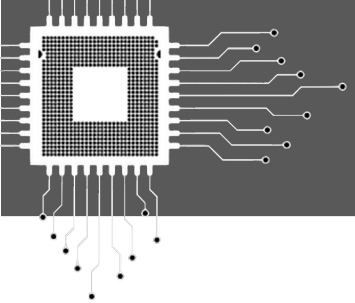
Project 1 - CPU Monitoring (Linux)

[Team 11]

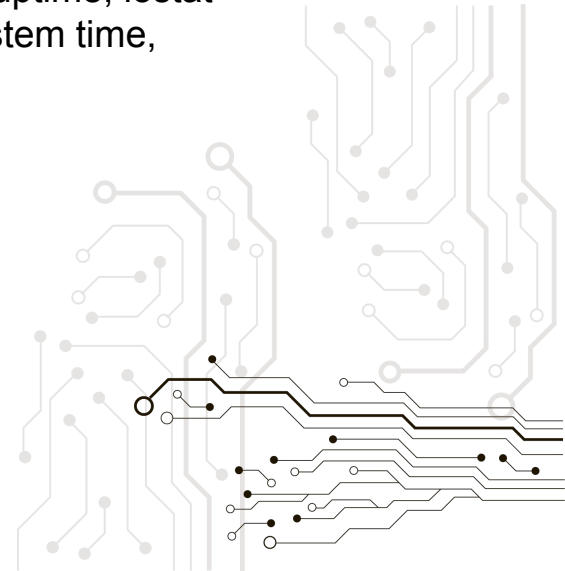
Chong Yihui, Gan Wan Cheng Isaac, Kenneth Goh Zhen Hao, Manish Kumar, Soedarsono

NUS National University of Singapore | School of Computing



Goals

- Use various tools to monitor performance (vmstat, htop, mpstat, uptime, iostat etc.) of the CPU and interpret the various metrics (User time, System time, Waiting I/O, Idle Time, Nice Time).
- Understand the x86 Hardware Performance Counters
- Learn how to optimise a program



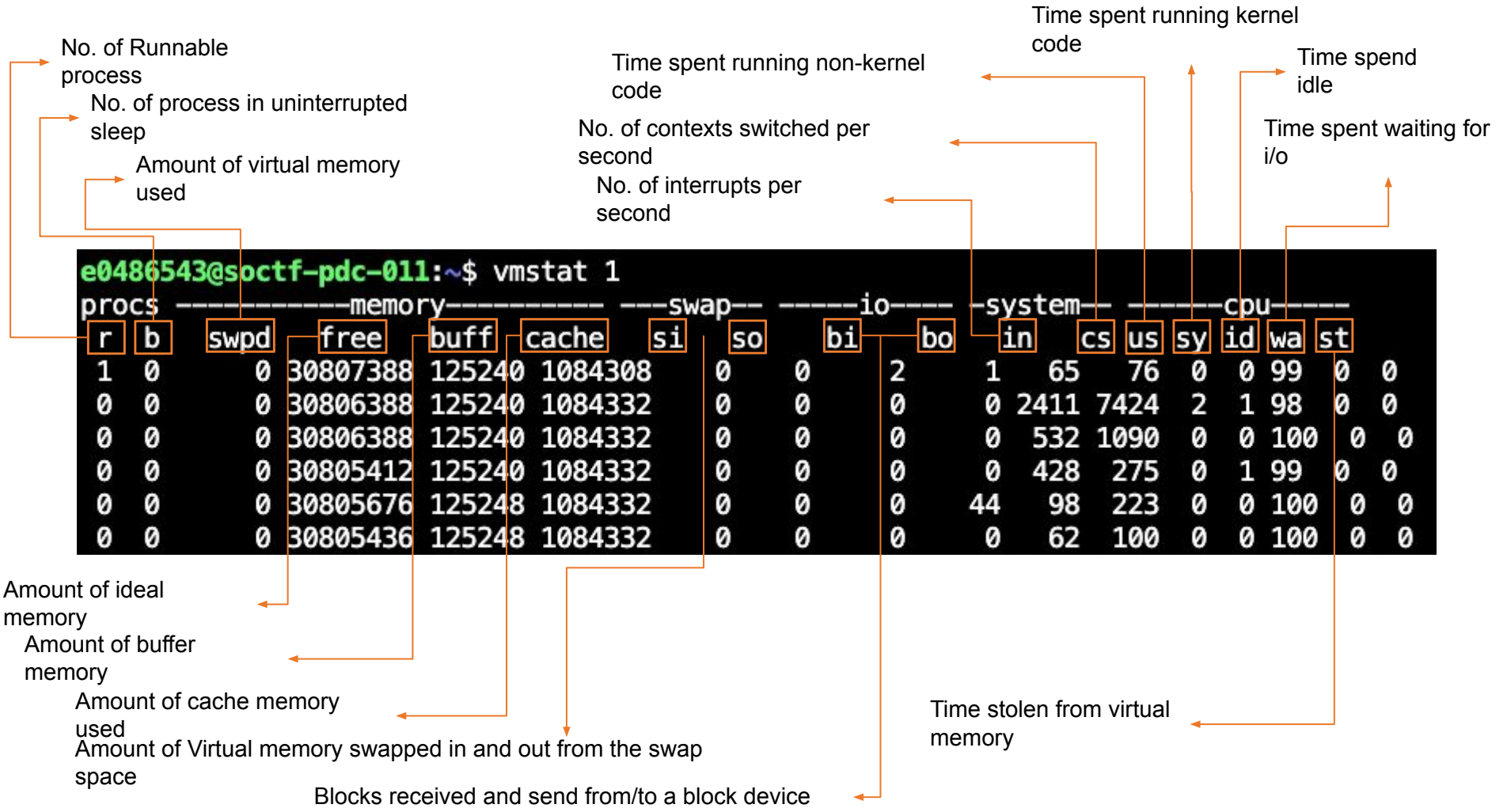
vmstat

vmstat gets its value from system calls to the proc file system

```
openat(AT_FDCWD, "/proc/sys/kernel/osrelease", O_RDONLY) = 3
newfstatat(3, "", {st_mode=S_IFREG|0444, st_size=0, ...}, AT_EMPTY_PATH) = 0
read(3, "5.15.0-52-generic\n", 1024)     = 18
close(3)                                 = 0
openat(AT_FDCWD, "/proc/meminfo", O_RDONLY) = 3
lseek(3, 0, SEEK_SET)                    = 0
read(3, "MemTotal:       32737204 kB\nMemF"..., 8191) = 1419
openat(AT_FDCWD, "/proc/stat", O_RDONLY) = 4
read(4, "cpu  300656 609 15486 1207146 84"..., 131071) = 4668
openat(AT_FDCWD, "/proc/vmstat", O_RDONLY) = 5
```

Disk statistics sorted by **descending order of total reads**

```
e0486543@soctf-pdc-011:~$ vmstat -nd | tail -n +3 | sort -nrk2
sda    25178  15891 2539526  817002  40932  29890 1355226  258252      0    273
loop2  16438      0   34920   80058      0      0       0       0      0      6
loop15    66      0    2130    3772      0      0       0       0      0      3
loop10    63      0    2104    4987      0      0       0       0      0      2
loop13    62      0    2112    4176      0      0       0       0      0      2
loop27    61      0    2156    2960      0      0       0       0      0      2
loop8     60      0    2120    1497      0      0       0       0      0      1
loop22    59      0    2108    2246      0      0       0       0      0      1
```


No. of Runnable process
No. of process in uninterrupted sleep
Amount of virtual memory used
Time spent running non-kernel code
No. of contexts switched per second
No. of interrupts per second
Time spent running kernel code
Time spend idle
Time spent waiting for i/o
e0486543@soctf-pdc-011:~$ vmstat 1
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa st
1 0 0 30807388 125240 1084308 0 0 2 1 65 76 0 0 99 0 0
0 0 0 30806388 125240 1084332 0 0 0 0 2411 7424 2 1 98 0 0
0 0 0 30806388 125240 1084332 0 0 0 0 532 1090 0 0 100 0 0
0 0 0 30805412 125240 1084332 0 0 0 0 428 275 0 1 99 0 0
0 0 0 30805676 125248 1084332 0 0 0 44 98 223 0 0 100 0 0
0 0 0 30805436 125248 1084332 0 0 0 0 62 100 0 0 100 0 0
Amount of ideal memory
Amount of buffer memory
Amount of cache memory used
Amount of Virtual memory swapped in and out from the swap space
Blocks received and send from/to a block device
Time stolen from virtual memory

mpstat

```
def factorial(n):
    sys.setrecursionlimit(sys.getrecursionlimit() + 1)
    return 1 if n == 0 \
           else n * factorial(n - 1)
```

nw=$((`nproc` - 1)) && taskset -c 1-$nw stress --cpu $nw

```
16:49:19      CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
16:49:20      all   90.40    0.00    0.12    0.00    0.00    0.00    0.00    0.00    0.00    9.48
16:49:20        0   24.75    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   75.25
16:49:20        1  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
16:49:20        2  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
16:49:20        3  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
16:49:20        4  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
16:49:20        5   99.01    0.00    0.99    0.00    0.00    0.00    0.00    0.00    0.00    0.00
16:49:20        6  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
16:49:20        7  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
```

```
16:49:51      CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
16:49:52      all   96.26    1.00    2.74    0.00    0.00    0.00    0.00    0.00    0.00    0.00
16:49:52        0  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
16:49:52        1   96.04    0.99    2.97    0.00    0.00    0.00    0.00    0.00    0.00    0.00
16:49:52        2   94.00    3.00    3.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
16:49:52        3   96.00    0.00    4.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
16:49:52        4   96.04    0.99    2.97    0.00    0.00    0.00    0.00    0.00    0.00    0.00
16:49:52        5   90.10    0.99    8.91    0.00    0.00    0.00    0.00    0.00    0.00    0.00
16:49:52        6  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
16:49:52        7   98.00    2.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
```

htop

```
  0[||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]   4[||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
  1[||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]   5[||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
  2[||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]   6[||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
  3[||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]   7[||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
  Mem[|||||||||||||||                                                    ]   Tasks: 168, 784 thr; 8 running
  Swp[                                                                   ]   Load average: 4.41 4.67 3.28
                                                                             Uptime: 00:56:23

CPU     PID USER      PRI  NI  VIRT   RES   SHR S CPU CPU%▽MEM%   TIME+  Command
  0   18318 ellie      20   0 32808 13352  5804 R   0 98.6  0.0  0:16.15 python3 fact_rcrs_unlimited.py 10000
  5   18312 ellie      20   0  3704   108     0 R   5 86.2  0.0  0:19.51 stress --cpu 7
  2   18317 ellie      20   0  3704   108     0 R   2 98.6  0.0  0:19.21 stress --cpu 7
  7   18316 ellie      20   0  3704   108     0 R   7 98.6  0.0  0:20.05 stress --cpu 7
  2   18314 ellie      20   0  3704   108     0 R   2 94.7  0.0  0:19.67 stress --cpu 7
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

To binds CPU stress workers on your odd-numbered cores:

taskset -c 1-`nproc`:2 stress --cpu $((nproc/2))

Without fixing the script to run on CPU 0, we occasionally get this:

Measurements

Write a script to measure the compression rate and the time required for each level

[Code Snippet]

```
for i in 0 1 2 3 4 5 6 7 8 9
    do
        Level+="$i,"
        ZipOutput="$((time zip -$i -v $var.zip $var) 2>&1))"
        # echo $ZipOutput
        array=($ZipOutput)
```

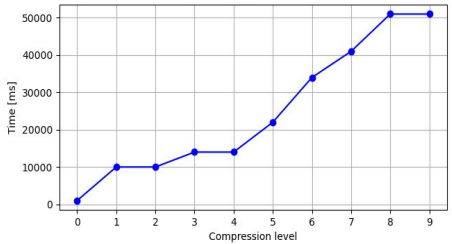
```
originalSize=$(stat -c %s $var)
compressedSize=$(stat -c %s $var.zip)
reducedSize=$(($originalSize-$compressedSize))
reducedScale=$(echo "scale=4; $reducedSize /
```

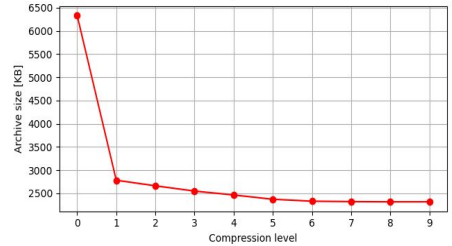
[Output]

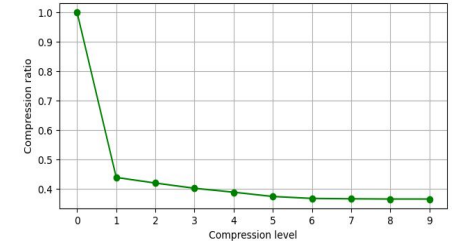
```
e0486543@soctf-pdc-011:~/cs5239-computer-system-performance-analysis-project$ bash compression.sh big.txt MARBLES.bmp RAY.bmp
File : big.txt
————————————
Level: 0,1,2,3,4,5,6,7,8,9,
Real Time: 0.022,0.096,0.108,0.146,0.142,0.222,0.335,0.412,0.497,0.502,
User Time: 0.013,0.096,0.100,0.142,0.142,0.222,0.335,0.412,0.493,0.498,
System Time: 0.008,0.000,0.008,0.004,0.000,0.000,0.000,0.000,0.004,0.004,
Initial Size: 6488666,6488666,6488666,6488666,6488666,6488666,6488666,6488666,6488666,6488666,
Compressed Size: 6488830,2848375,2724635,2609286,2522414,2428270,2385401,2377033,2372291,2372271,
Reduction Size: -164,3640291,3764031,3879380,3966252,4060396,4103265,4111633,4116375,4116395,
Reduction Scale: 0,.5610,.5800,.5978,.6112,.6257,.6323,.6336,.6343,.6343,

File : MARBLES.bmp
————————————
Level: 0,1,2,3,4,5,6,7,8,9,
Real Time: 0.012,0.088,0.092,0.098,0.115,0.140,0.145,0.148,0.157,0.160,
User Time: 0.012,0.080,0.088,0.090,0.115,0.140,0.141,0.148,0.153,0.156,
System Time: 0.000,0.008,0.004,0.008,0.000,0.000,0.004,0.000,0.004,0.004,
Initial Size: 4264316,4264316,4264316,4264316,4264316,4264316,4264316,4264316,4264316,4264316,
Compressed Size: 4264488,2583093,2535261,2506032,2529707,2516562,2512826,2512239,2511796,2511669,
Reduction Size: -172,1681223,1729055,1758284,1734609,1747754,1751490,1752077,1752520,1752647,
Reduction Scale: 0,.3942,.4054,.4123,.4067,.4098,.4107,.4108,.4109,.4110,

File : RAY.bmp
————————————
Level: 0,1,2,3,4,5,6,7,8,9,
Real Time: 0.005,0.017,0.020,0.028,0.024,0.038,0.066,0.092,0.183,0.200,
User Time: 0.005,0.017,0.016,0.028,0.025,0.034,0.066,0.092,0.183,0.200,
System Time: 0.000,0.000,0.004,0.000,0.000,0.004,0.000,0.000,0.000,0.000,
Initial Size: 1440054,1440054,1440054,1440054,1440054,1440054,1440054,1440054,1440054,1440054,
Compressed Size: 1440218,331253,310248,285757,255569,240879,231742,229192,227524,227390,
Reduction Size: -164,1108801,1129806,1154297,1184485,1199175,1208312,1210862,1212530,1212664,
Reduction Scale: -.0001,.7699,.7845,.8015,.8225,.8327,.8390,.8408,.8420,.8420,
```

Measurements

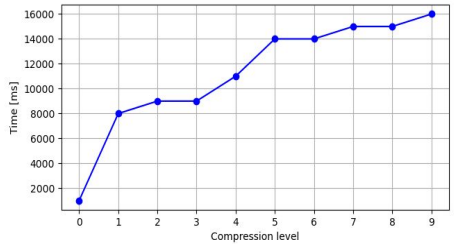
BIG

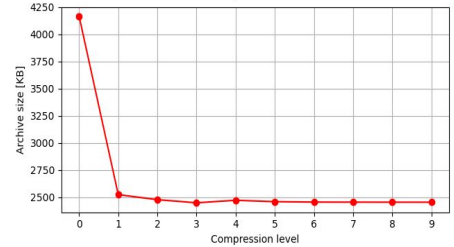


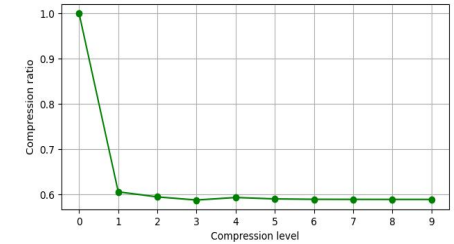




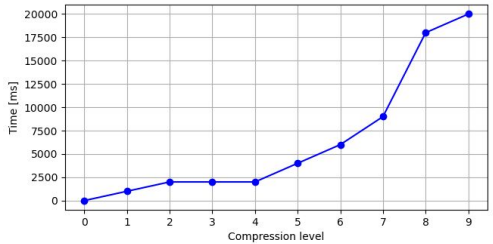
MARBLES

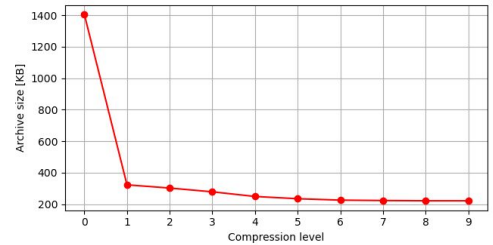


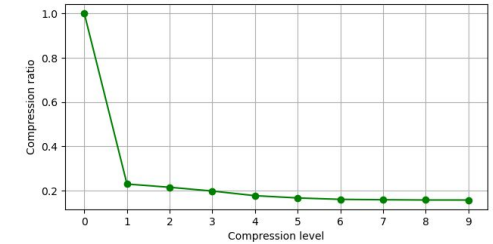




RAY







Hardware Counters

CPUID.EAX = 0AH (* Returns Architectural Performance Monitoring leaf. *)

Command: **cpuid -r**

```
0x0000000a 0x00: eax=0x07300404 ebx=0x00000000 ecx=0x00000000 edx=0x00000603
```

CPUID.0AH:EAX[15:8]: number of general purpose counters: 0x04: 4
CPUID.0AH:EAX[7:0] : version ID: 0x04: 4
CPUID.0AH:EDX[7:0] : number of fixed function counters: 0x03: 3

To start tracking L2 Misses

```
sudo wrmsr -a 0xc1 0x00
sudo wrmsr -a 0x186 0x413f24
```

Event Num.	Umask Value	Event Mask Mnemonic	Description
24H	3FH	L2_RQSTS.MISS	All requests that missed L2.

Hardware Counters

[Naive (IJK) Matrix Multiplication]

```
for (uint32_t i=0; i<N; i++)              /* line   */
    for (uint32_t j=0; j<N; j++)          /* column */
        for (uint32_t k=0; k<N; k++)
            r[i*N + j] += m1[i*N + k] * m2[k*N + j];
```

[Code Snippets]

```
rdpmc(0, eax, edx);
start_cnt  = ((int64_t)eax) | ((int64_t)edx << 32);
```

```
ellie@HAL9000:~/Github/Perf_Analysis/code/project1/hw_counter$ taskset 0x01 ./mat_mul 1024
L2 Cache Miss: 2092713129
Multiplication 1 finished in  10.79 s    << Naive Results
L2 Cache Miss: 184373924
Multiplication 2 finished in   3.55 s    << Better Loops Results
ellie@HAL9000:~/Github/Perf_Analysis/code/project1/hw_counter$
```

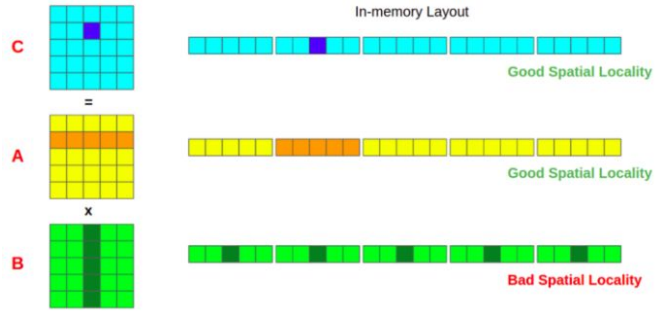
Optimising with Better Loops

```
/* perform fast(er) multiplication */
for (uint32_t i=0; i<N; i++)
    for (uint32_t k=0; k<N; k++)          /* line   */
        for (uint32_t j=0; j<N; j++)      /* column */
            r[i*N + j] += m1[i*N + k] * m2[k*N + j];
```

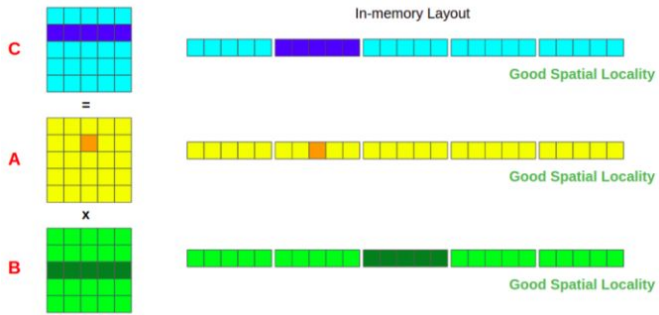
[Results]

Order	ijk	ikj	jik	jki	kij	kji
Time (s)	17.63	**3.74**	10.23	36.15	3.76	29.83
L2 Cache miss	2,146,934,417	**179,497,619**	2,287,344,732	4,260,117,593	184,105,100	4,305,142,993

[Standard Approach - IJK]



[Best Loop Approach - IKJ]



References: *The impact of cache locality on performance in C through matrix multiplication*

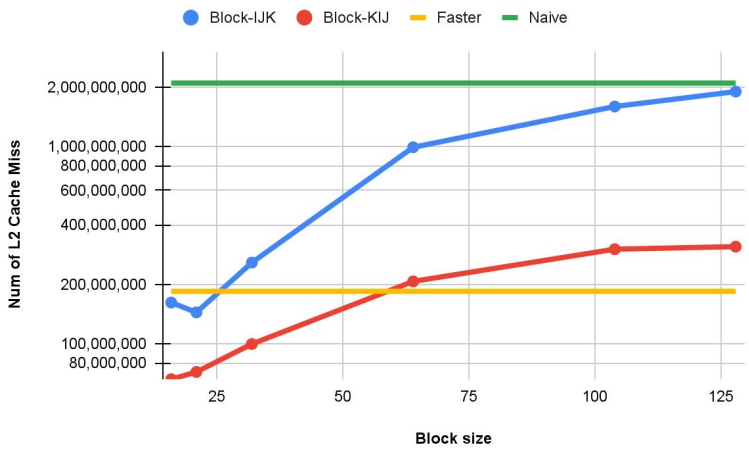
Optimising with Better Loops (Transpose)

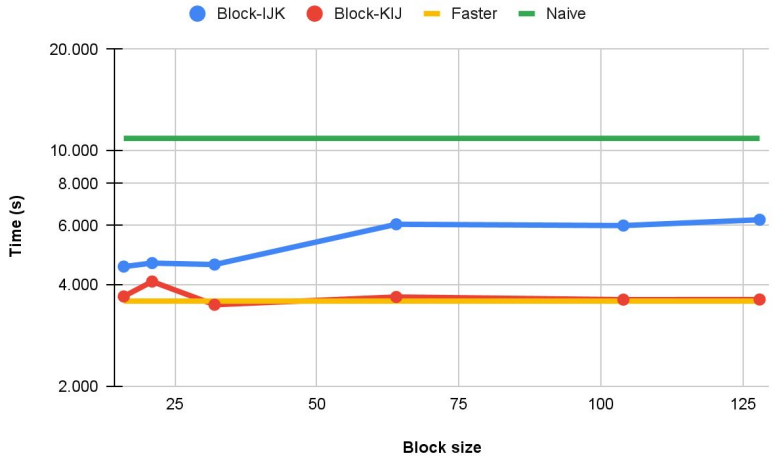
```
/* Transpose M2 */
for (uint32_t i=0; i<N; i++)          /* line   */
    for (uint32_t j=0; j<N; j++)      /* column */
        m2_t[j*N + i] = m2[i*N + j];

/* perform transpose multiplication */
for (uint32_t i=0; i<N; i++)          /* line   */
    for (uint32_t j=0; j<N; j++)      /* column */
        for (uint32_t k=0; k<N; k++)
            rBlk[i*N + j] += m1[i*N + k] * m2_t[j*N + k];
```

	Time (s)	L2 Cache Miss
IJK	10.889	2,095,685,036
KIJ	3.559	184,501,076
Transpose	3.567	181,702,620

Blocked (Tiling) Matrix Multiplication

Partition matrices into smaller blocks that can be wholly fitted into cache
⇒ Loop order should not matter
⇒ Reduces cache miss





[Limitation] Need to tune block size based on matrix size and CPU cache size

Optimising with Compiler

Compiler Optimization	Time Improvement
Auto-unrolling (-funroll-loops)	25.6 %
Auto-vectorization (-ftree-vectorize)	29.0 %
Math operation optimizations (-ffast-math)	30.5 %
Leveraging on AVX2 (-march=native)	**38.9 %**

Normal loop

```
int x;
for (x = 0; x < 100; x++)
{
    delete(x);
}
```

After loop unrolling

```
int x;
for (x = 0; x < 100; x+=5)
{
    delete(x);
    delete(x+1);
    delete(x+2);
    delete(x+3);
    delete(x+4);
}
```

Example of loop unrolling

```
cvtsi2ssq       -24(%rbp), %xmm0
movss   .LC4(%rip), %xmm1
divss   %xmm1, %xmm0
cvtss2sd        %xmm0, %xmm0

vcvtsi2ssq      -24(%rbp), %xmm0, %xmm0
vmovss  .LC4(%rip), %xmm1
vdivss  %xmm1, %xmm0, %xmm0
vcvtss2sd       %xmm0, %xmm0, %xmm0
```

Default instructions in RED
AVX instruction set in green

Optimising with Compiler

Optimisation Levels	Time (s)	L2 Cache Miss
Naive	10.889	2,095,685,036
O1-Naive	2.407	1,462,736,895
O2-Naive	2.273	1,489,176,762
O3-Naive	2.410	1,461,982,966
Ofast-Naive	**2.187 (~5x faster)**	**1,496,253,025**

Optimisation Levels	Time (s)	L2 Cache Miss
Faster	3.559	184,501,076
O1-Faster	1.020	181,810,513
O2-Faster	0.947	182,316,248
O3-Faster	0.893	182,334,093
Ofast-Faster	**0.873 (~4x faster)**	**182,119,835**

[Next Steps]

- Loop interchange
- Other compilers (ICC, Clang)

Multi-threading

[OpenMP]

- Need to optimize parallel code carefully

Matrix Size (N)	256	512	1024	2048	4096	
IJK	0.089	0.761	10.889	183.748	1691.498	
KIJ	0.050	0.406	3.559	28.216	233.656	
OMP1 (KIJ)	0.232	1.491	11.391	92.151	742.021	
OMP2 (KIJ)	**0.115**	**0.583**	**4.977**	**40.125**	**330.648**	**(~2x)**

CPU Time (s)

[OMP1 (Faster)] Uses shared variable (atomic) for array update
[OMP2 (Faster)] Uses reduction for array update

Multi-threading

[OpenMP]

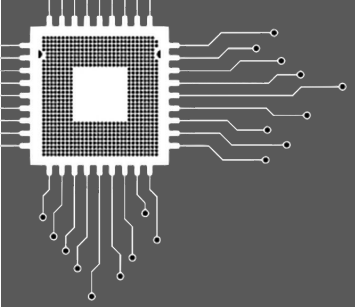
- Need to optimize parallel code carefully
  - Incorrect result or segmentation fault
    - Race conditions
    - Stack size limits

Time (s)	256	512	1024	2048	4096	8192
IJK	0.087	0.733	9.467	167.595	1491.353	14433.110
KIJ	0.050	0.396	3.218	26.411	211.032	1766.091
OMP (IJK)	0.044	0.210	1.988	42.144	469.860	3618.094
OMP1 (KIJ)	0.033	0.222	1.635	12.639	100.208	801.639
OMP2 (KIJ)	**0.025**	**0.126**	**0.977**	**7.404**	**62.220**	**476.490**
	(~3x)	(~6x)	(~10x)	(~23x)	(~24x)	(~30x)

4 hrs

8 mins

Wall Clock Time (s)



Conclusion

[Best Techniques Yet]

1. IKJ-order multiplication
2. Block matrix multiplication
3. Optimising w/ AVX2
4. Ofast GCC optimization level
5. Multi-threading w/ OpenMP

[Challenges]

- Requires Intel CPU and sudo rights
- Unfamiliarity in Computer Organisation, OS, Parallel Programming and ASM
- Long runtime for experiments w/ larger matrix

[Future Directions]

- Better loops
- Loop interchange and other compilers (ICC, Clang)
- Multi-threading w/ pthreads
- Combining techniques