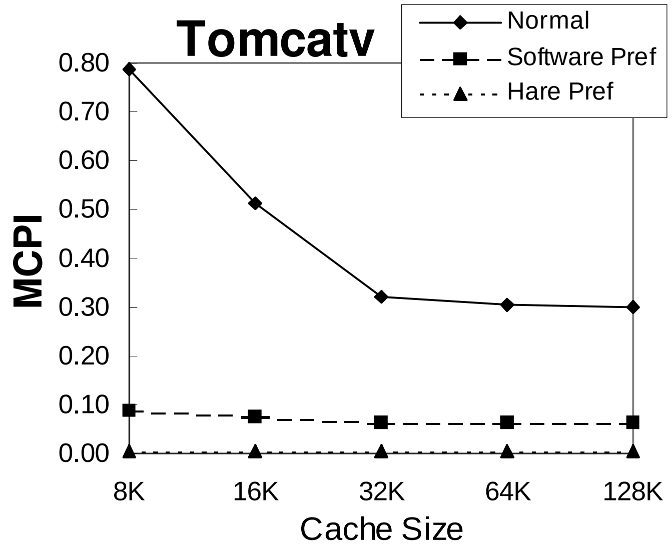
- Cycle-by-cycle trace driven simulation
- Direct-mapped cache with block size of 32 bytes
- Varying cache size from 8K to 128K
- Varying memory latency from 10 to 30
- Using SPEC benchmarks
- Emulate software prefetching by optimistic insertion
- Performance metric: MCPI

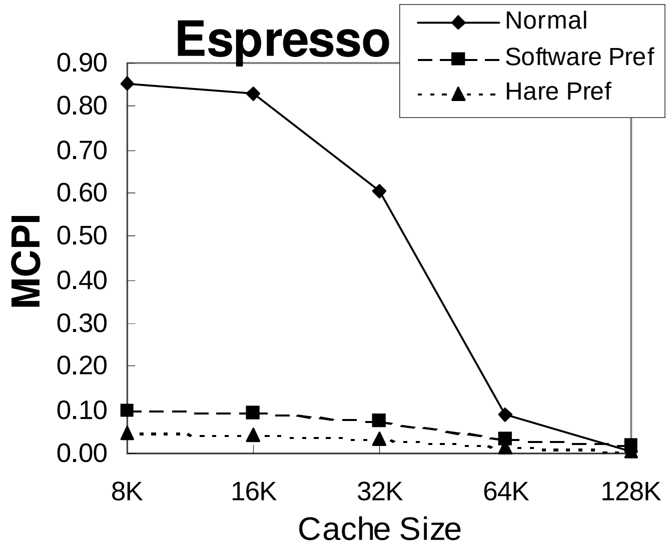
$$MCPI_{data\ access} = \frac{total\ data\ access\ penalty}{number\ of\ instructions\ executed}$$

Simulation Results - varying cache size

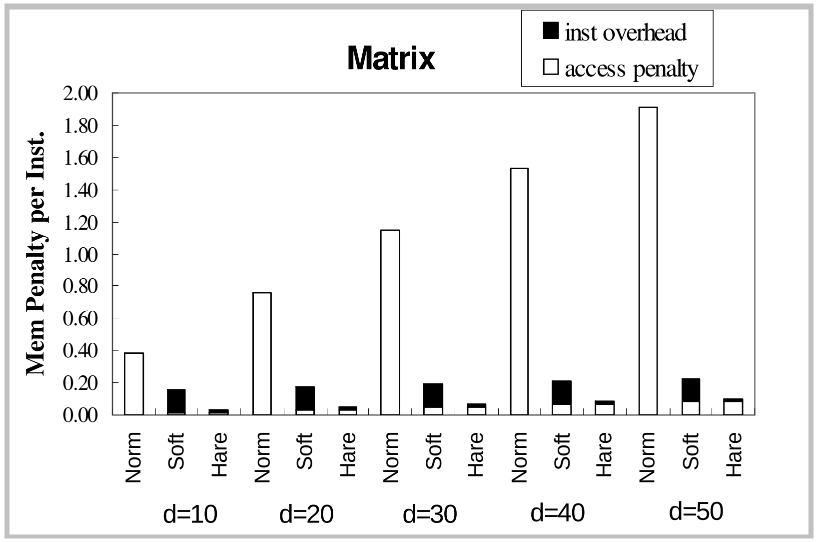


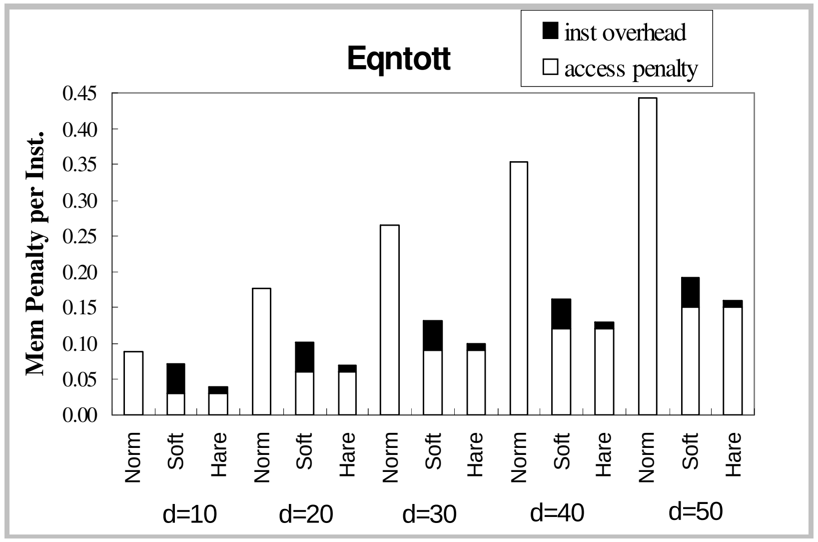


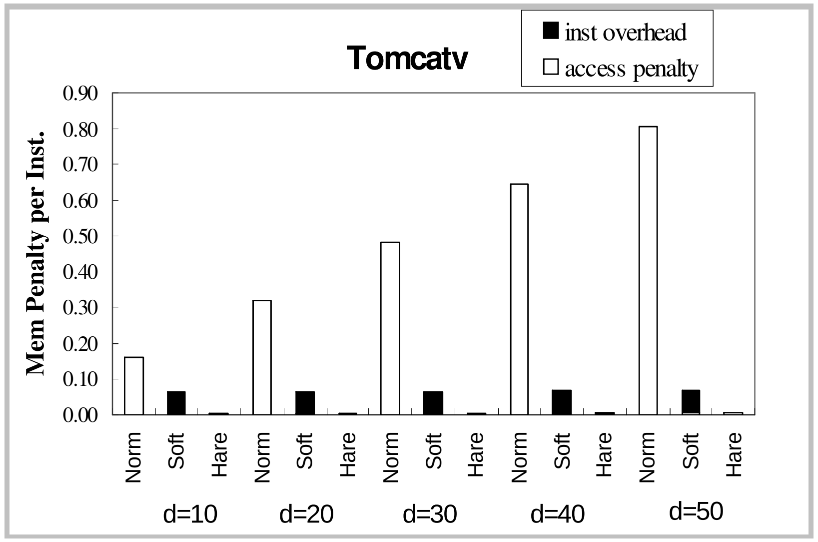


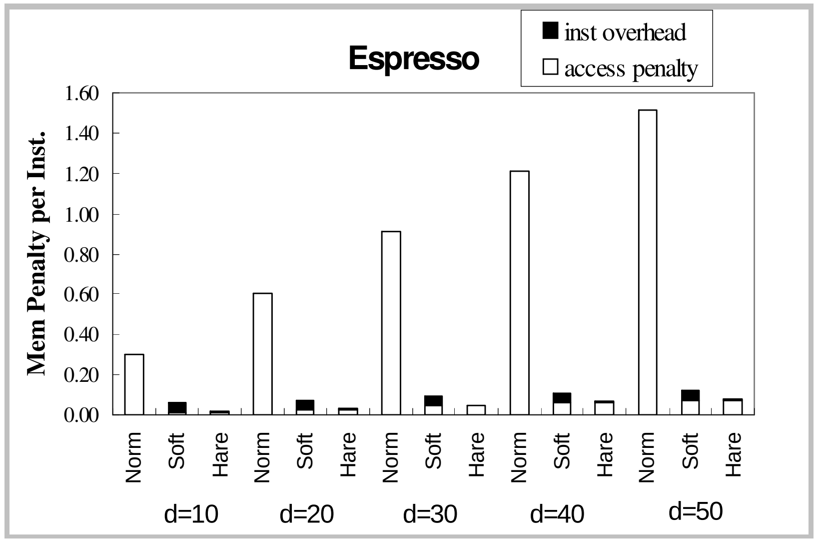


Simulation Results - varying latency









Conclusion

1. Propose an on-chip latency tolerating scheme for high-performance processors
2. A programmable prefetch engine controlled by user code
3. Take advantages of compiler support at run-time
4. Reduce significant access time by prefetching
5. Incur less instruction overhead than software prefetching
6. Future work incorporating branch prediction and speculative load in the programmable engine