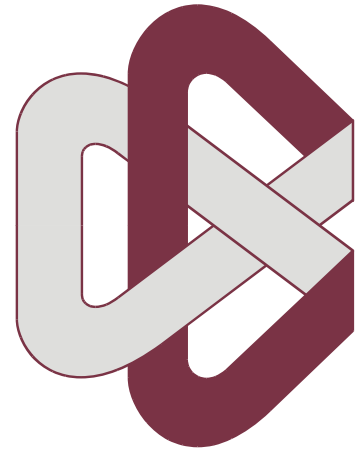


CIMAT

Metodo de subestructuracion de Schur

Miguel Vargas-Félix

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

Representación de una matriz rala como grafo

La idea básica es dividir un sistema de ecuaciones grande en problemas más pequeños que podamos repartir y calcular de forma independiente en paralelo.

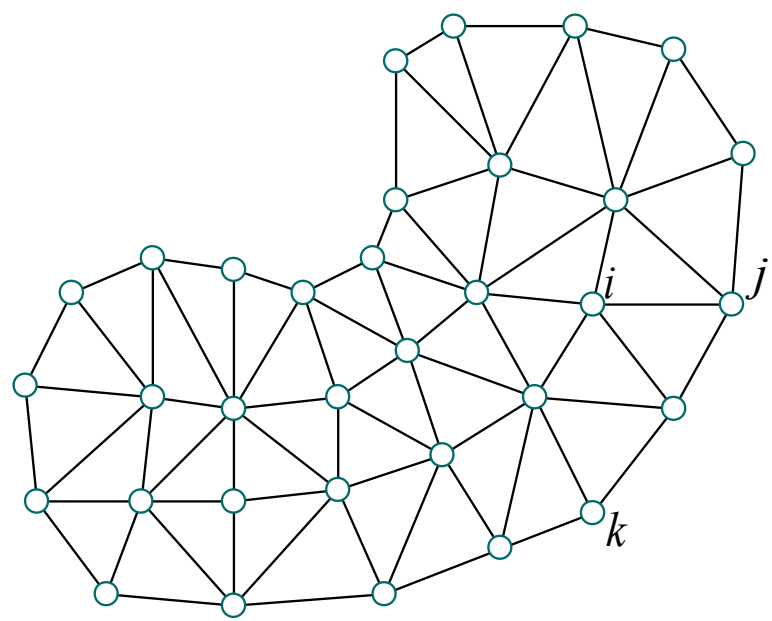
Partimos de un sistema de ecuaciones

$$\mathbf{A}\,\mathbf{x} = \mathbf{b}$$

donde $\mathbf{A}$ es una matriz **rala simétrica** positiva definida de tamaño $n \times n$.

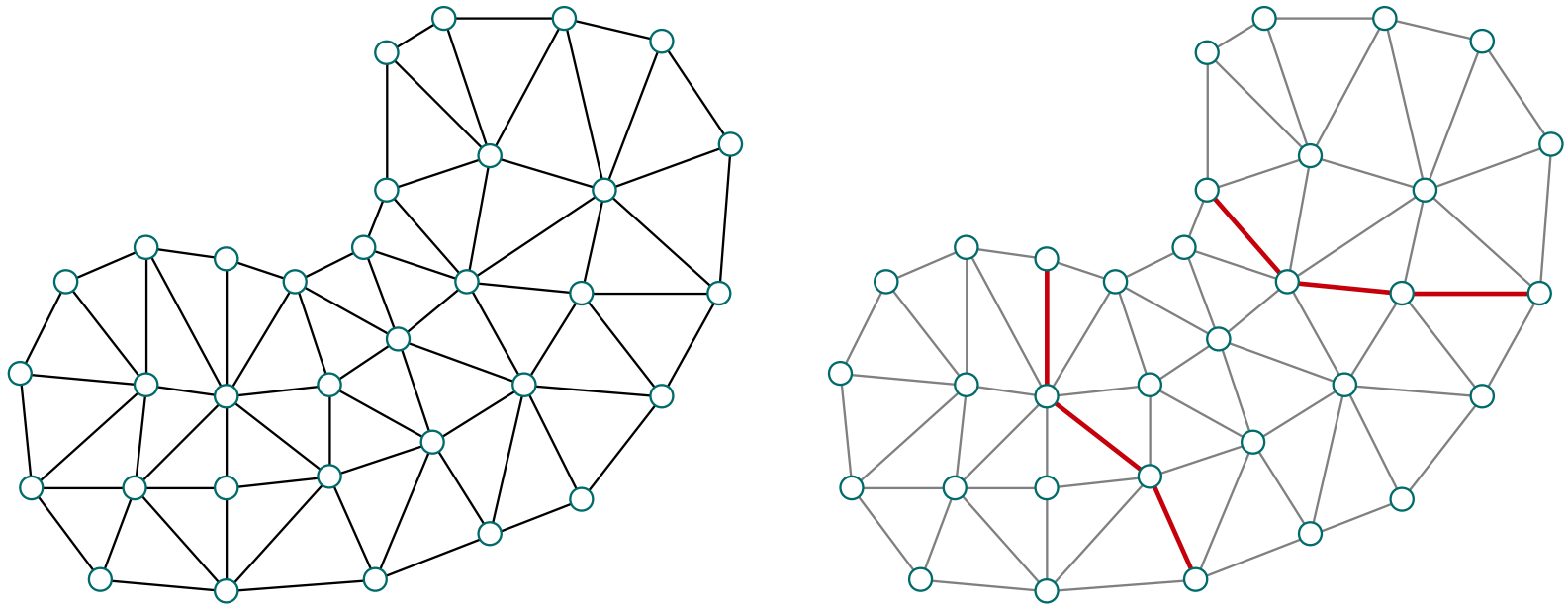
A partir de la estuctura rala de $\mathbf{A}$ podemos crear el grafo que la representa.

$$\mathbf{A} = \begin{pmatrix} \circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\ \circ & a_{ii} & \circ & a_{ij} & \circ & 0 & \circ & \cdots \\ \circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\ \circ & a_{ji} & \circ & a_{jj} & \circ & 0 & \circ & \cdots \\ \circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\ \circ & 0 & \circ & 0 & \circ & a_{kk} & \circ & \cdots \\ \circ & \circ & \circ & \circ & \circ & \circ & \circ & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$



Particionamiento de la matriz del sistema

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

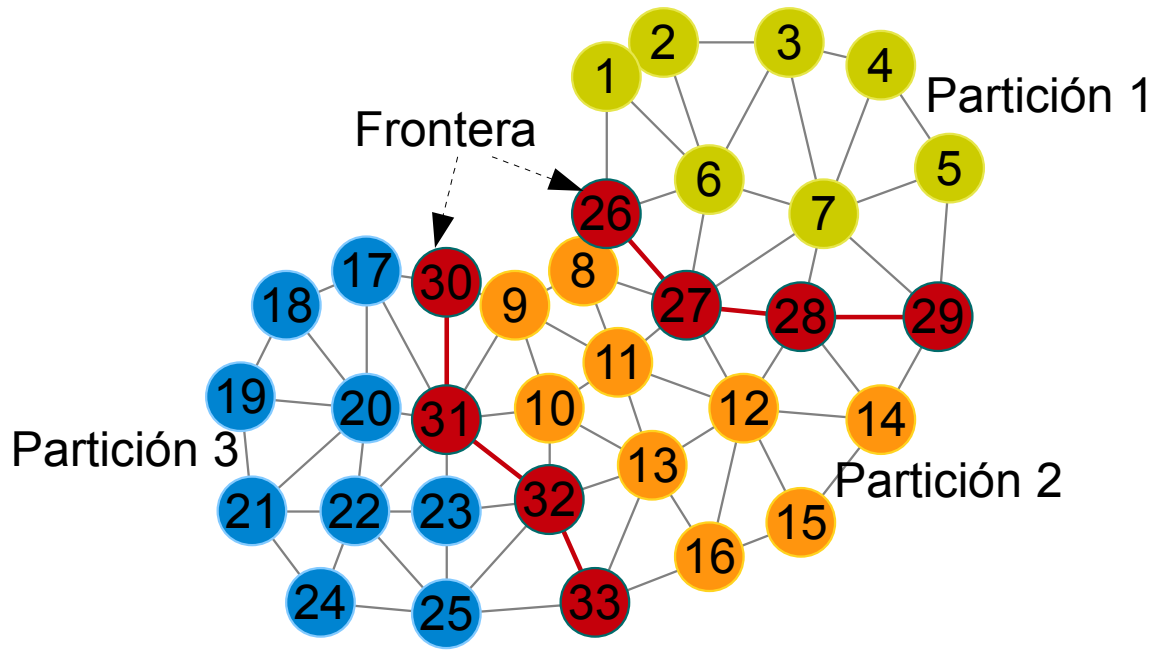


Grafo dividido en tres particiones.

- El particionamiento se hace sobre las aristas del grafo.
- Las particiones comparten nodos en la frontera.
- El particionamiento se hace tratando de poner un igual número de nodos en cada partición.

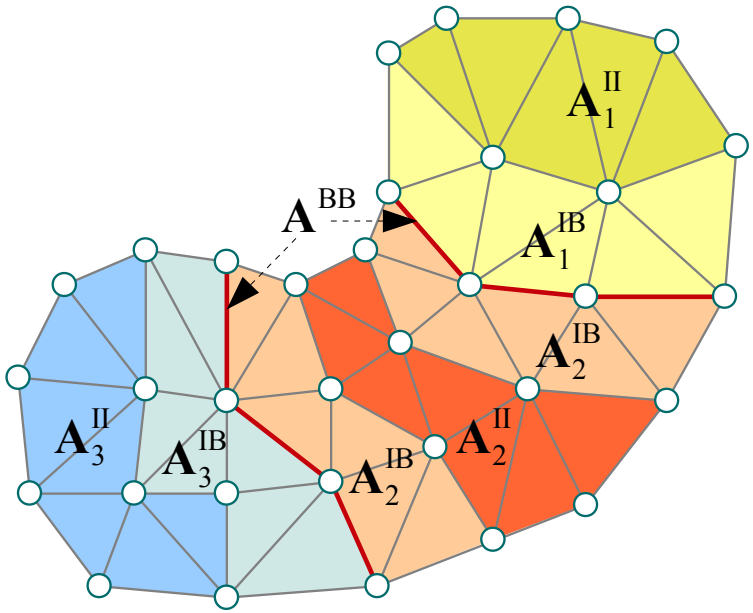
Reenumeración de los nodos

El siguiente paso es re-enumerar los nodos, de tal forma que los nodos de la primer partición tengan los primeros nodos, la segunda partición los siguientes, etc.



Los nodos de la frontera serán los últimos en re-enumerarse.

El efecto que esto produce en la estructura de la matriz es:



$$\begin{pmatrix} A_1^{II} & 0 & 0 & A_1^{IB} \\ 0 & A_2^{II} & 0 & A_2^{IB} \\ 0 & 0 & A_3^{II} & A_3^{IB} \\ A_1^{BI} & A_2^{BI} & A_3^{BI} & A^{BB} \end{pmatrix}$$

Los superíndices II denotan las entradas que capturan la relación entre los nodos en la parte interna de la partición.

BB indican las entradas de la matriz que relacionan los nodos en la frontera.

Finalmente IB y BI indican las entradas con valores que dependen de la relación entre los nodos dentro de la partición y los nodos frontera.

En un sistema de ecuaciónes con p particiones,

$$\begin{pmatrix} \mathbf{A}_1^{\mathrm{II}} & & \mathbf{0} & & & \mathbf{A}_1^{\mathrm{IB}} \\ & \mathbf{A}_2^{\mathrm{II}} & & & & \mathbf{A}_2^{\mathrm{IB}} \\ \mathbf{0} & & \mathbf{A}_3^{\mathrm{II}} & & & \mathbf{A}_3^{\mathrm{IB}} \\ \vdots & & & \ddots & & \vdots \\ & & & & \mathbf{A}_p^{\mathrm{II}} & \mathbf{A}_p^{\mathrm{IB}} \\ \mathbf{A}_1^{\mathrm{BI}} & \mathbf{A}_2^{\mathrm{BI}} & \mathbf{A}_3^{\mathrm{BI}} & \cdots & \mathbf{A}_p^{\mathrm{BI}} & \mathbf{A}^{\mathrm{BB}} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1^{\mathrm{I}} \\ \mathbf{x}_2^{\mathrm{I}} \\ \mathbf{x}_3^{\mathrm{I}} \\ \vdots \\ \mathbf{x}_p^{\mathrm{I}} \\ \mathbf{x}^{\mathrm{B}} \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1^{\mathrm{I}} \\ \mathbf{b}_2^{\mathrm{I}} \\ \mathbf{b}_3^{\mathrm{I}} \\ \vdots \\ \mathbf{b}_p^{\mathrm{I}} \\ \mathbf{b}^{\mathrm{B}} \end{pmatrix}.$$

Los superíndices I denotan valores en los nodos dentro de la partición.

Los superíndices B indican valores en los nodos de la frontera.

Juntando las matrices que tienen nodos en la partición i-ésima, con sus correspondientes vectores, podemos formar el sistema de ecuaciones

$$\begin{pmatrix} \mathbf{A}_i^{\mathrm{II}} & \mathbf{A}_i^{\mathrm{IB}} \\ \mathbf{A}_i^{\mathrm{BI}} & \mathbf{A}^{\mathrm{BB}} \end{pmatrix} \begin{pmatrix} \mathbf{x}_i^{\mathrm{I}} \\ \mathbf{x}^{\mathrm{B}} \end{pmatrix} = \begin{pmatrix} \mathbf{b}_i^{\mathrm{I}} \\ \mathbf{b}^{\mathrm{B}} \end{pmatrix}, \; i = 1 \ldots p.$$

El método de complemento de Schur

Para cada partición i,

$$\begin{pmatrix} \mathbf{A}_i^{\mathrm{II}} & \mathbf{A}_i^{\mathrm{IB}} \\ \mathbf{A}_i^{\mathrm{BI}} & \mathbf{A}^{\mathrm{BB}} \end{pmatrix} \begin{pmatrix} \mathbf{x}_i^{\mathrm{I}} \\ \mathbf{x}^{\mathrm{B}} \end{pmatrix} = \begin{pmatrix} \mathbf{b}_i^{\mathrm{I}} \\ \mathbf{b}^{\mathrm{B}} \end{pmatrix},$$

trabajando por bloques podemos despejar el vector de incógnitas $\mathbf{x}_i^{\mathrm{I}}$ a partir de

$$\mathbf{A}_i^{\mathrm{II}} \mathbf{x}_i^{\mathrm{I}} + \mathbf{A}_i^{\mathrm{IB}} \mathbf{x}^{\mathrm{B}} = \mathbf{b}_i^{\mathrm{I}},$$

así

$$\mathbf{x}_i^{\mathrm{I}} = \left(\mathbf{A}_i^{\mathrm{II}} \right)^{-1} \left(\mathbf{b}_i^{\mathrm{I}} - \mathbf{A}_i^{\mathrm{IB}} \mathbf{x}^{\mathrm{B}} \right).$$

Aplicando el método de reducción gaussiana por bloques en

$$\begin{pmatrix} \mathbf{A}_1^{\mathrm{II}} & & \mathbf{0} & & & \mathbf{A}_1^{\mathrm{IB}} \\ & \mathbf{A}_2^{\mathrm{II}} & & & & \mathbf{A}_2^{\mathrm{IB}} \\ \mathbf{0} & & \mathbf{A}_3^{\mathrm{II}} & & & \mathbf{A}_3^{\mathrm{IB}} \\ \vdots & & & \ddots & & \vdots \\ & & & & \mathbf{A}_p^{\mathrm{II}} & \mathbf{A}_p^{\mathrm{IB}} \\ \mathbf{A}_1^{\mathrm{BI}} & \mathbf{A}_2^{\mathrm{BI}} & \mathbf{A}_3^{\mathrm{BI}} & \cdots & \mathbf{A}_p^{\mathrm{BI}} & \mathbf{A}^{\mathrm{BB}} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1^{\mathrm{I}} \\ \mathbf{x}_2^{\mathrm{I}} \\ \mathbf{x}_3^{\mathrm{I}} \\ \vdots \\ \mathbf{x}_p^{\mathrm{I}} \\ \mathbf{x}^{\mathrm{B}} \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1^{\mathrm{I}} \\ \mathbf{b}_2^{\mathrm{I}} \\ \mathbf{b}_3^{\mathrm{I}} \\ \vdots \\ \mathbf{b}_p^{\mathrm{I}} \\ \mathbf{b}^{\mathrm{B}} \end{pmatrix},$$

multiplicando el renglón 1 por $\mathbf{A}_1^{\mathrm{BI}}\left(\mathbf{A}_1^{\mathrm{II}}\right)^{-1}$ y restándolo al último renglon,

$$\begin{pmatrix} \mathbf{A}_1^{\mathrm{II}} & & \mathbf{0} & & & \mathbf{A}_1^{\mathrm{IB}} \\ & \mathbf{A}_2^{\mathrm{II}} & & & & \mathbf{A}_2^{\mathrm{IB}} \\ \mathbf{0} & & \mathbf{A}_3^{\mathrm{II}} & & & \mathbf{A}_3^{\mathrm{IB}} \\ \vdots & & & \ddots & & \vdots \\ & & & & \mathbf{A}_p^{\mathrm{II}} & \mathbf{A}_p^{\mathrm{IB}} \\ 0 & \mathbf{A}_2^{\mathrm{BI}} & \mathbf{A}_3^{\mathrm{BI}} & \cdots & \mathbf{A}_p^{\mathrm{BI}} & \mathbf{A}^{\mathrm{BB}} - \mathbf{A}_1^{\mathrm{BI}}\left(\mathbf{A}_1^{\mathrm{II}}\right)^{-1}\mathbf{A}_1^{\mathrm{IB}} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1^{\mathrm{I}} \\ \mathbf{x}_2^{\mathrm{I}} \\ \mathbf{x}_3^{\mathrm{I}} \\ \vdots \\ \mathbf{x}_p^{\mathrm{I}} \\ \mathbf{x}^{\mathrm{B}} \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1^{\mathrm{I}} \\ \mathbf{b}_2^{\mathrm{I}} \\ \mathbf{b}_3^{\mathrm{I}} \\ \vdots \\ \mathbf{b}_p^{\mathrm{I}} \\ \mathbf{b}^{\mathrm{B}} - \mathbf{A}_1^{\mathrm{BI}}\left(\mathbf{A}_1^{\mathrm{II}}\right)^{-1}\mathbf{b}_1^{\mathrm{I}} \end{pmatrix}.$$

Si repetimos esta operación p veces para reducir los primeros p renglones, entonces la ecuación del último renglón quedará como

$$\left(\mathbf{A}^{\mathrm{BB}}-\sum_{i=1}^{p}\mathbf{A}_i^{\mathrm{BI}}\left(\mathbf{A}_i^{\mathrm{II}}\right)^{-1}\mathbf{A}_i^{\mathrm{IB}}\right)\mathbf{x}^{\mathrm{B}}=\mathbf{b}^{\mathrm{B}}-\sum_{i=1}^{p}\mathbf{A}_i^{\mathrm{BI}}\left(\mathbf{A}_i^{\mathrm{II}}\right)^{-1}\mathbf{b}_i^{\mathrm{I}}.$$

Éste es un sistema de ecuaciones, en el cual las incógintas son las $\mathbf{x}^{\mathrm{B}}$, todos los demás términos son conocidos.

En vez de resolver un sistema de ecuaciones grande, vamos a resolver un sistema de ecuaciones con el número de variables igual al número de nodos en la frontera (encontrar $\mathbf{x}^{\mathrm{B}}$).

Una vez que el vector $\mathbf{x}^{\mathrm{B}}$ es calculado, podemos encontrar los valores en los vectores internos $\mathbf{x}_i^{\mathrm{I}}$ utilizando la ecuación determinada anteriormente

$$\mathbf{x}_i^{\mathrm{I}}=\left(\mathbf{A}_i^{\mathrm{II}}\right)^{-1}\left(\mathbf{b}_i^{\mathrm{I}}-\mathbf{A}_i^{\mathrm{IB}}\mathbf{x}^{\mathrm{B}}\right).$$

Vamos a ver que no es necesario calcular la inversa $\left(\mathbf{A}_1^{\mathrm{II}}\right)^{-1}$ explícitamente.

Definamos

$$\bar{\mathbf{A}}_i^{\mathrm{BB}} = \mathbf{A}_i^{\mathrm{BI}} \left(\mathbf{A}_i^{\mathrm{II}} \right)^{-1} \mathbf{A}_i^{\mathrm{IB}},$$

para calcularla factorizaremos con Cholesky $\mathbf{A}_i^{\mathrm{II}}$, y procederemos columna por columna [Sori00], utilizando un vector extra $\mathbf{t}$, y resolviendo para $c = 1 \ldots n$

$$\mathbf{A}_i^{\mathrm{II}} \mathbf{t} = \left[\mathbf{A}_i^{\mathrm{IB}} \right]_c,$$

note que muchas columnas $\left[\mathbf{A}_i^{\mathrm{IB}} \right]_c$ son nulas. Podemos así completar $\mathbf{A}_i^{\mathrm{BB}}$ con,

$$\left[\bar{\mathbf{A}}_i^{\mathrm{BB}} \right]_c = \mathbf{A}_i^{\mathrm{BI}} \mathbf{t}.$$

Ahora definamos $\bar{\mathbf{b}}_i^{\mathrm{B}} = \mathbf{A}_i^{\mathrm{BI}} \left(\mathbf{A}_i^{\mathrm{II}} \right)^{-1} \mathbf{b}_i^{\mathrm{I}}$, en este caso sólo resolvemos un sistema de ecuaciones

$$\mathbf{A}_i^{\mathrm{II}} \mathbf{t} = \mathbf{b}_i^{\mathrm{I}},$$

entonces

$$\bar{\mathbf{b}}_i^{\mathrm{B}} = \mathbf{A}_i^{\mathrm{BI}} \mathbf{t}.$$

Tanto $\bar{\mathbf{A}}_i^{\mathrm{BB}}$ como $\bar{\mathbf{b}}_i^{\mathrm{B}}$ tienen la contribución de cada partición, éstas pueden ser escritas como

$$\left(\mathbf{A}^{\mathrm{BB}} - \sum_{i=1}^{p} \bar{\mathbf{A}}_i^{\mathrm{BB}} \right) \mathbf{x}^{\mathrm{B}} = \mathbf{b}^{\mathrm{B}} - \sum_{i=1}^{p} \bar{\mathbf{b}}_i^{\mathrm{B}},$$

éste es el sistema que tiene que ser resuelto.

La paralelización consistirá en que cada proceso de MPI calcule sus respectivos $\bar{\mathbf{A}}_i^{\mathrm{BB}}$ y $\bar{\mathbf{b}}_i^{\mathrm{B}}$.

Dado que $\mathbf{A}_i^{\mathrm{II}}$ es rala y su inversa tiene que utilizarse muchas veces, lo más eficiente es utilizar la factorización Cholesky parar matrices ralas en $\mathbf{A}_i^{\mathrm{II}}$.

En el sistema

$$\left(\mathbf{A}^{\mathrm{BB}} - \sum_{i=1}^{p} \bar{\mathbf{A}}_i^{\mathrm{BB}}\right)\mathbf{x}^{\mathrm{B}} = \mathbf{b}^{\mathrm{B}} - \sum_{i=1}^{p} \bar{\mathbf{b}}_i^{\mathrm{B}},$$

$\mathbf{A}_i^{\mathrm{II}}$ es rala, pero las $\bar{\mathbf{A}}_i^{\mathrm{BB}}$ no lo son. Para resolver este sistema de ecuaciones una versión distribuida del gradiente conjugado es necesaria (con MPI).

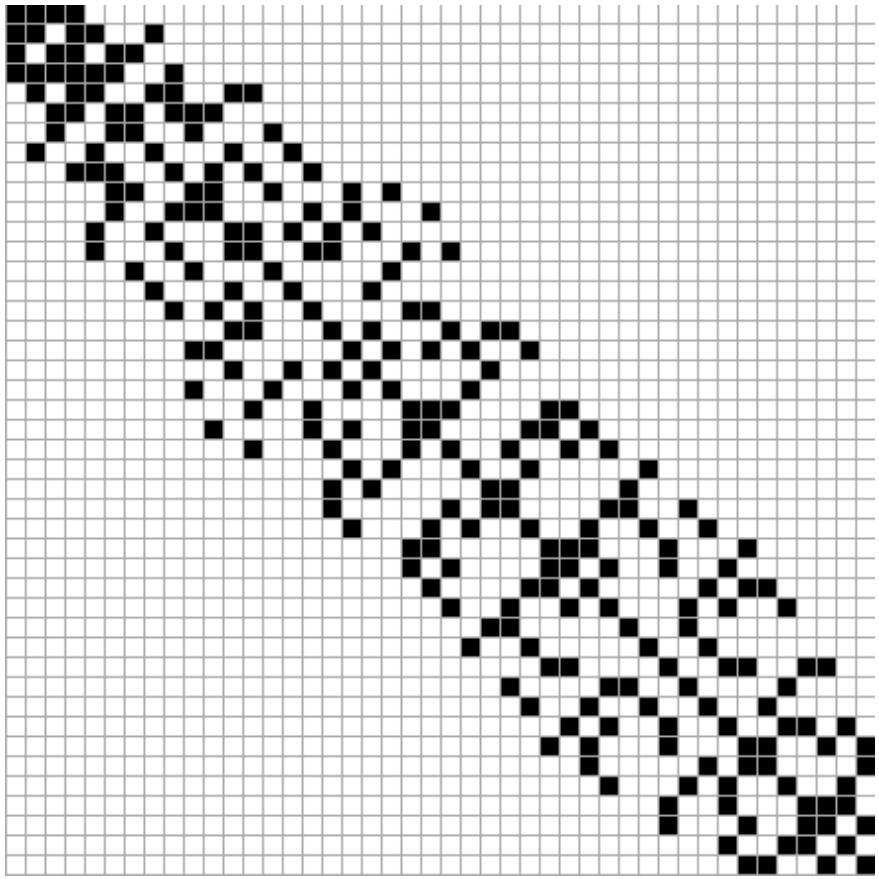
En el gradiente conjugado sólo es importante saber cómo multiplicar la matriz por la dirección de descenso $\mathbf{p}$, en esta implementación cada $\bar{\mathbf{A}}_i^{\mathrm{BB}}$ es mantenida en cada nodo esclavo y la multiplicación

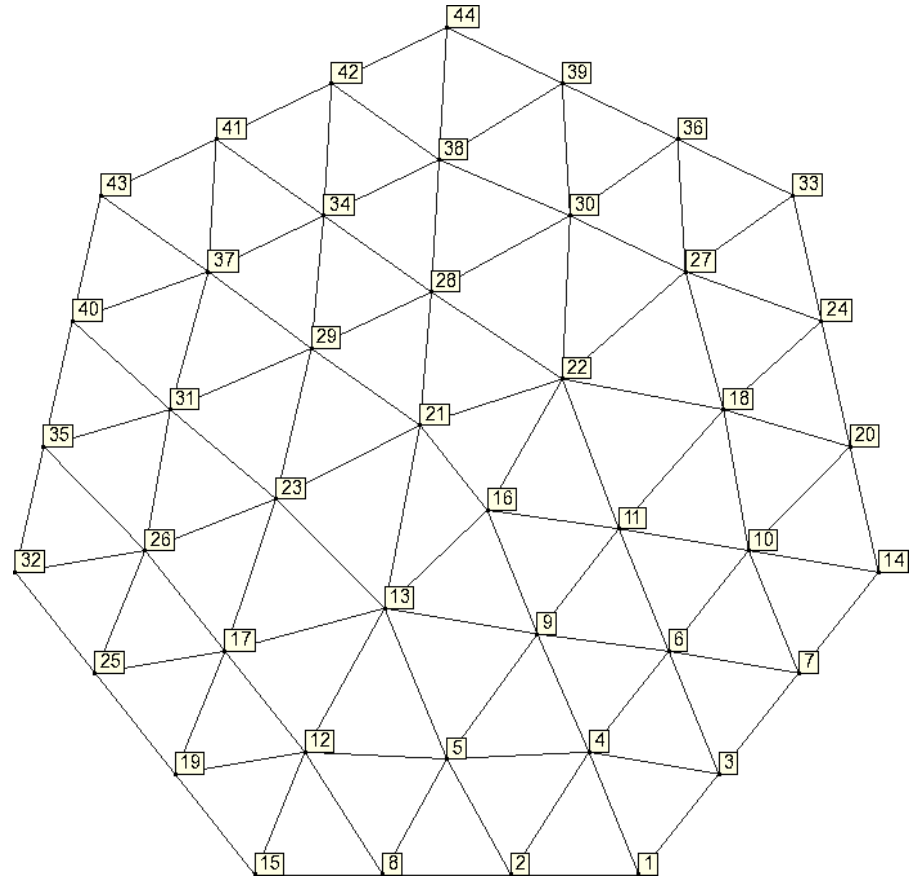
$$\mathbf{w} = \left(\mathbf{A}^{\mathrm{BB}} - \sum_{i=1}^{p} \bar{\mathbf{A}}_i^{\mathrm{BB}}\right)\mathbf{p}, \text{ es decir } \mathbf{w} = \left(\mathbf{A}^{\mathrm{BB}}\mathbf{p}\right) - \left(\bar{\mathbf{A}}_1^{\mathrm{BB}}\mathbf{p}\right) - \left(\bar{\mathbf{A}}_2^{\mathrm{BB}}\mathbf{p}\right) - \left(\bar{\mathbf{A}}_3^{\mathrm{BB}}\mathbf{p}\right) - \ldots - \left(\bar{\mathbf{A}}_p^{\mathrm{BB}}\mathbf{p}\right),$$

es realizada de forma separada, los resultados individuales de multiplicar $\bar{\mathbf{A}}_i^{\mathrm{BB}}\mathbf{p}$ en cada nodo serán acumulados en el nodo maestro.

Un beneficio de este método es que al resolver para $\mathbf{x}^{\mathrm{B}}$ se trabaja con un sistema reducido que tiene un número de condición menor, lo que reduce el número de iteraciones para converger.

Particionamiento de matrices ralas/grafos con METIS

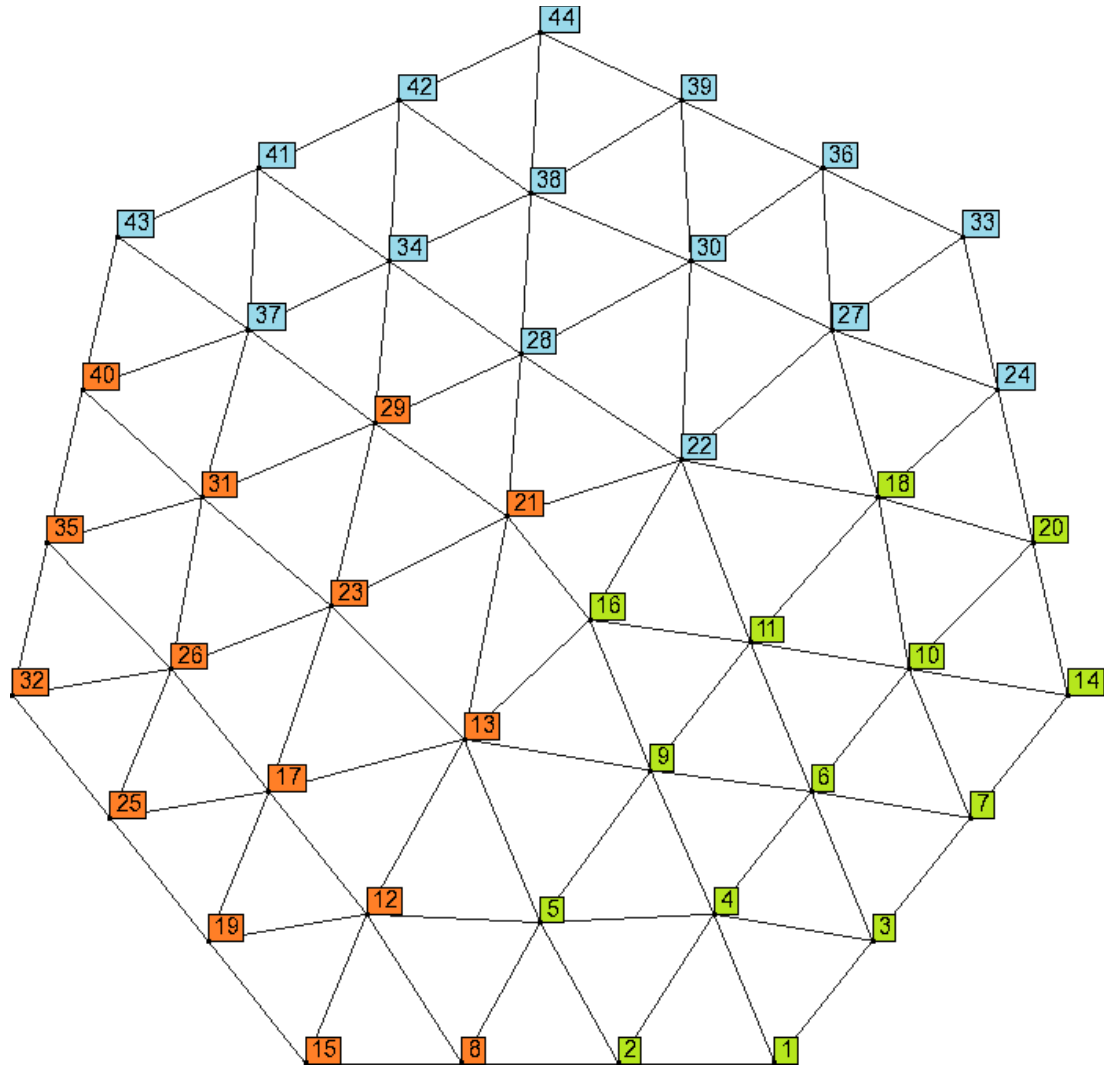





```
int xadj[N + 1] = {1, 4, 8, 12, 18, 24, 30, 34, 38, 44, 50, 56, 62, 69, 72, 75, 80, 86, 92, 96, 100, ...
int adjncy[EDGES] = {2, 3, 4, 1, 4, 5, 8, 1, 4, 6, 7, 1, 2, 3, 5, 6, 9, 2, 4, 8, 9, 12, 13, 3, 4, 7, 9, ...
int wgtflag = 0; // Sin pesos
int numflag = 1;
int nparts = 3; // Numero de particiones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

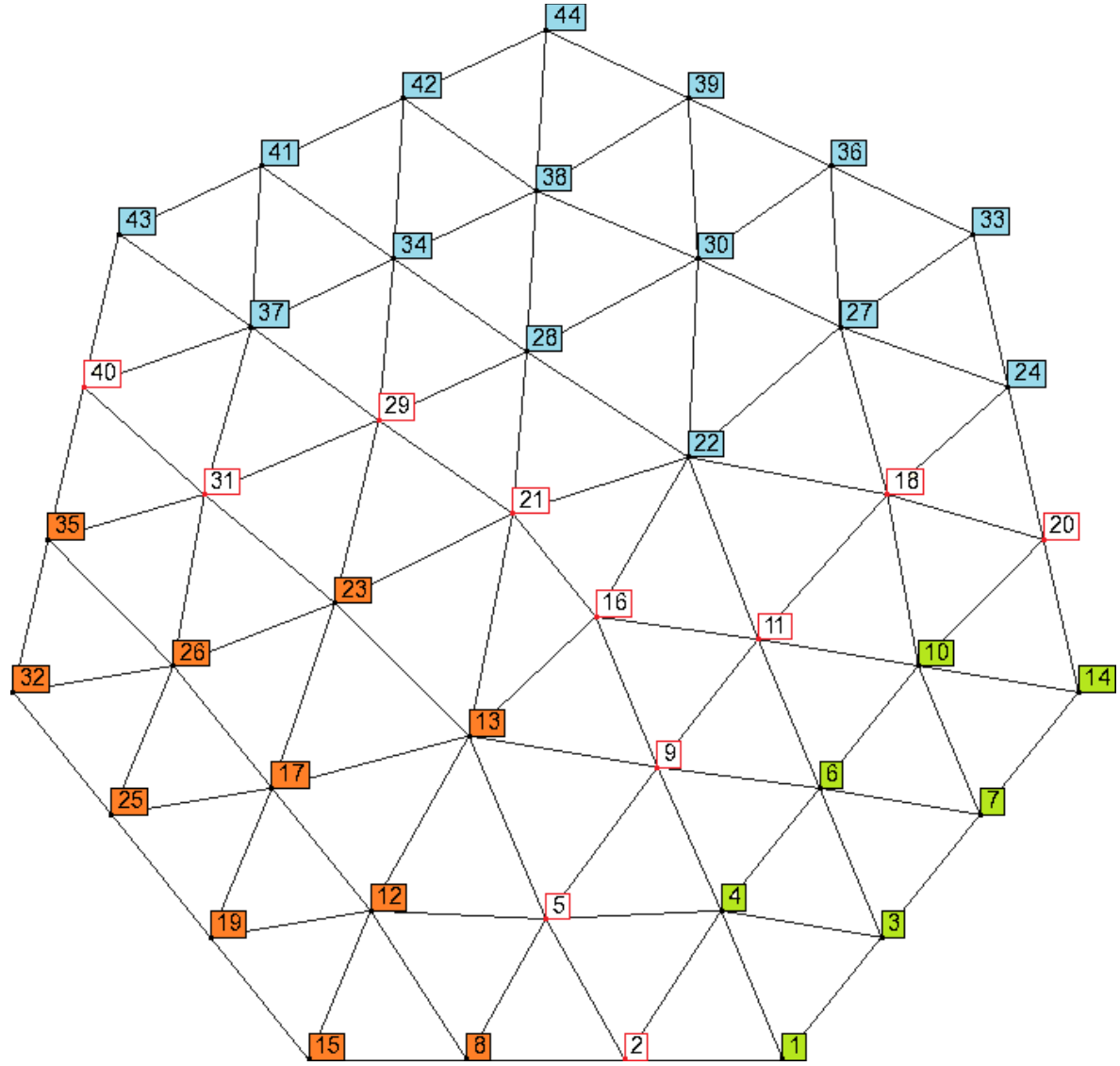
El resultado es:

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}



Ahora necesitamos detefinir nodos que serán frontera. Un algoritmo puede ser el siguiente

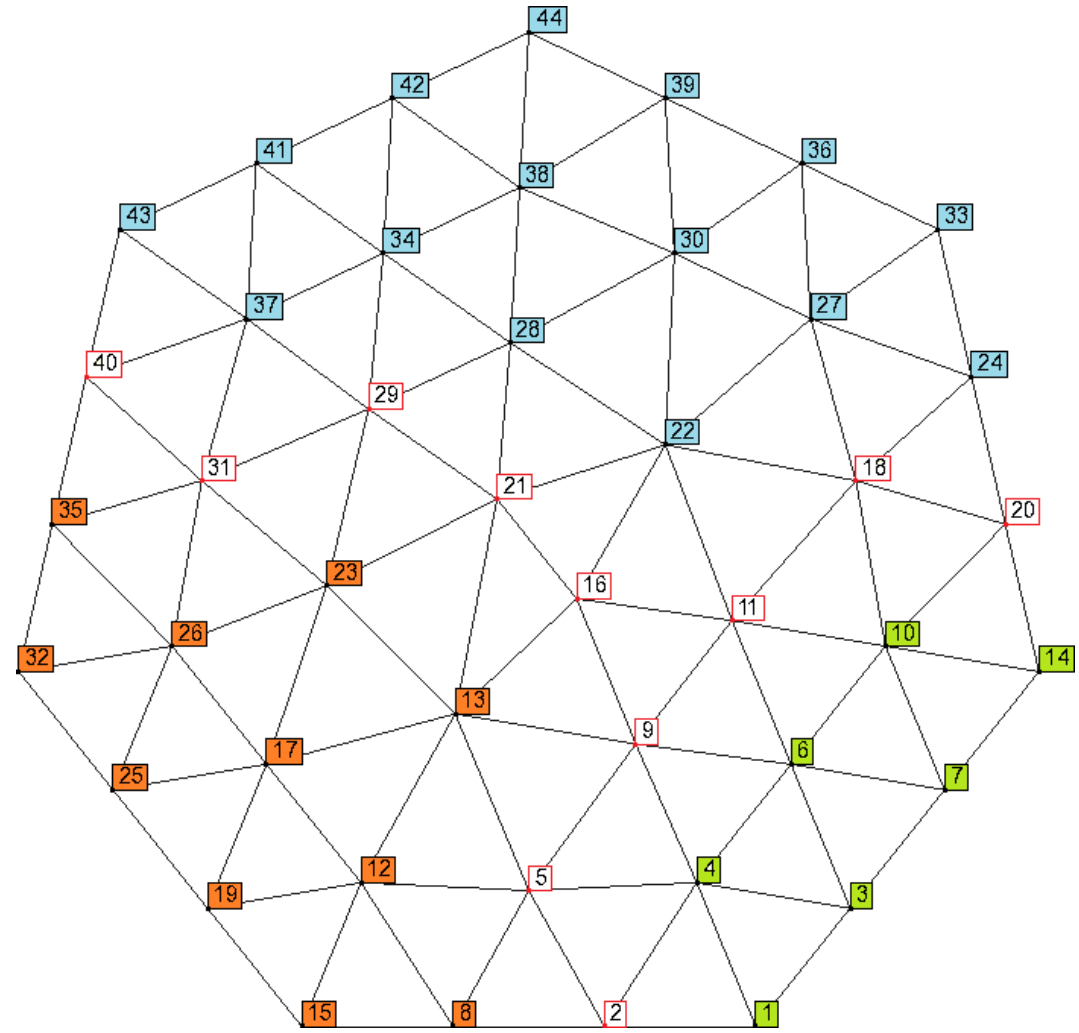
```
bool is_boundary[44] = {false};
for (int i = 0; i < n; ++i)
{
    int start = xadj[i] - 1;
    int size = xadj[i + 1] - xadj[i];
    int p = part[i];
    for (int j = 0; j < size; ++j)
    {
        int a = adjncy[start + j] - 1;
        int q = part[a];
        if (p < q)
        {
            is_boundary[i] = true;
            break;
        }
    }
}
```



El último paso del particionamiento es separar los índices para cada partición

```
int II[NPARTS][N];
int IB[NPARTS][N];
int BB[N];
int size_II[NPARTS] = {0};
int size_IB[NPARTS] = {0};
int size_BB = 0;

for (int i = 0; i < N; ++i) {
  if (is_boundary[i]) {
    BB[size_BB++] = i + 1;
    int start = xadj[i] - 1;
    int size = xadj[i + 1] - xadj[i];
    for (int j = 0; j < size; ++j) {
      int a = adjncy[start + j] - 1;
      if (!is_boundary[a]) {
        int p = part[a] - 1;
        if (size_IB[p] > 0)
          if (IB[p][size_IB[p] - 1] == i + 1) continue;
        IB[p][size_IB[p]++] = i + 1;
      }
    }
  }
  else {
    int p = part[i] - 1;
    II[p][size_II[p]++] = i + 1;
  }
}
```



El código completo lo encuentran al final de la presentación.

Generar las submatrices

Es necesario generar $\mathbf{A}_i^{\mathrm{II}}$, $\mathbf{A}_i^{\mathrm{IB}}$ y $\mathbf{b}_i^{\mathrm{I}}$ para cada partición a partir de la $\mathbf{A}$ y $\mathbf{b}$ originales. Esto se hace a partir de los índices obtenidos con el algoritmo anterior.

Para generar el vector $\mathbf{b}_i^{\mathrm{I}}$:

Sean $\left(\mathrm{II}_i\right)_k$ los subíndices de los nodos internos para la partición i, con $k = 1, 2, \ldots, \left|\mathrm{II}_i\right|$

for $k = 1, 2, \ldots, \left|\mathrm{II}_i\right|$

$r = \left(\mathrm{II}_i\right)_k$

$\left(\mathbf{b}_i^{\mathrm{I}}\right)_k = \left(\mathbf{b}\right)_r$

Para generar la matriz $\mathbf{A}_i^{\mathrm{II}}$ (utilizando la notación para matrices ralas $J_r(\mathbf{A})$ y $V_r(\mathbf{A})$ para los índices y valores del renglón r):

Sean $\left(\mathrm{II}_i\right)_k$ los subíndices de los nodos internos para la partición i, con $k = 1, 2, \ldots, \left|\mathrm{II}_i\right|$

Sea $\mathbf{g}$ un vector de números booleanos de tamaño n, con valor inicial $\mathbf{g} = (\mathrm{false}, \mathrm{false}, \ldots, \mathrm{false})$

for $k = 1, 2, \ldots, \left|\mathrm{II}_i\right|$
  $r = \left(\mathrm{II}_i\right)_k$
  $\mathbf{g}_r = \mathrm{true}$
$r_{\mathrm{shift}} = \left(\mathrm{II}_i\right)_1 - 1$
for $k = 1, 2, \ldots, \left|\mathrm{II}_i\right|$
  $r = \left(\mathrm{II}_i\right)_k$
  $\sigma = 0$
  for $\rho = 1, 2, \ldots, \left|J_r(\mathbf{A})\right|$
    $c = J_r^{\rho}(\mathbf{A})$
    if $\mathbf{g}_c = \mathrm{true}$
      $\sigma = \sigma + 1$
      $J_k^{\sigma}\left(\mathbf{A}_i^{\mathrm{II}}\right) = J_r^{\rho}(\mathbf{A}) - r_{\mathrm{shift}}$
      $V_k^{\sigma}\left(\mathbf{A}_i^{\mathrm{II}}\right) = V_r^{\rho}(\mathbf{A})$

Para generar la matriz $\mathbf{A}_i^{\mathrm{IB}}$:

Sean $\left(\mathrm{II}_i\right)_k$ los subíndices de los nodos internos para la partición i, con $k=1,2,\ldots,\left|\mathrm{II}_i\right|$
Sean $\left(\mathrm{IB}_i\right)_l$ los subíndices de los nodos frontera para la partición i, con $l=1,2,\ldots,\left|\mathrm{IB}_i\right|$
Sea $\mathbf{g}$ un vector de números booleanos de tamaño n, con valor inicial $\mathbf{g}=\left(\mathrm{false},\mathrm{false},\ldots,\mathrm{false}\right)$

for $l=1,2,\ldots,\left|\mathrm{IB}_i\right|$
    $c=\left(\mathrm{IB}_i\right)_l$
    $\mathbf{g}_c=\mathrm{true}$
$r_{\mathrm{shift}}=\left(\mathrm{IB}_i\right)_1-1$
for $k=1,2,\ldots,\left|\mathrm{II}_i\right|$
    $r=\left(\mathrm{II}_i\right)_k$
    $\sigma=0$
    for $\rho=1,2,\ldots,\left|J_r\left(\mathbf{A}\right)\right|$
        $c=J_r^{\rho}\left(\mathbf{A}\right)$
        if $\mathbf{g}_c=\mathrm{true}$
            $\sigma=\sigma+1$
            $J_k^{\sigma}\left(\mathbf{A}_i^{\mathrm{IB}}\right)=J_r^{\rho}\left(\mathbf{A}\right)-r_{\mathrm{shift}}$
            $V_k^{\sigma}\left(\mathbf{A}_i^{\mathrm{IB}}\right)=V_r^{\rho}\left(\mathbf{A}\right)$

Implementación con MPI

Si el número de particiones es p, entonces se deben ejecutar $p+1$ procesos

Proceso 0 (master)	**Procesos $i=1,2,\ldots,p$ (slaves)**
Leer $\mathbf{A}$, $\mathbf{b}$	
Particionar $\mathbf{A}$	
MPI_Init	MPI_Init
Para $i=1,2,\ldots,p$ Generar $\mathbf{A}_i^{\mathrm{II}}$, $\mathbf{A}_i^{\mathrm{IB}}$, $\mathbf{b}_i^{\mathrm{I}}$ MPI_Send $\mathbf{A}_i^{\mathrm{II}}$ a proceso i MPI_Send $\mathbf{A}_i^{\mathrm{IB}}$ a proceso i MPI_Send $\mathbf{b}_i^{\mathrm{I}}$ a proceso i	MPI_Recv $\mathbf{A}_i^{\mathrm{II}}$ de proceso 0 MPI_Recv $\mathbf{A}_i^{\mathrm{IB}}$ de proceso 0 MPI_Recv $\mathbf{b}_i^{\mathrm{I}}$ de proceso 0
Generar $\mathbf{A}^{\mathrm{BB}}$, $\mathbf{b}^{\mathrm{B}}$	$\mathbf{A}_i^{\mathrm{BI}}=\left(\mathbf{A}_i^{\mathrm{IB}}\right)^{\mathrm{T}}$ Resolver: $\bar{\mathbf{A}}_i^{\mathrm{BB}}=\mathbf{A}_i^{\mathrm{BI}}\left(\mathbf{A}_i^{\mathrm{II}}\right)^{-1}\mathbf{A}_i^{\mathrm{IB}}$, $\bar{\mathbf{b}}_i^{\mathrm{B}}=\mathbf{A}_i^{\mathrm{BI}}\left(\mathbf{A}_i^{\mathrm{II}}\right)^{-1}\mathbf{b}_i^{\mathrm{I}}$
Para $i=1,2,\ldots,p$ MPI_Recv $\mathbf{y}_i$ de proceso i $\mathbf{b}^{\mathrm{B}}=\mathbf{b}^{\mathrm{B}}-\bar{\mathbf{b}}_i^{\mathrm{B}}$	MPI_Send $\bar{\mathbf{b}}_i^{\mathrm{B}}$ a proceso 0

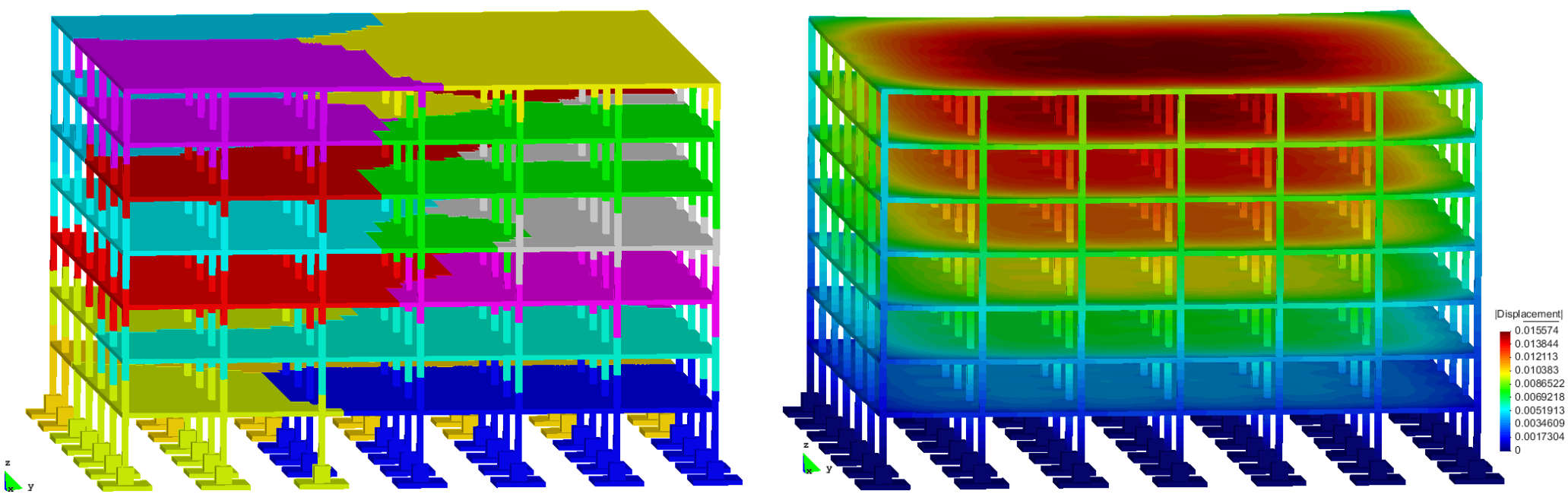
Sea $k \leftarrow 0$ Sea $\mathbf{x}_k^{\mathrm{B}}$ inicial	
Para $i = 1, 2, \ldots, p$ MPI_Send $\mathbf{x}_k^{\mathrm{B}}$ a proceso i	MPI_Recv $\mathbf{x}_k^{\mathrm{B}}$ de proceso 0
$\mathbf{g}_k = \mathbf{A}^{\mathrm{BB}} \mathbf{x}_k^{\mathrm{B}} - \mathbf{b}^{\mathrm{B}}$	$\mathbf{y}_i = \bar{\mathbf{A}}_i^{\mathrm{BB}} \mathbf{x}_k^{\mathrm{B}}$
Para $i = 1, 2, \ldots, p$ MPI_Recv $\mathbf{y}_i$ de proceso i $\mathbf{g}_k = \mathbf{g}_k - \mathbf{y}_i$	MPI_Send $\mathbf{y}_i$ a proceso 0
$\mathbf{p}_k = -\mathbf{g}_k$	
Mientras $\left\|\mathbf{g}_k\right\| > \varepsilon$	
Para $i = 1, 2, \ldots, p$ MPI_Send $\mathbf{p}_k$ a proceso i	MPI_Recv $\mathbf{p}_k$ de proceso 0
$\mathbf{w}_k = \mathbf{A}^{\mathrm{BB}} \mathbf{p}_k$	$\mathbf{y}_i = \bar{\mathbf{A}}_i^{\mathrm{BB}} \mathbf{p}_k$
Para $i = 1, 2, \ldots, p$ MPI_Recv $\mathbf{y}_i$ de proceso i $\mathbf{w}_k = \mathbf{w}_k - \mathbf{y}_i$	MPI_Send $\mathbf{y}_i$ a proceso 0
$\alpha_k = \dfrac{\mathbf{g}_k^{\mathrm{T}} \mathbf{g}_k}{\mathbf{p}_k^{\mathrm{T}} \mathbf{w}_k}$	
$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$	

$\mathbf{g}_{k+1} = \mathbf{g}_k + \alpha \mathbf{w}_k$	
$\beta_k = \dfrac{\mathbf{g}_{k+1}^{\mathrm{T}} \mathbf{g}_{k+1}}{\mathbf{g}_k^{\mathrm{T}} \mathbf{g}_k}$	
$\mathbf{p}_{k+1} = -\mathbf{g}_{k+1} + \beta_{k+1} \mathbf{p}_k$	
$k \leftarrow k+1$	
Para $i = 1, 2, \ldots, p$ MPI_Send $\mathbf{x}_k^{\mathrm{B}}$ a proceso i	MPI_Recv $\mathbf{x}_k^{\mathrm{B}}$ de proceso 0
Ensamblar $\mathbf{x}_k^{\mathrm{B}}$ en $\mathbf{x}$	$\mathbf{x}_i^{\mathrm{I}} = \left(\mathbf{A}_i^{\mathrm{II}} \right)^{-1} \left(\mathbf{b}_i^{\mathrm{I}} - \mathbf{A}_i^{\mathrm{IB}} \mathbf{x}_k^{\mathrm{B}} \right)$
Para $i = 1, 2, \ldots, p$ MPI_Recv $\mathbf{x}_i^{\mathrm{I}}$ de proceso i Ensamblar $\mathbf{x}_i^{\mathrm{I}}$ en $\mathbf{x}$	MPI_Send $\mathbf{x}_i^{\mathrm{I}}$ a proceso 0
Guardar $\mathbf{x}$	
MPI_Finalize	MPI_Finalize

Ejemplo

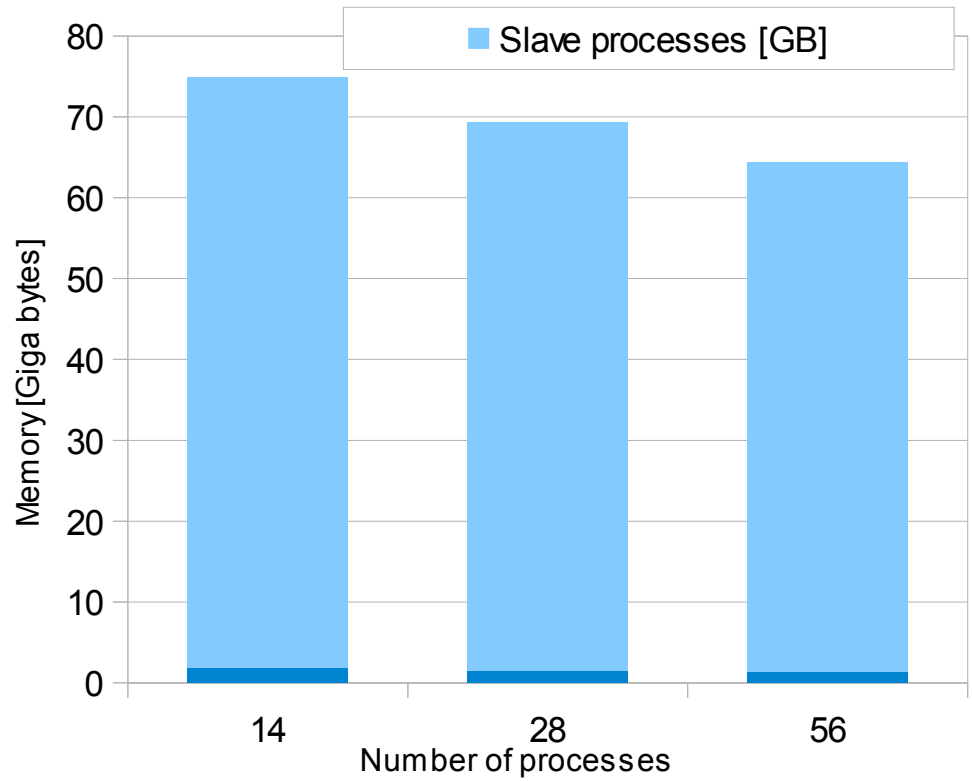
El problema es el análisis de deformación de una estructura sólida debido a su peso propio. La geometría está dividida en 1’336,832 elementos, con 1’708,273 nodos, con tres grados de libertad por nodo, el sistema de ecuaciones resultante tiene 5’124,819 de incógnitas. La tolaerancia utilizada es $1x10^{-10}$.

Utilizamos un cluster de 15 nodos, cada uno con dos procesadores dual core Intel Xeon E5502 (1.87GHz), un total de 60 cores.



Substructuration of the domain (left) resulting deformation (right)

Number of processes	Partitioning time [s]	Inversion time (Cholesky) [s]	Schur complement time (CG) [s]	CG steps	Total time [s]
14	47.6	18520.8	4444.5	6927	23025.0
28	45.7	6269.5	2444.5	8119	8771.6
56	44.1	2257.1	2296.3	9627	4608.9

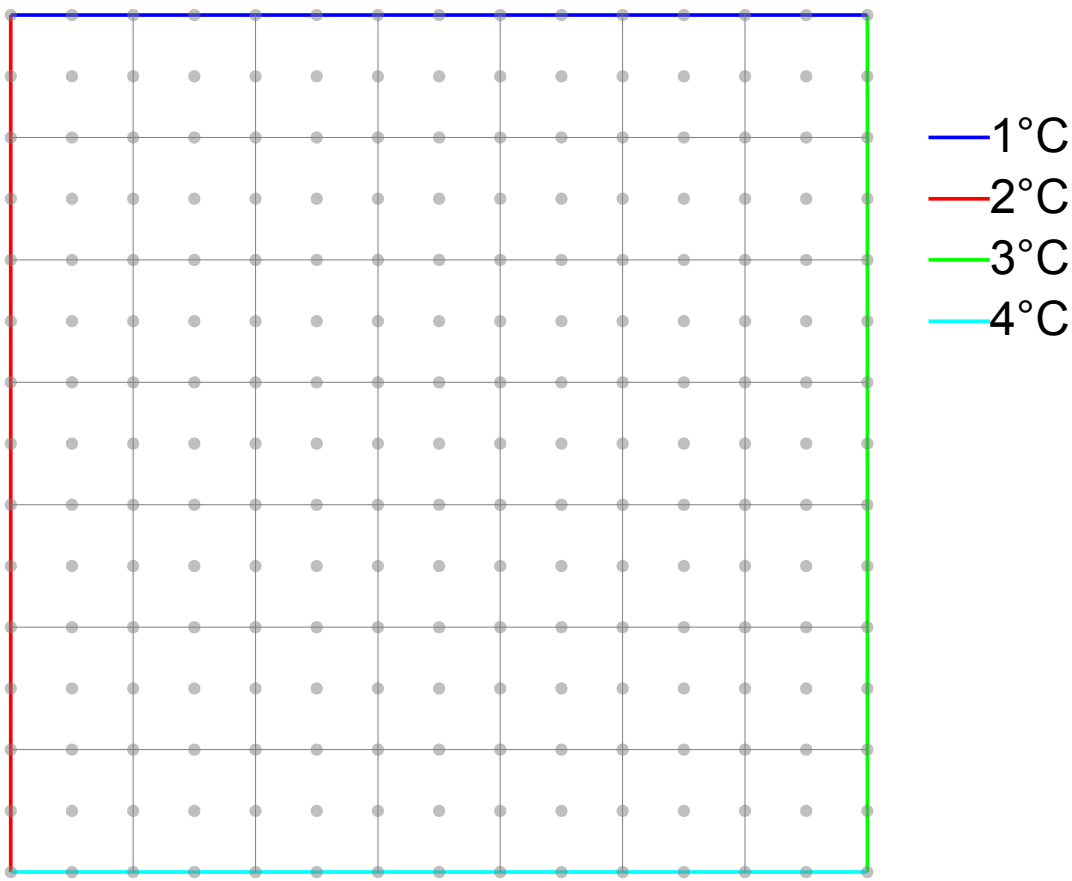


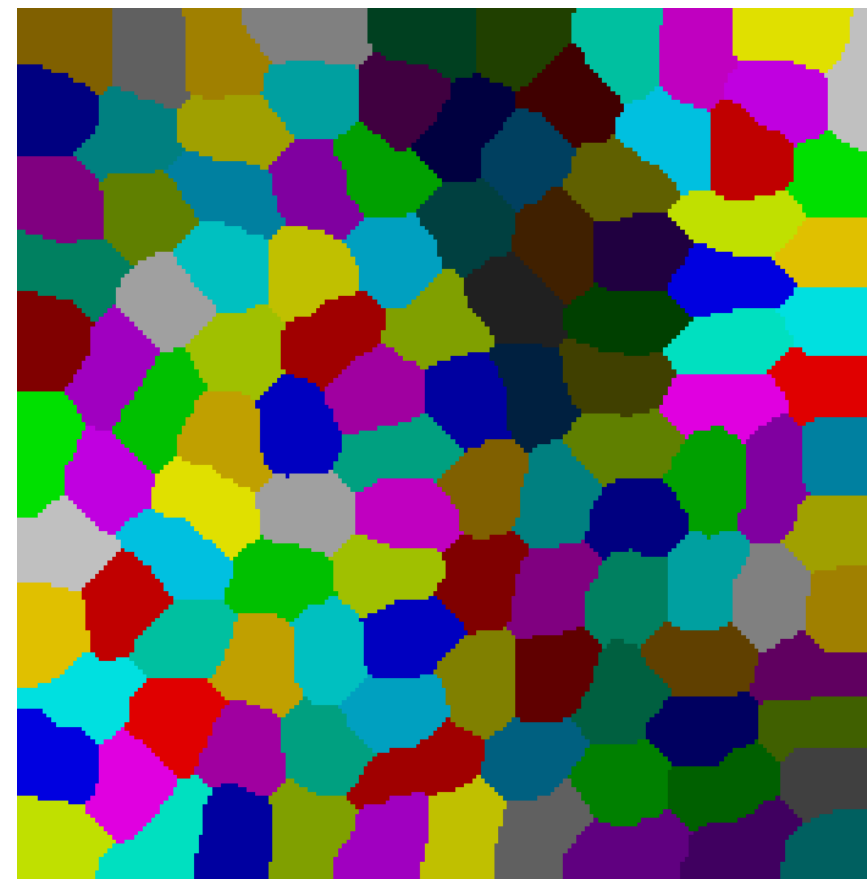
Number of processes	Master process [GB]	Slave processes [GB]	Total memory [GB]
14	1.89	73.00	74.89
28	1.43	67.88	69.32
56	1.43	62.97	64.41

Ejemplo con sistemas de ecuaciones “grandes”

Para estas pruebas utilizamos una geometría sencilla.

Resolvemos la ecuación de Poisson para calcular la distribución de temperatura en un cuadrádo unitario con condiciones Dirichlet en todas las fronteras





Dominio el problema (izquierda). Ejemplo de particionamiento (derecha).

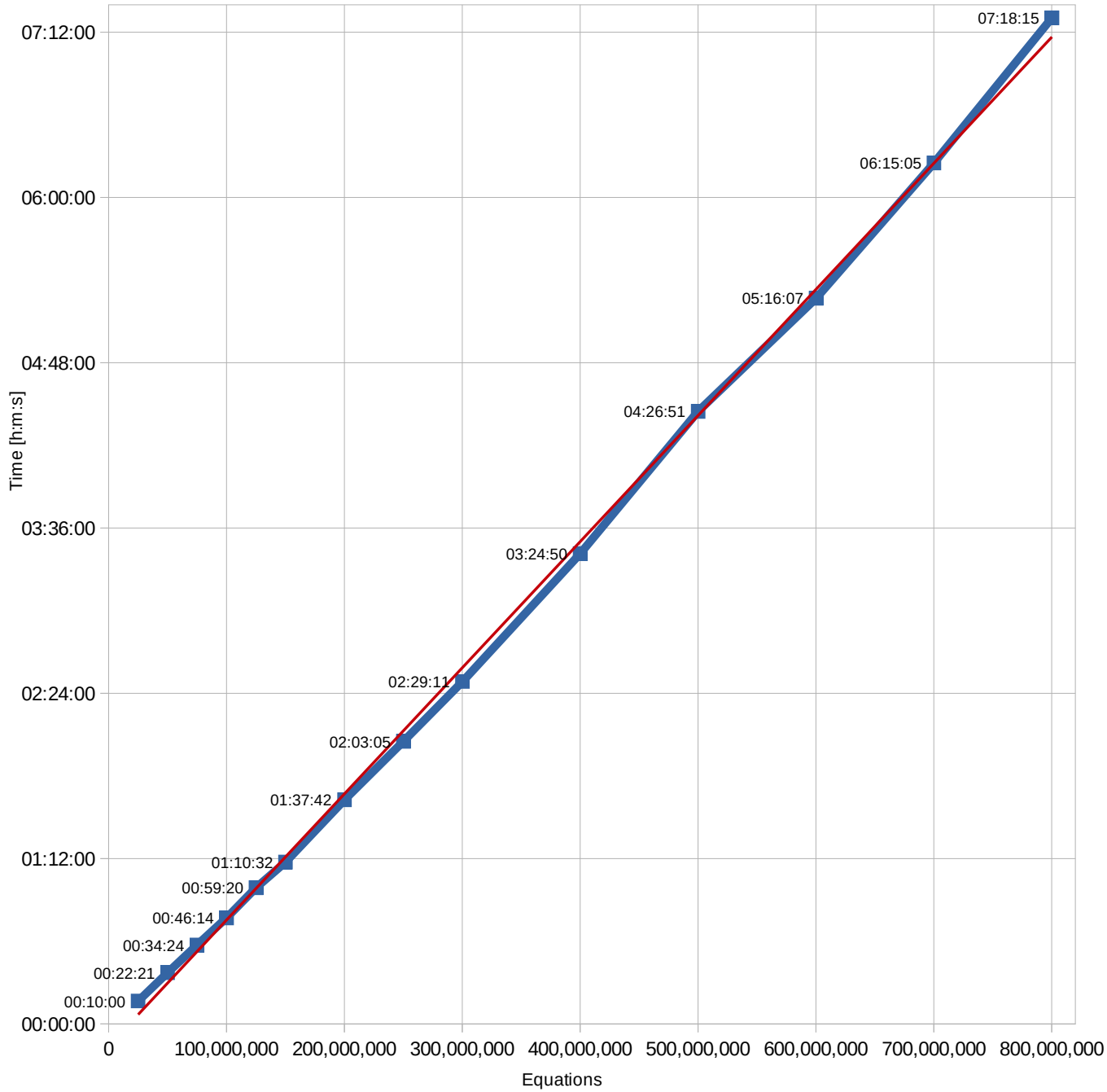
El dominio fue discretizado utilizando cuadriláteros con nueve nodos. Se hicieron 6 discretizaciones diferentes, variando el número de nodos, desde 25 millones a 800 millones.

En todos los casos, el dominio fue dividido en 960 particiones, cada partición fue resuelta en un core. Los tiempos de solución fueron:

Número de ecuaciones	Tiempo [h:m:s]	Memoria [bytes]
25,010,001	00:10:00	38,127,911,928
50,027,329	00:22:21	82,291,561,272
75,012,921	00:34:24	128,852,868,872
100,020,001	00:46:14	176,982,703,608
125,014,761	00:59:20	224,876,901,752
150,038,001	01:10:32	275,868,154,968
200,081,025	01:37:42	380,487,437,704
250,050,969	02:03:05	485,244,957,896
300,086,329	02:29:11	598,995,145,840
400,040,001	03:24:50	812,439,074,088
500,103,769	04:26:51	1,034,046,442,776
600,103,009	05:16:07	1,263,423,250,648
700,078,681	06:15:05	1,451,719,027,176
800,154,369	07:18:15	1,690,025,398,632

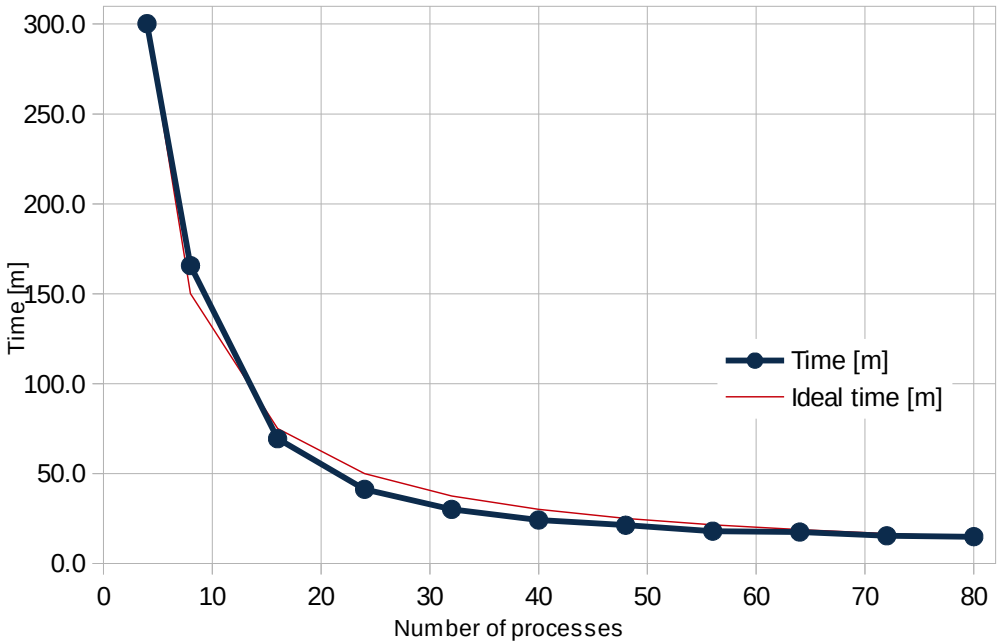
El cluster consta de 80 nodos, cada uno con 2 procesadores Intel Xeon CPU E5-2620 con 6 cores cada uno, un total de 960 cores.

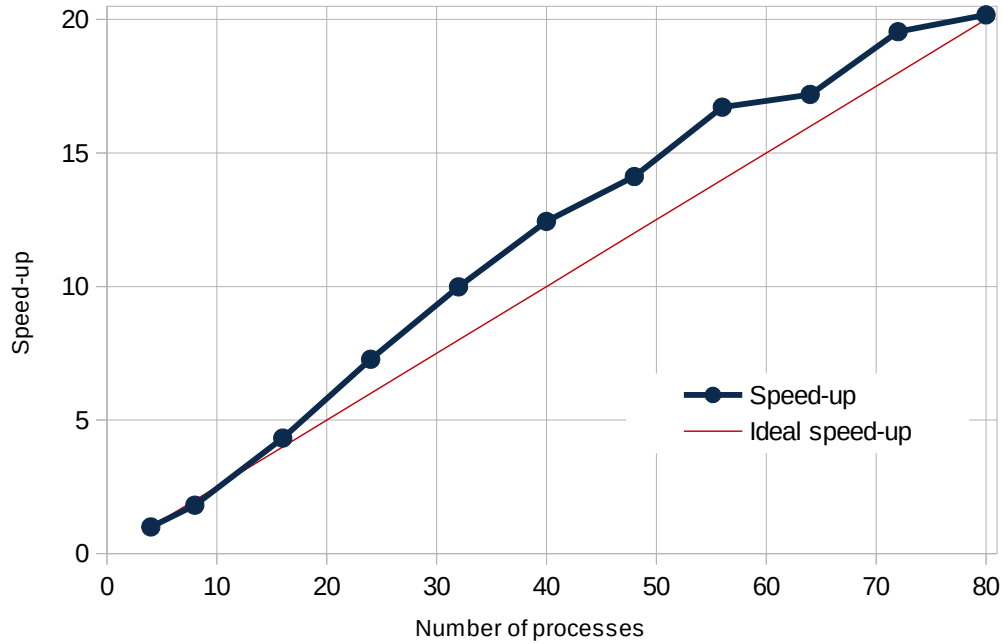
La gráfica de tiempos es:



Speed-Up

Resolviendo la ecuación de calor en 2D, con 25’010,001 ecuaciones con 4 a 80 procesos.



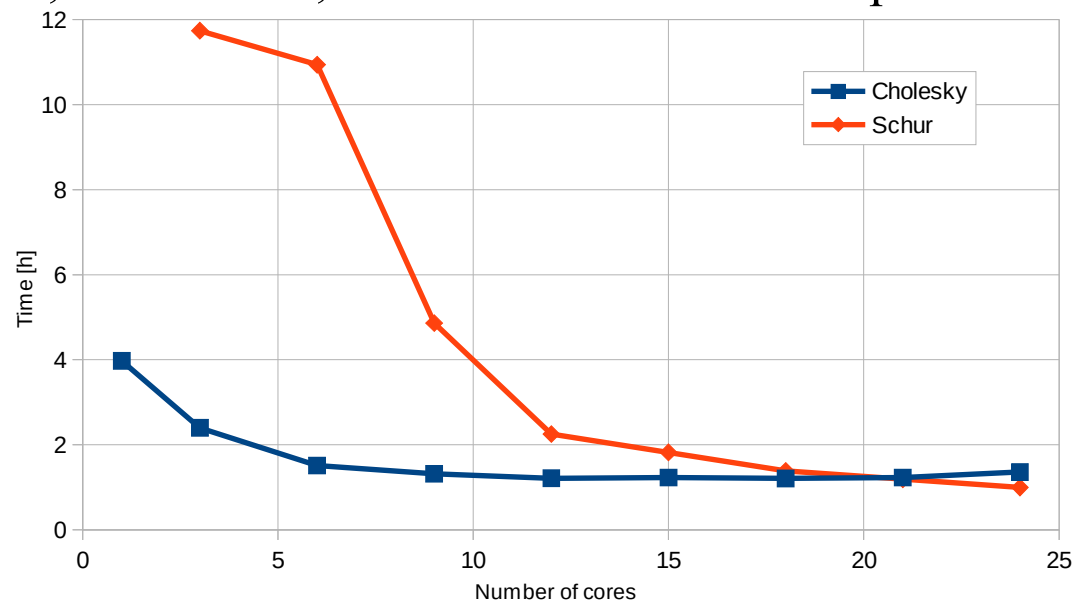


Processes	Time [min]	Memory [bytes]
4	300.2	76,957,081,832
8	165.6	67,884,179,856
16	69.4	59,350,255,936
24	41.3	55,276,709,976
32	30.1	53,020,499,256
40	24.1	51,385,832,424
48	21.3	49,783,503,696
56	18.0	48,852,531,880
64	17.5	48,360,392,536
72	15.4	47,395,019,288
80	14.9	47,303,011,304

Comparación con la factorización Cholesky

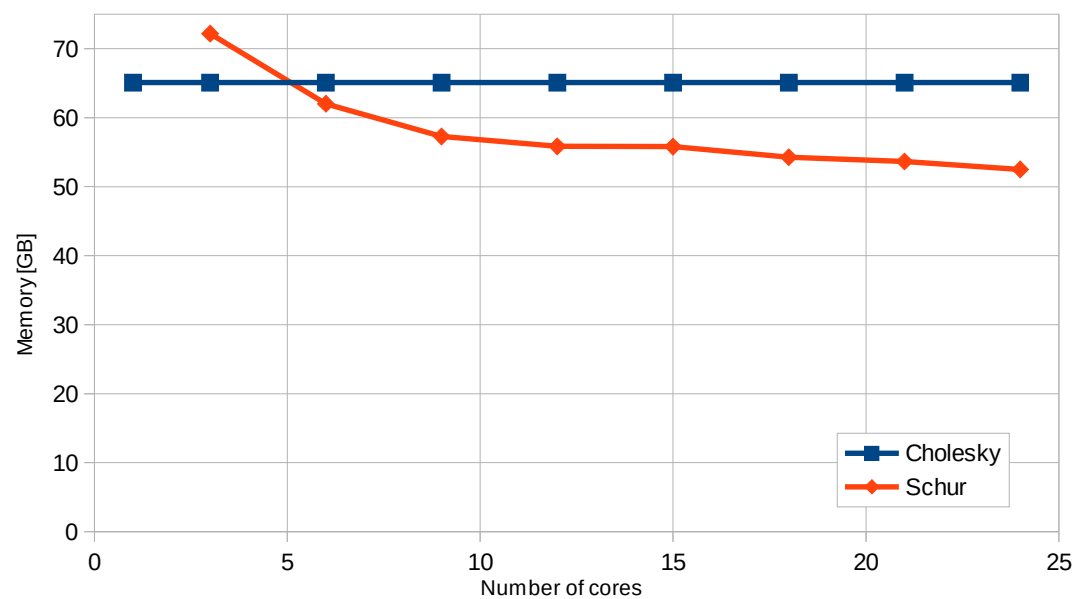
Resolviendo la ecuación de calor en 2D, con 25’010,001 ecuaciones. Los tiempos en horas fueron:

Cores	Cholesky	Schur
1	3.97	
3	2.40	11.74
6	1.51	10.93
9	1.32	4.86
12	1.21	2.25
15	1.23	1.82
18	1.21	1.39
21	1.23	1.19
24	1.36	1.00



La cantidad de memoria en gigabytes:

Cores	Cholesky	Schur
1	65.08	
3	65.08	72.19
6	65.08	62.02
9	65.08	57.29
12	65.08	55.84
15	65.08	55.81
18	65.08	54.28
21	65.08	53.66
24	65.08	52.50



¿Preguntas?

miguelvargas@cimat.mx

Referencias

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

[Sori00] M. Soria-Guerrero. Parallel multigrid algorithms for computational fluid dynamics and heat transfer. Universitat Politècnica de Catalunya, Departament de Màquines i Motors Tèrmics. 2000.

[Smit96] B. F. Smith, P. E. Bjorstad, W. D. Gropp. Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations. Cambridge Univeristy Press, 1996.

[Tose05] A. Toselli, O. Widlund. Domain Decomposition Methods - Algorithms and Theory. Springer, 2005.

Apéndice: Código de particionamiento

```
// g++ -I ~/metis -o partition partition.cpp -L ~/metis -lmetis
extern "C"
{
  #include <metis.h>
}
#include <stdio.h>

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

int main(void)
{
  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);

// Encontrar fronteras
bool is_boundary[N] = {false};
for (int i = 0; i < N; ++i)
{
  int start = xadj[i] - 1;
  int size = xadj[i + 1] - xadj[i];
  int p = part[i];
  for (int j = 0; j < size; ++j)
  {
    int a = adjncy[start + j] - 1;
    int q = part[a];
    if (p < q)
    {
      is_boundary[i] = true;
      break;
    }
  }
}

int II[NPARTS][N];
int IB[NPARTS][N];
int BB[N];
int size_II[NPARTS] = {0};
int size_IB[NPARTS] = {0};
```

```
int size_BB = 0;

// Definir indices
for (int i = 0; i < N; ++i)
{
  if (is_boundary[i])
  {
    BB[size_BB++] = i + 1;
    int start = xadj[i] - 1;
    int size = xadj[i + 1] - xadj[i];
    for (int j = 0; j < size; ++j)
    {
      int a = adjncy[start + j] - 1;
      if (!is_boundary[a])
      {
        int p = part[a] - 1;
        if (size_IB[p] > 0)
        {
          if (IB[p][size_IB[p] - 1] == i + 1)
          {
            continue;
          }
        }
        IB[p][size_IB[p]++] = i + 1;
      }
    }
  }
  else
  {
    int p = part[i] - 1;
```

```
      II[p][size_II[p]++] = i + 1;
    }
  }

  for (int p = 0; p < NPARTS; ++p)
  {
    printf("\nParticion %i\n", p + 1);
    printf("  II:");
    for (int i = 0; i < size_II[p]; ++ i)
    {
      printf(" %i", II[p][i]);
    }
    printf("\n  IB:");
    for (int i = 0; i < size_IB[p]; ++ i)
    {
      printf(" %i", IB[p][i]);
    }
  }
  printf("\nBoundary\n");
  printf("  BB:");
  for (int i = 0; i < size_BB; ++ i)
  {
    printf(" %i", BB[i]);
  }
  printf("\n\n");

  return 0;
}
```