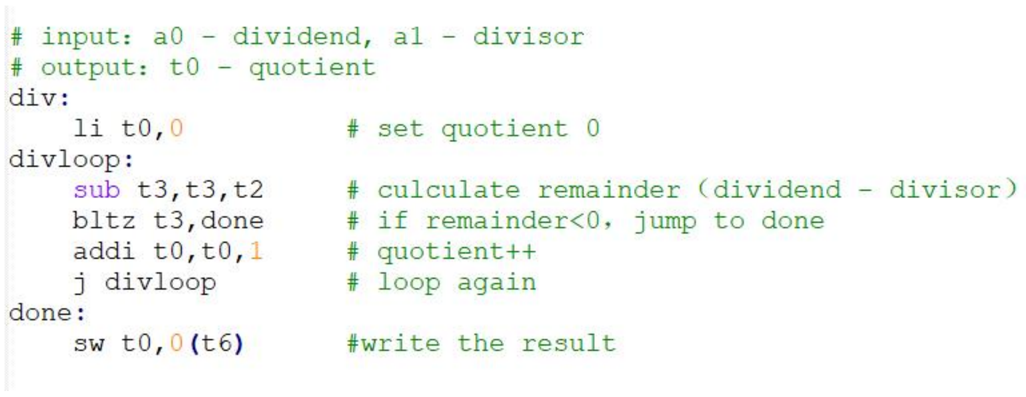
1. Group member

I'm the only member of my group.

2. Implementation of unsigned column average

Use integer truncation to divide unsigned numbers. Integer truncation is a simple division algorithm that calculates the quotient by iteratively subtracting the divisor and truncating the result to an integer part.The code is in the image below.

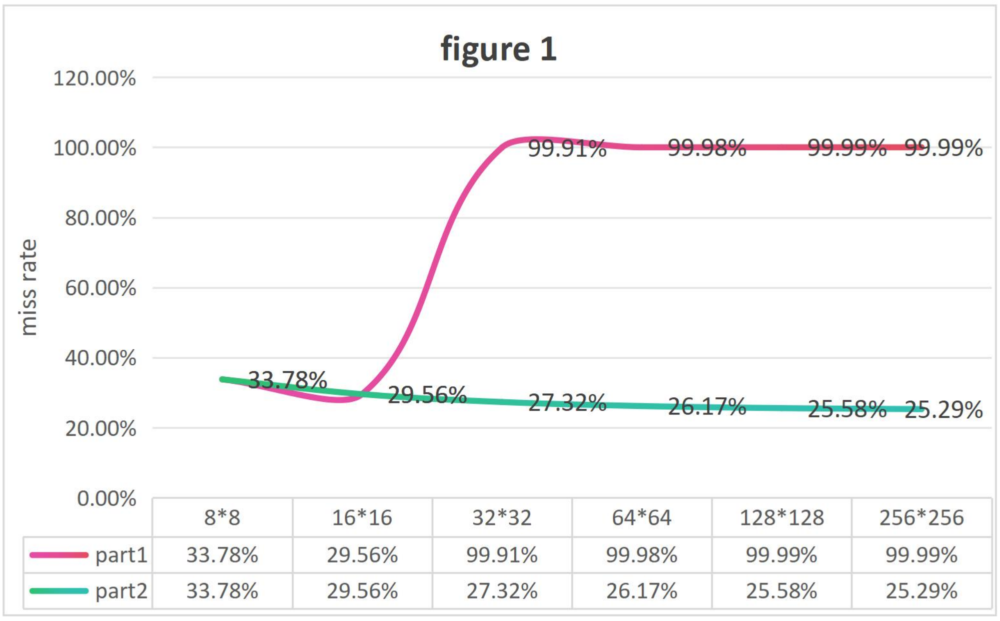


```
# input: a0 - dividend, a1 - divisor
# output: t0 - quotient
div:
    li t0,0             # set quotient 0
divloop:
    sub t3,t3,t2        # culculate remainder (dividend - divisor)
    bltz t3,done        # if remainder<0, jump to done
    addi t0,t0,1        # quotient++
    j divloop           # loop again
done:
    sw t0,0(t6)         #write the result
```

3. Analyses of performance

(1)

The data cache and program cache are all use Random replace policy,and data cache use write through-noallocate write back policy.Through my experiments I found that implementations from Part1 and Part2 have similar number of cycles and program cache miss rate,so their difference is depend on data cache miss rate.



	8*8	16*16	32*32	64*64	128*128	256*256
part1	33.78%	29.56%	99.91%	99.98%	99.99%	99.99%
part2	33.78%	29.56%	27.32%	26.17%	25.58%	25.29%

As we can see from the figure1, as the matrix size increases, the miss rate of the Part1 implementation is basically 100%, which can be costly in terms of time, so I think the Part2 implementation is better.

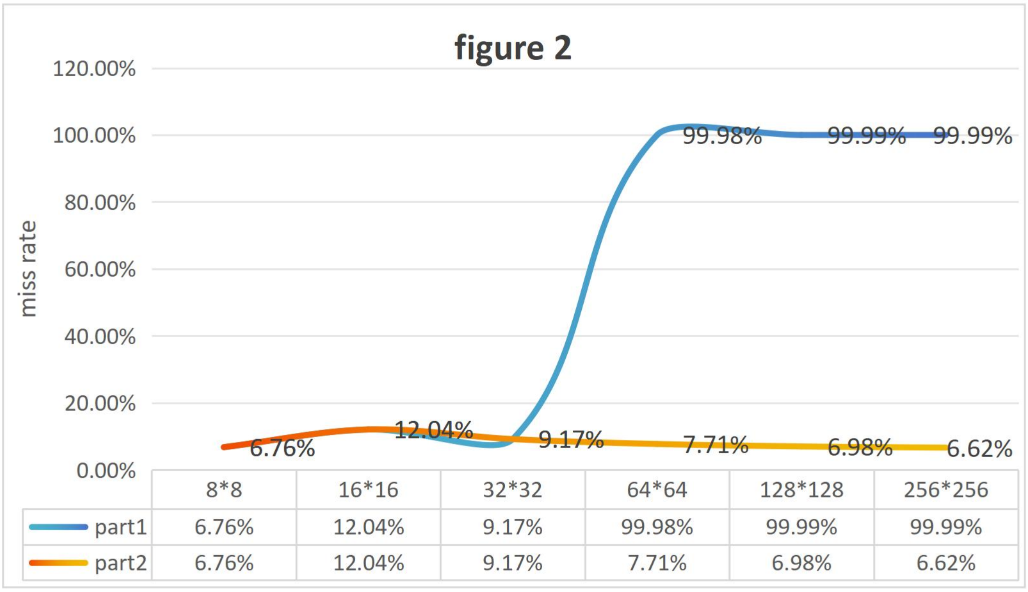
(2)

Use divu instruction just reduce the number of cycles,and it can accelerate program operation,it had same influence in both implementations.But the miss rate in both implementations had no change.So my conclusions don’t change.

(3)

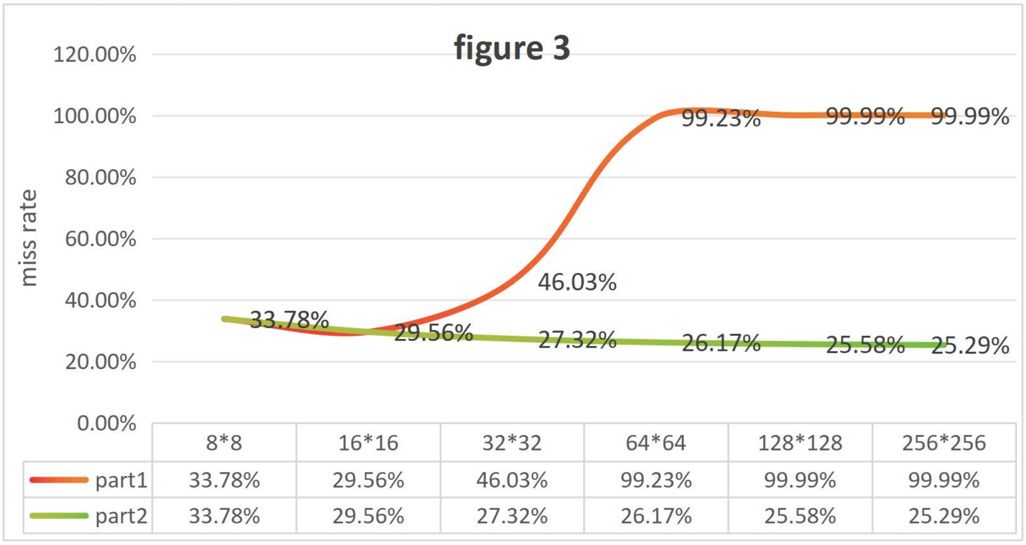
If quadrupling the block size,quadrupling the associativity or doubling the block size and doubling the associativity,all these three situations don’t change the number of cycles and program cache miss rate.So we only consider the data cache miss rate.

Consider quadrupling the block size,we can see the miss rate in figure2.



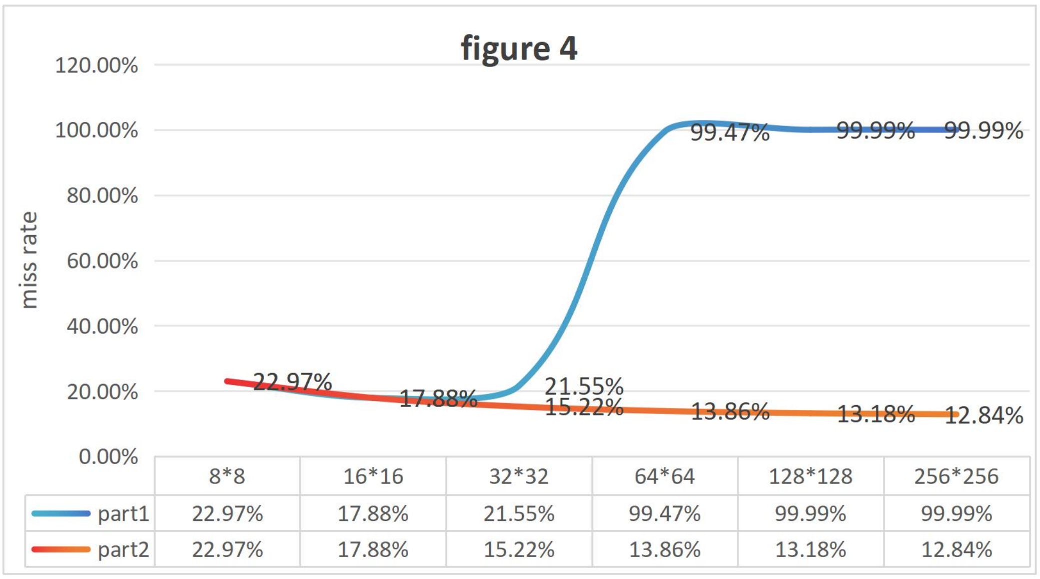
	8*8	16*16	32*32	64*64	128*128	256*256
part1	6.76%	12.04%	9.17%	99.98%	99.99%	99.99%
part2	6.76%	12.04%	9.17%	7.71%	6.98%	6.62%

Consider quadrupling the associativity,we can see the miss rate in figure3.



	8*8	16*16	32*32	64*64	128*128	256*256
part1	33.78%	29.56%	46.03%	99.23%	99.99%	99.99%
part2	33.78%	29.56%	27.32%	26.17%	25.58%	25.29%

Consider doubling the block size and doubling the associativity,we can see the miss rate in figure4.

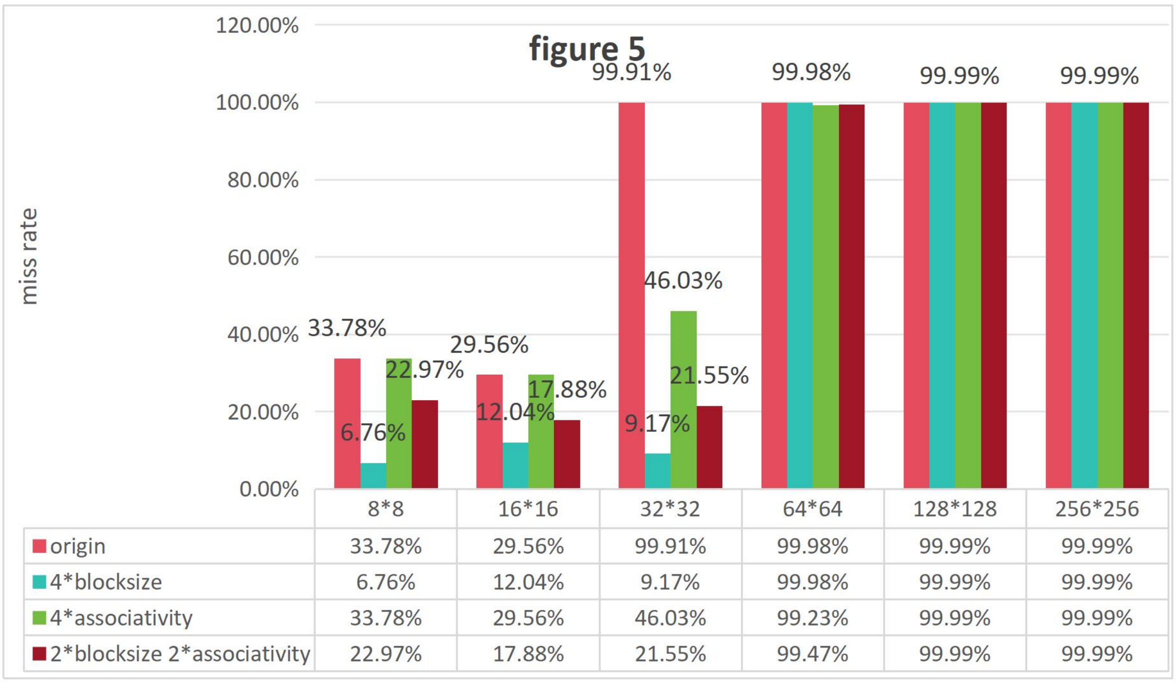


	8*8	16*16	32*32	64*64	128*128	256*256
part1	22.97%	17.88%	21.55%	99.47%	99.99%	99.99%
part2	22.97%	17.88%	15.22%	13.86%	13.18%	12.84%

In all three situations,the miss rate in part2 is less than or equal to that of the part1.So my conclusions don't change,the Part2 implementation is better.

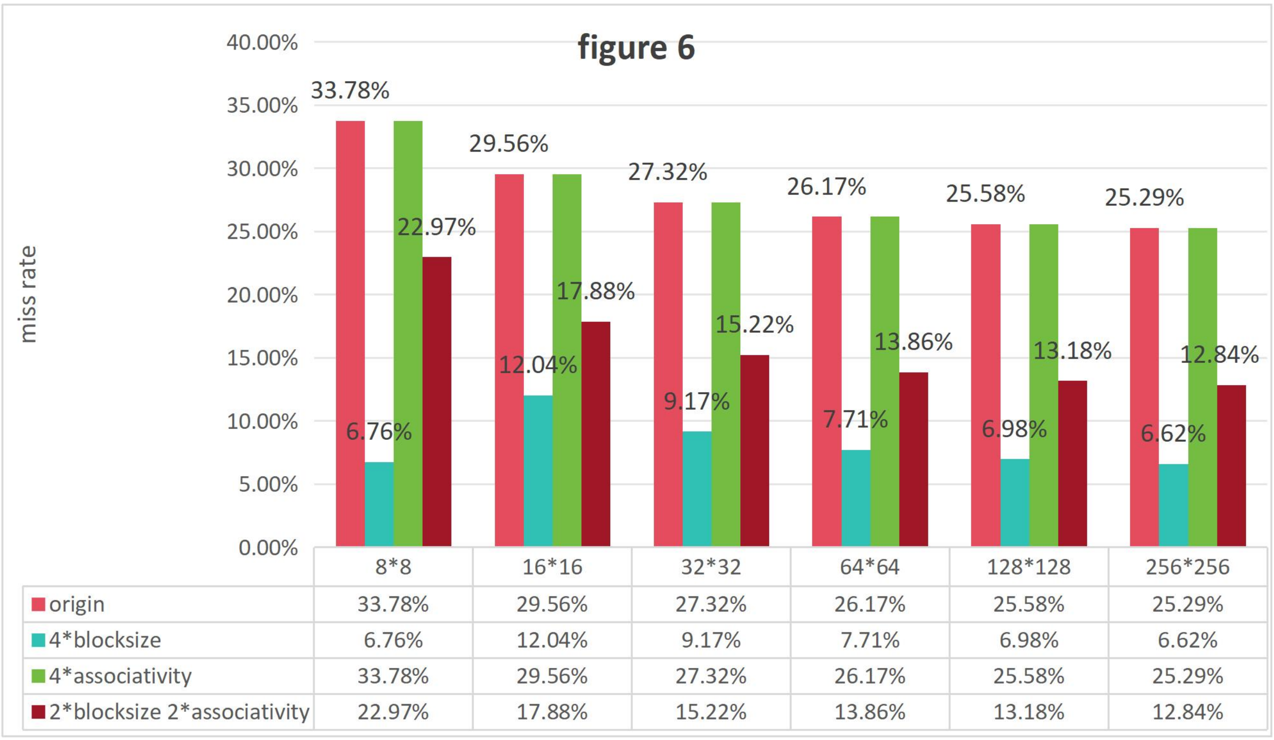
(4)

We can see the difference of miss date among different situations of part1 in figure5.We found all the three solutions to increase the size of the data cache can reduce the miss rate to varying degrees,but only useful in small matrix size,and quadrupling the block size can get the best results.



	8*8	16*16	32*32	64*64	128*128	256*256
🟥 origin	33.78%	29.56%	99.91%	99.98%	99.99%	99.99%
🟦 4*blocksize	6.76%	12.04%	9.17%	99.98%	99.99%	99.99%
🟩 4*associativity	33.78%	29.56%	46.03%	99.23%	99.99%	99.99%
🟥 2*blocksize 2*associativity	22.97%	17.88%	21.55%	99.47%	99.99%	99.99%

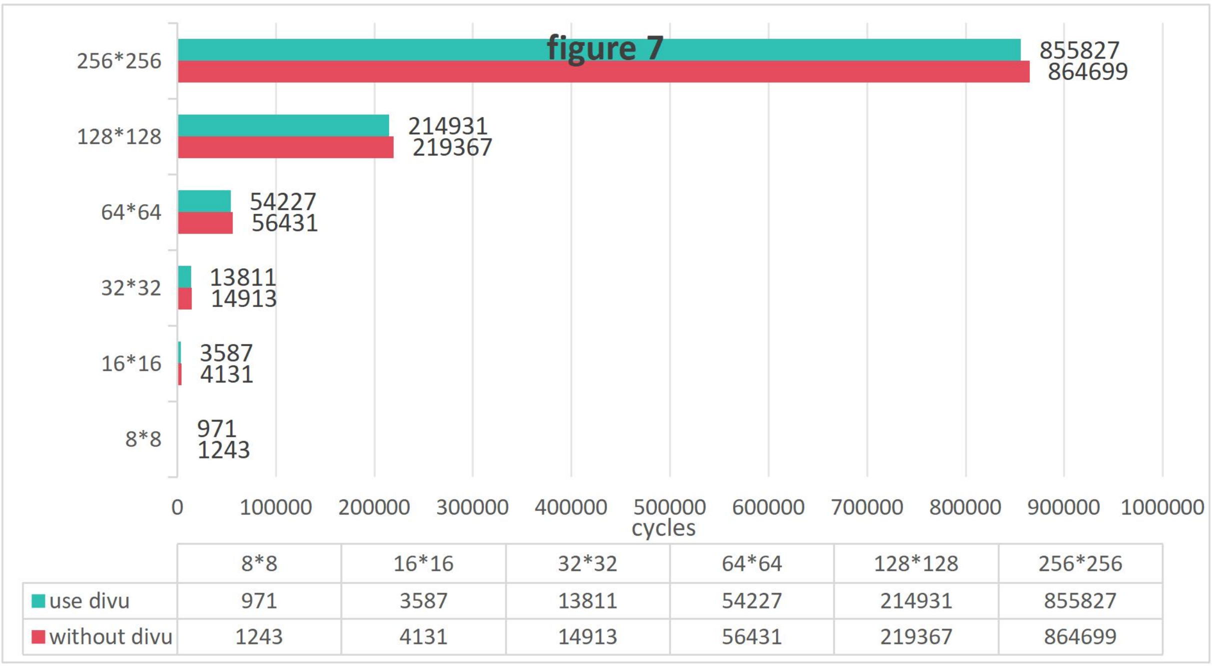
We can see the difference of miss date among different situations of part2 in figure6.We found except for quadrupling the associativity,other two situations both can reduce the cache rate in all matrix size.After the above analysis, I think the block size helps more than associativity.



	8*8	16*16	32*32	64*64	128*128	256*256
🟥 origin	33.78%	29.56%	27.32%	26.17%	25.58%	25.29%
🟦 4*blocksize	6.76%	12.04%	9.17%	7.71%	6.98%	6.62%
🟩 4*associativity	33.78%	29.56%	27.32%	26.17%	25.58%	25.29%
🟥 2*blocksize 2*associativity	22.97%	17.88%	15.22%	13.86%	13.18%	12.84%

4.recommendation

Because the cycles in part1 and part2 almost exactly the same,and change the data cache size will not change the cycles.So we only consider the cycles change in part1.We can see it in figure7.



	8*8	16*16	32*32	64*64	128*128	256*256
🟦 use divu	971	3587	13811	54227	214931	855827
🟥 without divu	1243	4131	14913	56431	219367	864699

We can see that the use of the divu instruction has a very limited effect on the reduction of the number of cycles, and this effect is essentially negligible when matrix size is big.In contrast, increasing the data cache size has a more significant impact.So if i had to choose between implementing division in hardware and quadrupling the size of the data cache,i would choose quadrupling the size of the data cache,and it's best to increase the block size.

5.other advice

Using multiple levels of data caches or improving memory bandwidth can also helpful to get better results.