SiWiR-I

Assignment 2(Elliptic PDE with Open MP)

2. Theoretical questions:

The PDE:

$-\Delta u(x, y) + k^2 u(x, y) = f(x, y) \qquad (x, y) \in \Omega$

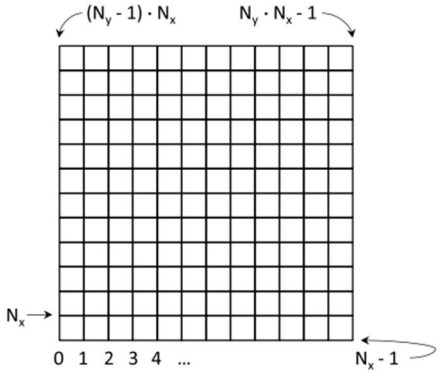
2.1Discretiztaion:

$$\frac{-u_{(x-h,y)}+2u_{(x,y)}-u_{(x+h,y)}}{h_x^2} + \frac{-u_{(x,y-h)}+2u_{(x,y)}-u_{(x,y+h)}}{h_y^2} + k^2 u_{(x,y)} = f_{(x,y)}$$

on rearranging terms:

$$\frac{-u_{(x-h,y)}-u_{(x+h,y)}}{h_x^2} + \frac{-u_{(x,y-h)}-u_{(x,y+h)}}{h_y^2} + \left(k^2 + \frac{2}{h_x^2} + \frac{2}{h_y^2}\right) u_{(x,y)} = f_{(x,y)}$$

The grid and chosen grid numbering:



As per the above grid numbering the size of the computing grid is $(N_x).(N_y)$ i.e (computational domain +boundary).

The boundary conditions would be applied at the outer edges .Therefore the matrix formation would start from point $(N_x + 1)$ to point $(N_y - 1).N_x - 2$

Matrix A:

Let $\left(k^2 + \frac{2}{h_x^2} + \frac{2}{h_y^2}\right)$=a; $\frac{-1}{h_x^2}$=b; and $\frac{-1}{h_y^2}$=c (evident that a > 2(|b| + |c|))

$$\begin{pmatrix} a & b & 0 & 0 & 0 & \dots & c & 0 & 0 & 0 & \dots & 0 \\ b & a & b & 0 & 0 & 0 & \dots & c & 0 & 0 & 0 & 0 \\ 0 & b & a & b & 0 & 0 & 0 & \dots & c & 0 & 0 & 0 \\ 0 & 0 & b & a & b & 0 & 0 & 0 & \dots & c & 0 & 0 \\ & & & & & & \vdots & & & & & \\ c & 0 & 0 & \dots & 0 & 0 & a & b & 0 & 0 & \dots & c \\ & & & & & & \vdots & & & & & \\ 0 & 0 & 0 & \dots & 0 & 0 & c & 0 & 0 & \dots & b & a \end{pmatrix}$$

Considering case nx=32, ny=32 :

we have a=2599.478, b=-256, c=-1024

Thus matrix A clearly satisfies the condition: $|a_{ii}| \geq \sum_{j \neq i} |a_{ij}| \quad for\ all\ i$

Hence, A is strictly diagonal dominant.

2.2 Jacobi Convergence:

The condition for Jacobi convergence is:

$$\rho(D^{-1}.R) < 1$$

where ρ is the spectral radius, D is the diagonal matrix(diagonal part of matrix A) and R =L+U;(where L and U are strictly the upper and lower part of the matrix A).

$$D=\begin{pmatrix} a & 0 & 0 & 0 & 0 & 0 \dots . 0 \\ 0 & a & 0 & 0 & 0 & 0 & 0 \dots . . \\ 0 & 0 & a & 0 & 0 & 0 & 0 \dots . . \\ & & & . & & & \\ & & & . & & & \\ & & & . & & & \\ \dots . . 0 & 0 & 0 & 0 & 0 & 0 & a \end{pmatrix} \quad R=\begin{pmatrix} 0 & b & 0 & 0 & 0 \dots . . c & 0 \dots . . \\ b & 0 & b & 0 & 0 \dots . . c & 0 \dots \\ 0 & b & 0 & b & 0 & 0 \dots . . c & 0 \\ & & & . & & & \\ & & & . & & & \\ & & & . & & & \\ 0 \dots . . c & 0 \dots . . 0 & 0 & b & 0 \end{pmatrix}$$

$$D^{-1}.R = 0 < 1$$

Additionally, if the matrix *A* is strictly or irreducibly diagonally dominant then the method is guaranteed to converge. In our case we have diagonally dominant matrix.

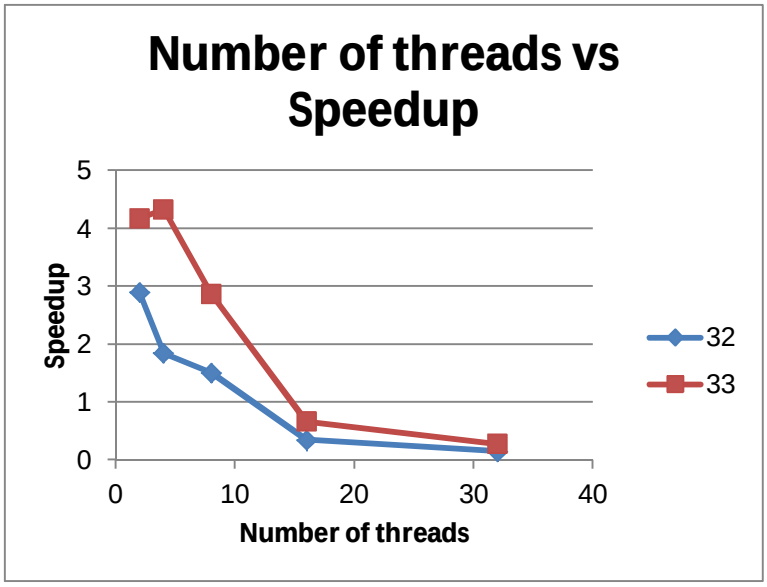
Calculating: case for 1 unknowns (i.e. 3x3 grid) spectral radius is 0
case for 4 unknowns (i.e. 4x4 grid) spectral radius is 0.4
case for 9 unknowns (i.e. 5x5 grid) spectral radius is 0.69

As the grid size increases the spectral radius increases but it will never reach 1.
Thus we can say that convergence factor is near to but <1, so it can be observed that method converges slowly as we increase grid size.

2.3 Performance graphs:

As we can see in data that code performs bad in parallelization for 32x32 and 33x33 because creating threads increases the overhead which is not compensated by increase in performance after 2 threads, especially after 8 threads when the program is run on different domains.

Threads		
Time	32	33
1	0.004656	0.006834
2	0.00161	0.001638
4	0.002527	0.00158
8	0.003096	0.002383
16	0.013476	0.010385
32	0.032625	0.025352
Speedup	32	33
2	2.892771	4.170703
4	1.842323	4.324171
8	1.503803	2.867857
16	0.345505	0.658032
32	0.142714	0.269549



While computing time for all threads with every size, the optimum choice of Processors (and hence ccNUMA domains) is done through GOMP_CPU_AFFINITY, where care was taken to divide load among all 4 domains. The order is show in Appendix 1.

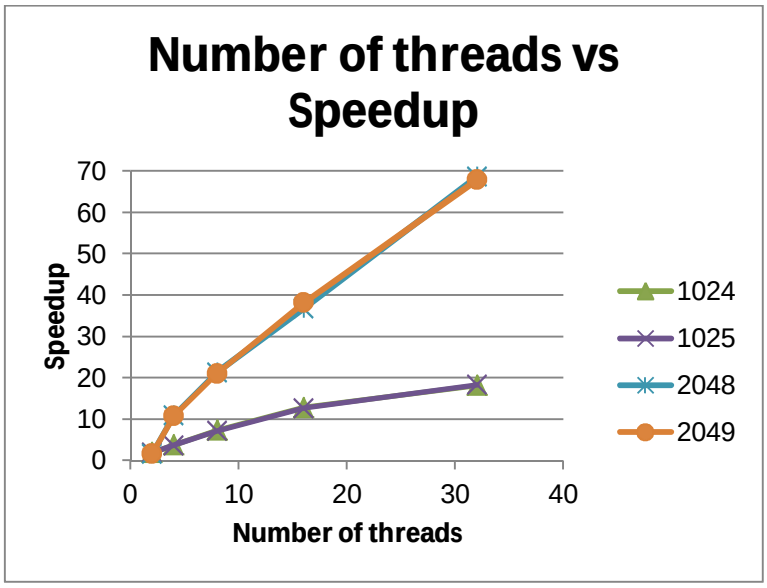
When running with a single size, the value for GOMP_CPU_AFFINITY and OMP_NUM_THREADS is chosen depending on the size, which has been optimized. This is noted in Appendix 2.

Obviously, the purposes of the two different executions will also result in different outputs, though the times will be same.

Apart from this, with respect to Open MP, firstprivate, schedule (static, chunk) etc. were extensively used

As seen in plots for larger size parallelization actually pays off and we get almost linear speed up of performance. In sizes 1024 and 1025 we are using all the threads but the bandwidth is not fully used while in 2048 and 2049 we are utilizing full bandwidth resulting in much more better speedup.
The graphs for 1024 & 1025 and 2048 & 2049 are overlapping as we have used padding of 1 if given an odd input size. So performance remains same for both sizes.

Threads				
Time	1024	1025	2048	2049
1	1.14048	1.00946	11.3812	11.2794
2	0.585823	0.514684	6.59994	6.49896
4	0.298583	0.268239	1.03671	1.03922
8	0.154713	0.13975	0.533044	0.535226
16	0.088672	0.07941	0.308489	0.295189
32	0.062574	0.054869	0.165718	0.166145
Speedup	1024	1025	2048	2049
2	1.9468	1.96132	1.72444	1.73557
4	3.819641	3.763286	10.97819	10.85372
8	7.371585	7.223327	21.35133	21.07409
16	12.86181	12.71207	36.89337	38.21077
32	18.2261	18.3976	68.67812	67.88889



Appendix 1: (excerpt from sc_all.sh)

```
i=1
while [ $i -lt 33 ]
do
        echo "============================================="
    export OMP_NUM_THREADS=$i
        if [ $i -eq 1 ]; then
            export GOMP_CPU_AFFINITY=0-1,8-8,16-16,24-24
        elif [ $i -eq 4 ]; then
                if [ $1 -lt 50 ]; then
                export GOMP_CPU_AFFINITY=0-3,8-8,16-16,24-24
                else
                    export GOMP_CPU_AFFINITY=0-0,8-8,16-16,24-24
                fi
    elif [ $i -eq 8 ]; then
         if [ $1 -lt 50 ]; then
                    export GOMP_CPU_AFFINITY=0-7,8-8,16-16,24-24
         else
              export GOMP_CPU_AFFINITY=0-1,8-9,16-17,24-25
         fi
    elif [ $i -eq 16 ] ; then
         if [ $1 -lt 50 ]; then
              export GOMP_CPU_AFFINITY=0-7,8-15,16-16,24-24
         else
              export GOMP_CPU_AFFINITY=0-4,8-11,16-19,24-27
         fi
    elif [ $i -eq 32 ]; then
         export GOMP_CPU_AFFINITY=0-31
        fi
        echo "OMP_NUM_THREADS: "   $OMP_NUM_THREADS
        echo "GOMP_CPU_AFFINITY: " $GOMP_CPU_AFFINITY
        echo "./rbgs2 $1 $2 $3"
        ./rbgs1 $1 $2 $3
        i=$[$i*2]
    echo "============================================="
done
```

Appendix2: (excerpt from sc.sh)

```
("$1" = nx  )
if [ "$1" -gt 510 ]; then
        export OMP_NUM_THREADS=32
        export GOMP_CPU_AFFINITY=0-31
        echo "Choice 1"
elif [ "$1" -gt 70 ]; then
        export OMP_NUM_THREADS=16
        export GOMP_CPU_AFFINITY=0-3,8-11,16-19,24-27
        echo "Choice 2"
else
        export OMP_NUM_THREADS=4
        export GOMP_CPU_AFFINITY=0,8,16,24
        echo "Choice 3"
fi

echo "================================================"
echo "OMP_NUM_THREADS: "   $OMP_NUM_THREADS
echo "GOMP_CPU_AFFINITY: " $GOMP_CPU_AFFINITY
echo "./rbgs1" $1 $2 $3
./rbgs1 $1 $2 $3
echo "================================================"
echo "Done"
```