ECE 6373 Advanced Computer Architecture

Lec 14 – Course Project & Advanced Cache Optimizations

Instructor: Dr. Xin Fu

ECE Department
University of Houston
xfu8@central.uh.edu

Course Project

- **In this course project, you will use SimpleScalar to evaluate the impact of a set of cache configurations on performance.**
- **To be done individually.**
- **Project report will due 11:59PM 12/9/2022.**
- **Details are released on the blackboard.**

SimpleScalar

- **How to install SimpleScalar**
  - Download the simplesim-3.0.tar.gz from the link (provided in course website)
  - Unpack the file
    - » tar -zxvf simplesim-3v0d.tgz
  - Compile
    - » Make sim-outorder
  - Copy the bin and the default.cfg (available in config/) to the benchmark directory
- **How to run Simplescalar**
  - ./sim-outorder -config default.cfg -fastfwd 1000000 -max:inst 1000000 (copy the command from benchmark.arg)
- **Understand the configuration file to configure your own machine**

Default.cfg in SimpleScalar

- **Cache configuration defined in sim-outorder.c**

```
 opt_reg_note(odb,
"  The cache config parameter <config> has the following format:\n"
"\n"
"    <name>:<nsets>:<bsize>:<assoc>:<repl>\n"
"\n"
"    <name>   - name of the cache being defined\n"
"    <nsets>  - number of sets in the cache\n"
"    <bsize>  - block size of the cache\n"
"    <assoc>  - associativity of the cache\n"
"    <repl>   - block replacement strategy, 'l'-LRU, 'f'-FIFO, 'r'-random\n"
"\n"
"    Examples:   -cache:dl1 dl1:4096:32:1:l\n"
"                -dtlb dtlb:128:4096:32:r\n"
           );
```

Default.cfg in SimpleScalar

- **You can also find the configuration setup for other structures in sim-outorder.c**

```
opt_reg_int(odb, "-issue:width",
        "instruction issue B/W (insts/cycle)",
        &ruu_issue_width, /* default */4,
        /* print */TRUE, /* format */NULL);
```

8. Reducing Misses by Compiler Optimizations

- **The optimized software can greatly reduce the miss rates**
- **Instructions**
  - **Reorder procedures in memory so as to reduce conflict misses, without affecting correctness**
  - **Profiling to look at conflicts(using tools they developed)**
  - **Entry point of basic block at the beginning of a block**
  - **Branch straightening**
- **Data**
  - ***Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays**
  - ***Loop Interchange*: change nesting of loops to access data in order stored in memory**
  - ***Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap**
  - ***Blocking*: Improve temporal locality by accessing "blocks" of data repeatedly vs. going down whole columns or rows**

Merging Arrays Example

- `/* Before: 2 sequential arrays */`
- `int val[SIZE];`
- `int key[SIZE];`

- `/* After: 1 array of stuctures */`
- `struct merge {`
- `    int val;`
- `    int key;`
- `};`
- `struct merge merged_array[SIZE];`

- **Reducing conflicts between val & key (they might conflict for a single block in the cache); improve spatial locality when they are accessed in a interleaved fashion**

Array Storage

- **Colum order: store first colum, then second colum, then third….**
- **Example:**
  - **6x6 array A**
  - **Each element 1 word**
  - **Block size: 4 word**
  - **For i=1 to 6; X+= a[1,i]; how many blocks are loaded into cache**

A(1,1)	A(2,1)	A(3,1)	A(4,1)	A(5,1)	A(6,1)	A(1,2)	A(2,2)	A(3,2)	A(4,2)	A(5,2)
A(6,2)	A(1,3)	A(2,3)	A(3,3)	A(4,3)	A(5,3)	A(6,3)	A(1,4)	A(2,4)	A(3,4)	A(4,4)
A(5,4)	A(6,4)	A(1,5)	A(2,5)	A(3,5)	A(4,5)	A(5,5)	A(6,5)	A(1,6)	A(2,6)	A(3,6)
A(4,6)	A(5,6)	A(6,6)								

Array Storage

- **Row order: store first row, then second row, then third….**
- **Example:**
  - **6x6 array A**
  - **Each element 1 word**
  - **Block size: 4 word**
  - **For i=1 to 6; X+= a[1,i]; how many blocks are loaded into cache**

A(1,1)	A(1,2)	A(1,3)	A(1,4)	A(1,5)	A(1,6)	A(2,1)	A(2,2)	A(2,3)	A(2,4)	A(2,5)
A(2,6)	A(3,1)	A(3,2)	A(3,3)	A(3,4)	A(3,5)	A(3,6)	A(4,1)	A(4,2)	A(4,3)	A(4,4)
A(4,5)	A(4,6)	A(5,1)	A(5,2)	A(5,3)	A(5,4)	A(5,5)	A(5,6)	A(6,1)	A(6,2)	A(6,3)
A(6,4)	A(6,5)	A(6,6)								

Loop Interchange Example

- **Assume the array storage is in row order**

```
for (k = 0; k < 100; k = k+1)
     for (j = 0; j < 100; j = j+1)
          for (i = 0; i < 5000; i = i+1)
               x[i][j] = 2 * x[i][j];
```

```
/* After */
for (k = 0; k < 100; k = k+1)
     for (i = 0; i < 5000; i = i+1)
          for (j = 0; j < 100; j = j+1)
               x[i][j] = 2 * x[i][j];
```

- **Sequential accesses instead of striding through memory every 100 words; improved spatial locality**

Loop Fusion Example (1)

```
/* Before */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        d[i][j] = a[i][j] + c[i][j];
```

- **Many programs have separate loops that operate on the same data**
- **Combining these loops allows a program to take advantage of temporal locality by grouping operations on the same (cached) data together.**

Loop Fusion Example (2)

```
/* After */
for (i = 0; i < N; i = i+1)
      for (j = 0; j < N; j = j+1)
      {      a[i][j] = 1/b[i][j] * c[i][j];
             d[i][j] = a[i][j] + c[i][j];}
```

- **2 misses per access to `a` & `c` vs. one miss per access; improve temporal locality**

Blocking Example – Matrix Multiply (1)

```
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
      {r = 0;
       for (k = 0; k < N; k = k+1){
            r = r + y[i][k]*z[k][j];};
       x[i][j] = r;
      };
```

- **Two Inner Loops:**
  - **Read all NxN elements of z[]**
  - **Read N elements of 1 row of y[] repeatedly**
  - **Write N elements of 1 row of x[]**
- **To calculate one entry of X: Sum the multiplication of every element in a single row of Y by every corresponding element in a single column of Z**

Blocking Example – Matrix Multiply (2)

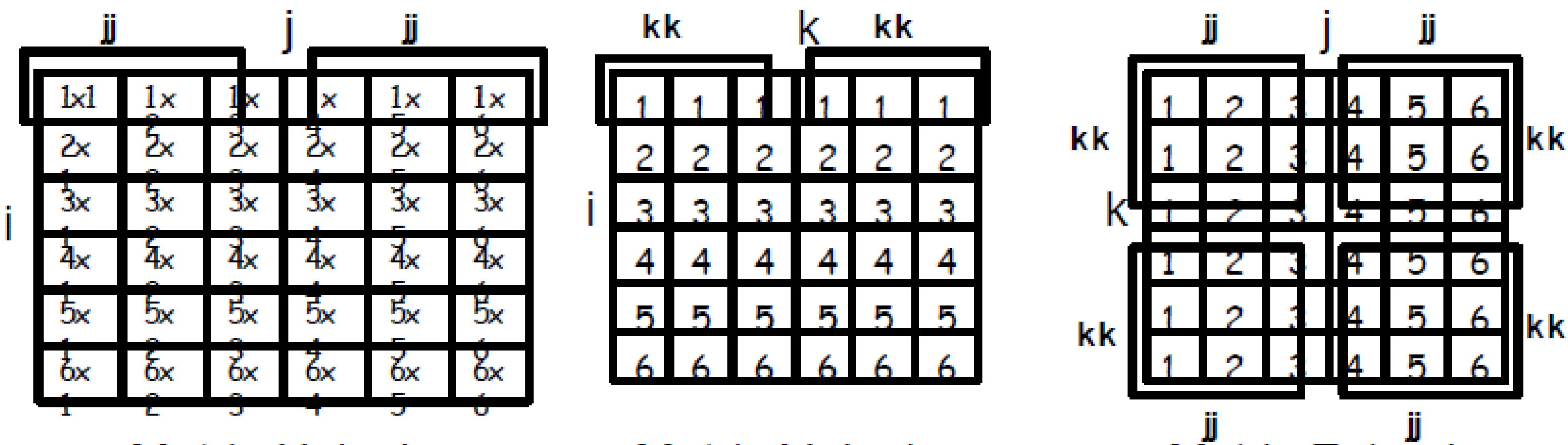
- **Capacity Misses Possibilities: Cache large enough to hold**
  - **all three N x N matrices**
  - **one N x N matrix, two rows of N**
  - **neither row of N nor one N x N matrix**
- **Conclusion: Capacity Misses a function of N & Cache Size**

Blocking Example – Matrix Multiply (3)

- **Problem:**
  - **Worst case capacity misses: 2N^3 + N^2**
  - **Working set of matrix elements is too large to fit in cache**

- **Solution:**
  - **Sub-divide matrices into smaller groups of working sets that will fit in cache, iterate through all subgroups (improves temporal locality)**
- **Maximize accesses to the data loaded into the cache before data is replaced**

Blocking Example – Matrix Multiply (4)



Matrix X: i x j

Matrix Y: i x k

Matrix Z: k x j

Blocking Example – Matrix Multiply (5)

```
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
    for (j = jj; j < min(jj+B-1,N); j = j+1)
      {r = 0;
       for (k = kk; k < min(kk+B-1,N); k = k+1) {
            r = r + y[i][k]*z[k][j];}
       x[i][j] = x[i][j] + r;
      }
```

- **B called *Blocking Factor* (the size of the block)**
- **Capacity misses?**

$2N^3 + N^2$ to $2N^3/B + N^2$

Blocking Example – Matrix Multiply (5)

```
• for (jj = 0; jj < N; jj = jj+B)
• for (kk = 0; kk < N; kk = kk+B)
• for (i = 0; i < N; i = i+1)
•     for (j = jj; j < min(jj+B-1,N); j = j+1)
•       {r = 0;
•        for (k = kk; k < min(kk+B-1,N); k = k+1) {
•            r = r + y[i][k]*z[k][j];}
•        x[i][j] = x[i][j] + r;
•       }
```

- **B called *Blocking Factor* (the size of the block)**
- **Capacity misses?**

$2N^3 + N^2$ to $2N^3/B + N^2$