Lenguaje de programación para cálculo paralelo.

Andrés Baamonde Lozano (andres.baamonde@udc.es)
Rodrigo Arias Mallo (rodrigo.arias@udc.es)

16 de diciembre de 2014

Introducción

Propósito

- Lenguaje de programación orientado al cálculo numérico.
- Objetivo: explotar al máximo la GPU y la CPU.

Introducción

Propósito

- Lenguaje de programación orientado al cálculo numérico.
- Objetivo: explotar al máximo la GPU y la CPU.

Lenguajes relacionados

- C, que implica una programación cuidadosa.
- OpenCL, cuyo objetivo es ser un estándar abierto en el ámbito computación paralela.
- Matlab, para las funciones sobre imágenes.

Paradigma

- Programación imperativa.
- Necesaria sencillez para operar con la GPU.
- Descartados otros paradigmas, no es necesario un alto nivel de abstracción.

Gestión de memoria

- A cargo del programador(como C).
- Gestión de memoria dinámica retiene la memoria más tiempo del necesario.

Tipado

- Tipado estático.
- Se establece para las variables dimensión y tipo.
- Incrementa el tiempo de desarrollo.
- En tiempo de ejecución requiere menos comprobaciones.

Tipos y modificadores

Tipos

- Básico(C).
- Complejos(Vector, Matriz,imaginario).

Tipos y modificadores

Tipos

- Básico(C).
- Complejos(Vector, Matriz,imaginario).

Modificadores

Además de los modificadores a nivel de función (Local) y a nivel de programa(Global).Existirán unos modificadores de tipos para cargar las variables en GPU o CPU.

Operadores

- Tradicionales de C.
- Sobrecarga de operadores para los tipos complejos.

Errores y características

Errores

- Matemáticos.
- Desbordamiento.
- Segmentación.

Errores y características

Errores

- Matemáticos.
- Desbordamiento.
- Segmentación.

Características

- Inserción de código OpenCL.
- Evaluación en cortocircuíto.

Las entrañas

GPGPU =

General Purpose computing on Graphics Processing Units

Usar la GPU como herramienta de computación

Las entrañas

GPGPU =

General Purpose computing on Graphics Processing Units

Usar la GPU como herramienta de computación

SIMD = Single Instruction, Multiple Data

Ejecutar una sóla instrucción a varios elementos a la vez.

Las entrañas

GPGPU =

General Purpose computing on Graphics Processing Units

Usar la GPU como herramienta de computación

SIMD = Single Instruction, Multiple Data

Ejecutar una sóla instrucción a varios elementos a la vez.

OpenCL = Open Computing Language

Lenguaje para cálculo paralelo independiente del hw (2009).

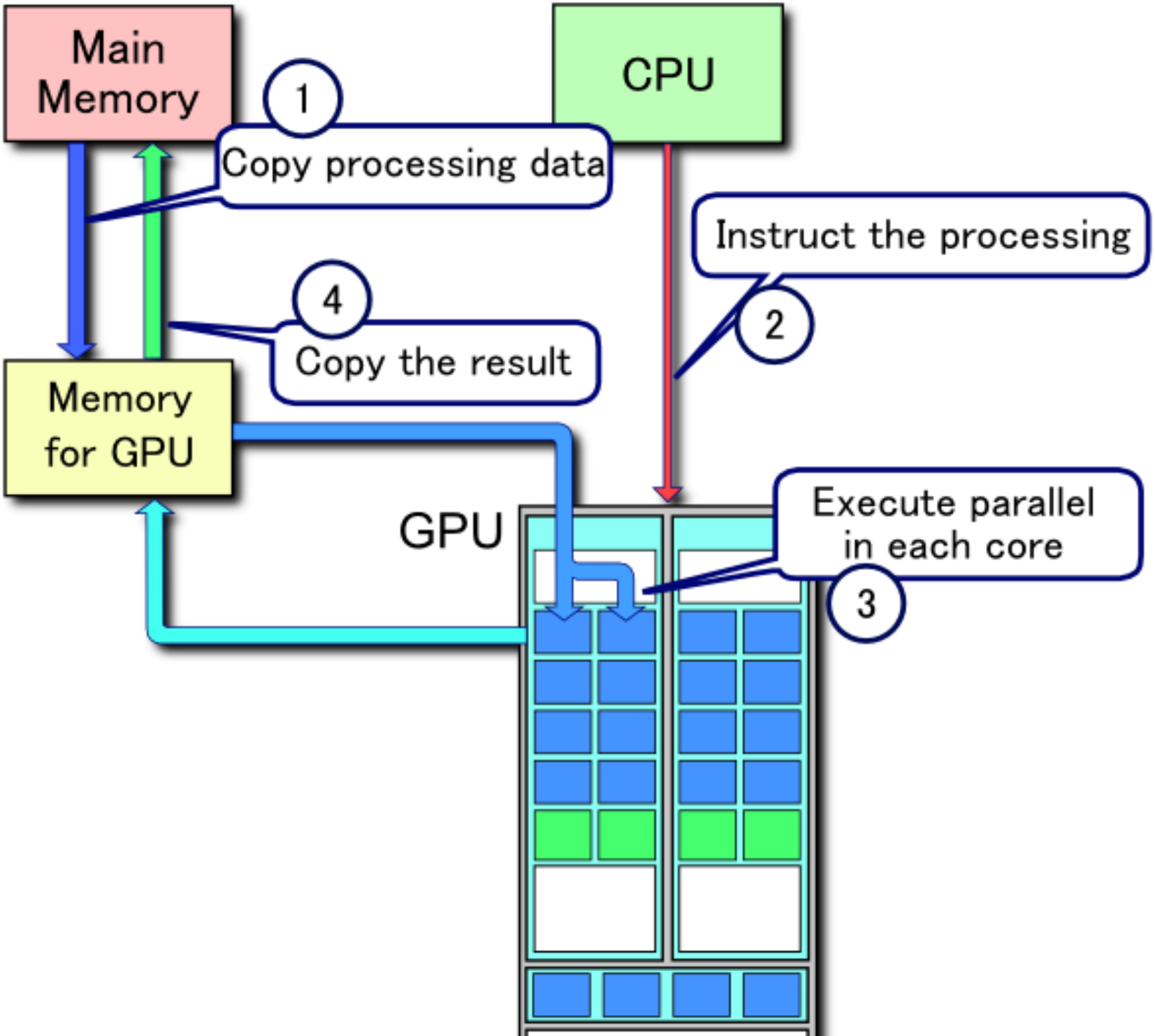
Rápida introducción a GPGPU en 3 pasos

- Enviamos datos desde la RAM a la GPU.
- La GPU realiza operaciones en su memoria interna.
- Transferimos los datos de la GPU a la RAM.

Nota: La GPU tiene una memoria interna.

Procesamiento paralelo

- La GPU tiene muchos núcleos.
- Pero sólo tiene una memoria.
- Enviamos datos, y un programa común.
- Cada núcleo ejecuta el mismo programa, con diferentes datos

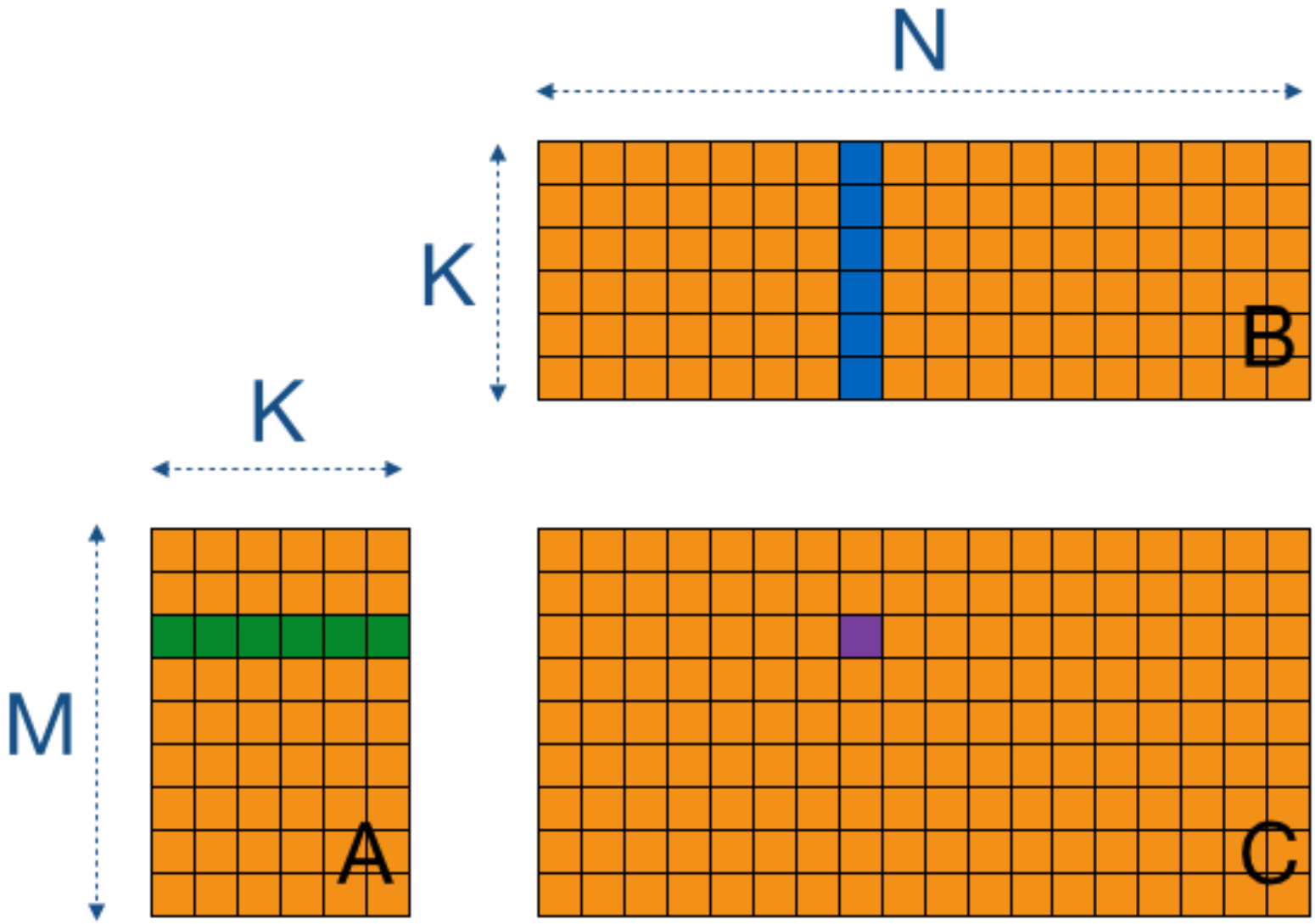

Main Memory
CPU
1
Copy processing data
Instruct the processing
2
4
Copy the result
Memory for GPU
GPU
Execute parallel in each core
3

Dónde está la mejora? Ejemplo de matrices.

A, B y C son matrices de NxN. Disponemos de P núcleos.

Calcular $C = A * B$. Complejidad $O(n^3)$.

- Dividir la matriz C en bloques de P elementos.
- Asignar cada elemento (i, j) del bloque a un núcleo:
- Repetir para todos los bloques.


N
K
B
K
M
A
C

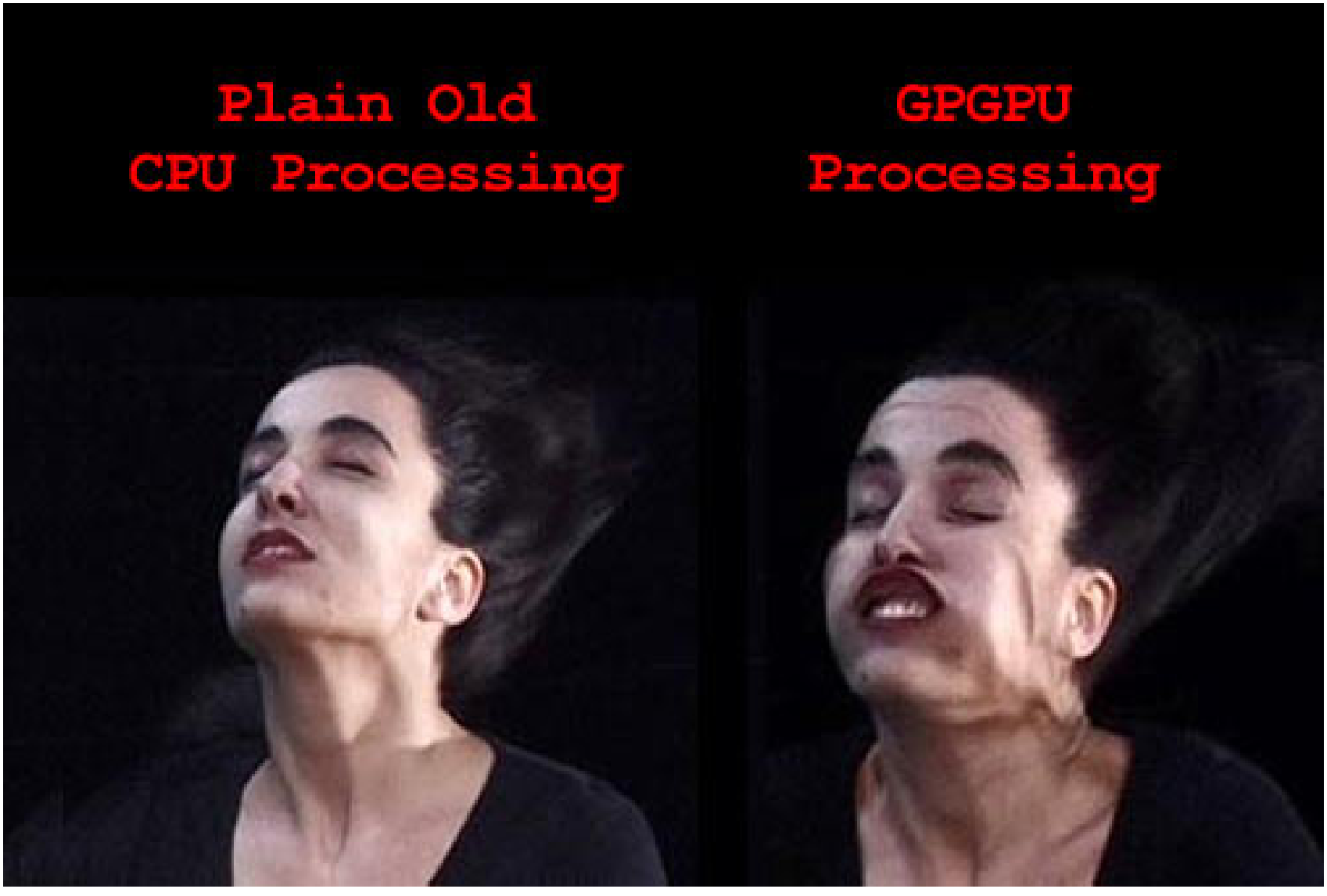
Procesamiento paralelo, en cada núcleo

```
C[i][j] = 0;
for(k = 0; k < K; k++)
    C[i][j] += A[i][k] * B[k][j];
```

Complejidad final de $O(n^3/P)$

Demo

Multiplicación de matrices.


Plain Old
CPU Processing
GPGPU
Processing

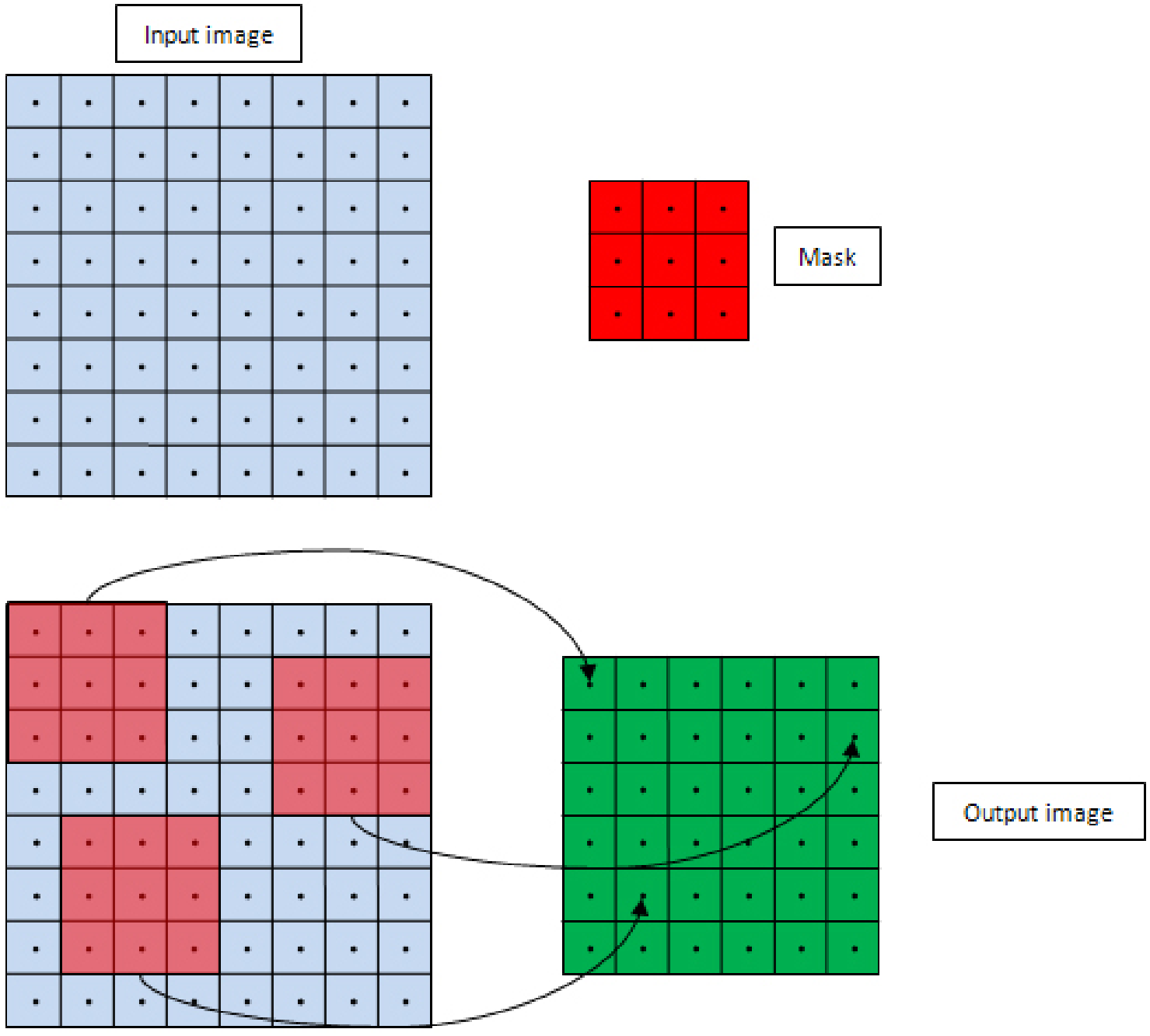
Ejemplo de filtro.

Cada núcleo ejecuta un pequeño programa, como sumar los valores de los píxeles vecinos.

El resultado se almacena en un lugar aparte, sin modificar la original.

Todos los núcleos se pueden ejecutar en paralelo.

La imagen resultante, regresa a la RAM.


Input image
Mask
Output image

Problema

Sabemos resolver muchos problemas, empleando algoritmos secuenciales. Convertir un algoritmo a uno paralelo, requiere replantearlo.

Solución?

Realizar cambios mecánicamente en el código secuencial para vectorizar aquello que sea posible. Rama de investigación, vectorización.

Ejemplo:

```
for(i = 0; i < N; i++)
{
     A[i]++;
     A[N-1-i]++;
}
```

El optimizador debe ser capaz de deducir:

```
A = A + 2;
```

Optimización de código secuencial

Tratar de paralelizar todo el código posible.

Crear un grafo de operaciones y simplificarlo.

Código de ejemplo

```
A = B*C + D*E
C = A+E + E*B
B = B*E + C
```

Descomposición:

```
1     F = B*C
2     G = D*E
3     A = F+G
4     H = A*E
5     C = H+B
6     I = B*E
7     B = I*C
```

Descomposición:

```
1      F = B*C
2      G = D*E
3      A = F+G
4      H = A*E
5      C = H+B
6      I = B*E
7      B = I*C
```

Tabla de operaciones:

```
   A  B  C  D  E  F  G  H  I
   ----------------------
1     .  .        1
2           .  .     2
3  3              .  .
4  .           .        4
5     .  5              .
6     .        .           6
7     7  .                 .
```

Tabla de operaciones:

```
   A  B  C  D  E  F  G  H  I
   -----------------------
1     .  .        1
2           .  .     2
3  3              .  .
4  .           .        4
5     .  5              .
6     .        .           6
7     7  .                 .
```

Optimización de operaciones.

```
   A  B  C  D  E  F  G  H  I
   -----------------------
1     .  .  .  .  1  2
2  3              .  .
3  .           .        4
4     .  5     .        .  6
5     7  .                 .
```

Optimización de operaciones.

```
   A  B  C  D  E  F  G  H  I
   -----------------------
1     .  .  .  .  1  2
2  3                 .  .
3  .              .     4
4     .  5        .        .  6
5     7  .                    .
```

Optimización del espacio.

```
   A  B  C  D  E  F  G  H  I
   -----------------------
1     .  .  .  .  1  2
2  3                 .  .
3  .              .  4
4     .  5        .  .  6
5     7  .              .
```

¿Preguntas?