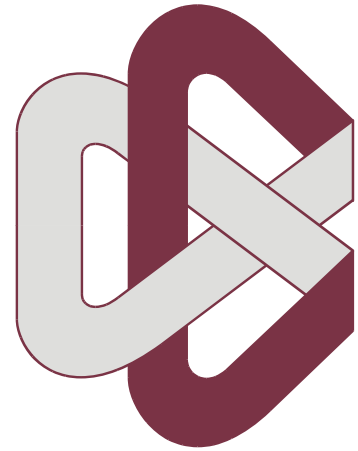


CIMAT

Gradiente conjugado con MPI

Miguel Vargas-Félix

miguelvargas@cimat.mx
http://www.cimat.mx/~miguelvargas

Gradiente conjugado con memoria distribuida

Vamos a partir del algoritmo serial de gradiente conjugado.

Input: $\mathbf{A}$, $\mathbf{x}_0$, $\mathbf{b}$,ε

$\mathbf{r}_0 \leftarrow \mathbf{A}\mathbf{x}_0 - \mathbf{b}$

$\mathbf{p}_0 \leftarrow -\mathbf{r}_0$

$k \leftarrow 0$

mientras $\|\mathbf{r}_k\| > \varepsilon$

$\mathbf{w} \leftarrow \mathbf{A}\mathbf{p}_k$

$\alpha_k \leftarrow \dfrac{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}{\mathbf{p}_k^{\mathrm{T}}\mathbf{w}}$

$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$

$\mathbf{r}_{k+1} \leftarrow \mathbf{r}_k + \alpha \mathbf{w}$

$\beta_k \leftarrow \dfrac{\mathbf{r}_{k+1}^{\mathrm{T}}\mathbf{r}_{k+1}}{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}$

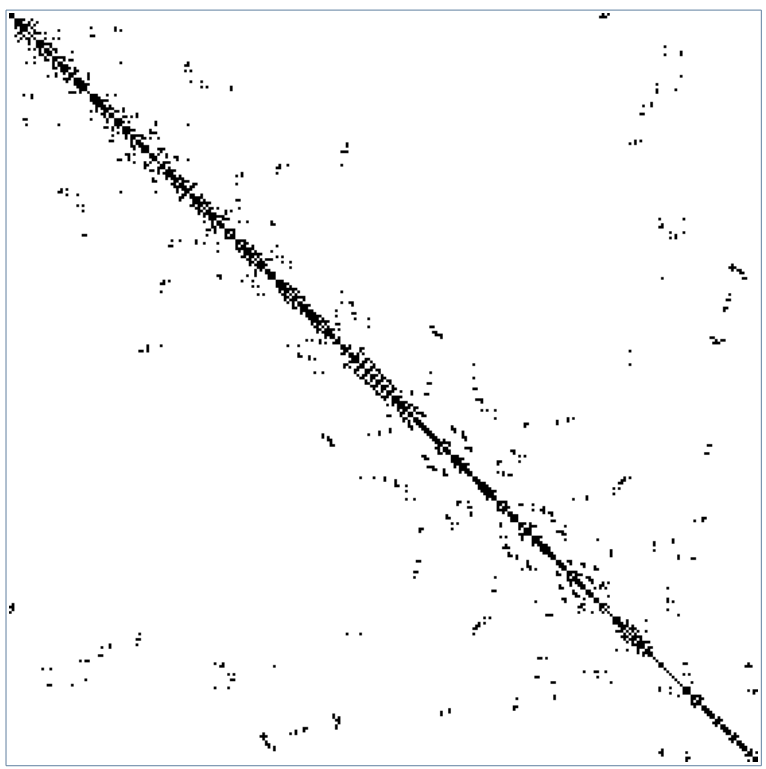
$\mathbf{p}_{k+1} \leftarrow -\mathbf{r}_{k+1} + \beta_{k+1}\mathbf{p}_k$

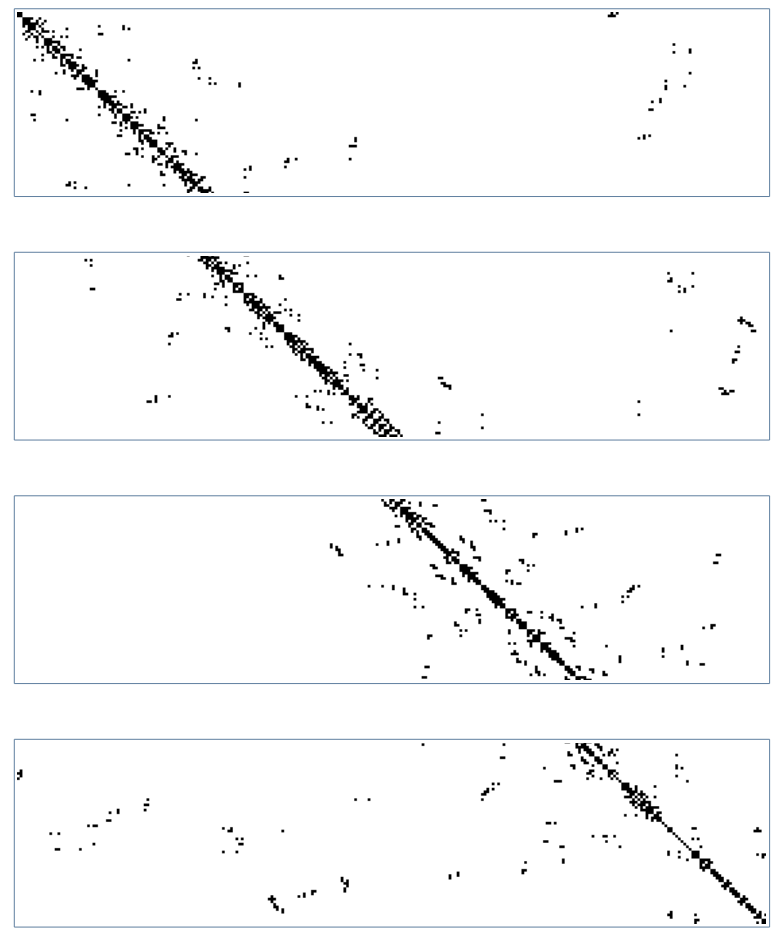
$k \leftarrow k+1$

La parte computacionalmente más costosa es la multiplicación matriz-vector. Vamos a paralelizar con MPI esa operación.

Primer algoritmo

El algoritmo de gradiente conjugado con MPI será entonces un esquema maestro-esclavos, en la cual el maestro realiza el algoritmo y los esclavos se encargan de la multiplicación matriz-vector.





La multiplicación matríz vector será por partes.

Pseudocódigo

El pseudocódigo del programa en MPI sería:

MPI_Init(); MPI_Comm_size(size); MPI_Comm_rank(rank);	Inicializar MPI

if (rank == 0)
{
Load(**A**, **b**); // Leer matriz A, leer vector b
n = size(**b**) // Sea n el número de incógnitas
num_slaves = size - 1; // Sea num_slaves el número de esclavos
	Inicia sección para maestro

for (s = 1; s <= num_slaves; ++s)
{
MPI_Send(n, 1, s); // Enviar el tamaño del sistema de ecuaciones
MPI_Send(num_rows, 1, s); // Enviar el número de renglones para el esclavo s
MPI_Send(***rows***, num_rows, s); // Enviar los índices de renglón
for (r = 1; r <= num_rows; ++r)
{
MPI_Send(row_size[r], 1, s);
MPI_Send(***row_index***[r], row_size[r], s);
MPI_Send(***row_values***[r], row_size[r], s);
}
}
	Enviar un grupo de renglones a cada esclavo

// Calcular r = A*x - b
for (s = 1; s <= num_slaves; ++s)
{
MPI_Send(**x**, n, s);
}
	Calcular residual y dirección de descenso iniciales

```
r = -b;
for (s = 1; s <= num_slaves; ++s)
{
  MPI_Recv(y_s, n, s);
  r = r + y_s;
}
p = -r;
```

```
while (norm(r) > tolerance)
{
  // Calcular w = A*p
  for (s = 1; s <= num_slaves; ++s)
  {
    multiplicar = 1;
    MPI_Send(multiplicar, 1, s); // Avisar al esclavo s que sigue una multiplicación
    MPI_Send(p, n, s); // Enviar p al esclavo s
  }
  w = 0
  for (s = 1; s <= num_slaves; ++s)
  {
    MPI_Recv(y_s, n, s);
    w = w + y_s;
  }

  // Calcular tamaño de paso y nueva dirección de descenso
  alpha = dot(r, r)/dot(p, w);
  x1 = x + alpha*p;
  r1 = r + alpha*w;
  beta = dot(r1, r1)/dot(r, r);
  p1 = -r1 + beta*p;
  x = x1;
```

Ciclo de gradiente conjugado en el maestro

```
  r = r1;
  p = p1;
 }
 for (s = 1; s <= num_slaves; ++s)
 {
  multiply = 0; // Avisar al esclavo s que terminaron las multiplicacines
  MPI_Send(multiply, 1, s);
 }
 Save(x); // Guardar la solución
}
else
{
 MPI_Recv(n, 1, 0); // Recibe el tamaño del sistema de ecuaciones
 MPI_Recv(num_rows, 1, 0);     // Recibe el número de renglones
 MPI_Recv(rows, num_rows, 0); // Recibir los índices de renglón
 for (r = 1; r <= num_rows; ++r)
 {
  MPI_Recv(row_size[r], 1, 0);
  MPI_Recv(row_index[r], row_size[r], 0);
  MPI_Recv(row_values[r], row_size[r], 0);
 }
 MPI_Recv(x, n, 0); // Recibir vector x
 y = 0;
 for (r = 1; r <= num_rows; ++r)
 {
  i = rows[r];
  y_i = 0;
  for (k = 1; k <= row_size[r]; ++k)
  {
```

Avisarle a los esclavos que no hay más multiplicaciones pendientes

Inicia sección para esclavos

Recibir el numero e indice de los renglones

Recibir las los renglones comprimidos

Hacer el la multiplicación de los renglones comprimidos por el vector x

```
    y_i += row_values[r][k]*x[row_index[r][k]];
  }
 }
 MPI_Send(y, n, 0); // Enviar vector y
```

```
 for ( ; ; )
 {
```

Ciclo de multiplicaciones

```
  MPI_Recv(multiply, 1, 0);
  if ( multiply == 0)
    break;
```

Condicion de salida del ciclo

```
  MPI_Recv(p, n, 0); // Recibir vector p
  for (r = 1; r <= num_rows; ++r)
  {
    i = rows[r];
    y_i = 0;
    for (k = 1; k <= row_size[r]; ++k)
    {
      y_i += row_values[r][k]*p[row_index[r][k]];
    }
  }
  MPI_Send(y, n, 0); // Enviar vector y
 }
}
```

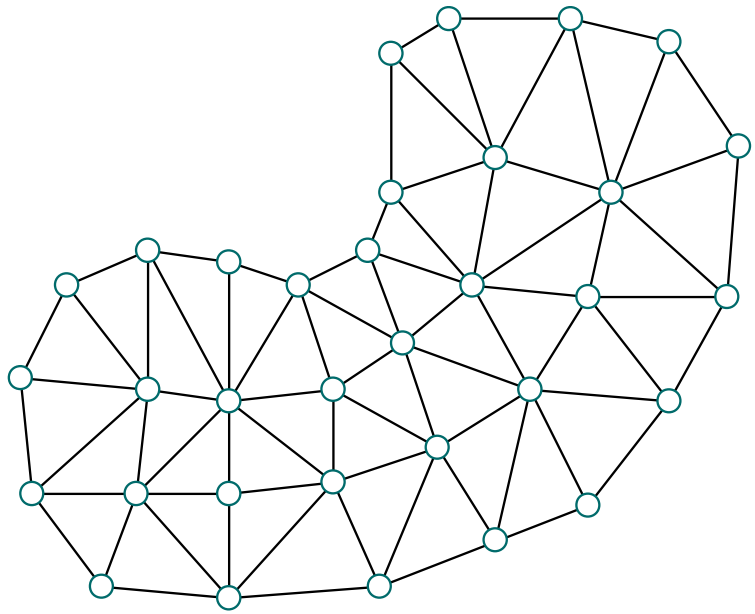
Hacer el la multiplicación de los renglones comprimidos por el vector p

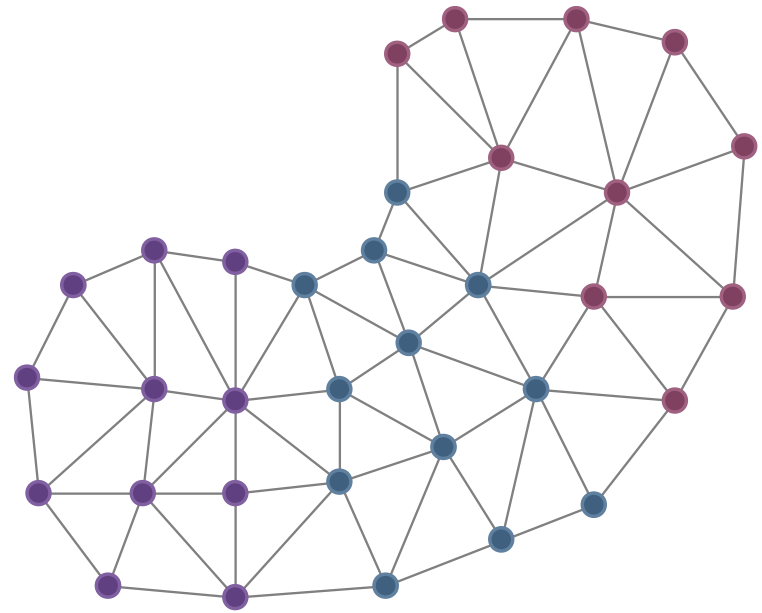
```
MPI_Finalize();
```

Finalizar MPI

Segundo algoritmo

La forma en que dividiremos la matriz del sistema será particionando el grafo de ésta.

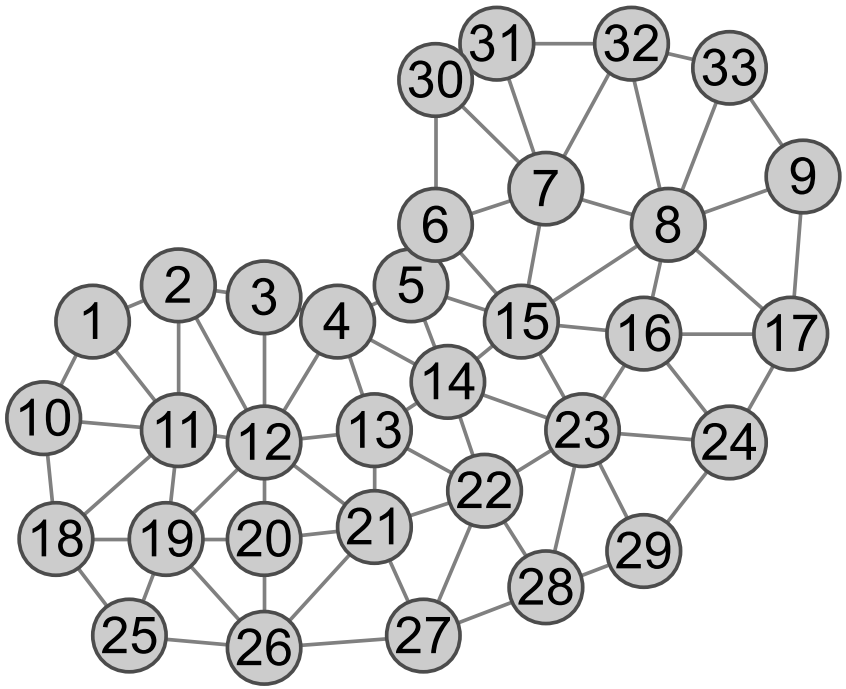


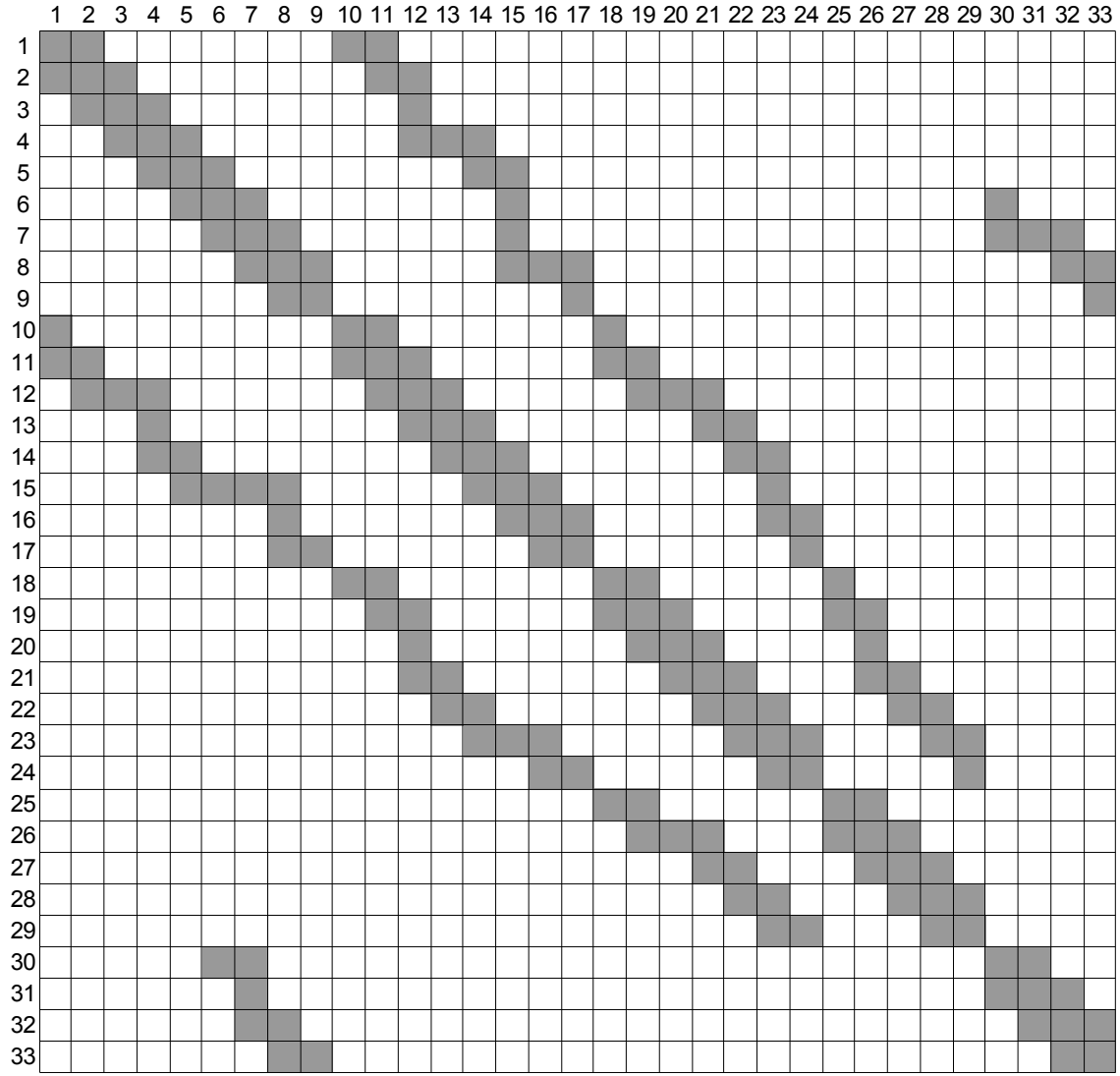


- El particionamiento se hace agrupando nodos.
- Las particiones comparten aristas.
- El particionamiento se hace tratando de poner un igual número de nodos en cada partición.

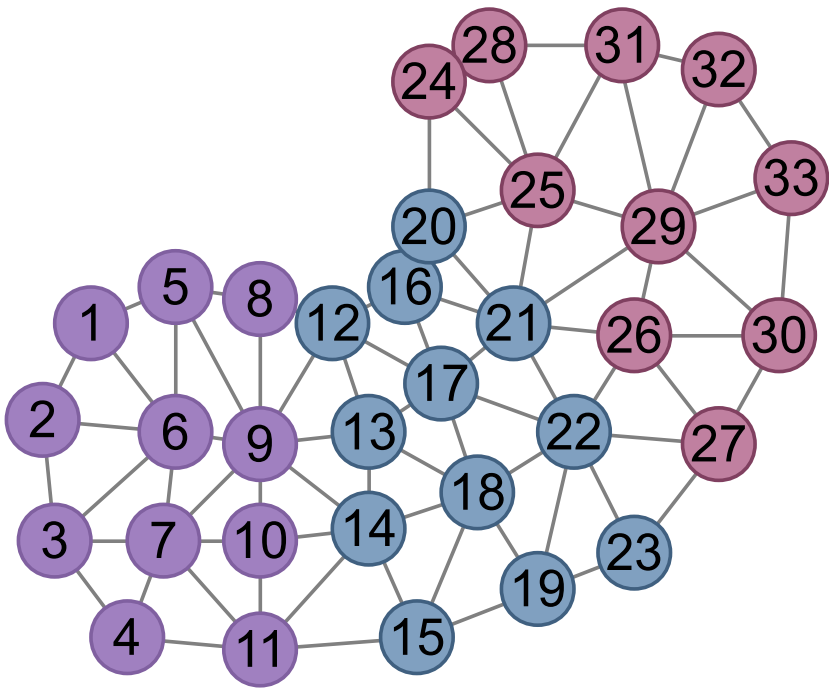
Reenumeración de los nodos

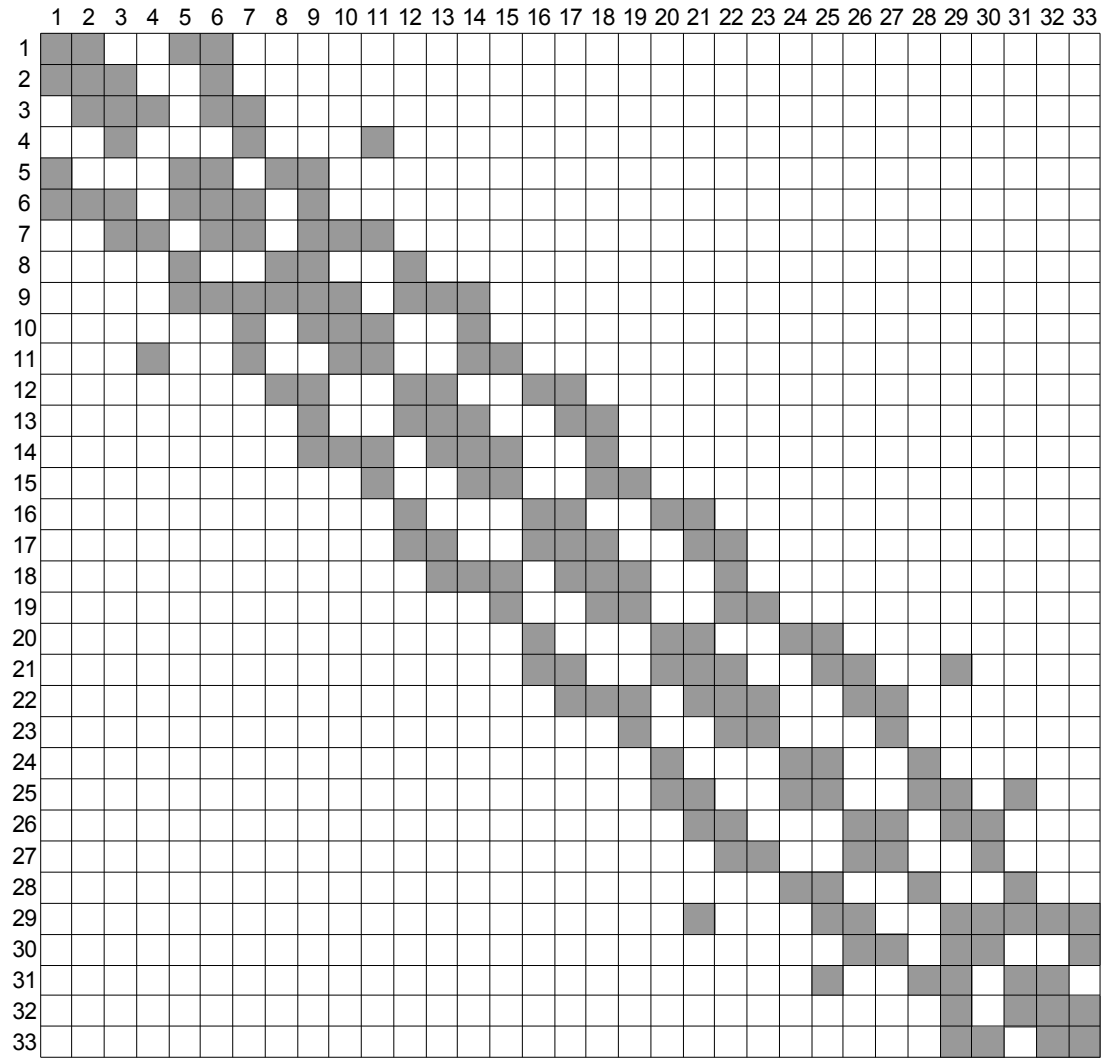
La siguiente figura muestra el grafo de un problema y su matríz dispersa.





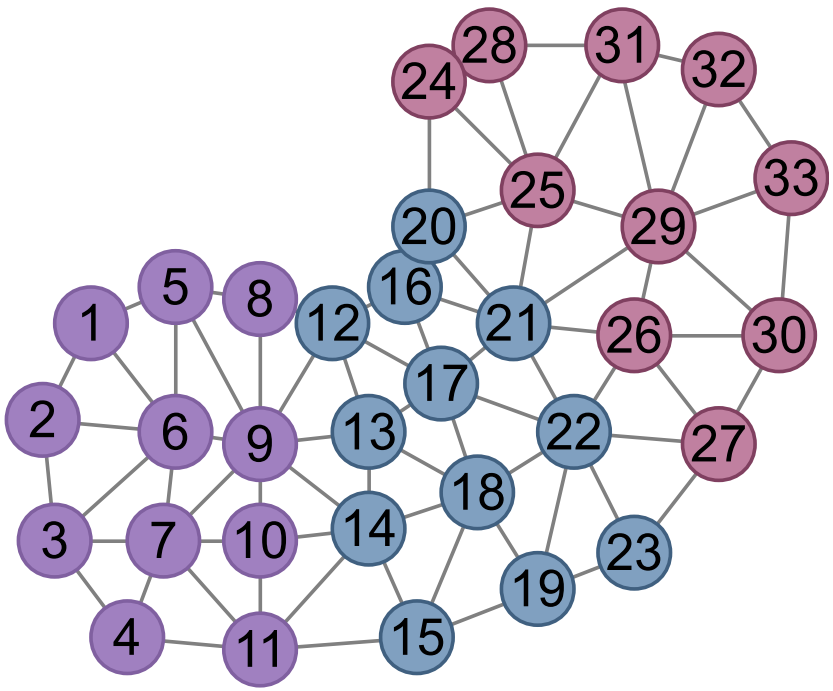
Vamos a particionar el grafo y a re-enumerar los nodos, de tal forma que los nodos de la primer partición tengan los primeros nodos, la segunda partición los siguientes, etc.

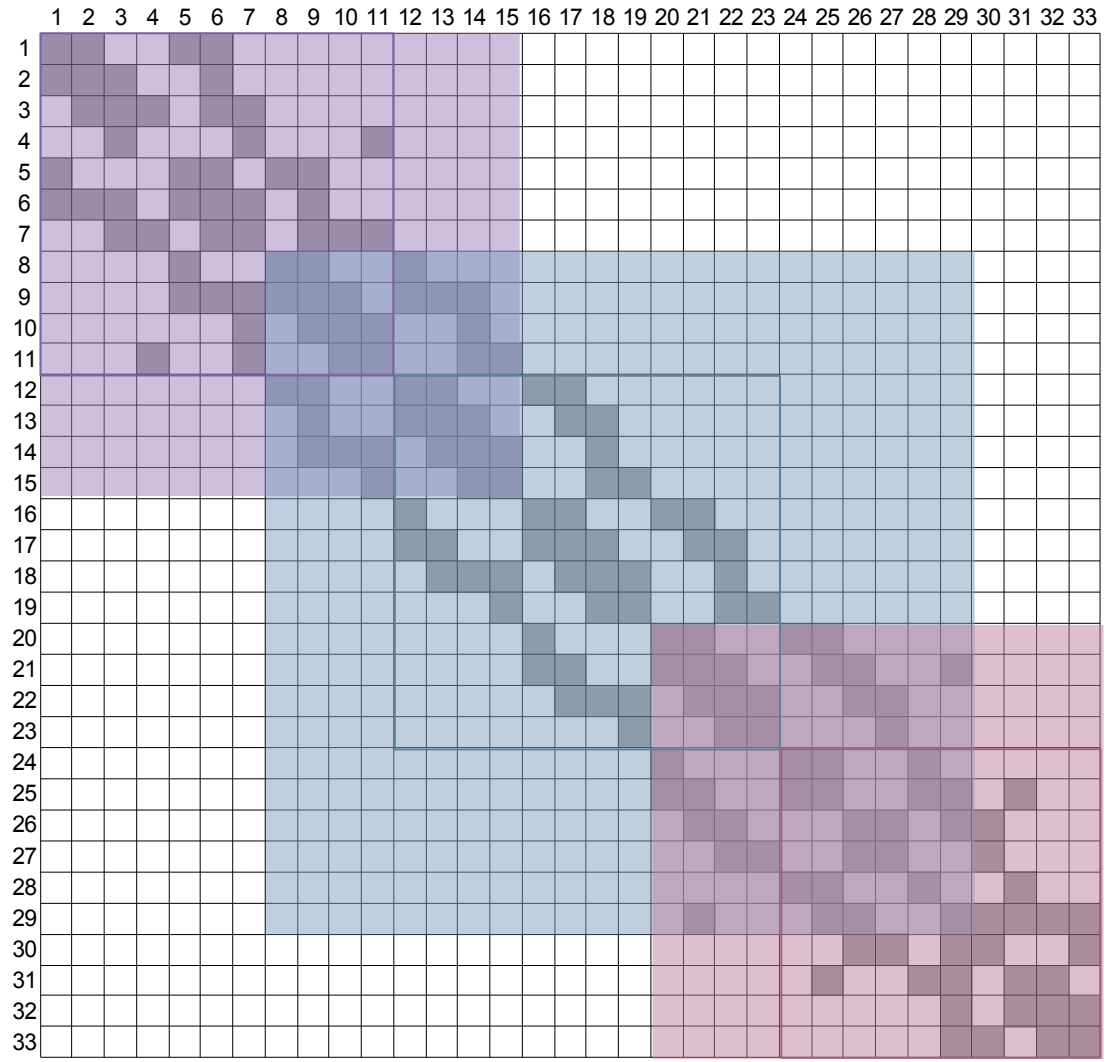




En la matríz dispersa se puede ver como se agrupan las entradas en bloques que se traslapan entre particiones.

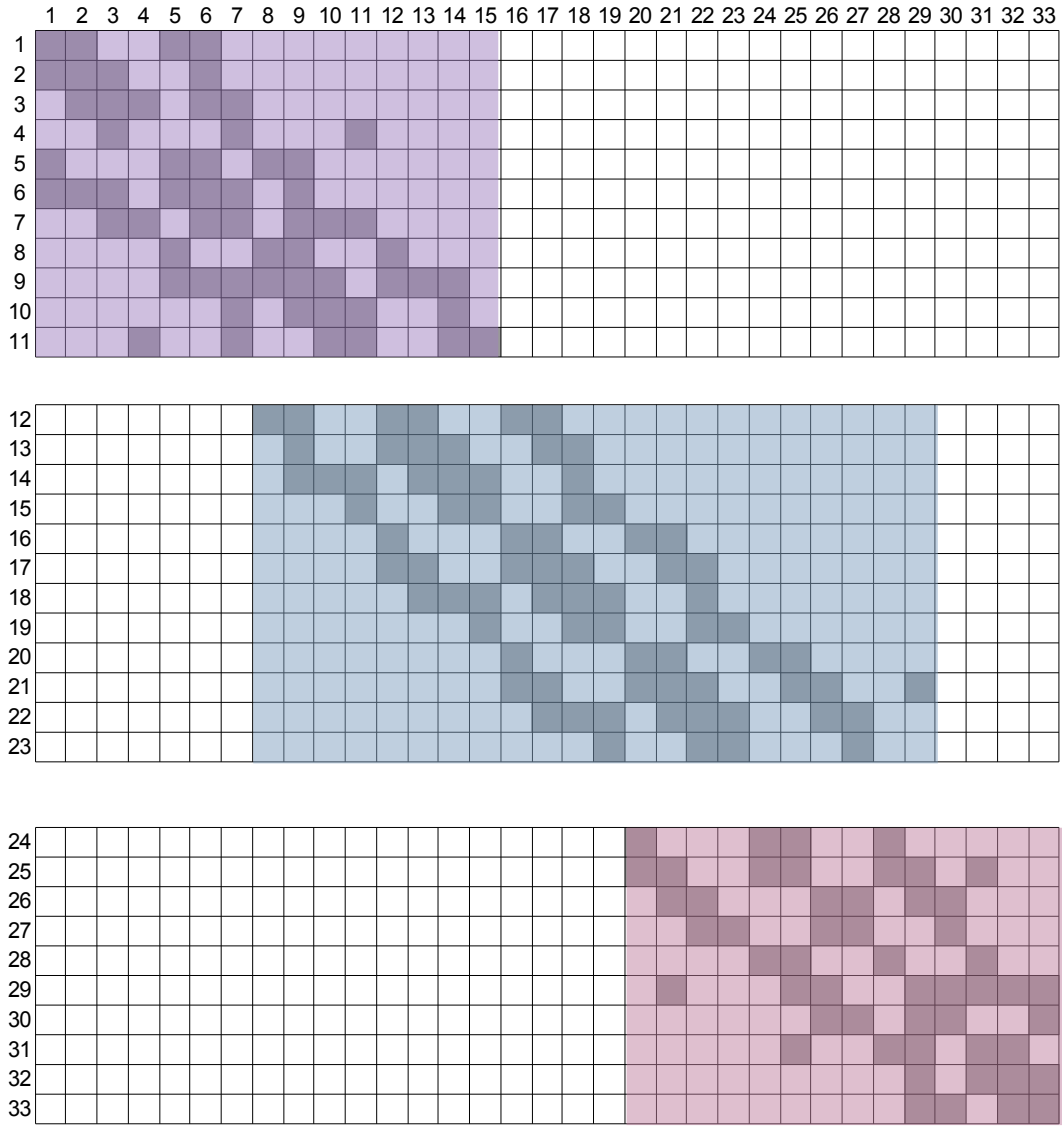
Vamos a particionar el grafo y a re-enumerar los nodos, de tal forma que los nodos de la primer partición tengan los primeros nodos, la segunda partición los siguientes, etc.





En la matríz dispersa se puede ver como se agrupan las entradas en bloques que se traslapan entre particiones.

La separación de la matriz entre los distintos esclavos sería:



$\mathbf{y}_1 \leftarrow \mathbf{A}_1^r \mathbf{p}_1$

$\mathbf{A}_1^r$ tiene 11 renglones
$\mathbf{p}_1$ tiene 15 renglones
$\mathbf{y}_1$ tiene 11 renglones

$\mathbf{y}_2 \leftarrow \mathbf{A}_2^r \mathbf{p}_2$

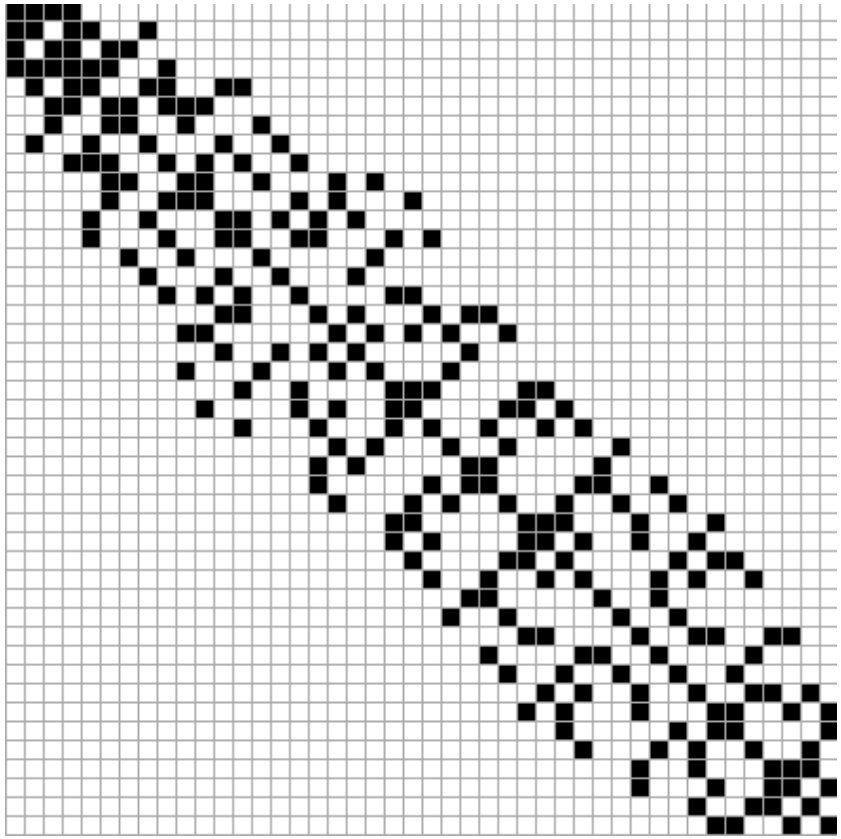
$\mathbf{A}_2^r$ tiene 12 renglones
$\mathbf{p}_2$ tiene 22 renglones
$\mathbf{y}_2$ tiene 12 renglones

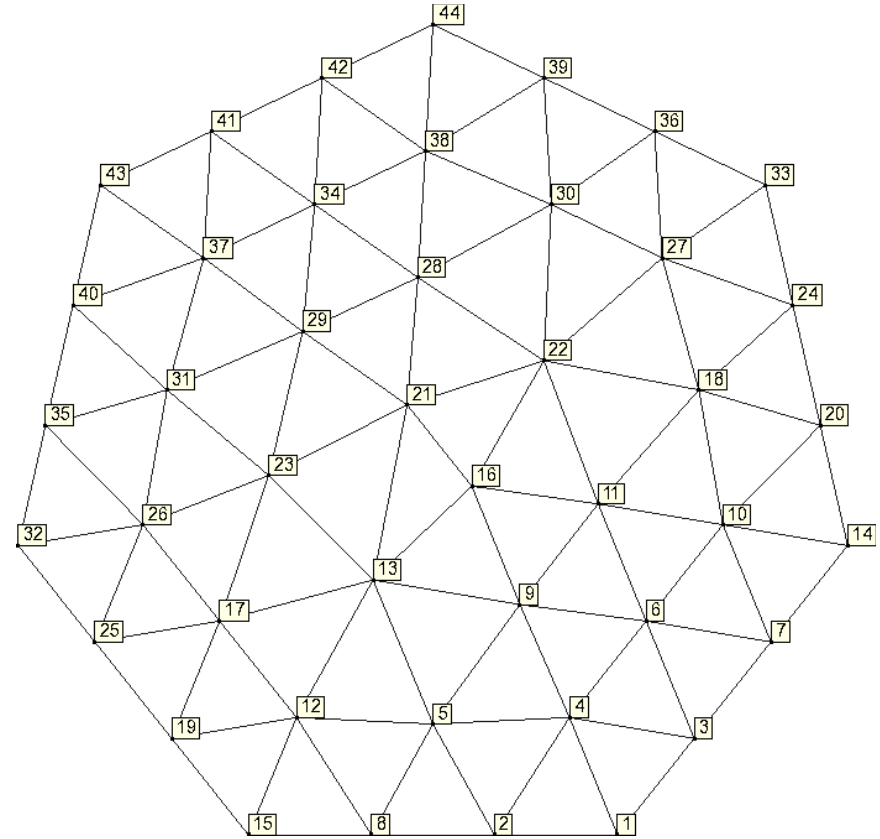
$\mathbf{y}_3 \leftarrow \mathbf{A}_3^r \mathbf{p}_3$

$\mathbf{A}_3^r$ tiene 10 renglones
$\mathbf{p}_3$ tiene 14 entradas
$\mathbf{y}_3$ tiene 10 entradas

La ventaja de esta formulación es que para hacer cada multiplicación matriz-vector no tenemos que enviar todo el vector a multiplicar. Ya que el tamaño máximo del vector está dado por el número de columnas de la partición.

Particionamiento con METIS

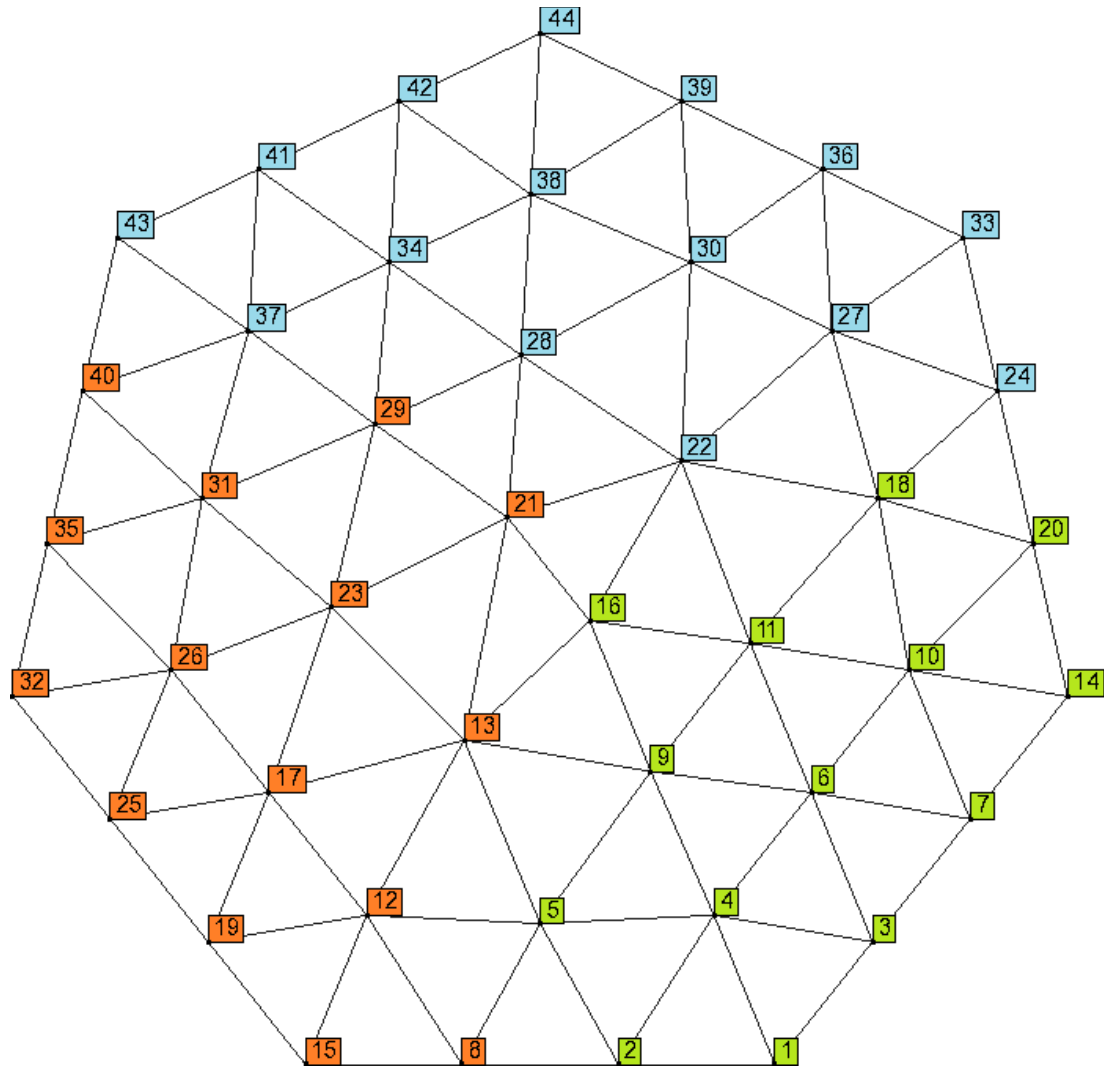





```
int xadj[n + 1] = {1, 4, 8, 12, 18, 24, 30, 34, 38, 44, 50, 56, 62, 69, 72, 75, 80, 86, 92, 96, 100, ...
int adjncy[EDGES] = {2, 3, 4, 1, 4, 5, 8, 1, 4, 6, 7, 1, 2, 3, 5, 6, 9, 2, 4, 8, 9, 12, 13, 3, 4, 7, 9, ...
int wgtflag = 0; // Sin pesos
int numflag = 1;
int nparts = 3; // Numero de particiones
int options[8] = {0, 0, 0, 0, 0, 0, 0, 0};
int edgecut;
int part[44];
METIS_PartGraphKway(&n, xadj, adjncy, NULL, NULL, &wgtflag, &numflag, &nparts, options, &edgecut, part);
```

El resultado es:

part = {1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 3, 2, 3, 2, 2, 3, 3, 2, 3, 2, 2, 3, 3, 2, 3, 3, 3, 3, 2, 3, 3, 3, 3}



La página web de METIS es:
http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

La función que estamos utilizando es:

void **METIS_PartGraphKway**(int* *n*, idxtype* *xadj*, idxtype* *adjncy*, idxtype* *vwgt*, idxtype* *adjwgt*, int* *wgtflag*, int* *numflag*, int* *nparts*, int* *options*, int* *edgecut*, idxtype* *part*)

Los parámetros son:

n	Número de vértices en el grafo
xadj	Arreglo de posición de inicio de las adyacencias
adjncy	Arreglo con las adyacencias
vwgt	Parámetro para los pesos (NULL)
adjwgt	Parámetro para los pesos (NULL)
wgtflag	Indicador de uso de pesos (0)
numflag	Indica si los vértices estan numerados iniciando en 0 o 1
nparts	Número de particiones
options	Arreglo de 5 enteros con parámetos para el algoritmo ({0, 0, 0, 0, 0})
edgecut	Regresa el número de aristas cortadas en el particionamiento.
part	Arreglo con la asignación de partición para cada vértice. La numeración comienza en 0 o 1 dependiendo de numflag.

Código de ejemplo completo

```
extern "C"
{
  #include <metis.h>
}
#include <stdio.h>

#define N 44
#define NPARTS 3
#define EDGES 216

int main()
{
  int n = N;

  int xadj[N + 1] = {1, 4, 8, 12, 18, 24, 30, 34, 38, 44, 50, 56, 62, 69, 72, 75, 80, 86, 92, 96, 100, 106, 113, 119, 123, 127, 133, 139, 145, 151, 157, 163, 166, 169,
175, 179, 183, 189, 195, 199, 203, 207, 211, 214, 217};
  int adjncy[EDGES] = {2, 3, 4, 1, 4, 5, 8, 1, 4, 6, 7, 1, 2, 3, 5, 6, 9, 2, 4, 8, 9, 12, 13, 3, 4, 7, 9, 10, 11, 3, 6, 10, 14, 2, 5, 12, 15, 4, 5, 6, 11, 13, 16, 6, 7, 11, 14, 18,
20, 6, 9, 10, 16, 18, 22, 5, 8, 13, 15, 17, 19, 5, 9, 12, 16, 17, 21, 23, 7, 10, 20, 8, 12, 19, 9, 11, 13, 21, 22, 12, 13, 19, 23, 25, 26, 10, 11, 20, 22, 24, 27, 12, 15, 17,
25, 10, 14, 18, 24, 13, 16, 22, 23, 28, 29, 11, 16, 18, 21, 27, 28, 30, 13, 17, 21, 26, 29, 31, 18, 20, 27, 33, 17, 19, 26, 32, 17, 23, 25, 31, 32, 35, 18, 22, 24, 30, 33,
36, 21, 22, 29, 30, 34, 38, 21, 23, 28, 31, 34, 37, 22, 27, 28, 36, 38, 39, 23, 26, 29, 35, 37, 40, 25, 26, 35, 24, 27, 36, 28, 29, 37, 38, 41, 42, 26, 31, 32, 40, 27, 30,
33, 39, 29, 31, 34, 40, 41, 43, 28, 30, 34, 39, 42, 44, 30, 36, 38, 44, 31, 35, 37, 43, 34, 37, 42, 43, 34, 38, 41, 44, 37, 40, 41, 38, 39, 42};
  int wgtflag = 0; // Sin pesos
  int numflag = 1;
  int nparts = NPARTS; // Numero de particiones
  int options[8] = {0, 0, 0, 0, 0, 0, 0, 0};
  int edgecut;
  int part[N];
  METIS_PartGraphKway(&n, xadj, adjncy, NULL, NULL, &wgtflag, &numflag, &nparts, options, &edgecut, part);
  for (int p = 1; p <= NPARTS; ++p)
  {
    printf("\nPartition %i:\n", p);
    for (int i = 0; i < N; ++i)
    {
      if (part[i] == p)
        printf("%i,", i + 1);
    }
    printf("\n");
  }
  return 0;
}
```

Como instalar METIS desde el código fuente

Bajar el código fuente de METIS de:
http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/OLD/metis-4.0.3.tar.gz

Ir al directorio donde se bajó METIS y ejecutar:

```
tar xzf metis-4.0.3.tar.gz
cd metis-4.0.3
make
mkdir -p ~/metis
cp Lib/*.h ~/metis/.
cp libmetis.a ~/metis/.
```

Esto compila e instala el METIS en su directorio *home*.

Para compilar el ejemplo anterior hay que usar:

```
g++ -o test -I ~/metis test.cpp -L ~/metis -lmetis
```

Los parámetos extra son:

-I ~/metis	Dónde buscar los headers de metis
-L ~/metis	Dónde buscar la librería de metis
-lmetis	Enlazar la librería metis con el código

Permutación del sistema de ecuaciones

Para reordenar **A** y **b**, hay que crear los vectores perm e iperm.

El algoritmo para llenar *perm* es:

```
c ← 0
para p ← 1, 2, ..., nparts
    para i ← 1, 2, ..., n
        si part_i = p
            perm_c ← i
            c ← c + 1
        fin_si
    fin_para
fin_para
```

Para *iperm* el algoritmo es:

```
para i ← 1, 2, ..., n
    k ← perm_i
    iperm_k ← i
fin_para
```

A partir de *perm* e *iperm* podemos obtener $\mathbf{A}^{\mathrm{r}} = \mathbf{P}\,\mathbf{A}\,\mathbf{P}^{\mathrm{T}}$ con el siguiente algoritmo:

para $i \leftarrow 1, 2, \ldots, n$
  $j \leftarrow perm_i$
  Reservar espacio en $\boldsymbol{J}_i\left(\mathbf{A}^{\mathrm{r}}\right)$, tal que $\left|\boldsymbol{J}_i\left(\mathbf{A}^{\mathrm{r}}\right)\right| = \left|\boldsymbol{J}_j\left(\mathbf{A}\right)\right|$
  para $k \leftarrow 1, 2 \ldots \left|\boldsymbol{J}_j\left(\mathbf{A}\right)\right|$
    $l \leftarrow \boldsymbol{J}_j^k\left(\mathbf{A}\right)$
    $m \leftarrow iperm_l$
    $\boldsymbol{J}_i^k\left(\mathbf{A}^{\mathrm{r}}\right) \leftarrow m$
    $\boldsymbol{V}_i^k\left(\mathbf{A}^{\mathrm{r}}\right) \leftarrow \boldsymbol{V}_j^k\left(\mathbf{A}\right)$
  fin_para
  Los índices de $\boldsymbol{J}_i\left(\mathbf{A}^{\mathrm{r}}\right)$ no estarán en orden ascendente, reordenar $\boldsymbol{J}_i\left(\mathbf{A}^{\mathrm{r}}\right)$ y $\boldsymbol{V}_i\left(\mathbf{A}^{\mathrm{r}}\right)$.
fin_para

Hay que reordenar $\mathbf{b}$ para obtener un vector $\mathbf{b}^{\mathrm{r}} = \mathbf{P}\,\mathbf{b}$.

para $i \leftarrow 1, 2 \ldots n$
  $j \leftarrow perm_i$
  $b_i^{\mathrm{r}} \leftarrow b_j$
fin_para

Podemos entonces resolver el sistema

$$\mathbf{A}^{\mathrm{r}}\,\mathbf{x}^{\mathrm{r}}=\mathbf{b}^{\mathrm{r}}.$$

Finalmente hay que reordenar $\mathbf{x}^{\mathrm{r}}$ con la permutación inversa para obtener $\mathbf{x}=\mathbf{P}^{\mathrm{T}}\,\mathbf{x}^{\mathrm{r}}$.

para $i \leftarrow 1, 2 \ldots n$
    $j \leftarrow iperm_i$
    $x_i \leftarrow x^{\mathrm{r}}_j$
fin_para

Pseudocódigo

El pseudocódigo del programa en MPI sería:

MPI_Init(); MPI_Comm_size(size); MPI_Comm_rank(rank);	Inicializar MPI

if (rank == 0)
{
Load(**A**, **b**); // Leer matriz A, leer vector b
n = size(**b**) // Sea n el número de incógnitas
num_slaves = size - 1; // Sea num_slaves el número de esclavos
	Inicia sección para maestro

part = **METIS_PartGraphKway** // Generar *perm* e *iperm* (ver algoritmo)	Particionar el grafo de la matriz A y generar los vectores perm e iperm
$\mathbf{A}^r = \mathbf{P}\,\mathbf{A}\,\mathbf{P}^t$ // Utilizando el algoritmo con *perm* e *iperm* $\mathbf{b}^r = \mathbf{P}\,\mathbf{b}$ // Utilizando el algoritmo con *perm*	Crear $\mathbf{A}^r$ y $\mathbf{b}^r$

for (s = 1; s <= num_slaves; ++s)
{
MPI_Send(n, 1, s); // Enviar el tamaño del sistema de ecuaciones
MPI_Send(row_min, 1, s); // Enviar los rangos de índices de renglón
MPI_Send(row_max, 1, s); // Enviar los rangos de índices de renglón
MPI_Send(col_min, 1, s); // Enviar los rangos de índices de columna
MPI_Send(col_max, 1, s); // Enviar los rangos de índices de columna
num_rows = row_max - row_min + 1;
for (r = 1; r <= num_rows; ++r)
{
MPI_Send(row_size[r], 1, s);
MPI_Send(***row_index***[r], row_size[r], s);
	Enviar un grupo de renglones a cada esclavo

```
    MPI_Send(row_values[r], row_size[r], s);
  }
}
```

```
// Calcular r = A^r*x - b^r
for (s = 1; s <= num_slaves; ++s)
{
  // Formar x_s a partir de x^r con las entradas que le corresponden
  x_s = extract( x^r );
  MPI_Send(x_s, n, s);
}
r = -b^r;
for (s = 1; s <= num_slaves; ++s)
{
  MPI_Recv(y_s, n, s);
  r = r + y_s; // Recordar que y_s es de tamaño reducido
}
p = -r;
```

Calcular residual y dirección de descenso iniciales

```
while (norm(r) > tolerance)
{
  // Calcular w = A^r*p
  for (s = 1; s <= num_slaves; ++s)
  {
    multiplicar = 1;
    MPI_Send(multiplicar, 1, s); // Avisar al esclavo s que sigue una multiplicación
    // Formar p_s a partir de p con las entradas que le corresponden
    p_s = extract( p );
    MPI_Send(p_s , n, s); // Enviar p al esclavo s
  }
  w = 0
  for (s = 1; s <= num_slaves; ++s)
```

Ciclo de gradiente conjugado en el maestro

```
  {
    MPI_Recv(y_s, n, s);
    w = w + y_s; // Recordar que y_s es de tamaño reducido
  }

  // Calcular tamaño de paso y nueva dirección de descenso
  alpha = dot(r, r)/dot(p, w);
  x1 = x^r + alpha*p;
  r1 = r + alpha*w;
  beta = dot(r1, r1)/dot(r, r);
  p1 = -r1 + beta*p;
  x^r = x1;
  r = r1;
  p = p1;
 }
```

Avisarle a los esclavos que no hay más multiplicaciones pendientes

```
 for (s = 1; s <= num_slaves; ++s)
 {
  multiply = 0; // Avisar al esclavo s que terminaron las multiplicacines
  MPI_Send(multiply, 1, s);
 }
```

Obtener la **x** sin reordenamiento

```
 x = P^T x^r; // Utilizar el algoritmo con iperm
 Save(x); // Guardar la solución
}
```

Inicia sección para esclavos

```
else
{
 MPI_Recv(n, 1, 0); // Recibe el tamaño del sistema de ecuaciones
 MPI_Recv(row_min, 1, 0); // Recibir los rangos de índices de renglón
 MPI_Recv(row_max, 1, 0); // Recibir los rangos de índices de renglón
 num_rows = row_max - row_min + 1;
```

Recibir el numero e indice de los renglones

Recibir las los renglones

```
MPI_Recv(col_min, 1, 0); // Recibir los rangos de índices de columna
MPI_Recv(col_max, 1, 0); // Recibir los rangos de índices de columna
num_cols = col_max - col_min + 1;
for (i = 1; i <= num_rows; ++i)
{
  MPI_Recv(row_size[i], 1, 0);
  MPI_Recv(row_index[i], row_size[i], 0);
  MPI_Recv(row_values[i], row_size[i], 0);
}
```

comprimidos

```
MPI_Recv(x, n, 0); // Recibir vector x, que es de tamaño num_cols
y = 0; // Es de tamaño num_rows
for (i = 1; i <= num_rows; ++i)
{
  y_i = 0;
  for (k = 1; k <= row_size[i]; ++k)
  {
    y_i += row_values[i][k]*x[row_index[i][k] - row_min];
  }
}
MPI_Send(y, n, 0); // Enviar vector y
```

Hacer el la multiplicación de los renglones comprimidos por el vector x

```
for ( ; ; )
{
```

Ciclo de multiplicaciones

```
  MPI_Recv(multiply, 1, 0);
  if ( multiply == 0)
    break;
```

Condicion de salida del ciclo

```
  MPI_Recv(p, n, 0); // Recibir vector p, que es de tamaño num_cols
  for (i = 1; i <= num_rows; ++i)
  {
    y_i = 0;
```

Hacer el la multiplicación de los renglones comprimidos por el vector p

```
    for (k = 1; k <= row_size[i]; ++k)
    {
      y_i += row_values[i][k]*p[row_index[i][k] - row_min];
    }
  }
  MPI_Send(y, n, 0); // Enviar vector y
 }
}
MPI_Finalize();
```

Finalizar MPI

¿Preguntas?

miguelvargas@cimat.mx

Referencias

[Noce06] J. Nocedal, S. J. Wright. Numerical Optimization. Springer, 2006.

[MPIF08] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, Version 2.1. University of Tennessee, 2008.