ECE 571 – Advanced Microprocessor-Based Design Lecture 16

Vince Weaver

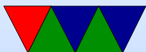
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

7 October 2020

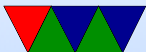
Announcements

- Don't forget HW#5 was posted, Caches



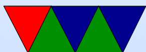
HW#4 (brpred) review – Measurements

- Bzip2 on Haswell-EP
  instr=19,2001M, branches=2,852M,conditional=2,561M
  branch:instr 15% (1:6),conditional 13.6% (1:7)%
  If way too low, entering command wrong (no file error low counts)
- equake_l on Haswell-EP
  instr=1,744B,branches=208B,conditional=178B
  branch:instr 12% (1:8), conditional 10% (1:10)
- Branch miss rate Haswell-EP bzip2 = 7.28%, equake_l



= 0.31%

- Speculative execution Haswell bzip2: roughly 74% retired, equake_l: roughly 54% retired
- AMD EPYC branch miss rate: bzip2 = 7.2% equake_l = 0.37%
- ARM64 bzip2 branch ratio:
  instr=20,141M, branches=3,344M, 17% (1:6)
  note on ARM does predicted include only conditional?
- ARM64 branch miss rate: bzip2=7.8%

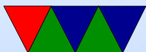


HW#4 (brpred) review – Questions

(a) Why BR% ratio differ?

Compiler being stupid? These are SPEC benchmarks so you can bet that these benchmarks are being optimized as completely as possible.

Floating point vs Integer code. Floating point, like equake, tends to have lots of regular loops over big blocks of calculations. Integer code like compilers and compression reads user data and makes decisions, so many more if/then loops on irregular data.



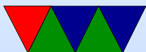
How could you determine the cause?

(b) BR ratio on bzip Haswell/arm64?
actually about the same considering arm64 is more typically RISC and x86 CISC. ARM32 is 1:16 (why? probably conditional execution. How could you tell?)

(c) Miss rate differ? FP vs Int program.
Loops easier to predict.

(d) Different branch predictors.

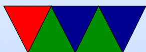
(e) Pi worse (17.6%) Lower end CPU?



Note, ARM64 *completely* different than ARM32
Smaller structures? 32-bit code? Fewer branches (Conditional execution) so ones left are harder? Cortex-A53
What could we do? Run the 32-bit version on Jetson and see?

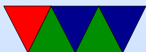
(f) Retired instructions. bzip2: roughly 74% retired, equake_l: roughly 54% retired So in theory bzip2 is more power efficient

(g) 50% benchmark. See below

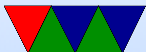


HW#4 brpred hardware

- 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



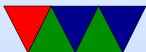
- 2-level dynamic predictor with Branch Target Buffer (BTB)
  - Static branch predictor.
  - Indirect predictor.
  - Return stack.
- Haswell
  - It's a secret (even Agner Fogg doesn't know)



HW#4 Writing a program to give 50%

- What kind of benchmark?
- Random number generation.
- How many branches in random()? Divide-based pseudo-number gen?
- Results below on ivybridge

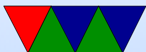
```
branch-mul (pseudo-random, 2 branches per loop)
5000138 4999862
        20,123,251      branches                  #  228.926 M/sec
         5,004,391      branch-misses             #   24.87% of all branches
branch-rand (rand(), 17 branches per loop)
170,143,753      branches                  #  726.443 M/sec
        10,358,447      branch-misses             #    6.09% of all branches
branch-random (random(), 15 branches per loop)
      150,139,161      branches                  #  719.563 M/sec
        10,622,205      branch-misses             #    7.07% of all branches
```



Cache Performance Measurement

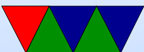
Matrix-Matrix multiply is the typical example.

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



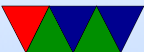
Naive Matrix-Matrix Multiply 1

```
#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];
        }
    }
}
```

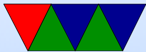


Naive Matrix-Matrix Multiply 1 – what's the issue?

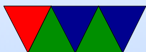
- Branch Misses?
- TLB Misses?
- ICache Misses?
- DCache Misses?
- L2 Cache Misses?



Naive Matrix-Matrix Multiply 1 – perf results

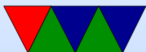


```
Matrix multiply sum: s=27665734022509.746094
 Performance counter stats for './matrix_multiply':

      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
```



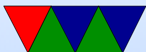
Naive Matrix-Matrix Multiply 1 – DCache results

```
vince@arm:~/class/ece571/lecture10_code$
perf stat -e cache-misses,cache-references ./matrix_multiply
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
```



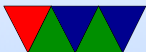
Naive Matrix-Matrix Multiply 1 – ICache results

```
vince@arm:~/class/ece571/lecture10_code$
perf stat -e l1-icache-load-misses,instructions ./matrix_multiply
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
```



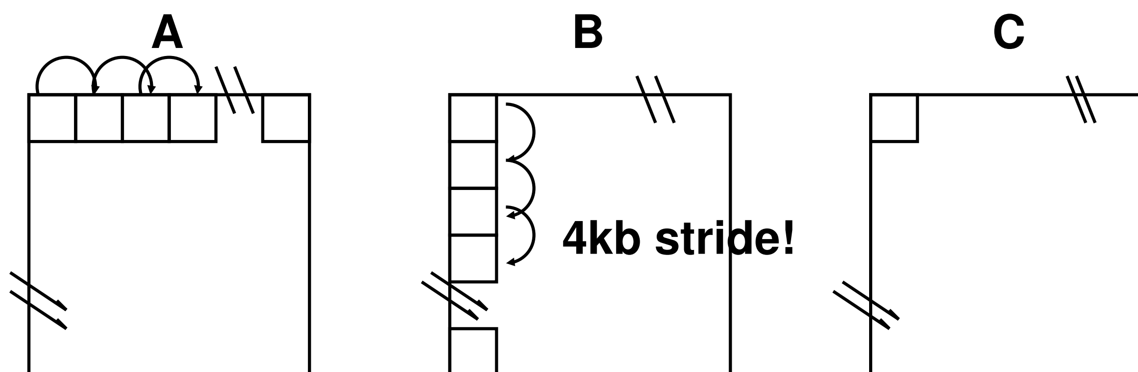
Naive Matrix-Matrix Multiply 1 – TLB Misses

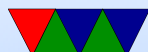
```
vince@arm:~/class/ece571/lecture10_code$
perf stat -e dTLB-load-misses,dTLB-store-misses ./matrix_multiply
Matrix multiply sum: s=27665734022509.746094

 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?

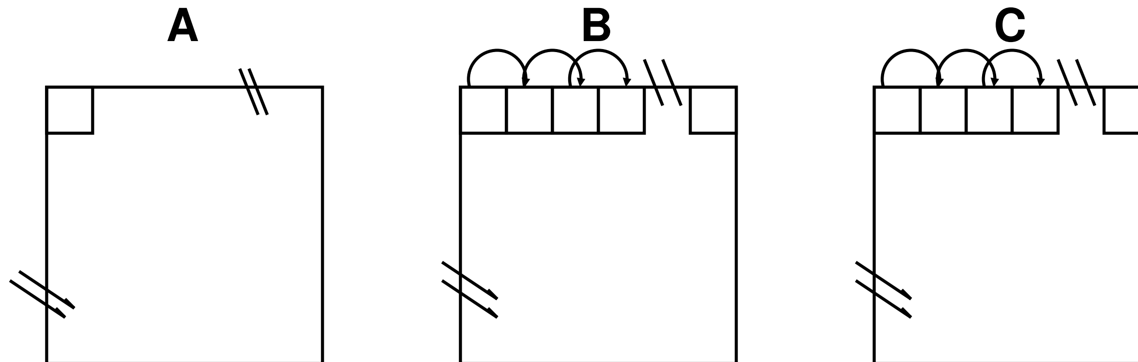
400M memory accesses about right (512x512x512x3)

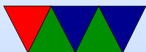




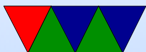
Switch the loop ordering

```
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];
      }
   }
}
```

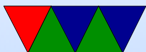




Naive Matrix-Matrix Multiply 2 – perf results

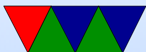


```
Matrix multiply sum: s=27665734022509.746094
 Performance counter stats for './matrix_multiply_swapped':

       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
```



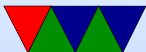
Naive Matrix-Matrix Multiply 2 – DCache results

```
vince@arm:~/class/ece571/lecture10_code$
perf stat -e cache-misses,cache-references ./matrix_multiply_swapped
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
```



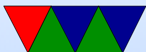
Naive Matrix-Matrix Multiply 2 – ICache results

```
vince@arm:~/class/ece571/lecture10_code$
perf stat -e l1-icache-load-misses,instructions ./matrix_multiply_swapped
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%
```



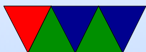
Naive Matrix-Matrix Multiply 1 – TLB Misses

```
vince@arm:~/class/ece571/lecture10_code$
perf stat -e dTLB-load-misses,dTLB-store-misses ./matrix_multiply_swapped
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
```



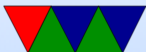
Other Ways to Optimize

- Tiling
- Parallelizing



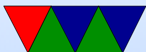
Use ATLAS/BLAS

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



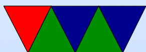
Matrix-Matrix Mul ATLAS – perf results

```
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 ./matrix_multiply_atlas
Matrix multiply sum: s=27665734022509.746094

 Performance counter stats for './matrix_multiply_atlas':

           224,299 dTLB-load-misses

       0.755981446 seconds time elapsed
```