ECE 571 – Advanced Microprocessor-Based Design Lecture 11

Vince Weaver

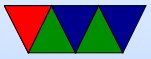
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

28 February 2017

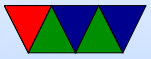
Announcements

- HW#6 was posted, Caches
- No homework over break
- Midterm: Thursday after spring break. Go over what's on it Tuesday. Basically everything up to Thursday's class.



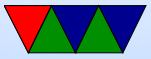
HW#5 (brpred) review

- Bzip2 on Haswell
  instr=19,161M, branches=2,852M,conditional=2,561M
  branch:instr 15% (1:6),conditional 13.4% (1:7)%
  If way too low, entering command wrong (no file error low counts)
- equake_l on Haswell
  instr=1,464B,branches=126B,conditional=93B
  branch:instr 8.8% (1:11), conditional 6.4% (1:15)
- Branch miss rate Haswell bzip2 = 7.27%, equake_l =

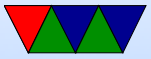


0.49%

- Speculative execution Haswell bzip2: roughly 75% retired, equake_l: roughly 55% retired
- ARM64 bzip2 branch ratio:
  instr=20,141M, branches=3,344M, 17% (1:6)
- ARM64 branch miss rate: 8.4%
- BR% ratio differ? FP vs Int
  Higher ratio because branches more?
- BR ratio on bzip haswell/arm64? actually about the same
- Miss rate differ? FP vs Int program

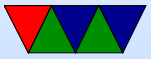


- Different branch predictors.
- Pi worse (17.6%) Lower end CPU? Smaller structures? 32-bit code? Conditional execution? Cortex-A53
- Cortex A-53
  - single entry Branch Target Instruction Cache (BTIC)
  - 256-entry Branch Target Address Cache (BTAC) to predict the target address of indirect branches.
  - The branch predictor is global, uses branch history registers, a 3072-entry pattern history prediction table
  - 8-entry return stack to accelerate returns from procedure calls

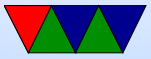


- Cortex-A57
  - 2-level dynamic predictor with Branch Target Buffer (BTB)
  - Static branch predictor.
  - Indirect predictor.
  - Return stack.
- Haswell
  - It's a secret
- power efficiency when lots of speculation?
- What kind of benchmark? random? on ivybridge

`branch-mul`



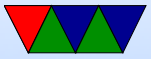
```
5000138 4999862
        20,123,251      branches
         5,004,391      branch-misses
branch-rand
170,143,753      branches                    #  7
        10,358,447      branch-misses
branch-random:
      150,139,161      branches
        10,622,205      branch-misses
```



Cache Example 1

512 Byte cache, 2-Way Set Associative, with 16 byte lines, LRU replacement.

24-bit tag, 16 lines (4 bits), 4-bit offset.

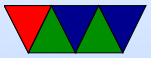


Cache Example – Instruction 1

```
ldb r1, 0x00000000
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	0	0000 00	0			
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
. . .								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold

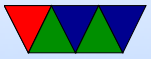


Cache Example – Instruction 2

```
ldb r1, 0x00000001
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	0	0000 00	0			
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Hit

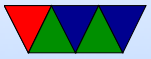


Cache Example – Instruction 3

```
ldb r1, 0x00000010
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	0	0000 00	0			
1	1	0	0	0000 00	0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
. . .								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold

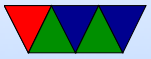


Cache Example – Instruction 4

```
ldb r1, 0x80000010
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	0	0000 00	0			
1	1	0	1	0000 00	1	0	0	8000 00
2	0				0			
3	0				0			
4	0				0			
5	0				0			
. . .								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold

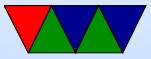


Cache Example – Instruction 5

```
ldb r1, 0xC0000010
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	0	0000 00	0			
1	1	0	0	C000 00	1	0	1	8000 00
2	0				0			
3	0				0			
4	0				0			
5	0				0			
. . .								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold (never in cache previously)

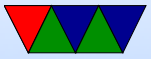


Cache Example – Instruction 6

```
ldb r1, 0xC0000002
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	1	0000 00	1	0	0	c000 00
1	1	0	0	C000 00	1	0	1	8000 00
2	0				0			
3	0				0			
4	0				0			
5	0				0			
. . .								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold

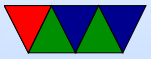


Cache Example – Instruction 7

```
ldb r1, 0x00000010
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	1	0000 00	1	0	0	c000 00
1	1	0	1	C000 00	1	0	0	0000 00
2	0				0			
3	0				0			
4	0				0			
5	0				0			
. . .								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Conflict

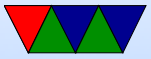


Cache Example – Instruction 8

```
stb r1, 0x00000005
```

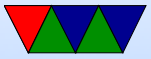
	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	1	0	0000 00	1	0	1	c000 00
1	1	0	1	C000 00	1	0	0	0000 00
2	0				0			
3	0				0			
4	0				0			
5	0				0			
. . .								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Hit



Capacity vs Conflict Miss

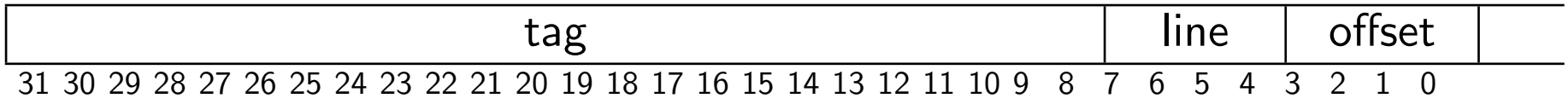
- It's hard to tell on the fly what kind of miss
- For example: to know if cold, need to keep list of every address that's ever been in cache
- To know if it's capacity, need to know if it would have missed even in a fully associative cache
- Otherwise, it's a conflict miss

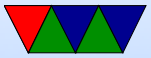


Cache Example Two

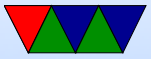
512 Byte cache, 2-Way Set Associative, with 16 byte lines, LRU replacement.

24-bit tag, 16 lines (4 bits), 4-bit offset.





Cache Example 2

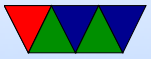


Cache Example – Instruction 1

```
ldrb r1, 0x00000000
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	0	0000 00	0			
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold

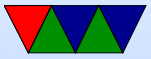


Cache Example – Instruction 2

```
str r1, 0x00000001
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	1	0	0000 00	0			
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Hit

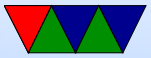


Cache Example – Instruction 3

```
strb r1, 0x00000105
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	1	1	0000 00	1	1	0	0000 01
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold

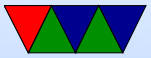


Cache Example – Instruction 4

```
ldr r1, 0x00000206
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	0	0000 02	1	1	1	0000 01
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold

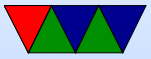


Cache Example – Instruction 5

```
ldb r1, 0x00000000
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	1	0000 02	1	0	0	0000 00
1	0				0			
2	0				0			
3	0				0			
4	0				0			
5	0				0			
...								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Conflict

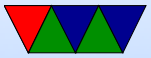


Cache Example – Instruction 6

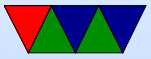
```
ldb r1, 0x00000030
```

	Way 0				Way 1			
line	V	D	LRU	Tag	V	D	LRU	Tag
0	1	0	1	0000 02	1	0	0	0000 00
1	0				0			
2	0				0			
3	1	0	0	0000 00	0			
4	0				0			
5	0				0			
. . .								
b	0				0			
c	0				0			
d	0				0			
e	0				0			
f	0				0			

Miss, Cold

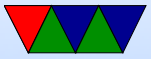


CMP Issues

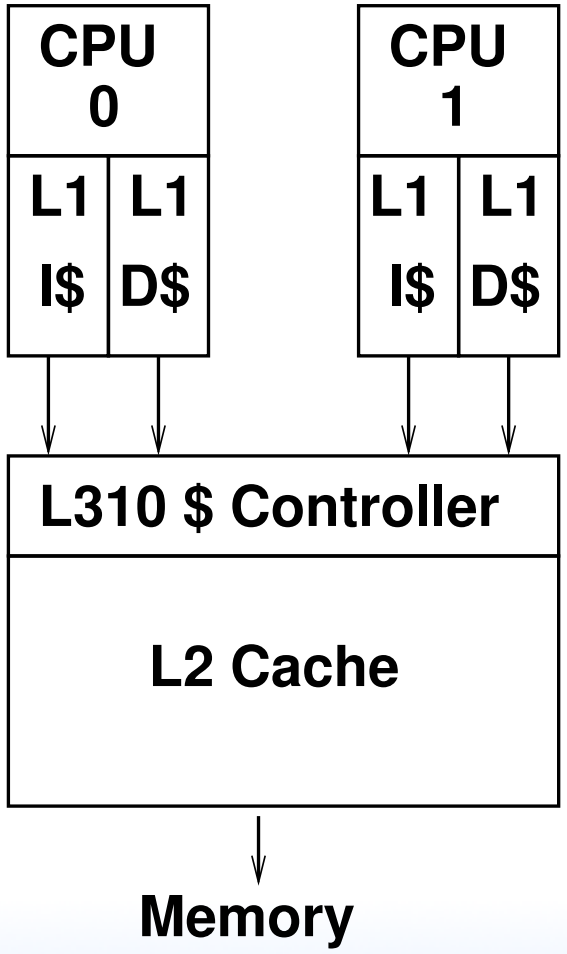


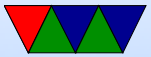
Cache Coherency

- Protocols such as MESI (Modified, Exclusive, Shared, Invalid)
- Snoopy vs Directory



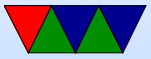
Cortex A9 Cache Layout





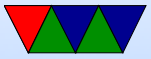
Cortex A9 Cache Layout

- OMAP4430 processor
- 32kB 4-way associative, separate L1-I and L1-D
  pseudo-round-robin or pseudo random replacement
  8-word line size (32B)
  critical-word first filling
  instruction: VIPT, data: PIPT
- Optional L2 cache controller
  Pandaboard has L310 L2 cache controller, 1MB 16-way

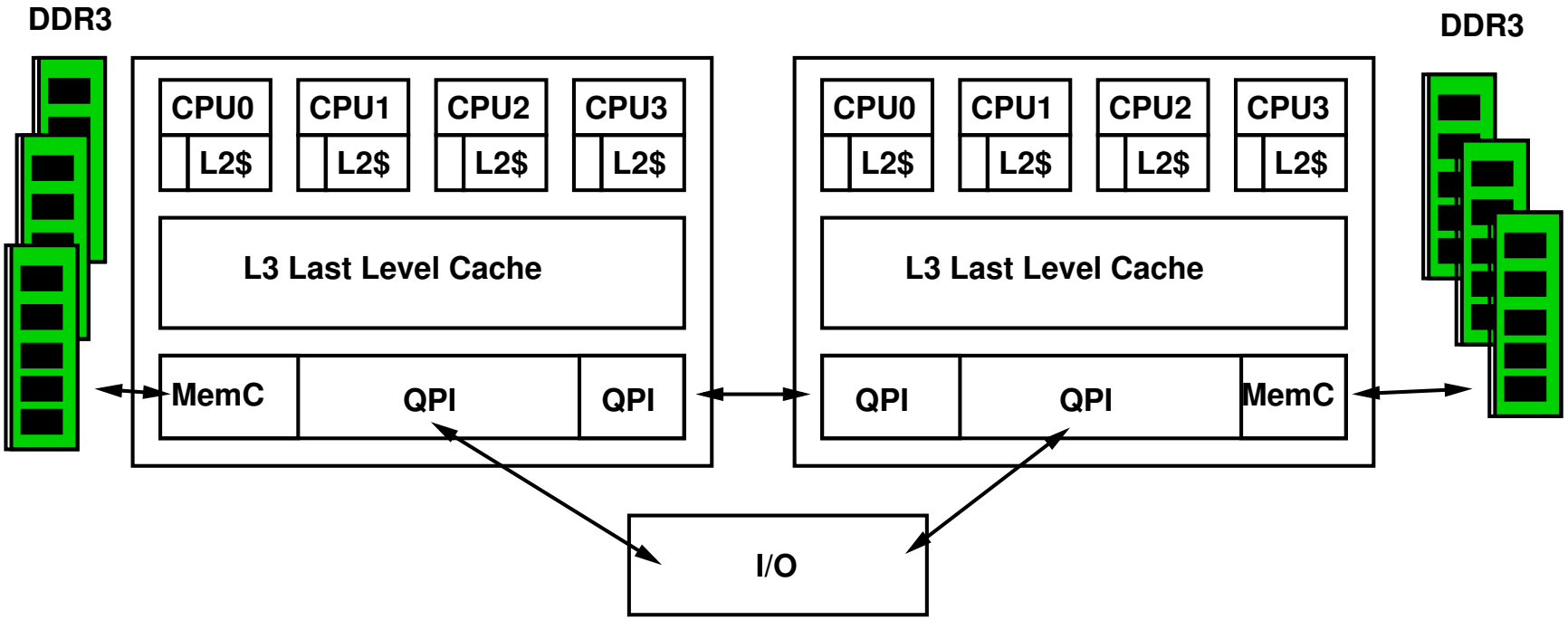


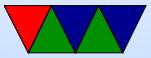
Optional prefetcher

- data cache reads/writes non-blocking, 4 outstanding misses
  write buffer: 4 64-bit, allowing write combining



SandyBridge Cache Layout



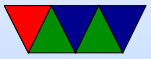


SandyBridge Cache Layout

- per core 32kB L1 I/D – 4 clocks
  64B/line, 8-Way
  (shared if hyper-threaded)
  writeback

- mOp cache? 1.5K instructions, 8-way, 6Mop/line
  Loop stream detector, can execute w/o accessing icache

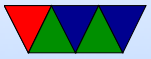
- per core 256kB L2 unified – 12 clocks



64B/line, 8-way
writeback. non-inclusive

- shared L3 1MB-20MB – 26-31 clocks
  64B/line. 12-way (varies)
  writeback, inclusive

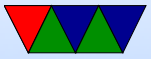
- various hw prefetchers operating



Cache Performance Measurement

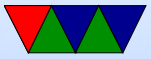
Matrix-Matrix multiply is the typical example.

Despite being a big deal in HPC, MMM happens in embedded world too.



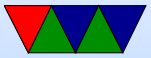
Naive Matrix-Matrix Multiply 1

```
#define MATRIX_SIZE 512
static double a[MATRIX_SIZE][MATRIX_SIZE];
static double b[MATRIX_SIZE][MATRIX_SIZE];
static double c[MATRIX_SIZE][MATRIX_SIZE];

for(j=0;j<MATRIX_SIZE;j++) {
    for(i=0;i<MATRIX_SIZE;i++) {
        for(k=0;k<MATRIX_SIZE;k++) {
            c[i][j]+=a[i][k]*b[k][j];
        }
    }
}
```

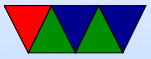


Naive Matrix-Matrix Multiply 1 – what's the issue?

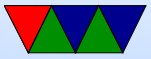
- Branch Misses?
- TLB Misses?
- ICache Misses?
- DCache Misses?
- L2 Cache Misses?



Naive Matrix-Matrix Multiply 1 – perf results

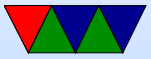


```
Matrix multiply sum: s=27665734022509.746094
 Performance counter stats for './matrix_multiply':

     11296.203614 task-clock                #    0.999 CPUs utilized
              20 context-switches          #    0.000 M/sec
               0 CPU-migrations            #    0.000 M/sec
           1,633 page-faults               #    0.000 M/sec
     9,032,356,979 cycles                  #    0.800 GHz
         6,547,102 stalled-cycles-frontend   #    0.07% frontend cycles id
     8,213,005,758 stalled-cycles-backend    #   90.93% backend  cycles id
     1,176,144,886 instructions              #    0.13  insns per cycle
                                             #    6.98  stalled cycles per
       137,651,296 branches                  #   12.186 M/sec
           795,064 branch-misses             #    0.58% of all branches

      11.303802490 seconds time elapsed
```



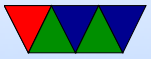
Naive Matrix-Matrix Multiply 1 – DCache results

```
vince@arm:~/class/ece571/lecture10_code$ perf stat -e cache-misses,cache-r
Matrix multiply sum: s=27665734022509.746094

 Performance counter stats for './matrix_multiply':

       171,786,441 cache-misses              #   42.072 % of all cache ref
       408,318,876 cache-references

      10.680664062 seconds time elapsed
```



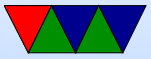
Naive Matrix-Matrix Multiply 1 – ICache results

```
vince@arm:~/class/ece571/lecture10_code$ perf stat -e l1-icache-load-misse
Matrix multiply sum: s=27665734022509.746094

 Performance counter stats for './matrix_multiply':

           536,719 l1-icache-load-misses
     1,174,927,869 instructions              #    0.00  insns per cycle

      12.203002930 seconds time elapsed

0.04% icache misses
```



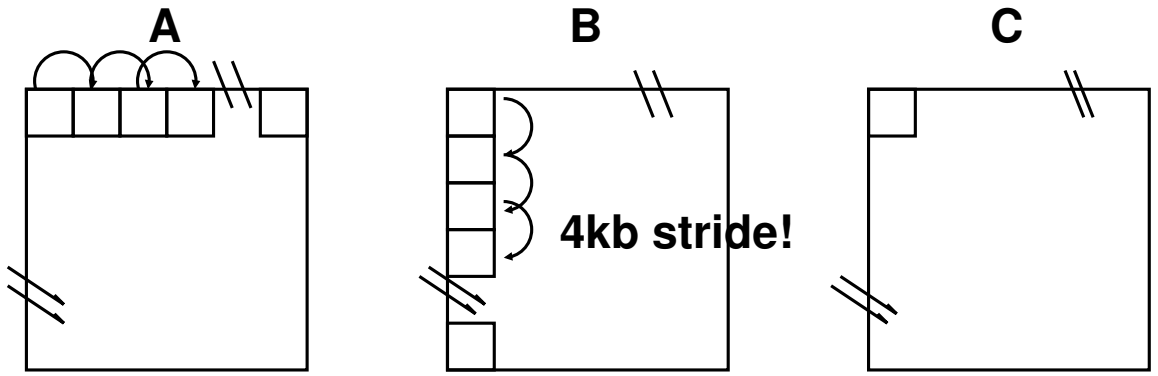
Naive Matrix-Matrix Multiply 1 – TLB Misses

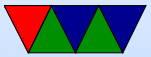
```
vince@arm:~/class/ece571/lecture10_code$ perf stat -e dTLB-load-misses,dTL
Matrix multiply sum: s=27665734022509.746094

 Performance counter stats for './matrix_multiply':

       135,253,464 dTLB-load-misses
       135,253,464 dTLB-store-misses

      12.443572998 seconds time elapsed
```



Naive Matrix-Matrix Multiply 1 – What's the Issue?

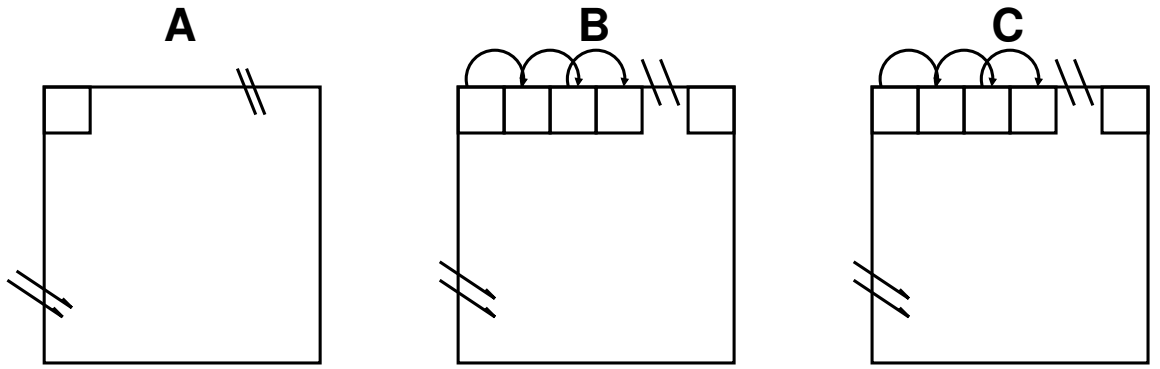
400M memory accesses about right (512x512x512x3)

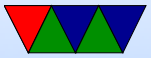




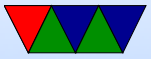
Switch the loop ordering

```
for(i=0;i<MATRIX_SIZE;i++) {
    for(k=0;k<MATRIX_SIZE;k++) {
        for(j=0;j<MATRIX_SIZE;j++) {
            c[i][j]+=a[i][k]*b[k][j];
        }
    }
}
```

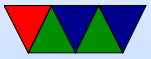




Naive Matrix-Matrix Multiply 2 – perf results

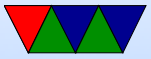


```
Matrix multiply sum: s=27665734022509.746094
 Performance counter stats for './matrix_multiply_swapped':

       3443.267822 task-clock                #    0.999 CPUs utilized
                 5 context-switches          #    0.000 M/sec
                 0 CPU-migrations            #    0.000 M/sec
             1,633 page-faults               #    0.000 M/sec
     2,849,573,010 cycles                    #    0.828 GHz
         2,913,607 stalled-cycles-frontend   #    0.10% frontend cycles id
     1,893,138,507 stalled-cycles-backend    #   66.44% backend  cycles id
       965,962,767 instructions              #    0.34  insns per cycle
                                             #    1.96  stalled cycles per
       136,649,964 branches                  #   39.686 M/sec
           553,643 branch-misses             #    0.41% of all branches

       3.447875977 seconds time elapsed
```



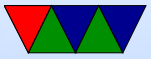
Naive Matrix-Matrix Multiply 2 – DCache results

```
vince@arm:~/class/ece571/lecture10_code$ perf stat -e cache-misses,cache-r
Matrix multiply sum: s=27665734022509.746094

 Performance counter stats for './matrix_multiply_swapped':

       38,628,043 cache-misses              #    9.409 % of all cache ref
      410,528,663 cache-references

      5.585754395 seconds time elapsed
```



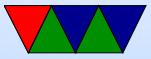
Naive Matrix-Matrix Multiply 2 – ICache results

```
vince@arm:~/class/ece571/lecture10_code$ perf stat -e l1-icache-load-misse
Matrix multiply sum: s=27665734022509.746094

 Performance counter stats for './matrix_multiply_swapped':

          254,041 l1-icache-load-misses
      964,335,795 instructions              #    0.00  insns per cycle

      4.245208740 seconds time elapsed

0.02%
```



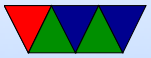
Naive Matrix-Matrix Multiply 1 – TLB Misses

```
vince@arm:~/class/ece571/lecture10_code$ perf stat -e dTLB-load-misses,dTL
Matrix multiply sum: s=27665734022509.746094

 Performance counter stats for './matrix_multiply_swapped':

           486,039 dTLB-load-misses
           486,039 dTLB-store-misses

       5.242126465 seconds time elapsed
```



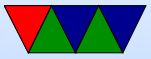
Other Ways to Optimize

- Tiling
- Parallelizing



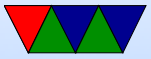
Use ATLAS/BLAS

```
cblas_dgemm(CblasRowMajor,
      CblasNoTrans,CblasNoTrans,
      512,512,512,
      1.0,(const double *)a,512,
      (const double *)b,512,
      1.0,(double *)c,512);
```



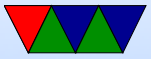
Matrix-Matrix Mul ATLAS – perf results

```
Matrix multiply sum: s=27665734022509.746094
 Performance counter stats for './matrix_multiply_atlas':

       1158.325193 task-clock                #    1.678 CPUs utilized
                12 context-switches          #    0.000 M/sec
                 1 CPU-migrations            #    0.000 M/sec
             2,017 page-faults               #    0.002 M/sec
       597,931,712 cycles                    #    0.516 GHz
         2,043,500 stalled-cycles-frontend   #    0.34% frontend cycles id
       258,860,537 stalled-cycles-backend    #   43.29% backend  cycles id
       519,715,833 instructions              #    0.87  insns per cycle
                                             #    0.50  stalled cycles per
        36,716,368 branches                  #   31.698 M/sec
           815,440 branch-misses             #    2.22% of all branches

       0.690429687 seconds time elapsed
```



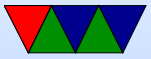
Matrix-Matrix Mul ATLAS – DCache

```
Matrix multiply sum: s=27665734022509.746094

 Performance counter stats for './matrix_multiply_atlas':

       11,988,047 cache-misses             #    8.128 % of all cache ref
      147,494,664 cache-references

      0.598632813 seconds time elapsed
```



Matrix-Matrix Mul ATLAS – TLB

```
vince@arm:~/class/ece571/lecture10_code$ perf stat -e dTLB-load-misses ./m
Matrix multiply sum: s=27665734022509.746094

 Performance counter stats for './matrix_multiply_atlas':

           224,299 dTLB-load-misses

       0.755981446 seconds time elapsed
```



False Sharing?