berserker1 - github account

For matrix of size n*n multiplying with other matrix of the same dimension.

Total floating point operation = (n+n-1) * n^2

N = 64 -> 520192
N = 128 -> 4117920
N = 256 -> 33488896
N = 512 -> 268173312
N = 1024 -> 2146435072
N = 2048 -> 17175674880

Problem 1.1

Here we are assuming no cache

Clock frequency 2GhZ
For **dgemmo** we have (per innermost iteration)

- 2 load (200 cycles)
- 1 store (100 cycles)
- 2 floating point operations (1 cycle)

For n=1000 total interations we will have is n^3.

Hence, it will take (200 + 100 + 1) * (10^9) cycles = (301 * 10^9)

Time spent on reading/writing (Converting this to seconds)

(300 * 10^(9)) /(2 * 10^(9)) = 150 sec

For **dgemm1** we have (n = 1000)
per innermost iteration

- 2 load (200 cycles)
- 2 floating point operations (1 cycle)

2nd loop

- One load (100 cycle)
- One Store (100 cycle)

Inner loop runs n^3 times = (200 + 1) * 10^9
2nd loop runs n^2 times = 200 * (10 ^ 6)

Time spent on reading/writing (Converting this to seconds)

((200) * (10^9) / 2 * 10^9) for inner loop

(200 * 10^6) / (2 * 10^9) for 2nd loop

Total = 100 + 0.1 = 100.1 sec

Problem 1.2

For dgemm0 we try the values of n given there and we get the following output

```
n=64
time=0.002019s
n=128
time=0.017137s
n=256
time=0.202748s
n=512
time=2.310106s
n=1024
time=17.662696s
n=2048
time=252.926354s
```

Problem 2

For dgemm2 the output is

```
n=64
time=0.000262s
n=128
time=0.002180s
n=256
time=0.020356s
n=512
time=0.229089s
n=1024
time=2.095926s
n=2048
time=44.806179s
dgemm2
```

Converting to Gflops respectively it will be

- 1.985465649
- 1.888954128
- 1.645160935
- 1.170607546
- 1.024098691
- 0.3833327292

Problem 3

For dgemm3 the output is

```
n=64
time=0.000273s
n=128
time=0.002223s
n=256
time=0.023768s
n=512
time=0.242686s
n=1024
time=1.879007s
n=2048
time=46.796065s
```

Converting to Gflops respectively it will be

- 1.905465201
- 1.852415655
- 1.408990912
- 1.105021765
- 1.142324149
- 0.3670324605

Somewhat unexpected decrease in performance as compared to dgemm3 that is because I decreased one register named “size” which was used to store the value of n and use every time instead of integer n, so overall register usage decreased but performance slightly downgraded. **Implemented dgmm3 in 10 registers**

Problem 4

Here “total cache miss” == “total read cache miss”

Ijk

Total cache miss for A in one iteration of the innermost loop = n/10
Total cache miss for B in one iteration of the innermost loop = n/10

Total write request = n^2
Total Read attempts = 2*(n^3)
Hence total cache miss = (2n/10) * (n^2)
Jki

Total cache miss for A in one iteration of the innermost loop = n/10
Total cache miss for B in one iteration of the innermost loop = 0
Total cache miss for C in one iteration of the innermost loop = n/10

Total write request = n^3
Hence total cache miss = (2n/10) * (n^2)
Total Read attempts = 2*(n^3) + n^2

Kji

Total cache miss for A in one iteration of the innermost loop = n/10
Total cache miss for B in one iteration of the innermost loop = 0
Total cache miss for A in one iteration of the innermost loop = n/10

Total write request = (n^3)
Hence total cache miss = (2n/10) * (n^2)
Total read attempts = 2*(n^3) + n^2

Ikj

Total cache miss for A in one iteration of the innermost loop = 0
Total cache miss for B in one iteration of the innermost loop = n/10
Total cache miss for C in one iteration of the innermost loop = n/10

Total write request = n^3
Total read attempts = 2*(n^3) + n^2
Hence total cache miss = (2n/10) * (n^2)

Kij

Total cache miss for A in one iteration of the innermost loop = 0
Total cache miss for B in one iteration of the innermost loop = n/10
Total cache miss for C in one iteration of the innermost loop = n/10

Total write requests = n^3
Total read attempts = 2*(n^3) + n^2
Hence total cache miss = (2n/10) * (n^2)

Jik

Total cache miss for A in one iteration of the innermost loop = n/10
Total cache miss for B in one iteration of the innermost loop = n/10

Total Write request = n^2
Total read attempts = 2(n^3)
Hence total cache miss = (2n/10) * (n^2)

If n/10 has quotient 0, then just round it of to 1 (no remainder)

Problem 5

Box size = 100
Matrix size = 10000^2
Cache line = 10 double
Cache size = 60 lines

We can easily see that 3*100 < 60 *100

Whole box will fit inside the cache

Total boxes in one iteration = 2 * (n/B)

For ijk

Miss in one cycle = (B^2) / 10 for each block (box A and box B)
Miss in one cycle in box C = 0

Hence total = ((B^2) / 10) * (2n/B)

For jki

Miss in box A = B^2/10
Miss in Box C = B^2/10
Miss in Box B = 0

Hence total = ((B^2) / 10) * (2n/B)

For kji

Miss in box A = B^2/10
Miss in Box C = B^2/10
Miss in Box B = 0

Hence total = ((B^2) / 10) * (2n/B)

For ikj

Miss in box A = 0
Miss in box B = B/10
Miss in box C = B/10

Hence total = ((B^2) / 10) * (2n/B)

For kij

Miss in box A = 0
Miss in box B = B/10
Miss in box C = B/10
Hence total = ((B^2) / 10) * (2n/B)

For jik

Miss in box A = B/10
Miss in Box B = B/10

Miss in box C = 0

Hence total = ((B^2) / 10) * (2n/B)

Problem 6

For non matrix operations we have the following output

```
n=2048
time=186.109614s
ijk
n=2048
time=265.962940s
jki
n=2048
time=153.756097s
jik
n=2048
time=255.656537s
kji
n=2048
time=26.420366s
kij
n=2048
time=26.246056s
Ikj
```

We see very less time for the last 2 (kij and ikj) which is natural as they had the least number of load/store operations

For matrix operations we have the following output for box size 256 * 256

```
n=2048
time=59.165108s
ijk2
n=2048
time=69.910760s
jki2
n=2048
time=82.598219s
jik2
n=2048
time=75.923451s
kji2
n=2048
time=27.186330s
kij2
n=2048
time=36.770345
ikj2
```

We see a lot of improvement obviously.

Now if we change the box size to 512*512 we get increased time duration

```
n=2048
time=89.242343s
ijk2
n=2048
time=143.001556s
jki2
n=2048
time=102.204625s
jik2
n=2048
time=116.107644s
kji2
n=2048
time=47.113896s
kij2
n=2048
time=43.177031s
ikj2
```

Problem 7

For using block size = 2 we get

```
n=2048
time=7.465826s
```

For using block size = 512 we get

```
n=2048
time=3.614114s
```

I did not go beyond this as if we go beyond this we will need to introduce padding as our parameter and padding will increase the computation unnecesarily