# Raptor Codes on GPUs

Masoud Alipour

Algo Day

Feb 11, 2011

# Raptor Codes on GPUs

- Standardized raptor codes
- Motivations:
  - Higher bandwidth for some applications
  - Off load the CPU
  - Scalable implementation
- Related works
  - LDPC on symmetric channels

# Standardized Raptor Codes

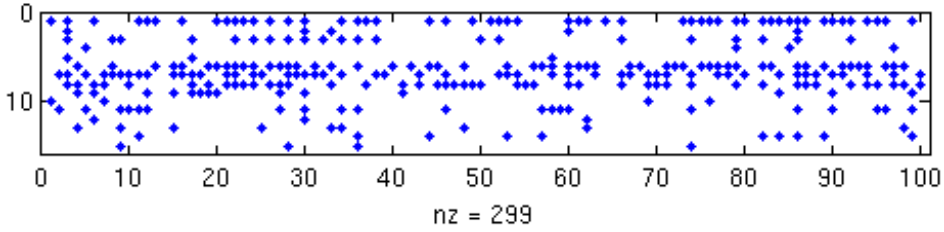
- Raptor 10
  - IETF RFC 5053, DVB, 3GPP, ITU, ATIS
  - Up to 8,192 source symbols
  - Up to 65,536 encoded symbols
- RaptorQ
  - Up to 56,403 source symbols
  - Up to 16,777,216 encoded symbols
  - Symbol size up to 65,535 bytes (max source block size of 3.4GB)

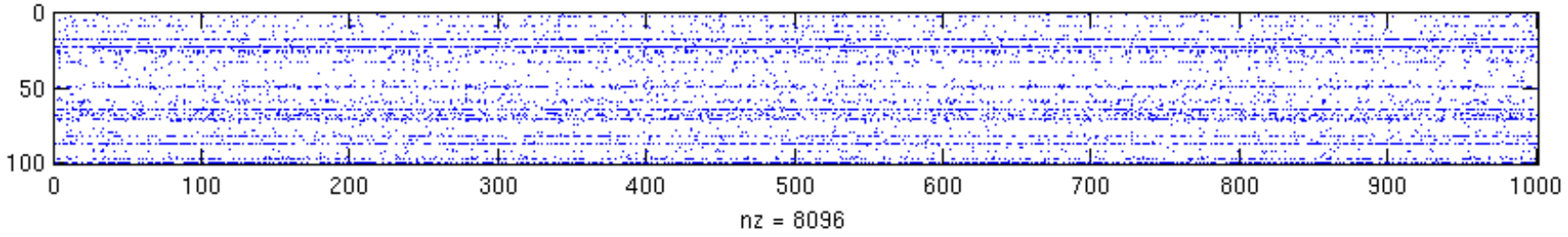
# Systematic Encoder

- Send the original source symbols + a few repair symbols
  - K = number of source symbols
  - R = number of repair symbols
  - N = symbol size
- First apply the decoder to source symbols to generate K temporary symbols, then apply the encoder to generate R repair symbols

# Systematic Encoder

- Offline Mode
- Store the generator matrix for desired parameters
- Independent of symbol size
- Mostly sparse rows with a few dense rows



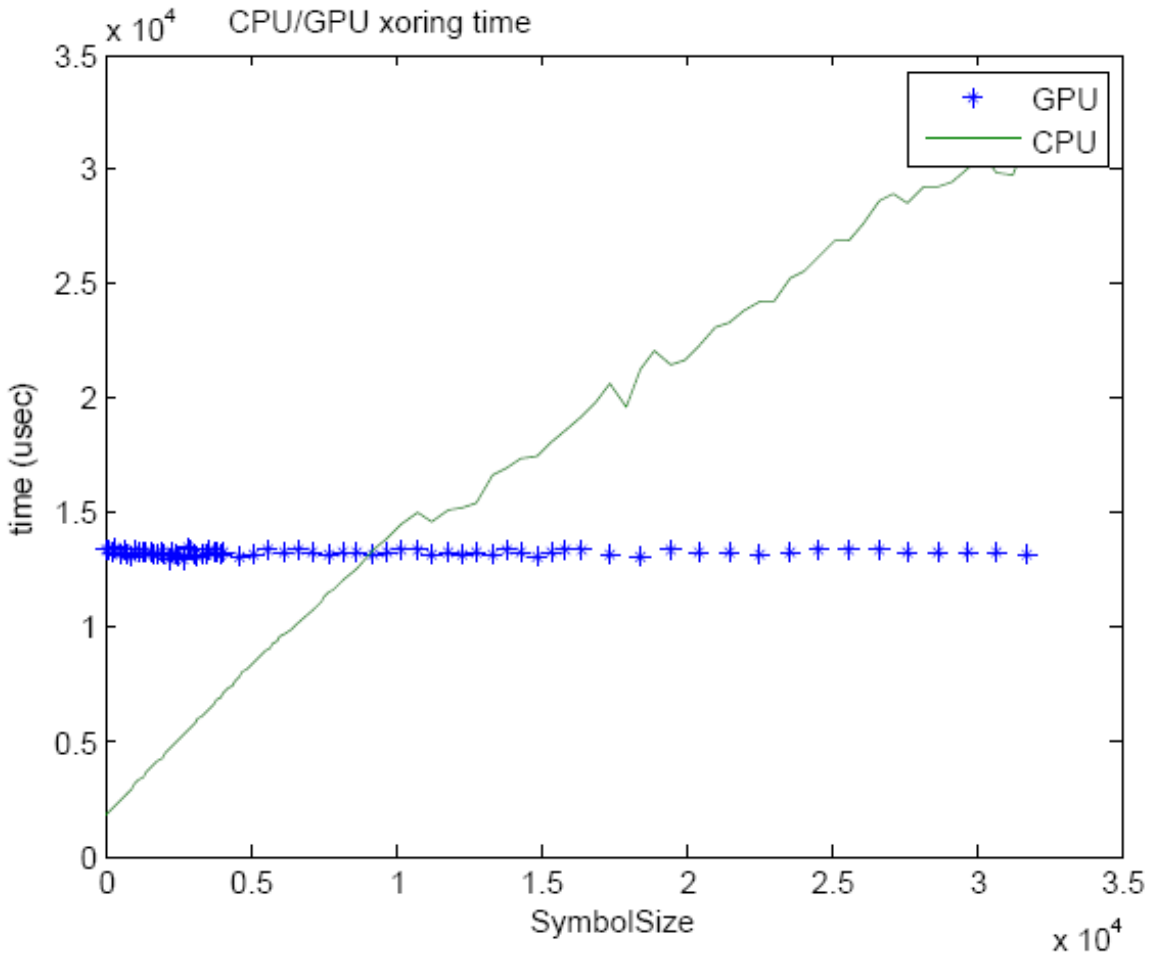



# Computational Cost

- Decoder
  - Inactivation decoding: Combination of message passing and matrix inversion
  - Requires Gaussian elimination (on smaller matrices) and efficient data structure (union-find) which are not parallelizeable
  - XOR of the symbols
- Encoder
  - XOR of the symbols according to the graph

# Computational Cost, XORing

- K = 100



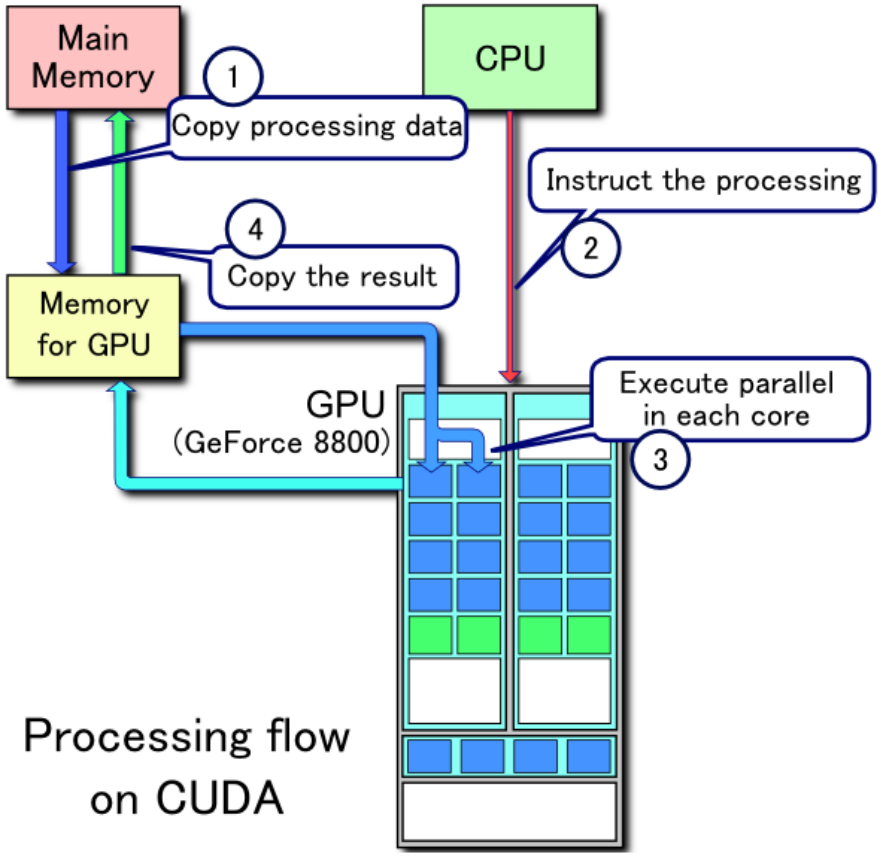
# Stream Processing

- Similar to SIMD
- Limited form of parallel processing
- Streams and Kernels
- Suitable for data-centric applications
- Many light threads

# GPU

## (Graphical Processing Units)

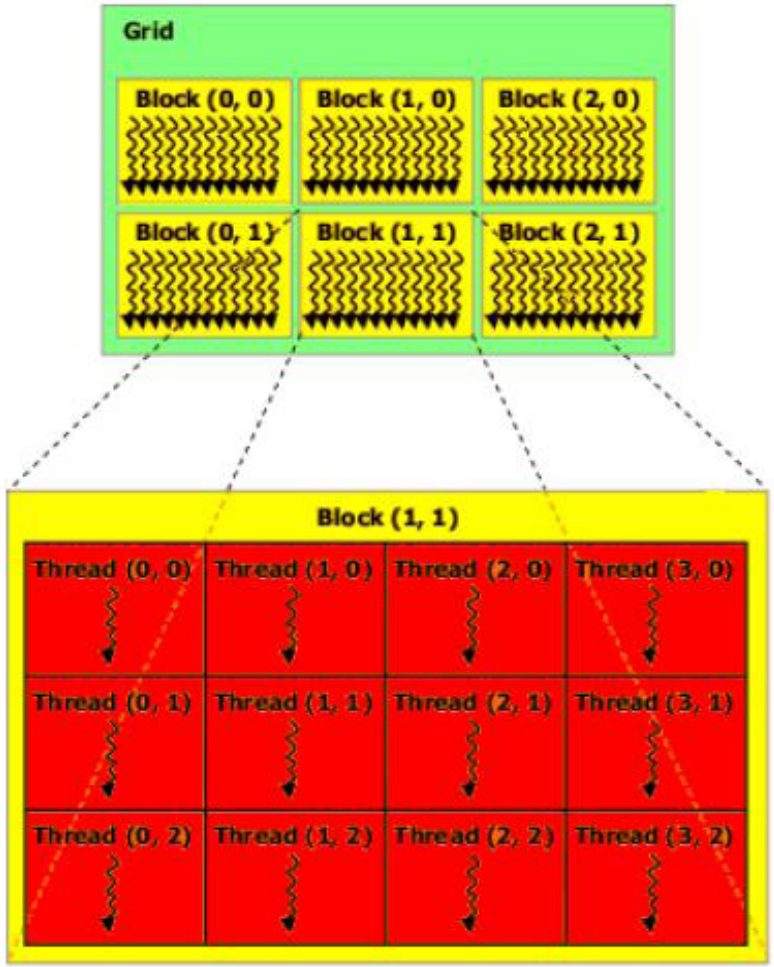
- Originally developed for 2D, 3D graphics
- GPGPU



# CUDA

## (Compute Unified Device Architecture)

- Organization of computational units: Grid, Block and threads
- Extension to C
- Scatter reads, shared memory
- Only for NVIDIA devices unlike OpenCL



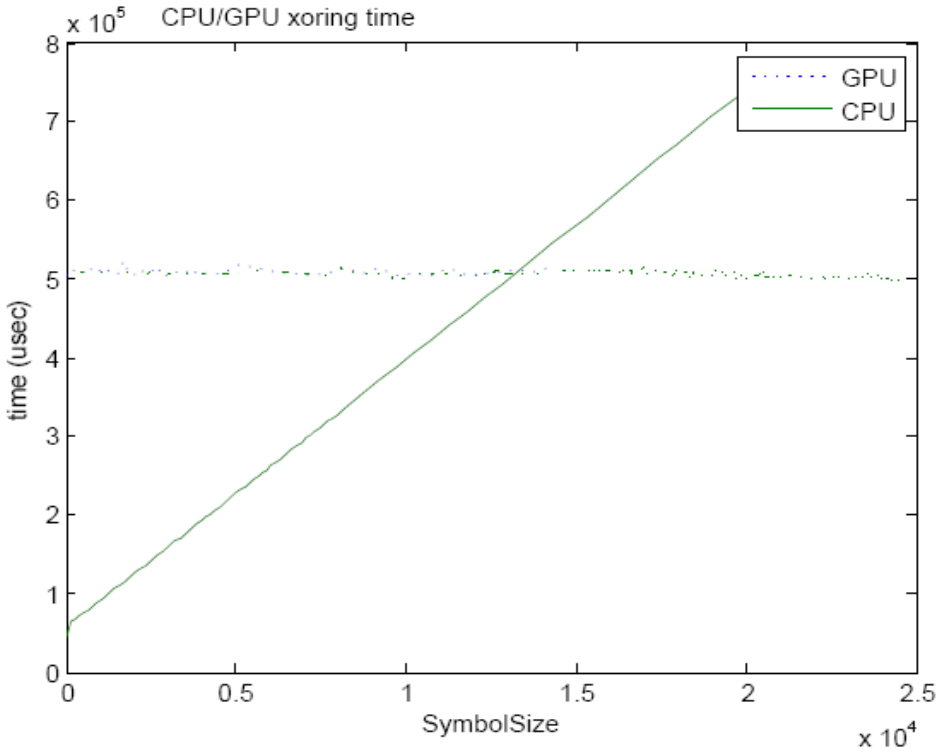
# Kernel

- Main working unit on CUDA
- Kernel locality
- The driver maps them to physical cores and threads

```
// Device code
__global__ void BinOp(const XOR_TYPE* A, const XOR_TYPE* B, XOR_TYPE* C, int N)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < N)
        C[i] = A[i] ^ B[i];
}
```

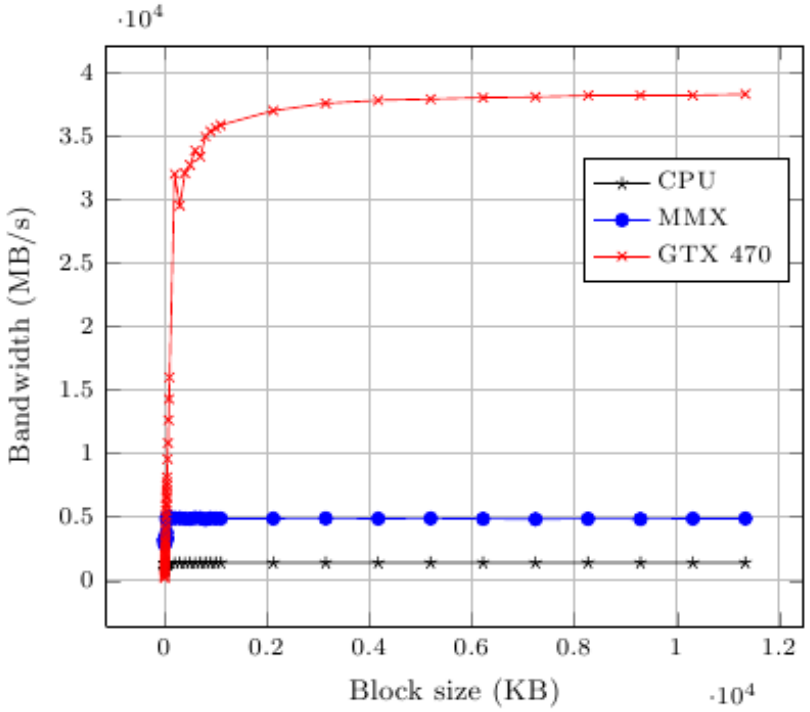
# Sequential Scheduler

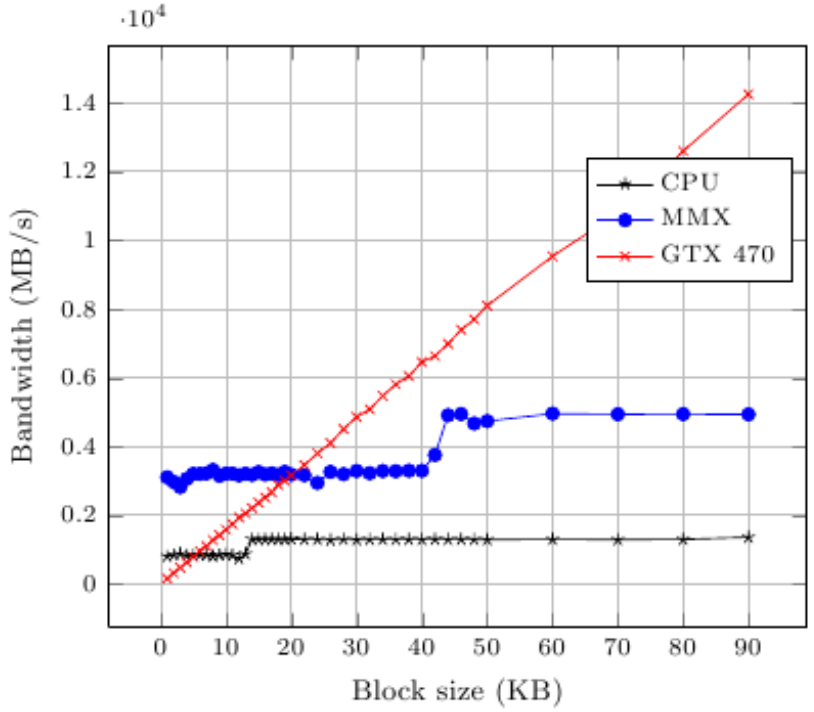
- Use the serial scheduler
- Load the source block to the device
- Call the kernel for on 2 symbols at a time



# Raw Performance

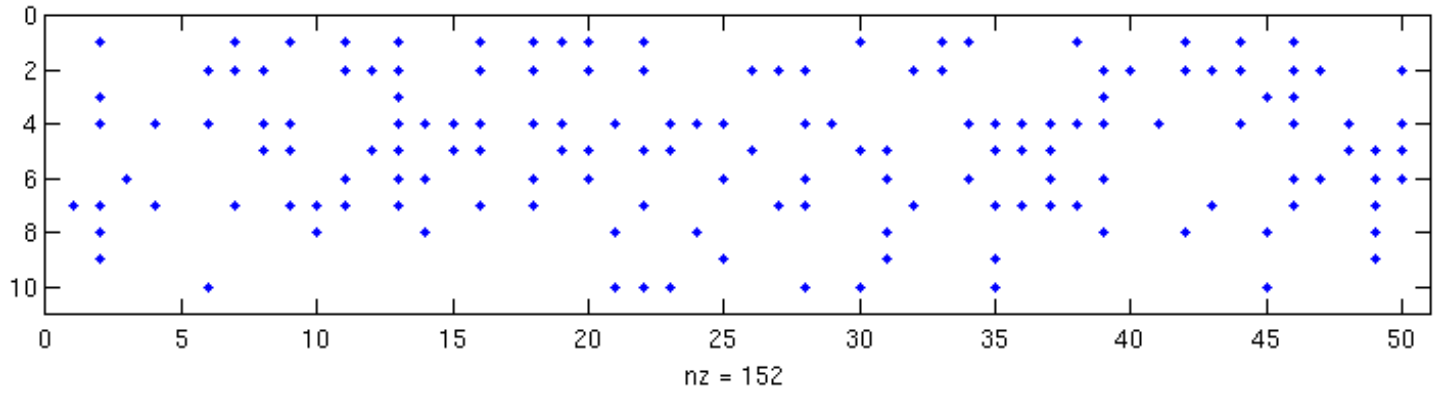
- Reference GPU:
  - Geforce GT 470
  - 448, 134 GB/s
- Reference CPU
  - 3 GHz Intel Core 2 Duo



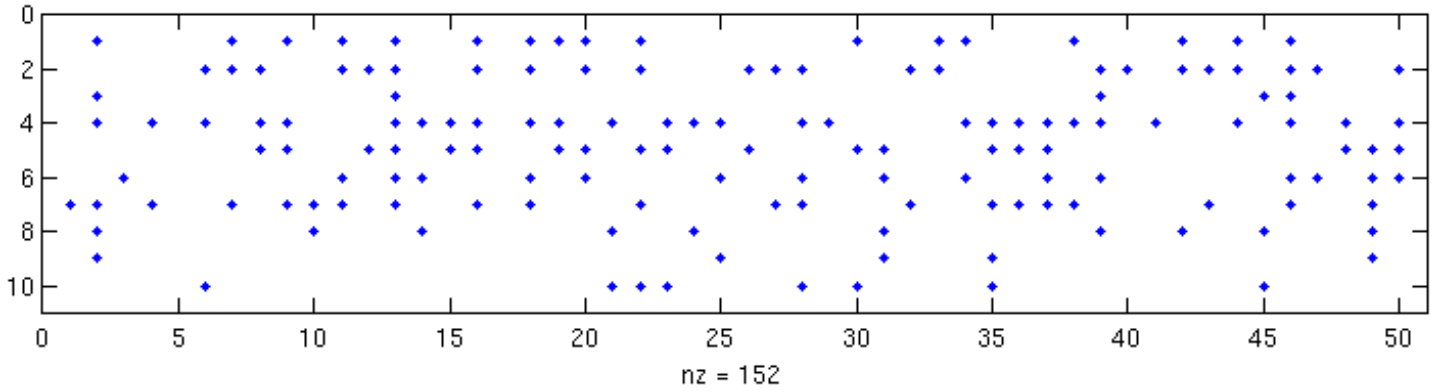
# Parallel Scheduler

- Design a new scheduler
  - Work with larger data blocks
  - Reduce number of kernel calls
  - At the cost of some unnecessary operations



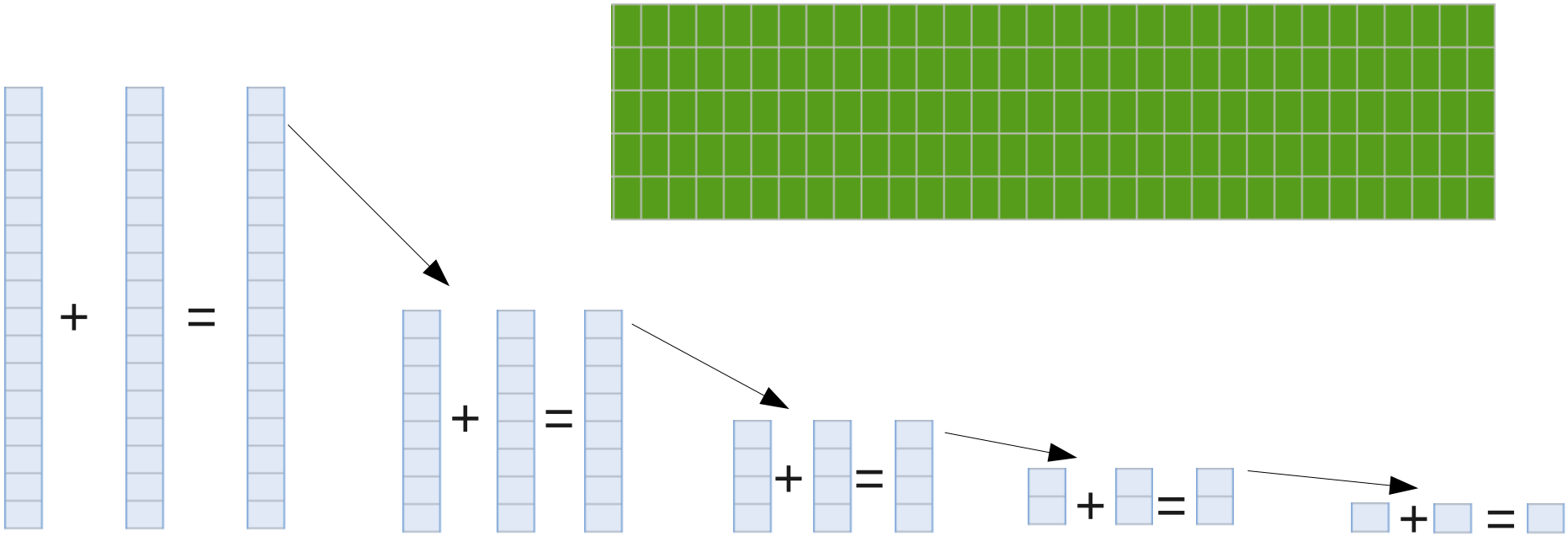
# First Approach

- Transfer the source block to the device
- At each step:
  - Copy the source symbol corresponding to none-zero elements in the generator matrix
  - Fill the other symbols to zero
  - Apply the kernel on the consecutive columns



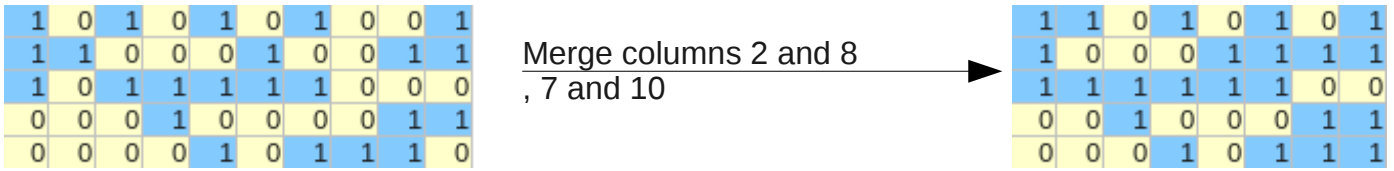
# Cooley–Tukey Technique

- Memory usage (at least two blocks of size K*R*N)
- Number of memory fills and transfers



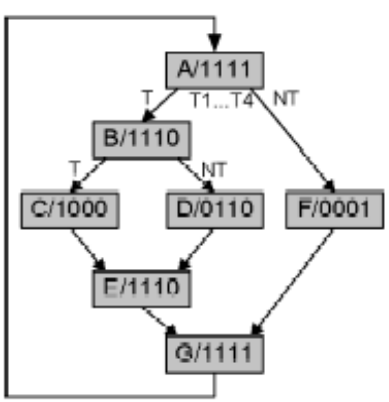
# Optimized Scheduler

- Try to avoid unnecessary and repetitious operations as much as possible while maintaining kernel locality
- Generate ESIs corresponding to sparse rows

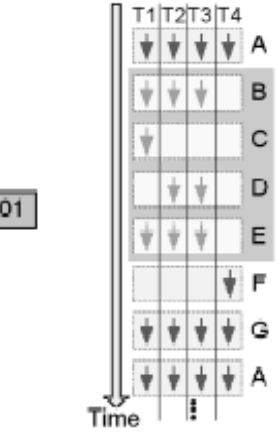


# Dynamic Warp Formation

- Out-of-order execution in pipe-lining
- Heuristics




(a) Control Flow Graph of an Example Program




(b) Resource Utilization vs. Time for Reconvergence at Immediate Postdominator of B

| | Next PC | Active Mask | Ret./Reconv. PC |
|---|---|---|---|
| | G | 1111 | - |
| | F | 0001 | G |
| TOS → | B | 1110 | G |

(c) Stack based Reconvergence: Initial State

| | Next PC | Active Mask | Ret./Reconv. PC | |
|---|---|---|---|---|
| | G | 1111 | - | |
| | F | 0001 | G | |
| | E | 1110 | G | (i) |
| | D | 0110 | E | (ii) |
| TOS → | C | 1000 | E | (iii) |

(d) Stack based Reconvergence: After Divergent Branch

| | Next PC | Active Mask | Ret./Reconv. PC |
|---|---|---|---|
| | G | 1111 | - |
| | F | 0001 | G |
| TOS → | E | 1110 | G |

(e) Stack based Reconvergence: After Reconvergence

Courtesy of TOR M. AAMODT