Full OpenMP implementation of Jacobi with dominant row

1 Introduction

Let

$$Ax = a \tag{1}$$

be our system of linear equations, with $A = (a_{ij})_{i,j\in\{1,...,m\}}$, $a = (a_1, ..., a_m)$ and $x = (x_1, ..., x_m)^T$.

Suppose $a_{ii} \neq 0$ for any $i \in \{1, ..., m\}$ then we can note $D = \begin{pmatrix} a_{11} & 0 \\ 0 & a_{mm} \end{pmatrix}$ and $D^{-1} = \begin{pmatrix} 1/a_{11} & 0 \\ 0 & 1/a_{mm} \end{pmatrix}$.

The system (1) could be transform in $D^{-1}Ax = D^{-1}a$ or

$$\left(I - \left(\underbrace{I - D^{-1}A}_{B}\right)\right) x = \underbrace{D^{-1}a}_{b}$$

which is equivalent with

$$(I - B)x = b \tag{2}$$

Theorem 1 *If*

$$|a_{ii}| > \sum_{j=1, j\neq i}^{m} |a_{ij}|, \forall i \in \{1, ..., m\} \tag{3}$$

then the system (2) has the unique solution z *and* $\forall x^{(0)} \in R^m$ *the string* $(x^{(n)})_{n\in N}$ *with* $x^{(n+1)} = Bx^{(n)} + b$ *convert to* z *and it takes palaces the following relations:*

$$\left\|x^{(n)} - z\right\|_\infty \leq \frac{q}{1-q} \left\|x^{(n)} - x^{(n-1)}\right\|_\infty \leq \frac{q^n}{1-q} \left\|x^{(1)} - x^{(0)}\right\|_\infty \tag{4}$$

where

$$q = \max_{1\leq i\leq m} \sum_{j=1, j\neq i}^{m} \left|\frac{a_{ij}}{a_{ii}}\right|$$

2 Implementation in C

Where in conformity with system (1)

- mat is matrix A of coefficients
- ty is the free vector a

- tx is the solution x of the system
- err is the error
- thread is the number of processors.

```
void jacobi_omp(double **mat,double *ty,double *tx,int dim,double err,int thread)
{
double *xn_1;
long i,j;
int th;
double q,sum,temp;
double *sum_p;
    xn_1=(double *)calloc(dim,sizeof(double));
    sum_p=(double *)calloc(thread,sizeof(double));
//JACOBI
    q=0.0;
    omp_set_num_threads(thread);
    #pragma omp parallel private(th,i)
    {
        #pragma omp for
            for(i=0;i<dim;i++)
                tx[i]=ty[i]/mat[i][i];
        //compute q
        #pragma omp for reduction(+:q)
            for(i=1;i<dim;i++)
                q+=fabs(mat[0][i]/mat[0][0]);
        th=omp_get_thread_num();
        sum_p[th]=q;
        #pragma omp for private(temp,j)
        for(i=1;i<dim;i++)
        {
            temp=0.0;
            for(j=0;j<dim;j++)
                if(i!=j) temp+=fabs(mat[i][j]/mat[i][i]);
            if(sum_p[th]<temp) sum_p[th]=temp;
        }
        #pragma omp single
        {
            q=sum_p[0];
            for(i=1;i<thread;i++)
                if(q<sum_p[i])
                    q=sum_p[i];
        }
        sum_p[th]=fabs(ty[th]/mat[th][th]);
        for(i=th+thread;i<dim;i=i+thread)
        {
            if(sum_p[th]<fabs(ty[i]/mat[i][i]))
                    sum_p[th]=fabs(ty[i]/mat[i][i]);
```

```
            }
            #pragma omp barrier
            #pragma omp single
            {
                  sum=sum_p[0];
                  for(i=1;i<thread;i++)
                    if(sum<sum_p[i]) sum=sum_p[i];
                  sum=sum*q/(1-q);
            }
            while(fabs(sum)>err)
            {
                  #pragma omp for
                        for(i=0;i<dim;i++) xn_1[i]=tx[i];
                  #pragma omp for private(j)
                        for(i=0;i<dim;i++)
                        {
                              tx[i]=ty[i]/mat[i][i];
                              for(j=0;j<dim;j++)
                                if(j!=i) tx[i]-=mat[i][j]/mat[i][i]*xn_1[j];
                        }
                  sum_p[th]=fabs(tx[th]-xn_1[th]);
                  for(i=th+thread;i<dim;i=i+thread)
                        if(sum_p[th]<fabs(tx[i]-xn_1[i])) sum_p[th]=fabs(tx[i]-xn_1[i]);
                  #pragma omp barrier
                  #pragma omp single
                  {
                        sum=sum_p[0];
                        for(i=1;i<thread;i++)
                              if(sum<sum_p[i]) sum=sum_p[i];
                        sum=sum*q/(1-q);
                  }
            }
      }
      free(xn_1);
}
```

Because I want to have a random solution but also to can verified the solution and the system to be compatible with systems solved by Jacobi with row dominant I use the following generator which will assure me that I have the following solution rez=(1,2,...,m) for a m dimensional problem.

```
      for(i=0;i<dim;i++) rez[i]=(double)i+1;
      for(i=0;i<dim;i++)
            for(j=0;j<dim;j++) if(i!=j)
                  {
                        mat[i][j]=20000*rand()/(double)RAND_MAX;
                        if((rand()/(double)RAND_MAX)<0.5) mat[i][j]=-mat[i][j];
                  }
      for(i=0;i<dim;i++)
      {
            temp=0.0;
```

```
        for(j=0;j<dim;j++) if(j!=i) temp+=fabs(mat[i][j]);
        mat[i][i]=temp+(20000+temp)*rand()/(double)RAND_MAX+0.00001;
        if((rand()/(double)RAND_MAX)<0.5) mat[i][i]=-mat[i][i];
    }
    for(i=0;i<dim;i++)
    {
        y[i]=0.0; x[i]=0.0;
        for(j=0;j<dim;j++) y[i]+=mat[i][j]*rez[j];
    }
```

3 Results

I have compile the parallel program with two openMP compilers: Omni 1.6 and Intel C Compiler 8.0 for LINUX and the serial with gcc and Intel C Compiler 8.0 for LINUX both with maximum optimization "-O3" and for Intel C Compiler I've put also "-mcpu=pentiumpro -tpp6" for maximum optimization.

The executable were run on a dual pentium II at 500MHz with 256MB RAM and with LINUX Fedora Core 1.

The following results were made for a average or 10 runs for serial and parallel programs and with red is plotted the results from ICC and with blue the results from Omni.