Gaussian elimination:

Gaussian elimination is the concept that is used to solve a system of linear equations. In this method the equations are substituted and the matrix is then transformed into row major form, this is also called upper triangular matrix, In this part of converting the matrix in row major form, we have some basic steps, first is to convert the matrix into the augmented matrix (adding an extra column at the right side of the matrix, this extra column is usually the right-side constant values that are being passed) and after converting the matrix into that form, we perform some operations so that we can make the elements under the diagonal of the matrix as zero. In the below figure (Source Wikipedia), we can see how the system of linear equations are taken and then the converted to

System of equations	Row operations	Augmented matrix
$\begin{aligned} 2x + y - z &= 8 \\ -3x - y + 2z &= -11 \\ -2x + y + 2z &= -3 \end{aligned}$		$\left[\begin{array}{ccc\|c} 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{array}\right]$
$\begin{aligned} 2x + y - z &= 8 \\ \tfrac{1}{2}y + \tfrac{1}{2}z &= 1 \\ 2y + z &= 5 \end{aligned}$	$\begin{aligned} L_2 + \tfrac{3}{2}L_1 &\to L_2 \\ L_3 + L_1 &\to L_3 \end{aligned}$	$\left[\begin{array}{ccc\|c} 2 & 1 & -1 & 8 \\ 0 & \frac{1}{2} & \frac{1}{2} & 1 \\ 0 & 2 & 1 & 5 \end{array}\right]$
$\begin{aligned} 2x + y - z &= 8 \\ \tfrac{1}{2}y + \tfrac{1}{2}z &= 1 \\ -z &= 1 \end{aligned}$	$L_3 + -4L_2 \to L_3$	$\left[\begin{array}{ccc\|c} 2 & 1 & -1 & 8 \\ 0 & \frac{1}{2} & \frac{1}{2} & 1 \\ 0 & 0 & -1 & 1 \end{array}\right]$
The matrix is now in echelon form (also called triangular form)		
$\begin{aligned} 2x + y &= 7 \\ \tfrac{1}{2}y &= \tfrac{3}{2} \\ -z &= 1 \end{aligned}$	$\begin{aligned} L_2 + \tfrac{1}{2}L_3 &\to L_2 \\ L_1 - L_3 &\to L_1 \end{aligned}$	$\left[\begin{array}{ccc\|c} 2 & 1 & 0 & 7 \\ 0 & \frac{1}{2} & 0 & \frac{3}{2} \\ 0 & 0 & -1 & 1 \end{array}\right]$
$\begin{aligned} 2x + y &= 7 \\ y &= 3 \\ z &= -1 \end{aligned}$	$\begin{aligned} 2L_2 &\to L_2 \\ -L_3 &\to L_3 \end{aligned}$	$\left[\begin{array}{ccc\|c} 2 & 1 & 0 & 7 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & -1 \end{array}\right]$
$\begin{aligned} x &= 2 \\ y &= 3 \\ z &= -1 \end{aligned}$	$\begin{aligned} L_1 - L_2 &\to L_1 \\ \tfrac{1}{2}L_1 &\to L_1 \end{aligned}$	$\left[\begin{array}{ccc\|c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & -1 \end{array}\right]$

augmented matrix and then the operations are performed to achieve the required output.

Figure 1. Gaussian Elimination

Parallelization strategies:

I have used two strategies. In the first one I normally parallelized the compute gauss function with the pthreads library and then in the second technique I used the method of matrix blocking, in this method the matrix is divided into blocks of size n and every block is assigned to a thread.

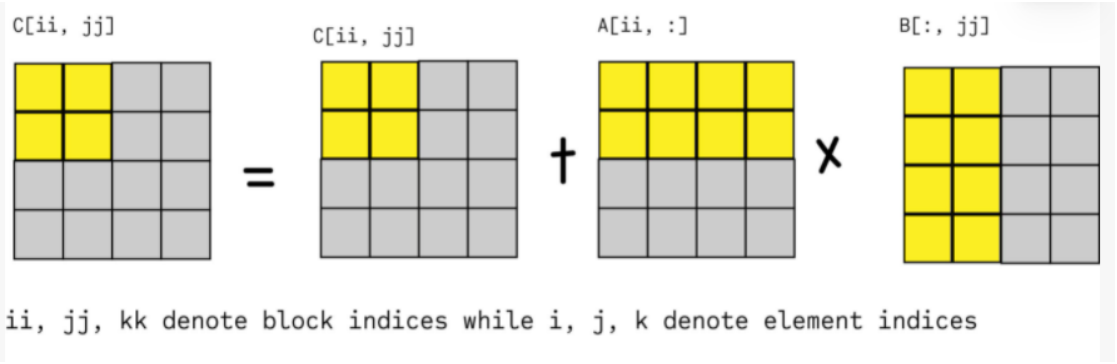

Figure 2. Matrix blocking visualization.

To evaluate that my strategies are working, I used the check(provided values) that was given in the assignment by assigning verify=1 and the values that I have received were as mentioned in the assignment (from -0.5 to 0.5) and while implementing the solution there was time, when I did a mistake and the values were very different compared to the given values(their limit was increased).

To asses the mistakes in the external elimination file, I have checked the error rates and compared them with the sequential implementation and found out that it was similar for the given matrices and

Apart from this I have checked the approaches that I have made on the different matrices which I have listed below and I have also checked the correctness of the program on different number of threads and matrix size.

Evaluation of method 1 and Method 2:

Environment of program execution:

```
Thread(s) per core:              2
Core(s) per socket:              14
Socket(s):                       2
NUMA node(s):                    2
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           63
Model name:                      Intel(R) Xeon(R) CPU E5-2695 v3 @ 2.30GHz
Stepping:                        2
CPU MHz:                         1319.430
CPU max MHz:                     3300.0000
CPU min MHz:                     1200.0000
BogoMIPS:                        4599.80
Virtualization:                  VT-x
L1d cache:                       896 KiB
L1i cache:                       896 KiB
L2 cache:                        7 MiB
L3 cache:                        70 MiB
NUMA node0 CPU(s):               0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,3
                                 4,36,38,40,42,44,46,48,50,52,54
NUMA node1 CPU(s):               1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,3
                                 5,37,39,41,43,45,47,49,51,53,55
Vulnerability Itlb multihit:     KVM: Mitigation: VMX disabled
Vulnerability L1tf:              Mitigation; PTE Inversion; VMX conditional cach
                                 e flushes, SMT vulnerable
Vulnerability Mds:               Mitigation; Clear CPU buffers; SMT vulnerable
Vulnerability Meltdown:          Mitigation; PTI
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled v
                                 ia prctl
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user
                                  pointer sanitization
Vulnerability Spectre v2:        Mitigation; Full generic retpoline, IBPB condit
                                 ional, IBRS_FW, STIBP conditional, RSB filling
Vulnerability Srbds:             Not affected
Vulnerability Tsx async abort:   Not affected
Flags:                           fpu vme de pse tsc msr pae mce cx8 apic sep mtr
                                 r pge mca cmov pat pse36 clflush dts acpi mmx f
                                 xsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rd
                                 tscp lm constant_tsc arch_perfmon pebs bts rep_
                                 good nopl xtopology nonstop_tsc cpuid aperfmper
                                 f pni pclmulqdq dtes64 monitor ds_cpl vmx smx e
                                 st tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid dca s
                                 se4_1 sse4_2 x2apic movbe popcnt tsc_deadline_t
                                 imer aes xsave avx f16c rdrand lahf_lm abm cpui
                                 d_fault epb invpcid_single pti ssbd ibrs ibpb s
                                 tibp tpr_shadow vnmi flexpriority ept vpid ept_
                                 ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms
                                  invpcid cqm xsaveopt cqm_llc cqm_occup_llc dth
                                 erm ida arat pln pts md_clear flush_l1d
```

Fig 3. Environment of execution

Environment:

Number of threads:56

Sockets: 2

Model: Intel(R) Xeon(R) CPU E5-2695 v3 @2.30 GHz

Gaussain_internal_input **Approach 1 normal parallelization**

Number of threads	2048	1024	512	256	128
1	16.013	2.274	0.395	0.071	0.0165
4	5.1834	0.744	0.187	0.0519	0.0231

16	3.0522	0.683	0.424	0.1482	0.0792
32	2.55310	1.290	0.728	0.4330	0.1509
56	3.9633	2.28	0.905	0.440	0.2419

Guassian_internal input **Approach 2 parallelization using blocking**

Number of threads	2048	1024	512	256	128
1	17.93	2.490	0.409	0.0818	0.018
4	6.55	1.1122	0.1918	0.05199	0.217
16	4.557	0.710	0.281	0.1597	0.822
32	3.845	1.07	0.504	0.421	0.149
56	4.744	2.99	0.932	0.443	0.242

Guassian_External_Input **Approach 1 normal parallelization**:

Number of threads	Matrix-jpwh_991	Matrix-jpwh_991 Error rate	Matrix orgseg	Matrix orgseg Error rate
1	2.08	4.814e -15	19.861008	2.221e-10
16	0.68694	4.814e -15	3.6890	2.221e-10
32	1.079	4.814e -15	3.0633	2.221e-10
56	1.632	4.814e -15	4.450767	2.221e-10

Guassian_External_Input **Approach 2 parallelization using blocking**:

Number of threads	jpwh_991	jpwh_991 Error Rate	Orgseg_1	Orgseg_1 Error rate
1	2.1913	4.814e-15	21.066201	2.221e-10
16	0.07088	4.814e-15	3.787	2.221e-10
32	0.9784	4.814e-15	3.782	2.221e-10
56	1.93339	4.814e-15	4.9830	2.221e-10

gaussian_internal_input **Sequential**:

Matrix size	Application Time
128	Application time: 0.003989 Secs
256	Application time: 0.029544 Secs
512	Application time: 0.164806 Secs
1024	Application time: 1.615068 Secs
2048	Application time: 10.078325 Secs

gaussian_external_input: **Sequential**:

Matrix name	Time	Error
jpwh_991	Time: 1.982013 seconds	Error: 4.814101e-15
orsreg_1	Time: 12.012935 seconds	Error: 2.221803e-10

After evaluating all the results, I have observed that the performance of the gaussian elimination (for matrix with big sizes) is best near 32 threads after comparing its running time with other threads on the same number if the matrix size, I assessed the time taken from 2048 matrix to come to this conclusion. We can see from the above information that when the matrix size is less and if we increase the number of processors, at this situation the time consumption is more as the processor is using a lot of time to allocate the threads.

In the two approaches that I have done, I found out that parallelizing the matrix normally takes almost similar or better time compared to the matrix blocking method, this is due to the reason of cache hits and miss, because even if we do matrix blocking there will be many elements that will be read but it will necessarily not be computed in that particular attempt. Whereas doing the normal parallelizing of matrix using row wise computation, we can reduce the number of cache misses.

Comparing the above two approaches with the sequential time, I can see that the time difference is almost 6 to 7 seconds, which is a huge amount, here I am taking the time of 32 threads and the sequential program. After assessing this I ran the gaussian elimination program on the two matrices and the best output difference [between JPwh_991[Approach 1] and sequential] that I have got from both of them was 1.2 seconds this was when the number of threads were 16. Apart from this the difference between the next matrix was [Orgseg_1[Approach 1] and sequential] almost 9 seconds. Whereas the error rate for every particular matrix was constant even after increasing the matrix size.