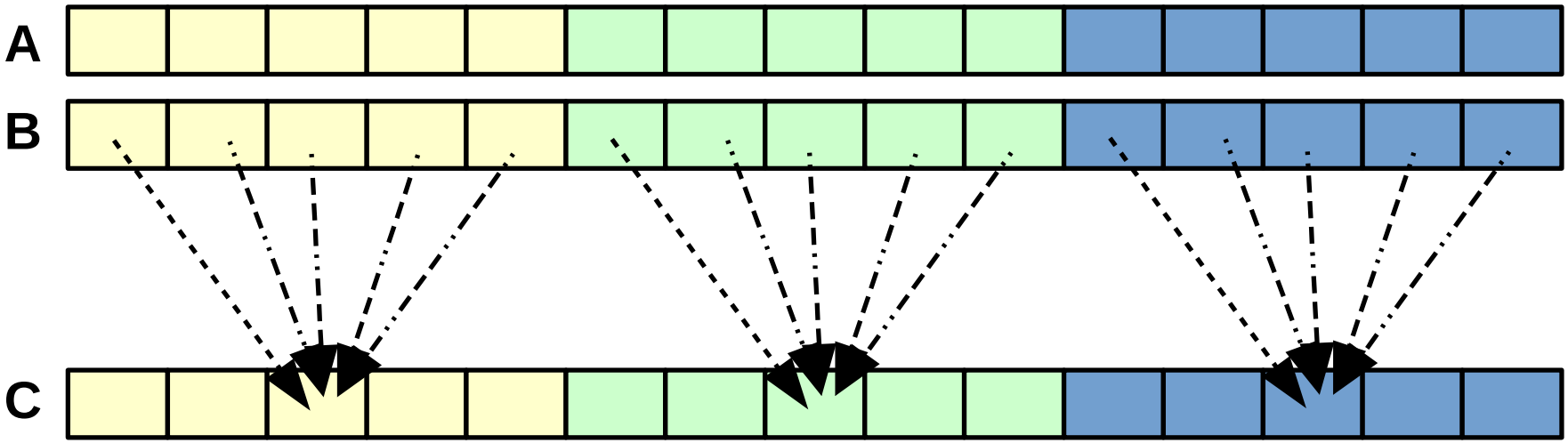
1.2 Implementación CUDA de una operación vectorial (1/4)

N = k × M , donde M ∈ {64, 128, 256} y k ∈ N



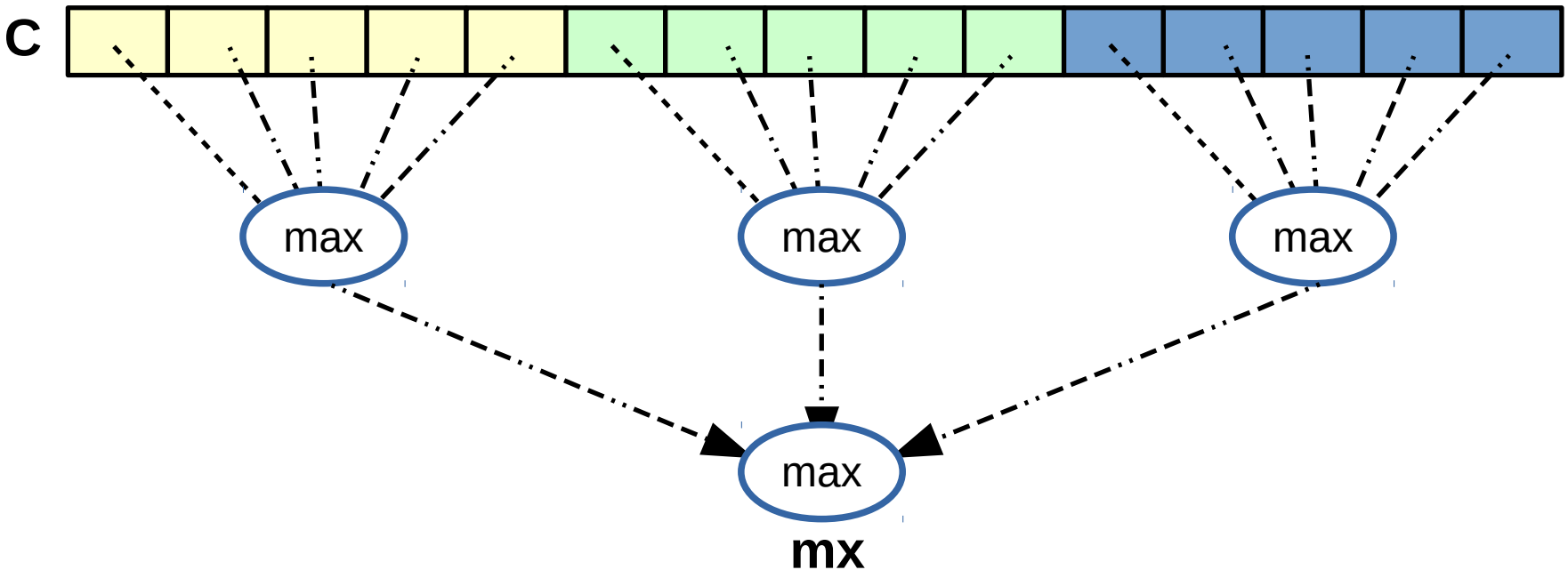
$$C[i] = \sum_{j \in M_i} (A[j] \cdot i + B[j]), \quad \text{Si} \quad ceil\,(A[j] \cdot i)) \quad \text{es par.}$$

$$C[i] = \sum_{j \in M_i} (A[j] \cdot i - B[j]), \quad \text{Si} \quad ceil\,(A[j] \cdot i) \quad \text{es impar.}$$

Mi = conjunto de M índices (M = tamaño de bloque) al que pertenezca la posición i.

1.2 Implementación CUDA de una operación vectorial (2/4)

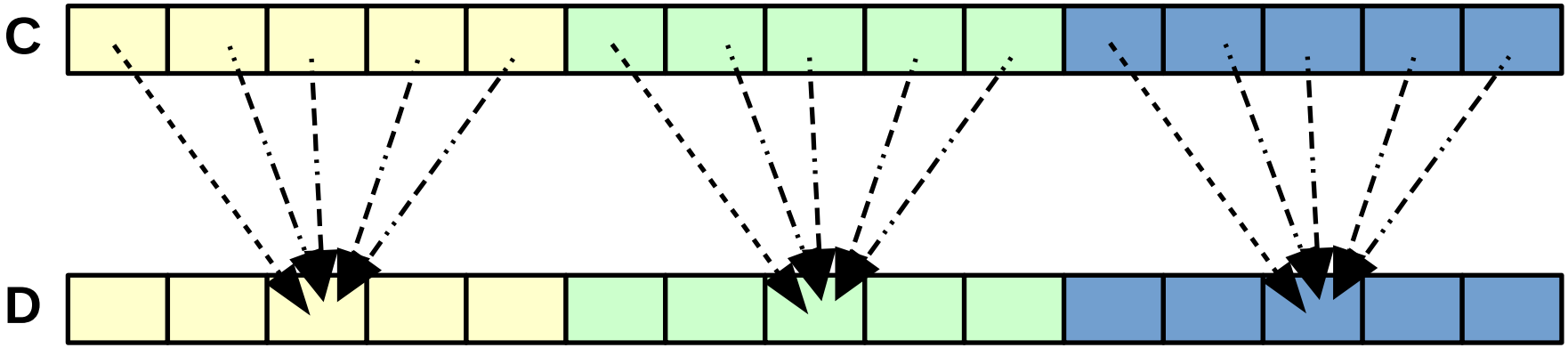
N = k × M , donde M ∈ {64, 128, 256} y k ∈ N



$$mx = \max\{C[i], \quad i = 0...N-1\}.$$

1.2 Implementación CUDA de una operación vectorial (3/4)

N = k × M , donde M ∈ {64, 128, 256} y k ∈ N



$$D[j] = \sum_{i \in B_j} C[j], \qquad j = 1, ..., k$$

Bj = conjunto de M índices del j-ésimo bloque.

1.2 Implementación CUDA de una operación vectorial (4/4)

- En la plataforma PRADO tenéis el código **transformacion.cc** que realiza estos cálculos en CPU.

Ejercicios Propuestos

1. Desarrollar **dos implementaciones CUDA C**: una que realiza el cálculo del vector C utilizando **variables en memoria compartida** y otra que solo utilice variables en memoria global. Para el resto de fases se recomienda usar memoria compartida.

2. **Comparar los resultados obtenidos**, tanto en tiempo de ejecución como en la exactitud de los valores obtenidos con respecto a las salidas del código CPU. Para ello, se utilizarán valores altos para N (N > 20000), probando los 3 tamaños de bloque indicados y 2 versiones desarrolladas.

```
// Compute C[i],  D[K]  and mx
for (int k=0; k<NBlocks;k++)
{ int istart=k*Bsize;
  int iend  =istart+Bsize;
  D[k]=0.0;
  for (int i=istart; i<iend;i++)
  { C[i]=0.0;
    for (int j=istart; j<iend;j++)
     { float a=A[j]*i;
       if ((int) ceil(a) % 2 ==0) C[i]+= a + B[j];
       else                       C[i]+= a - B[j];
     }
   D[k]+=C[i];
   mx=(i==1)?C[0]:max(C[i],mx);
  }
}
```