UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

Tarjetas Gráficas y Aceleradores

CUDA – Propuestas de Proyectos

Agustín Fernández

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya

C=A·B, Algoritmo de Strassen

Producto de matrices C = A · B (tamaño N × N)

- Matrices cuadradas, dimensiones potencia de 2 ($N = 2^n = 2^{2m}=2M$)

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \quad A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$$
$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$
$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$$
$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$$

8 Operaciones de $O(M^3)$
4 Operaciones de $O(M^2)$

MbyM $O(N^3)$
Strassen $\cong O(N^{2.807})$

C=A·B, Algoritmo de Strassen

Producto de matrices C = A · B (tamaño N × N)

- Matrices cuadradas, dimensiones potencia de 2 ($N = 2^n = 2^{2m}=2M$)

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \quad A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$M_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$$
$$M_2 = (A_{21} + A_{22}) \cdot B_{11}$$
$$M_3 = A_{11} \cdot (B_{12} - B_{22})$$
$$M_4 = A_{22} \cdot (B_{21} - B_{11})$$
$$M_5 = (A_{11} + A_{12}) \cdot B_{22}$$
$$M_6 = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$$
$$M_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$

8 Operaciones de $O(M^3)$
4 Operaciones de $O(M^2)$

7 Operaciones de $O(M^3)$
10 Operaciones de $O(M^2)$

C=A·B, Algoritmo de Strassen

Producto de matrices C = A · B (tamaño N × N)

- Matrices cuadradas, dimensiones potencia de 2 ($N = 2^n = 2^{2m}=2M$)

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \quad A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} = M_1 + M_4 - M_5 + M_7$$
$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22} = M_3 + M_5$$
$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21} = M_2 + M_4$$
$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22} = M_1 - M_2 + M_3 + M_6$$

8 Operaciones de $O(M^3)$
4 Operaciones de $O(M^2)$

7 Operaciones de $O(M^3)$
10 Operaciones de $O(M^2)$

8 Operaciones de $O(M^2)$

Strassen $\cong O(N^{2.807})$

C=A·B, Algoritmo de Strassen

Producto de matrices C = A · B (tamaño N × N)

- Matrices cuadradas, dimensiones potencia de 2 ($N = 2^n = 2^{2m}=2M$)

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \quad A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$M_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$$
$$M_2 = (A_{21} + A_{22}) \cdot B_{11}$$
$$M_3 = A_{11} \cdot (B_{12} - B_{22})$$
$$M_4 = A_{22} \cdot (B_{21} - B_{11})$$
$$M_5 = (A_{11} + A_{12}) \cdot B_{22}$$
$$M_6 = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$$
$$M_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$

$$C_{11} = M_1 + M_4 - M_5 + M_7$$
$$C_{12} = M_3 + M_5$$
$$C_{21} = M_2 + M_4$$
$$C_{22} = M_1 - M_2 + M_3 + M_6$$

Implementar el Algoritmo de Strassen en
- 1 GPU
- 2 GPUs
- 4 GPUs

C=A·B, Algoritmo de Strassen

2 GPUs

$$C_{11} = M_1 + M_4 - M_5 + M_7$$
$$C_{12} = M_3 + M_5$$

gpu0

$$C_{21} = M_2 + M_4$$
$$C_{22} = M_1 - M_2 + M_3 + M_6$$

gpu1

4 GPUs

$$C_{11} = M_1 + M_4 - M_5 + M_7$$ gpu0

$$C_{12} = M_3 + M_5$$ gpu1

$$C_{21} = M_2 + M_4$$ gpu2

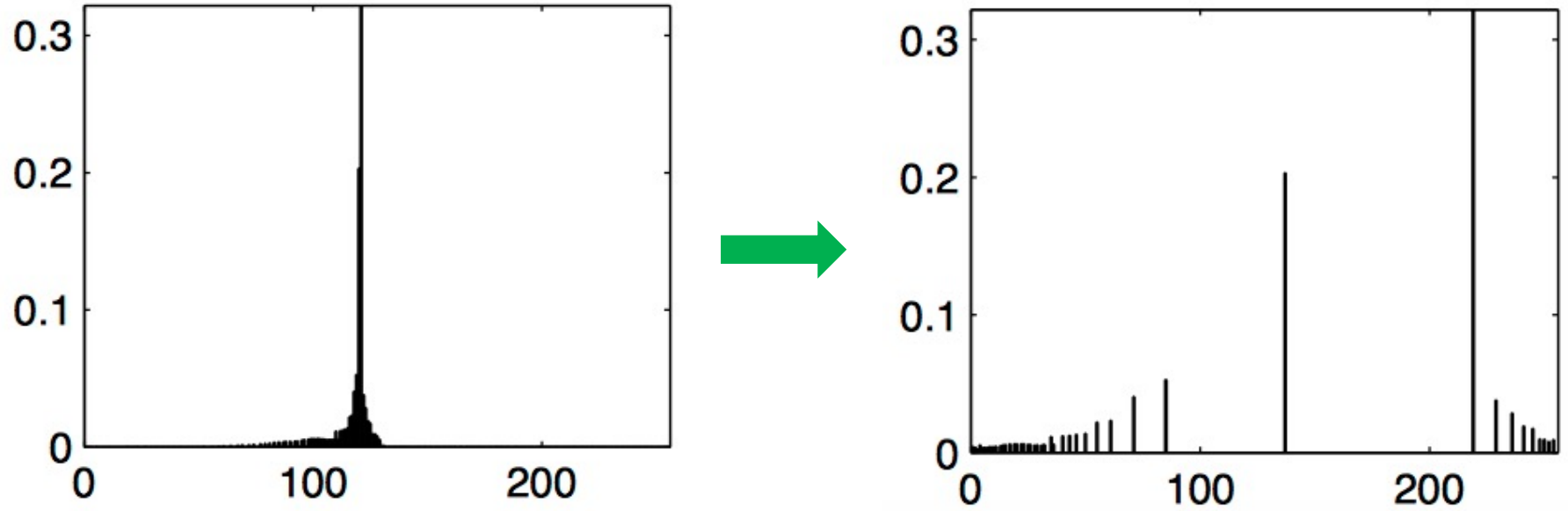
$$C_{22} = M_1 - M_2 + M_3 + M_6$$ gpu3

- Equilibrio de carga
- Replicar cálculos vs calcular y transmitir
- Comunicación GPU – GPU sin pasar por la CPU

Histogram Equalization

1. Generar un histograma de la imagen
2. A partir del histograma, en función de los valores máximos y mínimos, generar una tabla de traducción (equalización).
3. Equalizar la imagen.

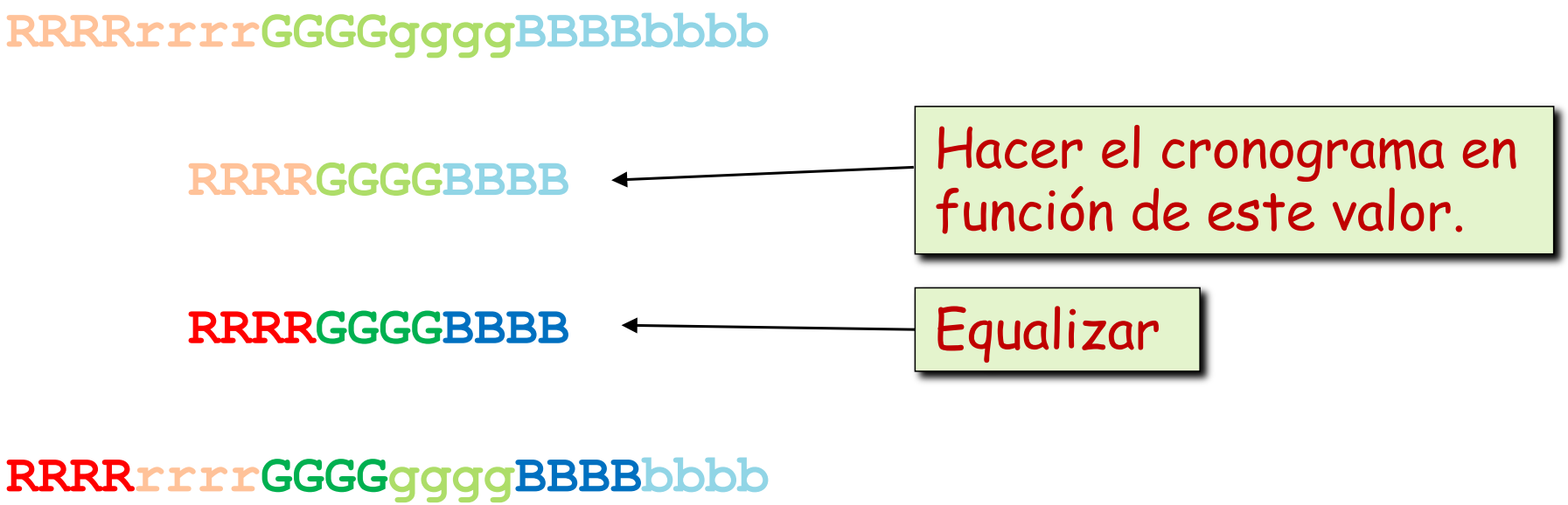
Histogram Equalization



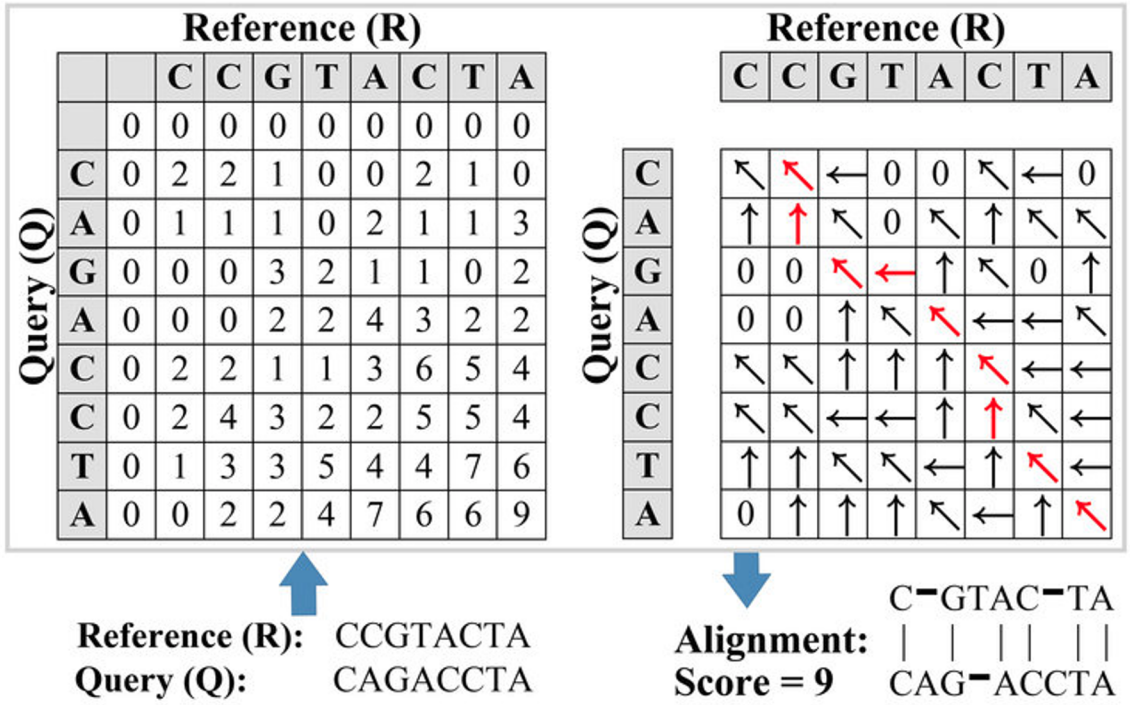
- Generar la tabla de equalización, puede ser tan simple como pasar de una distribución [min:max] a otra [0:N-1].
- Podemos montar una tabla de traducción, o hacer los cálculos necesarios para cada píxel.
- Si tenemos una imagen en blanco y negro, cada píxel es un valor entre 0 y 255, el algoritmo es simple de implementar.

Histogram Equalization

- Si trabajamos con RGB (24 bits) el algoritmo puede tener muchas variantes.
- Podríamos hacer un histograma para cada color y equalizar cada color por separado.
- Podríamos tomar la parte alta de cada color, por ejemplo los 4 bits de mayor peso de cada color y hacer el histograma con los 12 bits resultantes.



Smith-Waterman

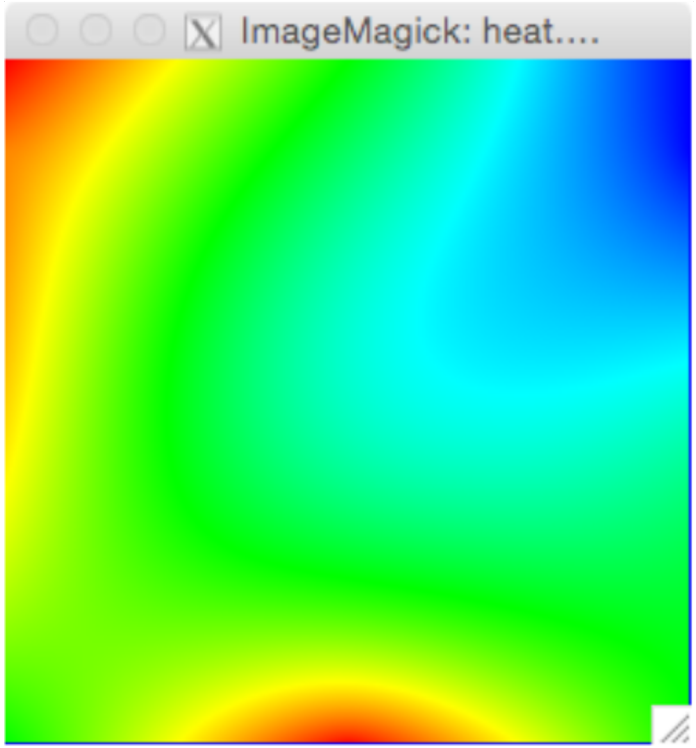


Smith-Waterman

- **Procesamiento de una matriz usando un algoritmo de programación dinámica**
- **Estructura de datos:**
  - Dos secuencias (vectores de caracteres)
  - Una matriz de scoring (matriz bidimensional)
- **Complejidad: O(n^2) espacio y tiempo**
- **Código:**

```
void smith_waterman(int h[N+1][N+1], char a[N], char b[N], int sim[20][20]) {
   int i,j;
   int diag, down, right;
   for (i=0;i<=N;i++) {
        h[0][i]=0;
        h[i][0]=0;
   }
   for (i=1;i<=N;i++)
     for (j=1;j<=N;j++) {
           diag = h[i-1][j-1] + sim[getIndex(a[i-1])][getIndex(b[j-1])];
           down = h[i-1][j] + 4;
           right = h[i][j-1] + 4;
           h[i][j] = MAX4(diag,down,right,0);
     }
}
```

Heat Equation - Jacobi



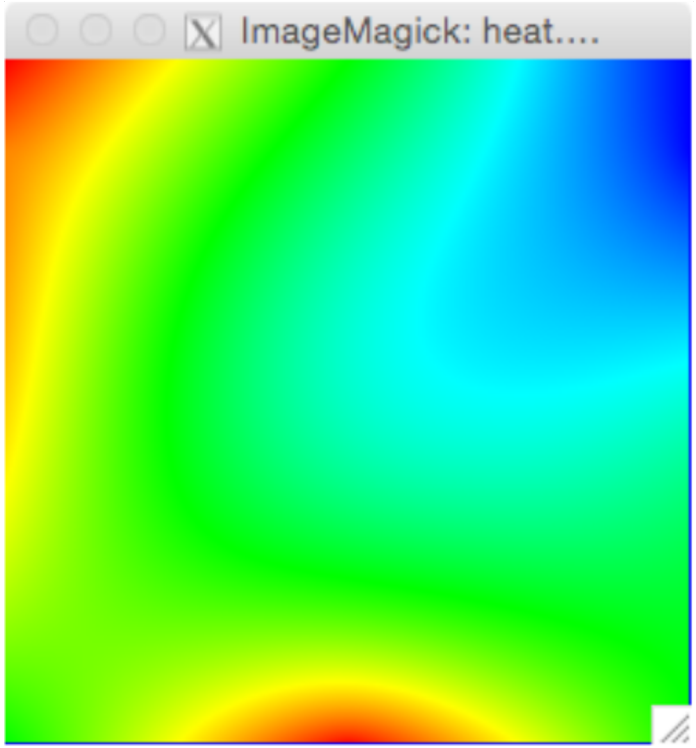
Heat Equation - Jacobi

- **Iterative sequential code**
- **Procesamiento de una matriz bidimensional**
  - Jacobi opera y deja el valor en otra matriz. Luego hay swap de la matriz.

```c
/*
 * Blocked Jacobi solver: one iteration step
 */
double relax_jacobi (double *u, double *utmp, unsigned sizex, unsigned sizey)
{
    double diff, sum=0.0;

    int howmany=4;
    for (int blockid = 0; blockid < howmany; ++blockid) {
      int i_start = lowerb(blockid, howmany, sizex);
      int i_end = upperb(blockid, howmany, sizex);
      for (int i=max(1, i_start); i<= min(sizex-2, i_end); i++) {
        for (int j=1; j<= sizey-2; j++) {
            utmp[i*sizey+j]= 0.25 * ( u[ i*sizey     + (j-1) ]+  // left
                                      u[ i*sizey     + (j+1) ]+  // right
                                      u[ (i-1)*sizey + j     ]+  // top
                                      u[ (i+1)*sizey + j     ]); // bottom
            diff = utmp[i*sizey+j] - u[i*sizey + j];
            sum += diff * diff;
        }
      }
    }

    return sum;
}
```

```c
// starting time
runtime = wtime();

iter = 0;
while(1) {
    switch( param.algorithm ) {
        case 0: // JACOBI
                residual = relax_jacobi(param.u, param.uhelp, np, np);
                // Copy uhelp into u
                copy_mat(param.uhelp, param.u, np, np);
                break;
        case 1: // GAUSS
                residual = relax_gauss(param.u, np, np);
                break;
        }

    iter++;

    // solution good enough ?
    if (residual < 0.00005) break;

    // max. iteration reached ? (no limit with maxiter=0)
    if (param.maxiter>0 && iter>=param.maxiter) break;
}
```

Heat Equation – Gauss-Seidel



Heat Equation – Gauss-Seidel

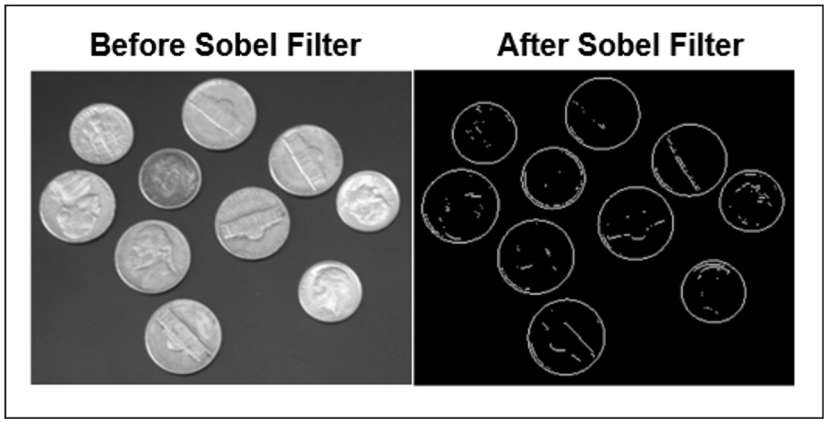
- **Iterative sequential code**
- **Procesamiento de una matriz bidimensional**
  - Gauss Seidel opera sobre la misma matriz

```
/*
 * Blocked Gauss-Seidel solver: one iteration step
 */
double relax_gauss (double *u, unsigned sizex, unsigned sizey)
{
    double unew, diff, sum=0.0;

    int howmany=4;
    for (int blockid = 0; blockid < howmany; ++blockid) {
      int i_start = lowerb(blockid, howmany, sizex);
      int i_end = upperb(blockid, howmany, sizex);
      for (int i=max(1, i_start); i<= min(sizex-2, i_end); i++) {
        for (int j=1; j<= sizey-2; j++) {
            unew= 0.25 * ( u[ i*sizey   + (j-1) ]+  // left
                           u[ i*sizey   + (j+1) ]+  // right
                           u[ (i-1)*sizey       + j     ]+  // top
                           u[ (i+1)*sizey       + j     ]); // bottom
            diff = unew - u[i*sizey+ j];
            sum += diff * diff;
            u[i*sizey+j]=unew;
        }
      }
    }

    return sum;
}
```

```
// starting time
runtime = wtime();

iter = 0;
while(1) {
    switch( param.algorithm ) {
        case 0: // JACOBI
                residual = relax_jacobi(param.u, param.uhelp, np, np);
                // Copy uhelp into u
                copy_mat(param.uhelp, param.u, np, np);
                break;
        case 1: // GAUSS
                residual = relax_gauss(param.u, np, np);
                break;
        }

    iter++;

    // solution good enough ?
    if (residual < 0.00005) break;

    // max. iteration reached ? (no limit with maxiter=0)
    if (param.maxiter>0 && iter>=param.maxiter) break;
}
```

Edge Detection

- **Procesamiento de imagenes para detectar figuras en una imagen**



Edge-Detection

- Procesamiento de una imagen haciendo una convolución de dos matrices
- Código: Algo estilo esto....

```
/* We will be ignoring the 1 pixel border around image for edge cases */
for (int i = 1; i < height-1; i++) {
  for (int j = 1; j < width-1; j++) {

    /* Get pixel and the neighbours and then apply onto the mask matrix */

    if (mask_size == 3) { // Multiplies the 3x3 matrix with a pixel and its 8 neighbours
      temp = (work_image[i-1][j-1] * mask[0][0]) + (work_image[i-1][j] * mask[0][1]) +
             (work_image[i-1][j+1] * mask[0][2]) + (work_image[i][j-1] * mask[1][0]) +
             (work_image[i][j] * mask[1][1]) + (work_image[i][j+1] * mask[1][2]) +
             (work_image[i+1][j-1] * mask[2][0]) + (work_image[i+1][j] * mask[2][1]) +
             (work_image[i+1][j+1] * mask[2][2]);

    } else if (mask_size == 2) { // Multiplies the 2x2 matrix with pixel and its 3 neighbours
      temp = (work_image[i][j] * mask[0][0]) + (work_image[i+1][j+1] * mask[1][1]) +
             (work_image[i][j+1] * mask[0][1]) + (work_image[i+1][j] * mask[1][0]);
    }

    out_image[i][j] = temp;
  }
}
```

Otras ideas

- Face detection
- Criptografia
- Sorting : Quicksort, Mergesort, Radix sort
- Etc.

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

Tarjetas Gráficas y Aceleradores

CUDA – Propuestas de Proyectos

Agustín Fernández

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya