Lecture 15: Application-level Cache Optimizations

Adapted from UCB CS252 S01, Revised by Zhao Zhang in ISU CPRE 585 F04

Reducing Misses by Compiler Optimizing Memory Layout or Access Pattern

- McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache, 4 byte blocks in software
- Instructions
  - Reorder procedures in memory so as to reduce conflict misses
  - Profiling to look at conflicts
- 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 potential conflicts between val & key;
- Improve spatial locality

Loop Interchange Example

```
/* Before */
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

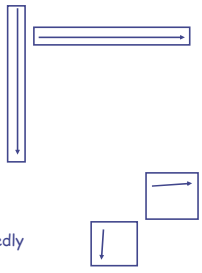
Loop Fusion Example

```
/* 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];
/* 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: Dense Matrix Multipication

```
/* Before */
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;
    };
```



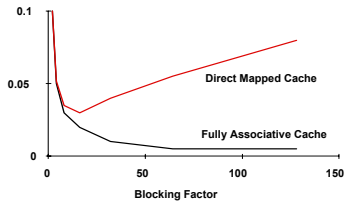
- Two Inner Loops:
  - Read all NxN elements of z[]
  - Read N elements of 1 row of y[] repeatedly
  - Write N elements of 1 row of x[]
- Capacity Misses a function of N & Cache Size:
  - $2N^3 + N^2$ => (assuming no conflict; otherwise …)
- Idea: compute on BxB submatrix that fits

Blocking Example

```
/* After */
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
- Capacity Misses from $2N^3 + N^2$ to $N^3/B+2N^2$
- But may suffer from conflict misses

Reducing Conflict Misses by Blocking



Conflict misses in caches not FA vs. Blocking size
- Choose the best blocking factor

Reducing Conflict Misses by Copying or Padding

- Copying: If some date cause severe cache conflict misses, copy them to another region [Gatlin et al. HPCA'99]
- Padding: Insert padding space to change the mapping of the data onto cache [zhang et al. SC'99]
- Automated approaches: let compiler chooses the best method after analysis

OS Methods in Reducing L2 Cache Conflicts

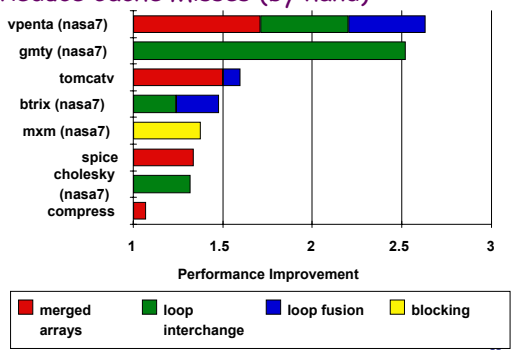
Conflicts are caused by "bad" mapping; can mapping be changed?

Note L2 cache is usually physically indexed!

OS Approaches: change the mapping between virtual memory and physical memory
- Dynamically detected memory pages that cause severe conflict misses
- Change the physical page of those pages so that they are not mapped onto the same sets in cache
- Needs hardware support (cache miss lookaside buffer)

[bershad et al, ISCA'94]

Summary of Compiler Optimizations to Reduce Cache Misses (by hand)



Reducing Misses by Software Prefetching Data

- Data Prefetch
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution
- Prefetching comes in two flavors:
  - Binding prefetch: Requests load directly into register.
    - Must be correct address and register!
  - Non-Binding prefetch: Load into cache.
    - Can be incorrect. Frees HW/SW to guess!
- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

Cache Optimization Summary

	Technique	MP	MR	HT	Complexity
miss penalty	Multilevel cache	+			2
	Critical work first	+			2
	Read first	+			1
	Merging write buffer	+			1
miss rate	Victim caches	+	+		2
	Larger block	-	+		0
	Larger cache		+	-	1
	Higher associativity		+	-	1
	Way prediction		+		2
	Pseudoassociative		+		2
	Compiler techniques		+		0

13

Cache Optimization Summary

	Technique	MP	MR	HT	Complexity
miss penalty	Nonblocking caches	+			3
	Hardware prefetching	+			2/3
	Software prefetching	+	+		3
hit time	Small and simple cache		-	+	0
	Avoiding address translation			+	2
	Pipeline cache access			+	1
	Trace cache			+	3

14