UYBHM Yaz Çalıştayı
14 – 24 Haziran 2011

Parallel Programming Application: Partial Differential Equations – Laplace Equation

Şenol Pişkin (senol@be.itu.edu.tr)

PDES in Life

- Laplace Equation
- Poissons EQuation
- Heat Equation
- Wave Equation
- Euler Equation
- Burgers Equation
- Helmholtz Equation
- Navier-Stokes Equation

PDES in Life (cont)

- Laplace Equation (steady state heat equation)
- Poissons EQuation (steady state heat equation with source)
- Heat Equation (unsteady state heat equation)
- Wave Equation (unsteady state wave equation)
- Euler Equation (Inviscid fluid flow equation)
- Burgers Equation (Wave process: acoustics, hydrodynamics)
- Helmholtz Equation(Wave, acoustics)
- Navier-Stokes Equation (Fluid flow equation)

PDES in Life (cont)

$$\nabla^2 u = 0 \qquad \nabla^2 u = f \qquad \nabla^2 u + k^2 u = 0$$

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u \qquad \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$$

$$\frac{\partial u}{\partial t} = k\nabla^2 u \qquad \rho\left(\frac{\partial}{\partial t} + \vec{u}\cdot\vec{\nabla}\right)\cdot\vec{u} + \vec{\nabla}p = 0$$

$$\rho\left(\frac{\partial}{\partial t} + \vec{u}\cdot\vec{\nabla}\right)\cdot\vec{u} = -\vec{\nabla}p + \mu\nabla^2\vec{u} + \vec{f}$$

PDES in Life (cont)

$\nabla^2 u = 0$ LE

$\nabla^2 u = f$ PE

$\nabla^2 u + k^2 u = 0$ HE

$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u$ WE

$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$ BE

$\frac{\partial u}{\partial t} = k\nabla^2 u$ HE

$\rho\left(\frac{\partial}{\partial t} + \vec{u}\cdot\vec{\nabla}\right)\cdot\vec{u} + \vec{\nabla}p = 0$ EE

$\rho\left(\frac{\partial}{\partial t} + \vec{u}\cdot\vec{\nabla}\right)\cdot\vec{u} = -\vec{\nabla}p + \mu\nabla^2\vec{u} + \vec{f}$ NSE

Laplaces Equation

- Laplaces equation is a mathematical model for heat flow and viscous stresses.
- It is a good test case for approaches to numerical programming.
- Scalar Laplaces equation is:

$$\nabla^2 u = 0$$

where u is a scalar variable.

Laplaces Equation (cont)

- In our case 2D Laplaces equation is used.
- It is a 2D unsteady heat equation.
- So the Laplaces equation becomes:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

Laplaces Equation (cont)

- u is the temperature in our domain.
- y and y are the coordinates of our coordinate system.
- Our domain is a rectangular area.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

Finite Difference Approximation (ODE)

- On a regular grid (i.e. dx and dy are constant), we can substitute the following first order finite difference for an ordinary derivative:

$$\frac{du}{dx} = \frac{u_{i+1} - u_i}{\Delta x} + O(\Delta x)$$

Finite Difference Approximation (PDE)

- On a regular grid (i.e. dx and dy are constant), we can substitute the following first order finite differences for partial derivatives:

$$\frac{\partial u}{\partial x} = \frac{u_{i+1,j} - u_{i,j}}{\Delta x} + O(\Delta x)$$

$$\frac{\partial u}{\partial y} = \frac{u_{i,j+1} - u_{i,j}}{\Delta y} + O(\Delta y)$$

Finite Difference Approximation (PDE)

- **On a regular grid (i.e. dx and dy are constant), we can substitute the following second order finite differences for the partial derivatives:**

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + O((\Delta x)^2)$$

$$\frac{\partial^2 u}{\partial y^2} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} + O((\Delta y)^2)$$

Finite Difference Approximation (Cont)

- After substituting finite differences into Laplaces equation and rearrange, we can have:

$$\frac{u_{i+1,j}^{n} - 2u_{i,j}^{n+1} + u_{i-1,j}^{n}}{(\Delta x)^2} + \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n+1} + u_{i,j-1}^{n}}{(\Delta y)^2} = O((\Delta x)^2, (\Delta y)^2)$$

- We can neglect the error term, and assume that Δx and Δy are equal in length.

Finite Difference Approximation (Cont)

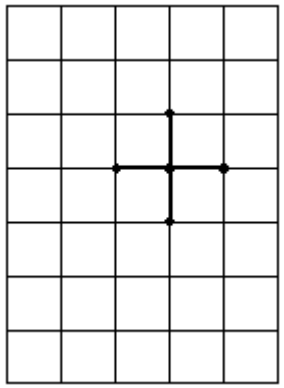
- If we solve the equation for $u_{i,j}^{n+1}$ and subtract $u_{i,j}^{n}$ from both sides, we have the following equations:

$$\Delta u_{i,j}^{n+1} = \frac{u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}}{4} - u_{i,j}^{n}$$

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \Delta u_{i,j}^{n+1}$$

Finite Difference Approximation (Cont)

$$u(i, j) = 0.25*(\; uold(i - 1, j) + uold(i + 1, j) + uold(i, j - 1) + uold(i, j + 1) \;)$$



Initial and Boundary Conditions

- We may assume the following initial and boundary conditions:

$$0 \le x \le L$$

$$0 \le y \le L$$

$$u(0, y) = 0$$

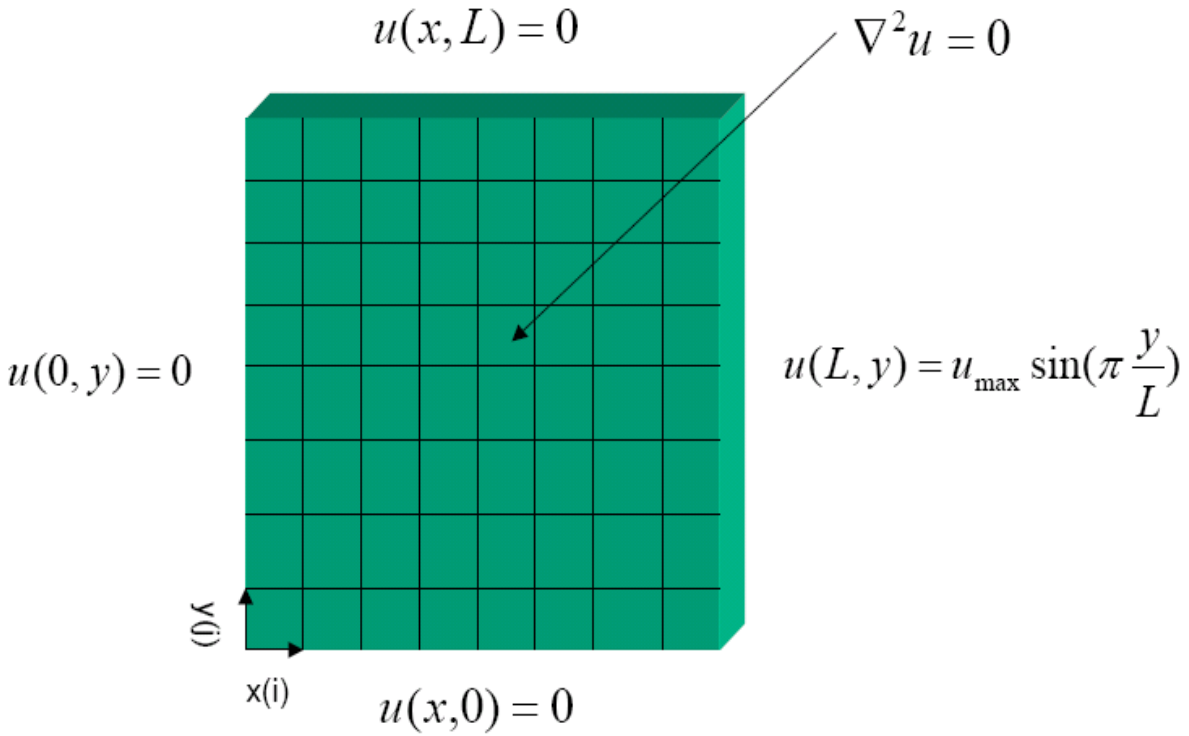
$$u(L, y) = u_{\max} \sin(\pi \frac{y}{L})$$

$$u(x,0) = 0$$

$$u(x, L) = 0$$

$$u_{i,j}^{0} = 0$$

Domain, Boundary and Initial Conditions



Serial Implementation (C)

```
for (i=1; i <= NR; i++)
    for (j=1; j <= NC; j++)
        u[i][j] = 0.25*( uold[i+1][j] +
                         uold[i-1][j] +
                         uold[i][j+1] +
                         uold[i][j-1] );
```

Serial Implementation (Fortran)

```
DO j = 1,NC
   DO i = 1,NR
      u(i,j) = 0.25*( uold(i-1,j)
.                    +uold(i+1,j)
.                    +uold(i,j-1)
.                    +uold(i,j+1) )
   ENDDO
ENDDO
```

Serial Implementation Error Term

- *du* is the difference term for point *(i,j)* at some iteration step.
- *dumax* is the maximum difference term for all points at some iteration step.

```
dumax=max( dumax, abs(du(i,j) )
```

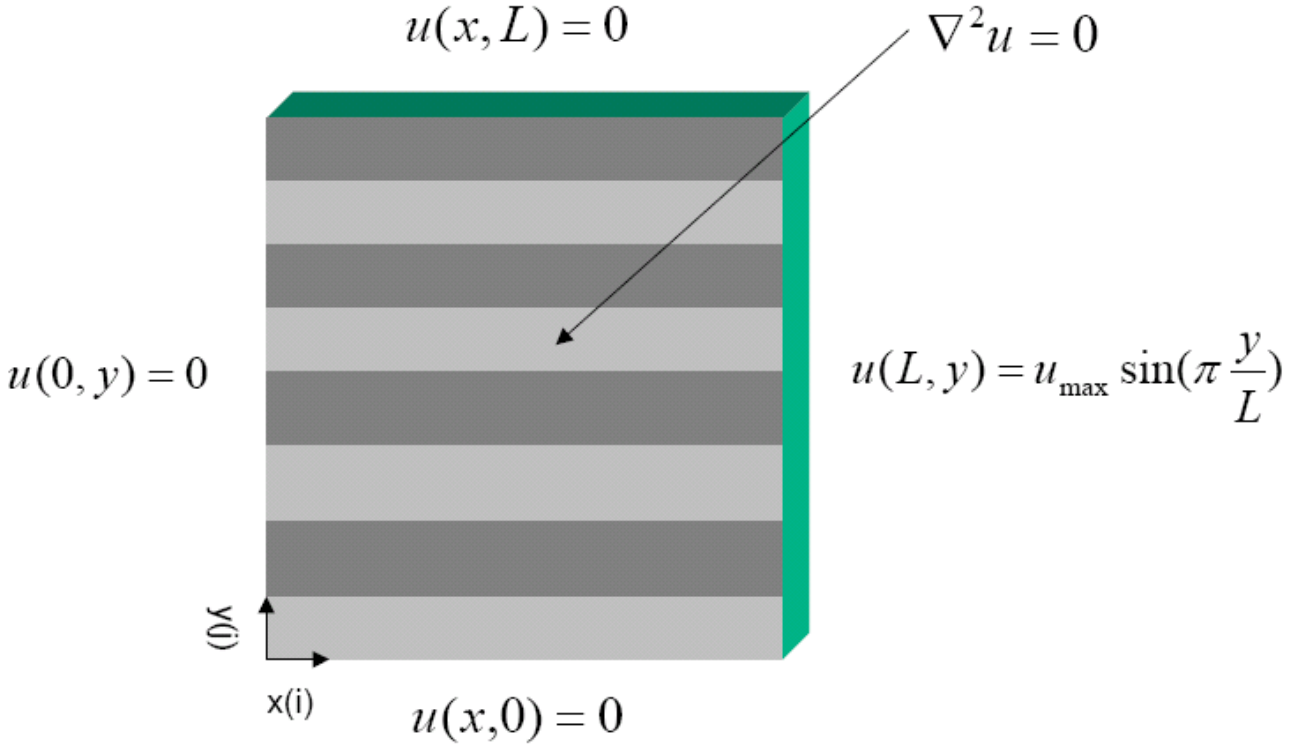
Serial Implementation Laplace (cont.)

- See example serial code

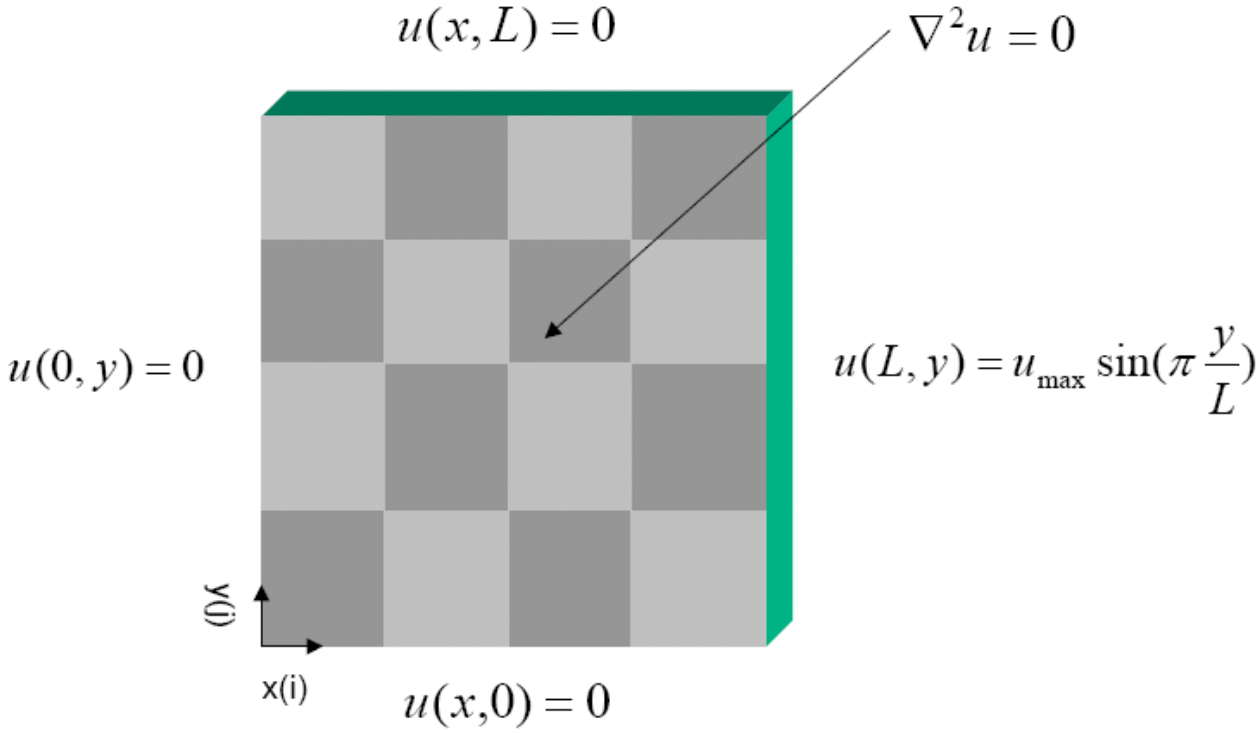
```
      program lpcache
      integer imax,jmax,im1,im2,jm1,jm2,it,itmax
      parameter (imax=2001,jmax=2001)
      parameter (im1=imax-1,im2=imax-2,jm1=jmax-1,jm2=jmax-2)
      parameter (itmax=100)
      real*8 u(imax,jmax),du(imax,jmax),umax,dumax,tol,pi
      parameter (umax=10.0,tol=1.0e-6,pi=3.1415)

! Initialize
      do j=1,jmax
         do i=1,imax-1
            u(i,j)=0.0
            du(i,j)=0.0
         enddo
         u(imax,j)=umax*sin(pi*float(j-1)/float(jmax-1))
      enddo
```

```
! Main computation loop
      do it=1,itmax
         dumax=0.0
         do j=2,jm1
            do i=2,im1
               du(i,j)=0.25*(u(i-1,j)+u(i+1,j)+u(i,j-1)
     +                    +u(i,j+1))-u(i,j)
               dumax=max(dumax,abs(du(i,j)))
               u(i,j)=u(i,j)+du(i,j)
            enddo
         enddo
         write (1,*) it,dumax
      enddo
      stop
      end
```

1D Domain Decomposition



2D Domain Decomposition



Data Construction (C)

- C writes row wise into the memory.

0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7

Data Construction (Fortran)

- Fortran writes column wise into the memory.

1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8
2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8
3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8
4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8

Derived Data Type (Column Vector)

- double x[4][8];
- MPI_Type_vector(4,1,8,MPI_DOUBLE,&coltype);
- MPI_Type_commit(&coltype);

0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	1,0	...

0,0
1,0
2,0
3,0

Derived Data Type (Row Vector)

- real x(4,8);
- Call MPI_TYPE_VECTOR(8,1,4,MPI_REAL, rowtype, err);
- Call MPI_TYPE_COMMIT(&rowtype,err);

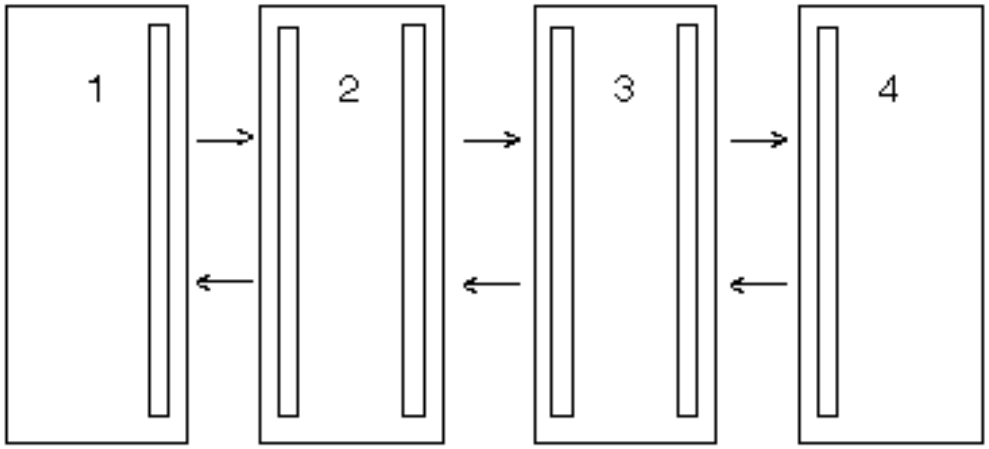
1,1
2,1
3,1
4,1
1,2
2,2
3,2
4,2
1,3
...

1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8

1D Virtual Topology

- MPI_Cart_create(MPI_COMM_WORLD,1,dim,period,reorder,&vu);
- MPI_Cart_shift(vu,0,1,&left,&right);
- printf("P:%d My neighbors are r: %d l:%d, \n",rank,right,left);

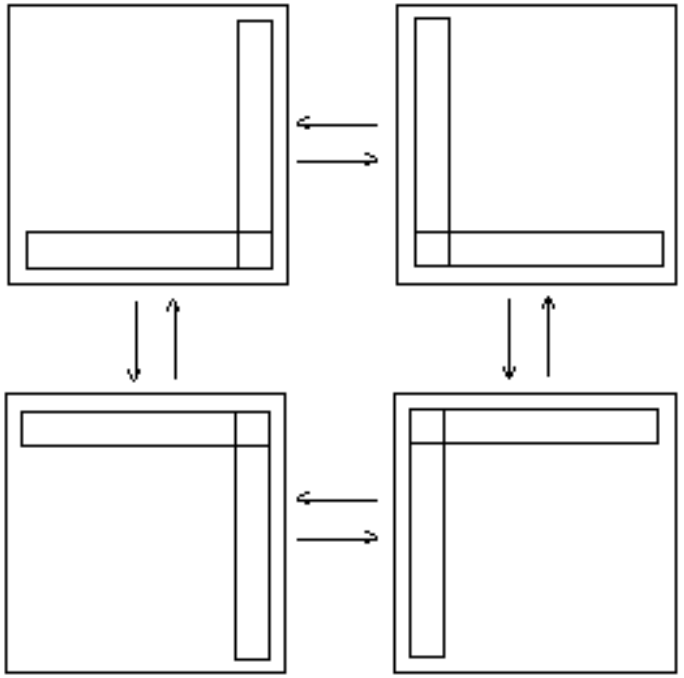
1D Communication



2D Virtual Topology

- MPI_Cart_create(MPI_COMM_WORLD,2,dim,period,reorder,&vu);
- MPI_Cart_shift(vu,0,1,&left,&right);
- MPI_Cart_shift(vu,1,1,&up,&down);
- printf("P:%d My neighbors are r: %d d:%d 1:%d u:%d\n",rank,right,down,left,up);

2D Communication



3D Virtual Topology

- MPI_Cart_create(MPI_COMM_WORLD,3,dim,period,reorder,&vu);
- MPI_Cart_shift(vu,0,1,&left,&right);
- MPI_Cart_shift(vu,1,1,&up,&down);
- MPI_Cart_shift(vu,2,1,&rear,&front);
- printf("P:%d My neighbors are r: %d d:%d re:%d l:%d u:%d f:%d\n",rank,right,down,rear,left,up,front);

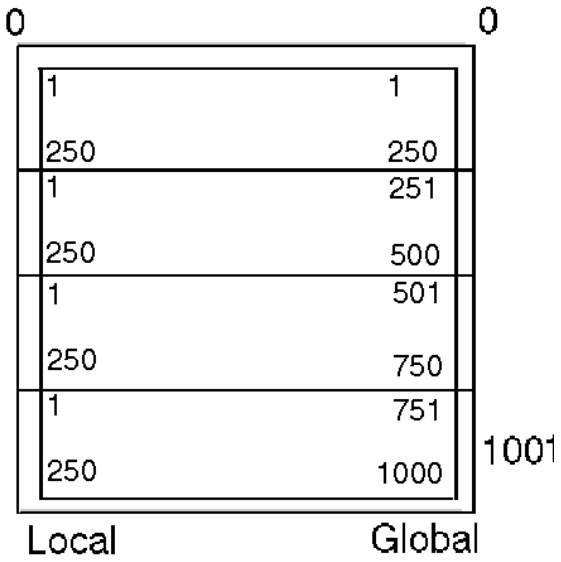
Before Parallel Imlementation (BPI)

- Some notifications and explanations before parallel implementation

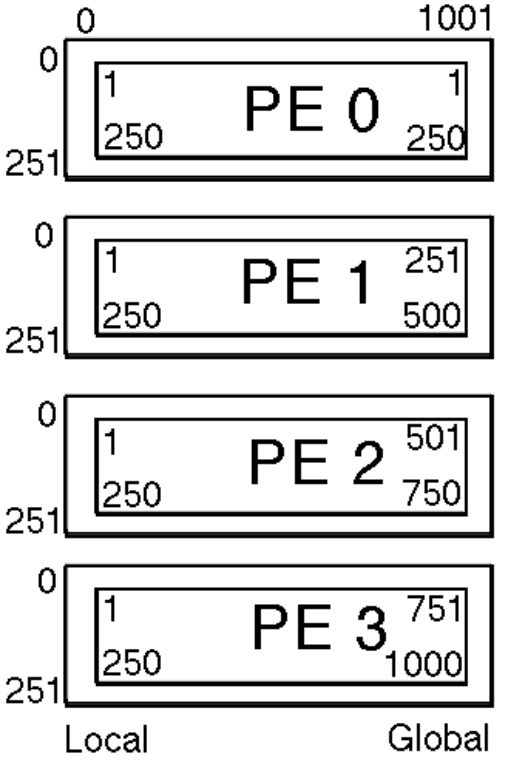
BPI – Data Types

- MPI_Datatype dt, ddt;
- dt=MPI_DOUBLE;
- MPI_Type_vector(1, n, n, dt, &ddt);
- MPI_Type_commit(&ddt);
- MPI_Send(&a(sr,0), n/(size-1), ddt, i+right, 5, comm);

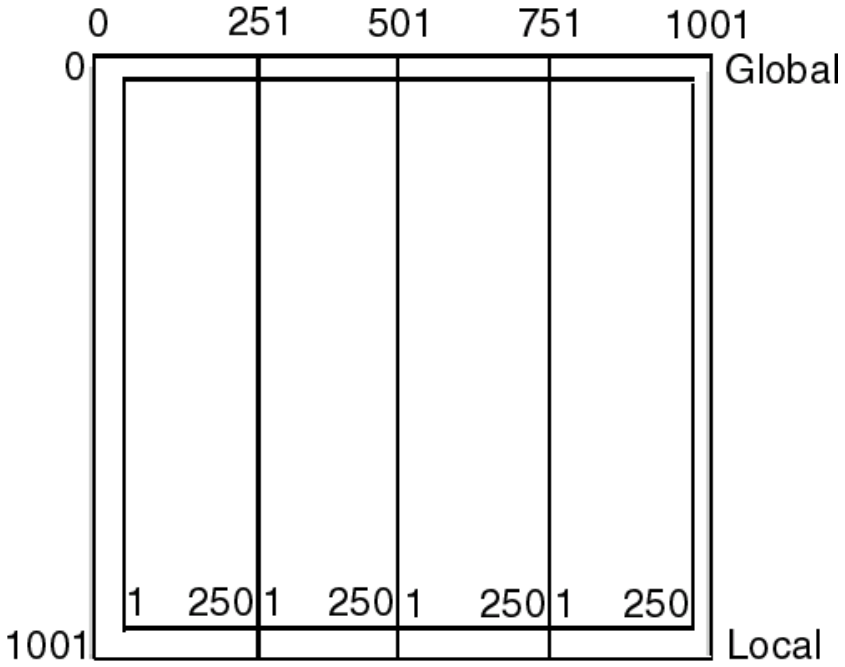
BPI - Decomposition for C



BPI - Decomposition for C (Ghost Cells)



BPI - Decomposition for Fortran



BPI - Sending Strips

- for (i=0;i<size-1;i++){
- sr=i*n/(size-1);
- MPI_Send(&a(sr,0), n/(size-1), ddt, i+right, 5, comm);
- }

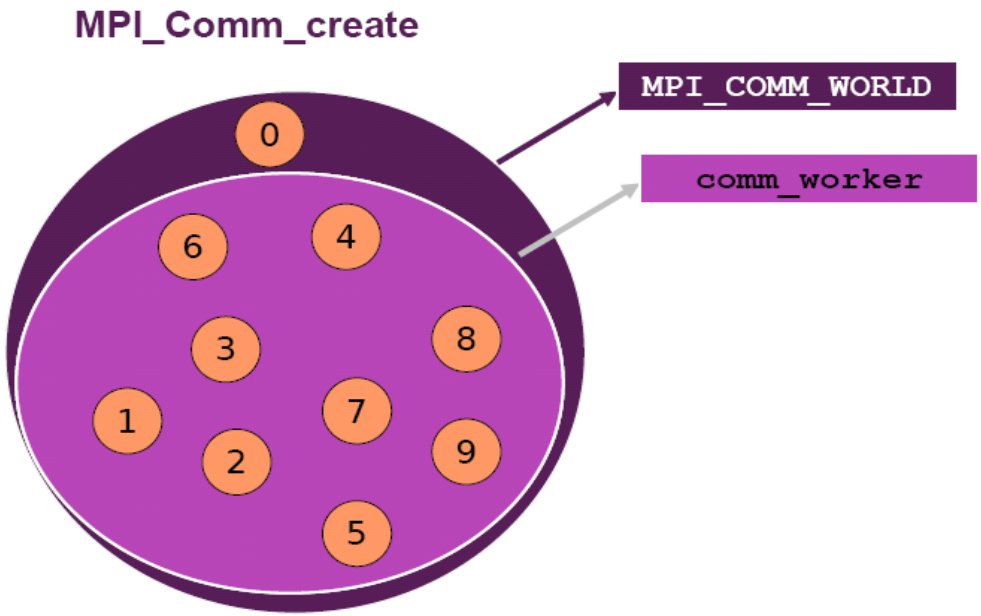
BPI - Topology

- int dim[1], period[1], coord[1], reorder, TRUE=1, FALSE=0;
- period[0]=FALSE; reorder=TRUE;
- MPI_Comm comm, vu, new_comm;
- MPI_Cart_create(comm, 1, dim, period, reorder, &vu);
- MPI_Cart_shift(vu, 0, 1, &down, &up);

BPI - Grouping

- for(i=0;i<size-1;i++)
- members[i]=i+1;
- MPI_Group group_world, new_group;
- MPI_Comm_group(comm, &group_world);
- MPI_Group_incl(group_world, size-1, members, &new_group);
- MPI_Group_rank(new_group, &new_rank);
- MPI_Comm_create(comm, new_group, &new_comm);
- MPI_Barrier(new_comm);

BPI – Grouping (cont)



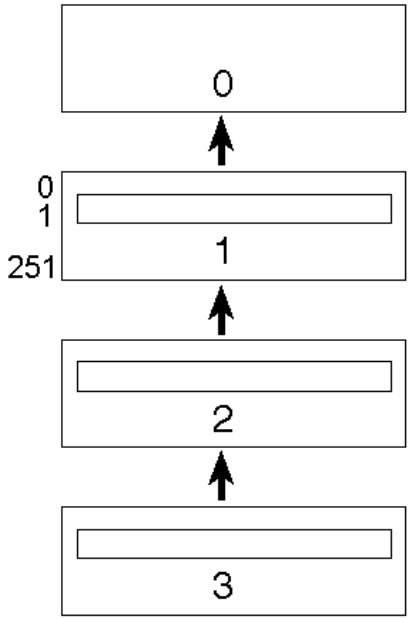
BPI Main Computation Loop

```
for (m=1;m<115;m++){
    for (j=jstart;j<=jend;j++)
        for (i=istart;i<=iend;i++)

            b(j,i) =b(j,i) + 1.0 * (0.25 * ( b(j-1,i) + b(j+1,i)+b(j,i-1)+b(j,i+1) ) - b(j,i) );
```

BPI – Sending Down

```
if (down>0){ //if baslar
    j=1;
    for (i=istart;i<=iend;i++){
      dobuf[j]=b((jstart),i);
      j++;
    }
    length=iend-istart+1;
    MPI_Bsend(&dobuf[1], length, dt, down, 5, comm);
```

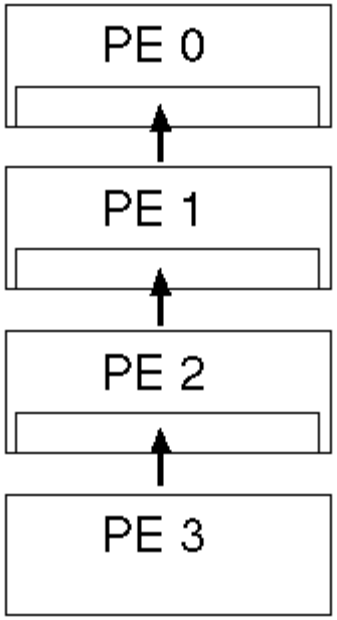
BPI – Sending Down (Cont)



BPI – Receiving From Up

```
if (up<size && up>0){//if baslar
    length=iend-istart+1;
    MPI_Recv(&uibuf[1], length, dt, up, 5, comm, &istat);
     j=1;
    for (i=istart;i<iend;i++){
       b(jend+1,i)=uibuf[j];
       j++;
    }
```

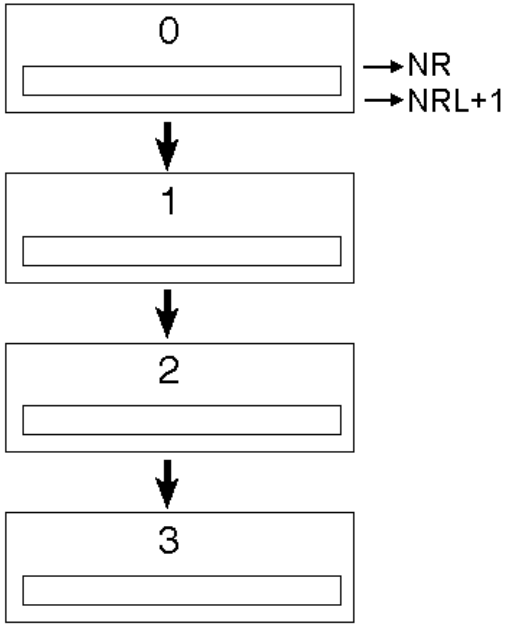
BPI – Receiving From Up (Cont)



BPI – Sending Up

```
if (up<size && up>0) { //if baslar
   j=1;
   for (i=istart;i<iend;i++){
      uobuf[j]=b(jend,i);
      j++;
   }
   length=iend-istart+1;
   MPI_Bsend(&uobuf[1], length, dt, up, 5, comm);
```

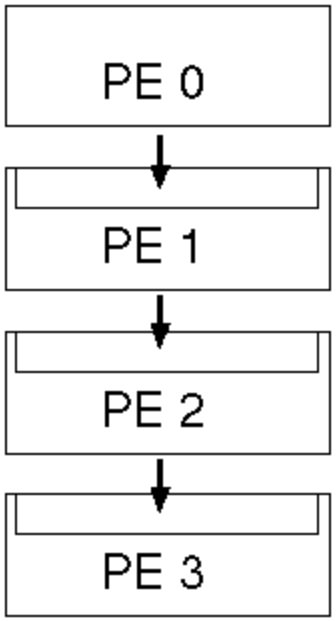
BPI – Sending up (Cont)



BPI – Receiving From Down

```
if (down>0){//if baslar
    length=iend-istart+1;
    MPI_Recv(&dibuf[1], length, dt, down, 5, comm, &istat);
    j=1;
    for (i=istart;i<iend;i++){
      b(jstart-1,i)=dibuf[j];
      j++;
    }
```

BPI – Receiving From Down (Cont)



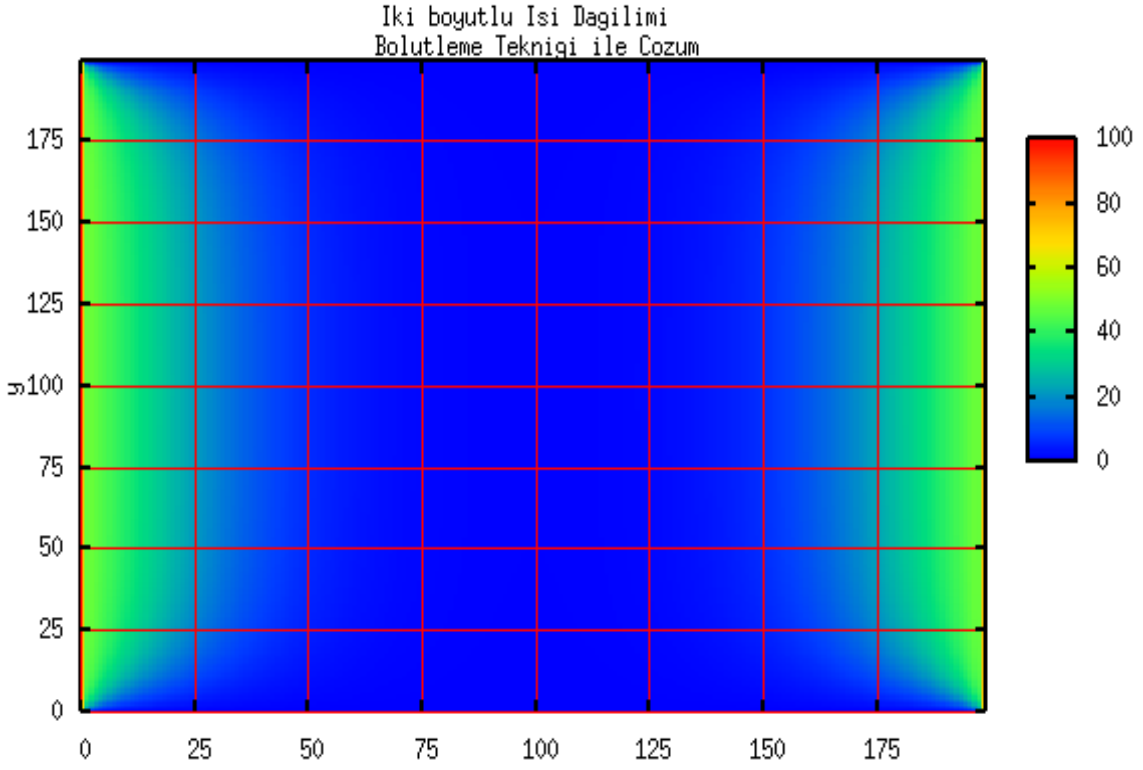
BPI – Attach Detach

- MPI_Buffer_attach(malloc(100000), 100000);
- MPI_Buffer_detach(&buff, &buffsize);

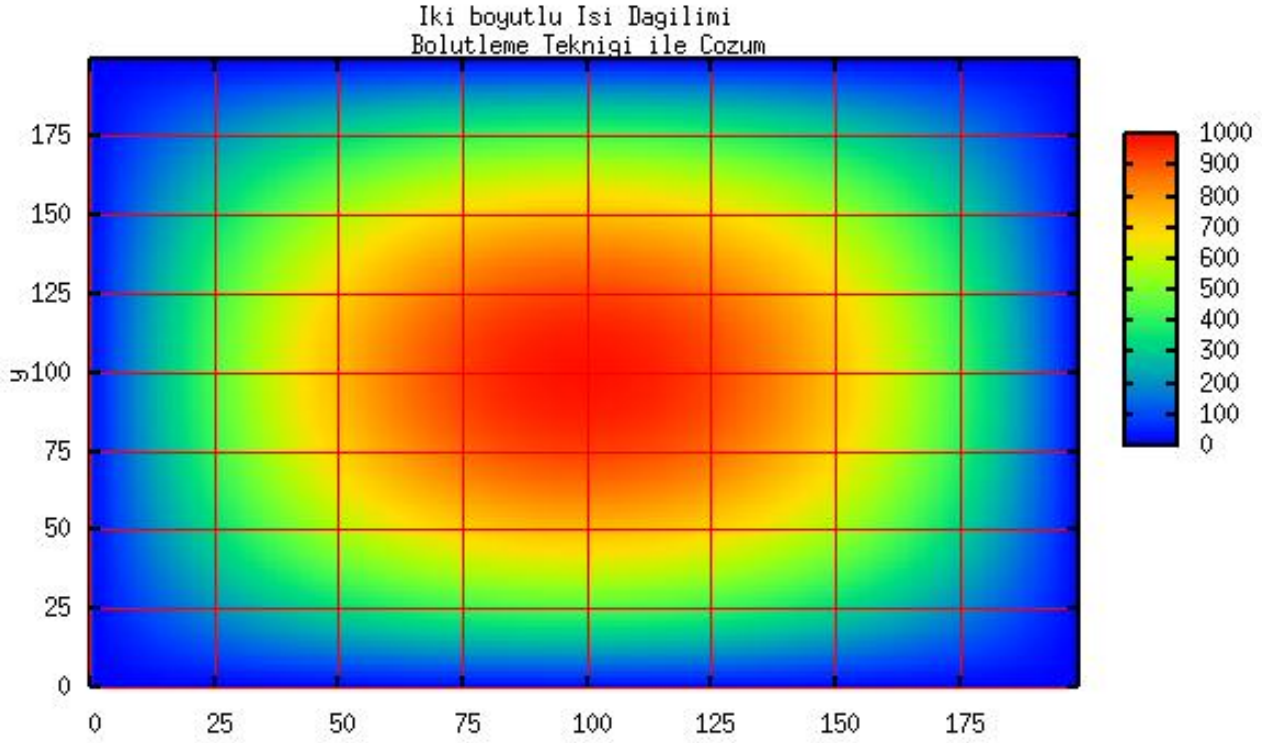
Parallel Imlementation (code)

```
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"

//#define n 16
#define a(j,k) a[k+j*n]
#define b(j,k) b[k+j*n]

int main(int argc, char *argv[]){
int n=16;
double *a=NULL, *b=NULL; //2d matrices
int i, j, k, l;
//MPI Parameters
int size, rank;
int dim[1], period[1], reorder, TRUE=1, FALSE=0;
int coord[1], id;
int left, right, up, down;
int sr;
double t1, t2;

int imax=n-1, jmax=n-1;
int istart, iend, jstart, jend, jsize, length;
double dobuf[n], dibuf[n];
double uobuf[n], uibuf[n];
int m, members[8];
int new_rank;
.....................................................
```

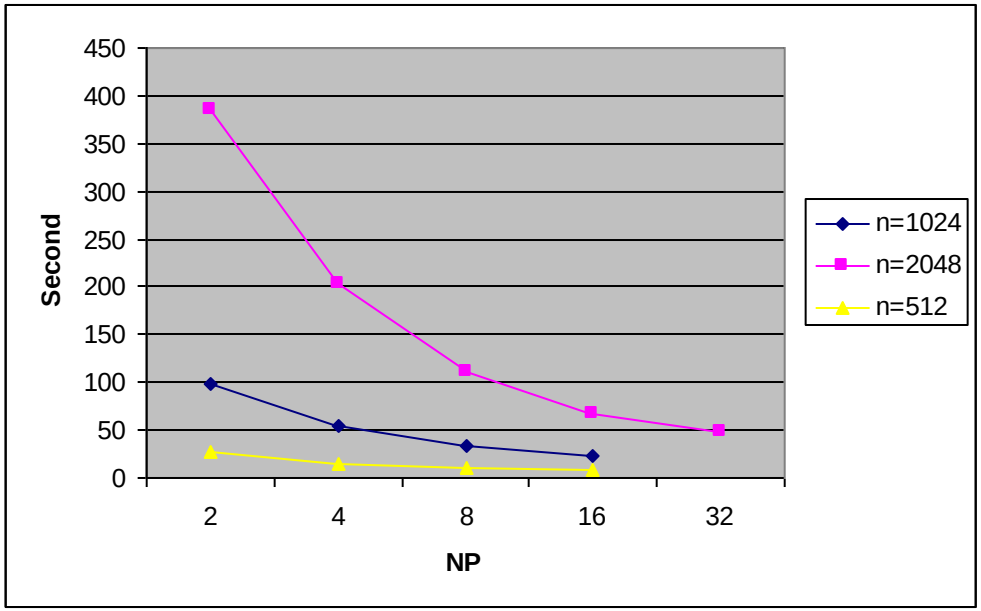
Results


Iki boyutlu Isi Dagilimi
Bolutleme Tekniqi ile Cozum
175
150
125
100
75
50
25
0
0
25
50
75
100
125
150
175
100
80
60
40
20
0

Results

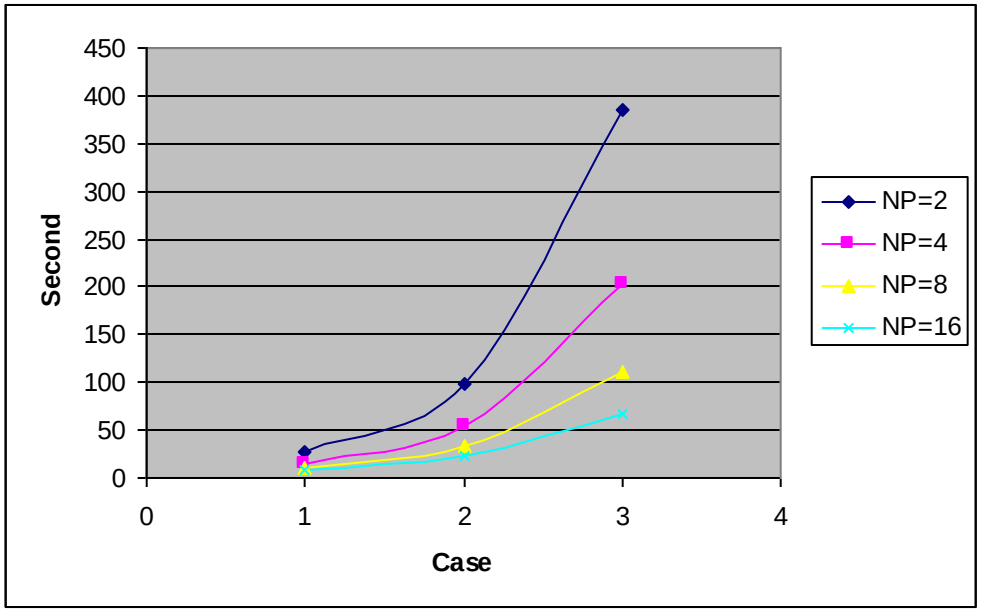

Iki boyutlu Isi Dagilimi
Bolutleme Tekniqi ile Cozum
175
150
125
100
75
50
25
0
0
25
50
75
100
125
150
175
1000
900
800
700
600
500
400
300
200
100
0

Results



- NP: Numper of processes

Results



- Case 1: 512x512 points
- Case 2: 1024x1024 points
- Case 3: 2048x2048 points

References

- http://www.sc-2.psc.edu/workshop/jan01/Code_Development/Code_Development.html#22

- Jim Giuliani, Ohio Sumpercomputer Center, Science and Technology Support High Performance Computing


UYBHM
ULUSAL YÜKSEK BAŞARIMLI
HESAPLAMA MERKEZİ
İSTANBUL TEKNİK ÜNİVERSİTESİ