5th International Conference on
Logic and Application – LAP 2016
Dubrovnik, Croatia, September 19-23, 2016

Computational Efficiency of the GF and the RMF Transforms for Quaternary Logic Functions on CPUs and GPUs

Dušan B. Gajić[1], Radomir S. Stanković[2]

[1]Dept. of Computing and Control, Faculty of Technical Sciences, University of Novi Sad
Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia

[2]Dept. of Computer Science, Faculty of Electronic Engineering, University of Niš
Aleksandra Medvedeva 14, 18000 Nis, Serbia

E-mail: [1]dusan.b.gajic@gmail.com, [2]radomir.stankovic@gmail.com

Presentation Outline

1. **The Galois field** (GF) and **the Reed-Muller-Fourier** (RMF) transforms

2. **Graphics processing units** (GPUs) and **GPGPU**

3. Computing **GF and RMF transforms** of **quaternary logic functions on CPUs and GPUs**

4. Experimental results

5. Closing remarks

Spectral Transforms

signal (function)

↓ apply

spectral transform

↓ achieve

redistribution of information content

↓ perform in spectral domain

1. easier observation of some properties of signals
2. more efficient computation of certain operations

Applications:
Digital logic design
(spectral transforms over GF(p) and ring of integers modulo p),
Digital signal processing, pattern recognition…

Spectral Transforms

Spectral transforms are mathematical operators in linear vector spaces which assign to a function *f* a corresponding spectrum S_f defined as

$$\mathbf{S}_f = \mathbf{T}^{-1}\mathbf{F},$$

$\mathbf{T}$ - Matrix with basis functions as columns

$\mathbf{F}$ - Functional vector for *f*

$$f:\{0,1,\ldots,p-1\}^n \rightarrow \{0,1,\ldots,p-1\} \Rightarrow \mathbf{F} = [f(0), f(1),\ldots, f(p^n-1)]^T$$

$$\mathbf{S}_f = [s_f(0), s_f(1),\ldots, s_f(p^n-1)]^T \quad \mathbf{S}_f = \begin{bmatrix} transform \\ matrix \end{bmatrix} \cdot \mathbf{F} \Rightarrow O(N^2)$$

Function is reconstructed from the spectrum as: $\mathbf{F} = \mathbf{T}\mathbf{S}_f$

Fast algorithms are based on the **factorization** of the **transform matrix** into **sparse matrices** $\Rightarrow O(N \log N)$

Quaternary Logic Functions

- Quaternary logic functions (p = 4) are of special interest since they can be **easily encoded by binary values**
- They can be **realized by two-stable state circuits** in binary devices
- Genetic code can be viewed as a quaternary logic function – research in bioinformatics

Galois Field (GF) Transform for Quaternary Logic Functions

Polynomial expressions for a quaternary logic function of *n* variables

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=0}^{4^n-1} g_i \phi_i$$

$$g_i \in \{0,1,2,3\}$$

ϕ_i - basis functions (products of powers of variables)

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

·	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

$$\mathbf{F} = [f(0), f(1), \ldots, f(4^n - 1)]^T$$

$$\mathbf{S}_{f,4GF} = \mathbf{G}_{4GF}(n)\mathbf{F}$$

$$\mathbf{G}_{4GF}(n) = \bigotimes_{i=1}^{n} \mathbf{G}_{4GF}(1), \quad \mathbf{G}_{4GF}(1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 3 & 2 \\ 0 & 1 & 2 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Operations in the GF Transform

Field operations depend on the order of the considered finite (Galois) field.

p **prime** ⟹ programming implementation:

1. % operator from high-level languages
2. lookup tables (LUTs)

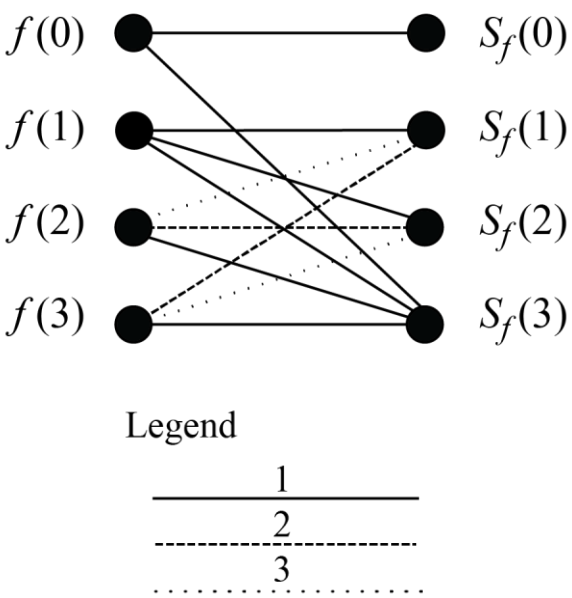
p **composite** ⟹ programming implementation:

1. lookup tables (LUTs)

Example: GF(4), $n = 2$

Basic transform matrix for GF(4):

$$\mathbf{G}_{4GF}(1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 3 & 2 \\ 0 & 1 & 2 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$



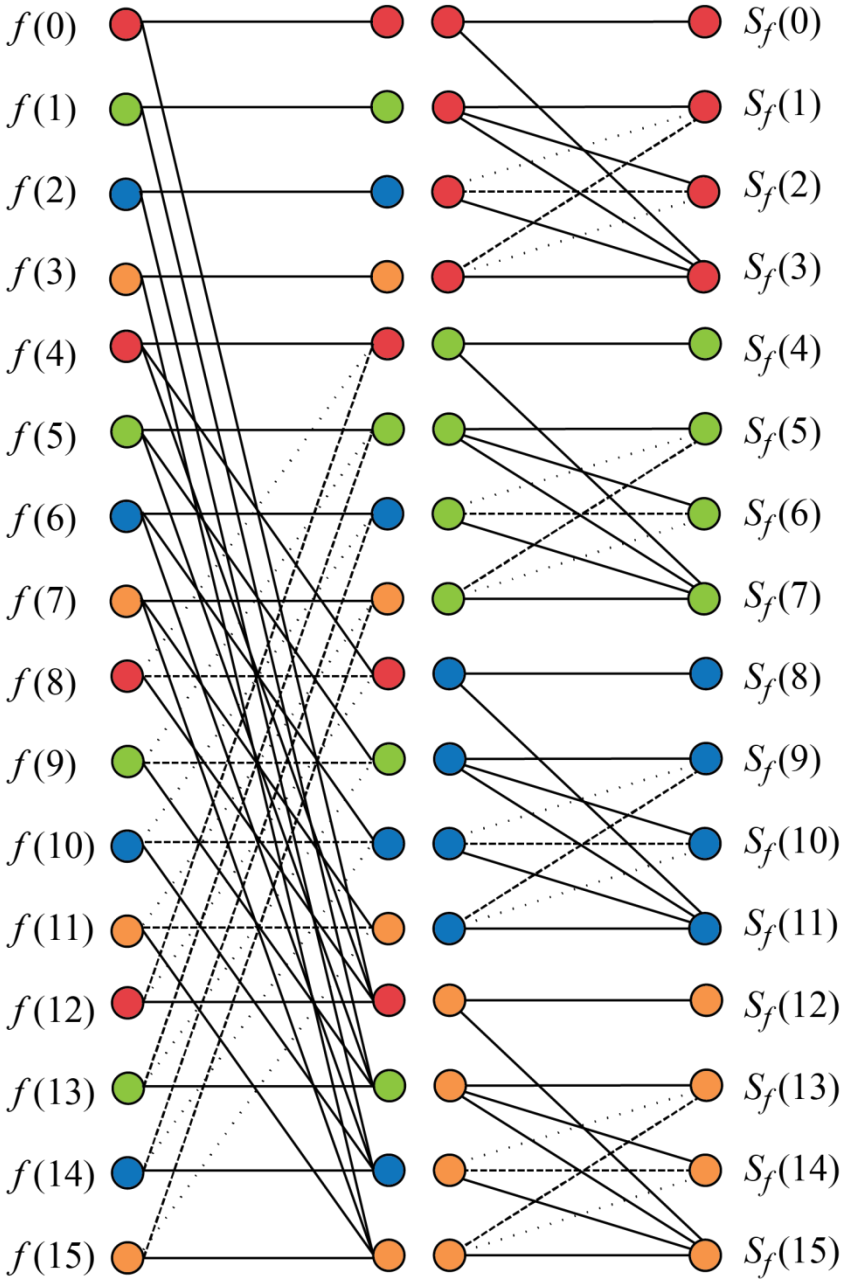
Cooley-Tukey factorization:

$$\mathbf{C}_1 = \mathbf{G}_{4GF}(1) \otimes \mathbf{I}$$

$$\mathbf{C}_2 = \mathbf{I} \otimes \mathbf{G}_{4GF}(1)$$

Example: GF(4)

$n = 2$



Reed-Muller-Fourier (RMF) Transform for Quaternary Logic Functions

Polynomial expressions for a quaternary logic function of *n* variables

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=0}^{4^n-1} g_i \phi_i$$

$$g_i \in \{0,1,2,3\}$$

ϕ_i - basis functions (products of powers of variables)

$\oplus$	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

$\cdot$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

$$\mathbf{F} = [f(0), f(1), \ldots, f(4^n - 1)]^T$$

$$\mathbf{S}_{f,4RMF} = \mathbf{R}_{4RMF}(n)\mathbf{F}$$

$$\mathbf{R}_{4RMF}(n) = \bigotimes_{i=1}^{n} \mathbf{R}_{4RMF}(1), \quad \mathbf{R}_{4RMF}(1) = 3\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 3 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 1 & 1 & 3 & 3 \end{bmatrix}$$

Operations in the RMF Transform

Introduced by **changing the underlying algebraic structure** into the **Gibbs algebra**

Group operation is **modulo *p* addition** for **all positive integer values of *p***, while **multiplication** is a **convolution-wise (Gibbs) multiplication**

all positive integer values of *p*

↓

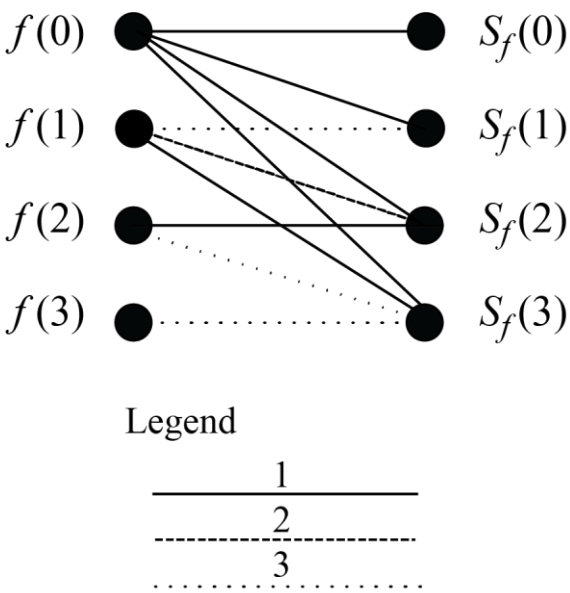
programming implementation:

1. % operator from high-level languages
2. lookup tables (LUTs)

Example: RMF(4), $n = 2$

Basic transform matrix for RMF(4):

$$\mathbf{R}_{4RMF}(1) = 3\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 3 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 1 & 1 & 3 & 3 \end{bmatrix}$$



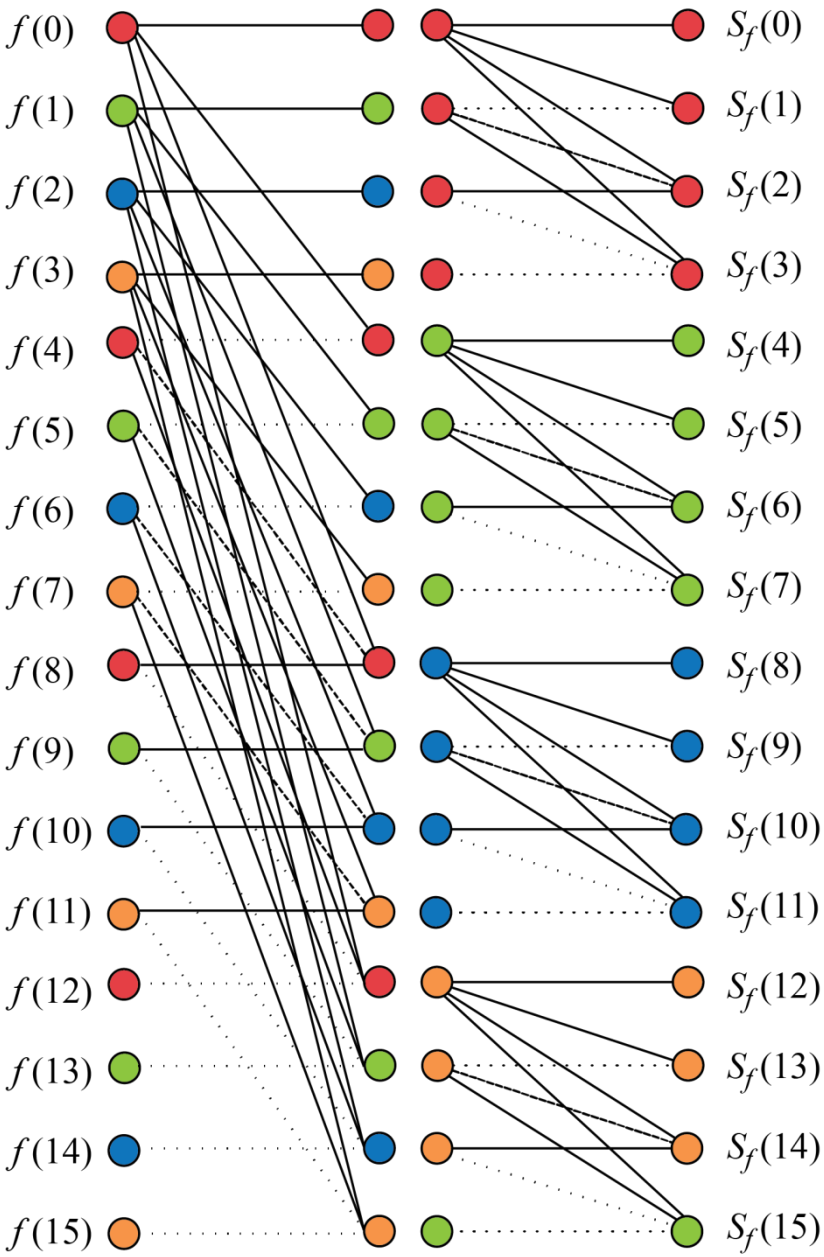
Cooley-Tukey factorization:

$$\mathbf{C}_1 = \mathbf{R}_{4RMF}(1) \otimes \mathbf{I}$$

$$\mathbf{C}_2 = \mathbf{I} \otimes \mathbf{R}_{4RMF}(1)$$

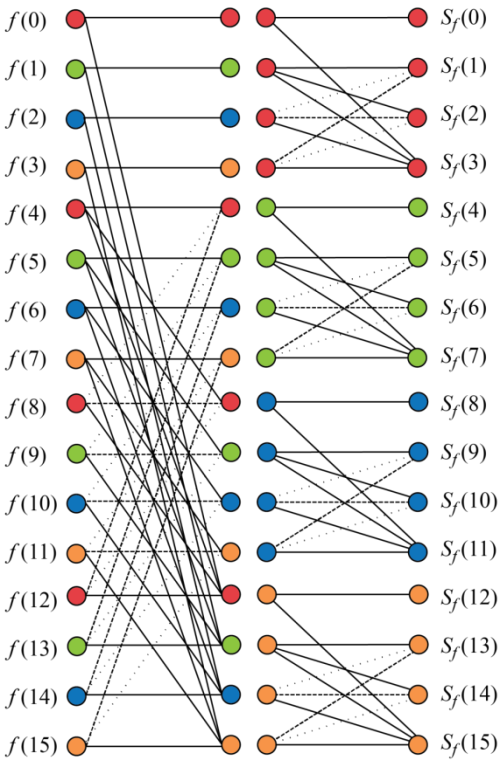
Example: RMF(4)

$n = 2$

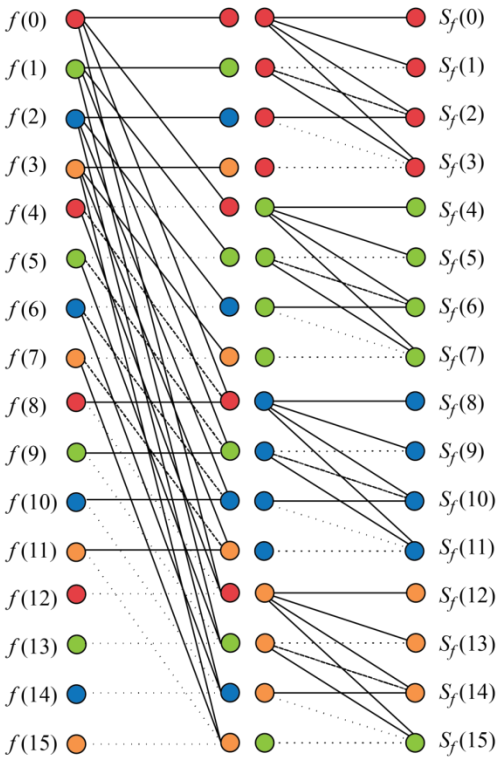


Comparison of Algorithms

GF(4)



RMF(4)



RMF has a triangular transform matrix (smaller number of operations)
RMF for many functions offers less non-zero spectral coefficients
Different arithmetic operations, modulo p instead GF-operations

Graphics Processing Unit (GPU)

Graphics processing unit (GPU) is a hardware device originally specialized for rendering computer graphics

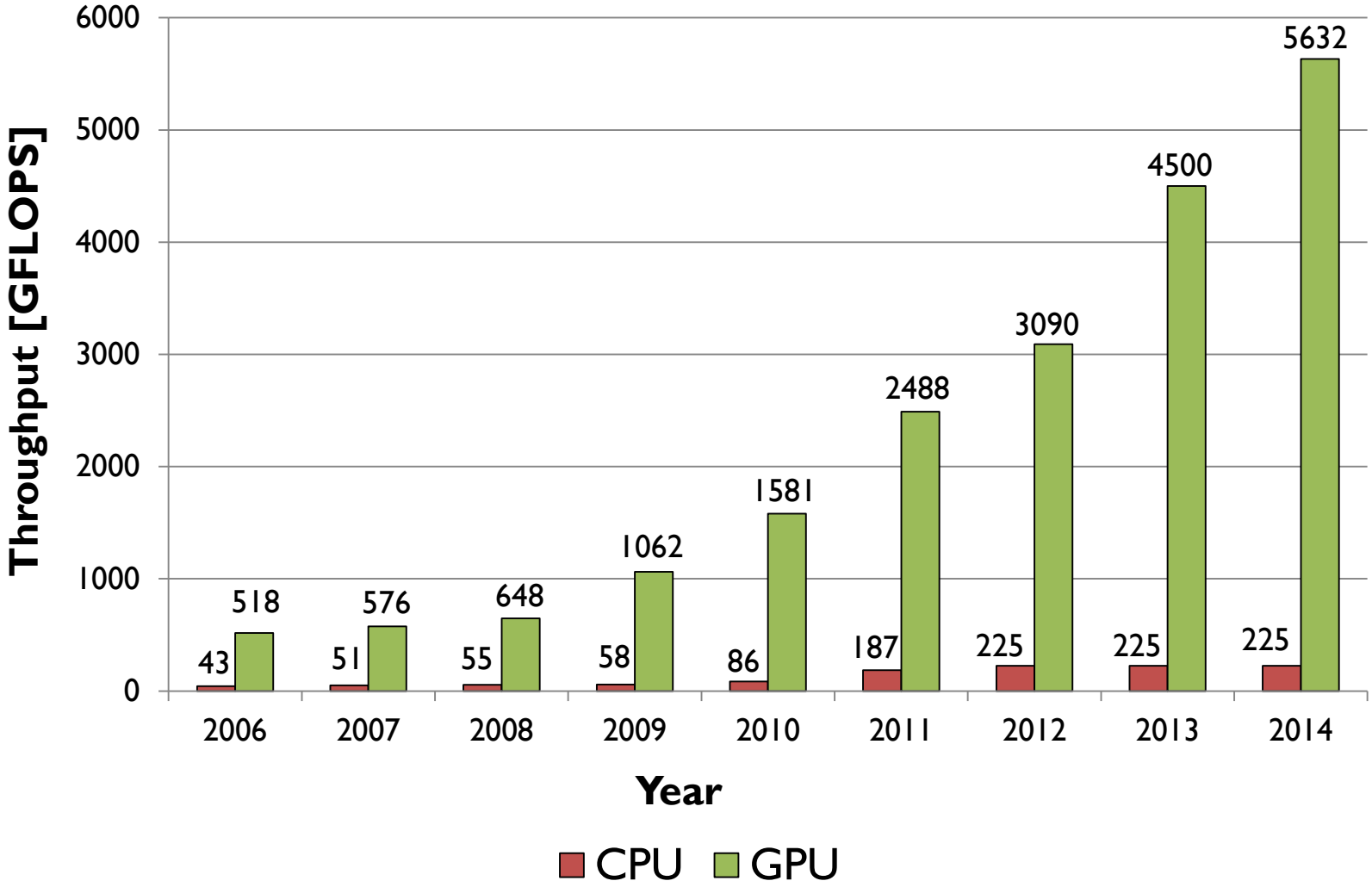
The first GPU appeared in 1999

Early 2000s: fixed-function processors dedicated to rendering computer graphics

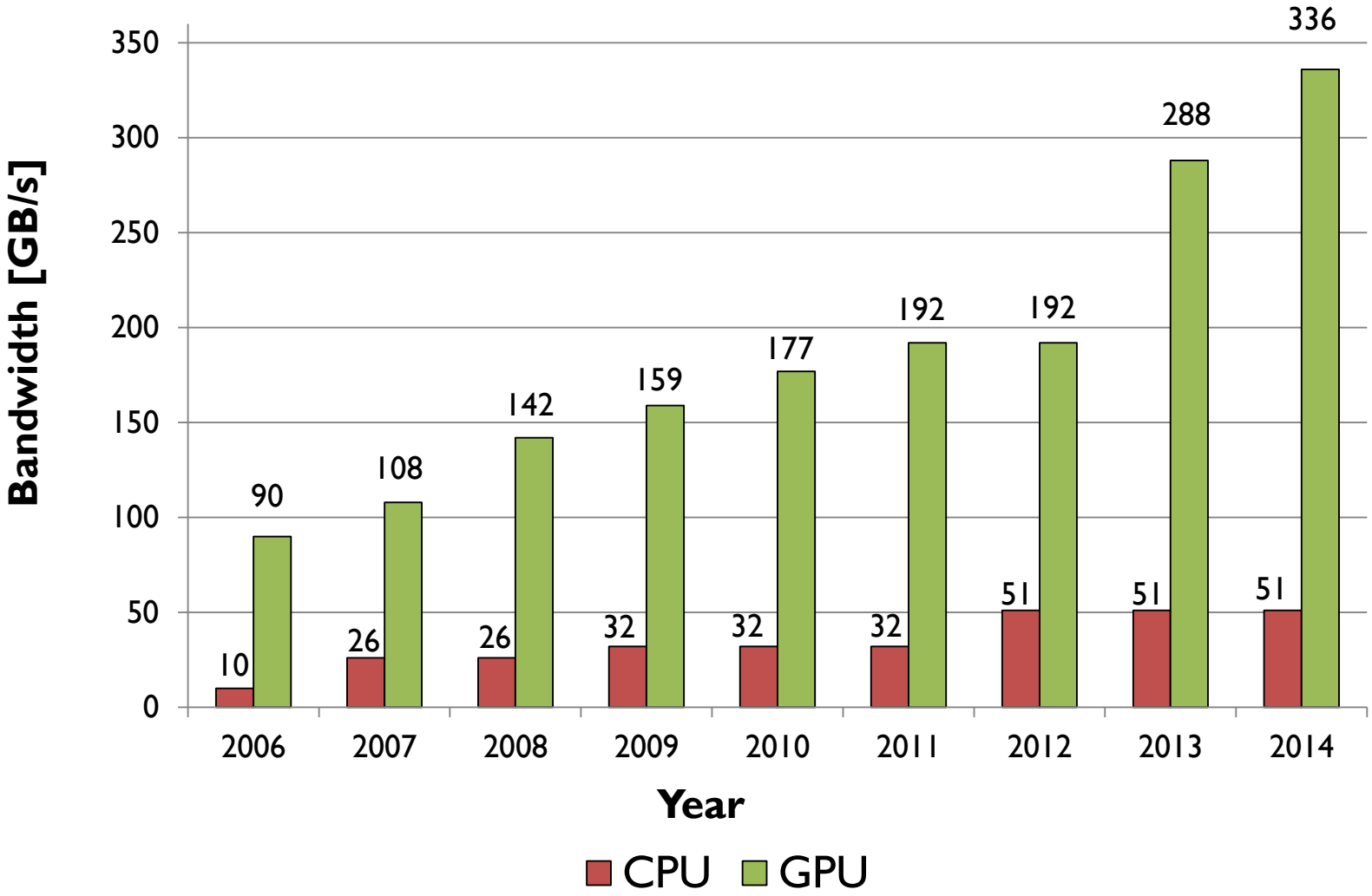
Presently**: a unified programmable graphics processor** and **a parallel computing platform**

GPU design philosophy is oposite to the design of CPUs (**throughput** vs **latency**) ⟹ different programming philosophy

CPU and GPU Throughput



CPU and GPU Bandwidth



GPU Computing (GPGPU)

General purpose computations on the GPU (GPGPU or GPU computing)

GPU features:

- **manycore architecture**
- **high throughput** and **processing power**
- **lower cost** and **smaller energy consumption**

Suitable for **intensive computations** and **large data processing**

Nvidia CUDA (high performance, exclusive for *Nvidia* GPUs), appeared in 2007

OpenCL (open standard, acceleration on heterogeneous devices (CPUs, GPUs, DSPs, FPGAs), appeared in 2009

GPU Computing Programs

A GPGPU program is composed of:

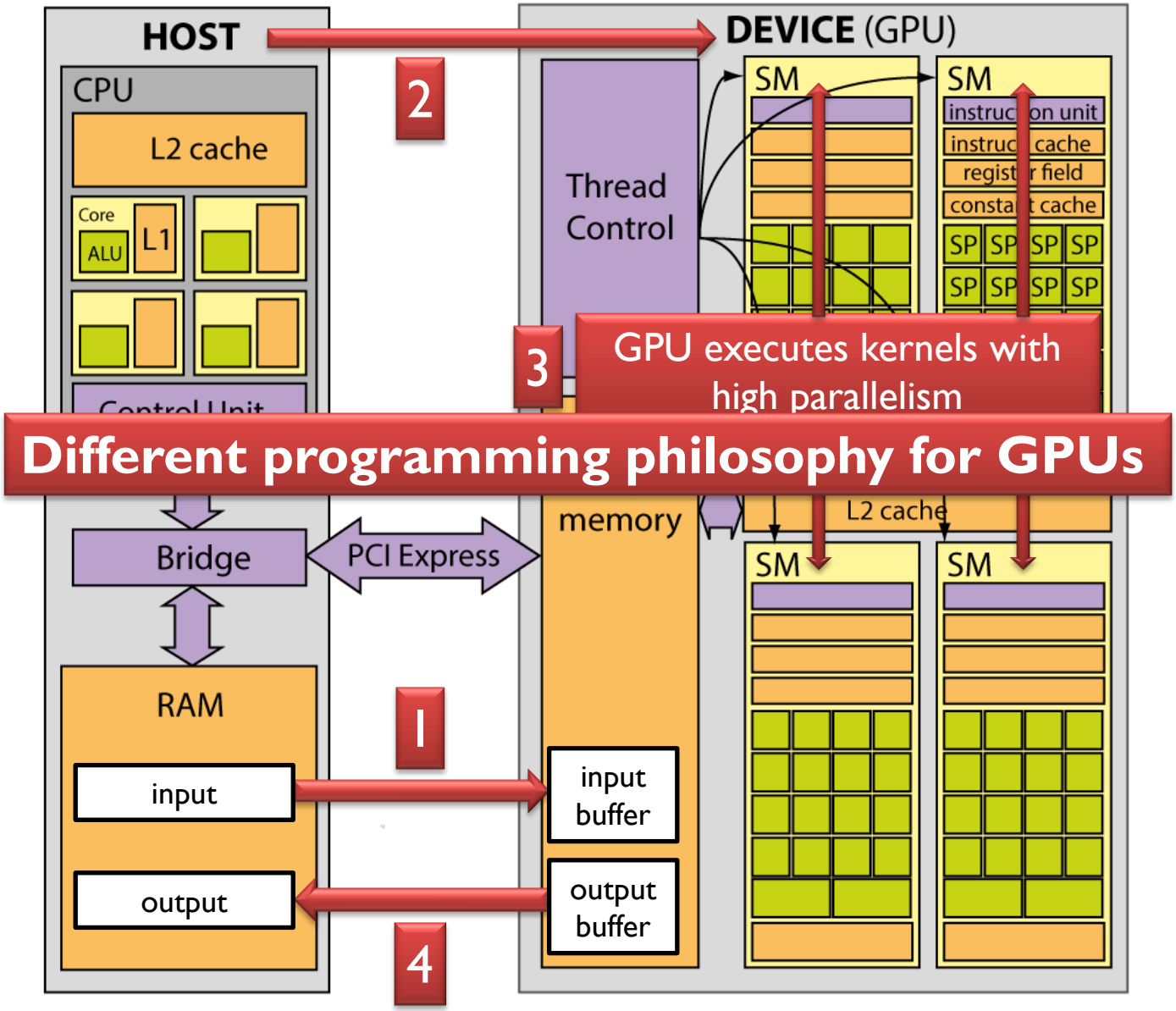
1. **host program** (processed on CPUs, controls execution) and
2. **device program** (processed on GPUs, implements kernels)

Kernel is a data-parallel function executed on a GPU

Each **kernel** describes computations performed by a **single thread**

Block (set of threads) and **grid** (set of blocks) configurations defined in the host program

GPU Architecture and Computing Model



Implementation of Operations for $p = 4$

Randomly generated quaternary logic function vectors **F**(n)

On the **CPU C++**, on the **GPU CUDA C**

Group operation was implemented in C++ and CUDA C using

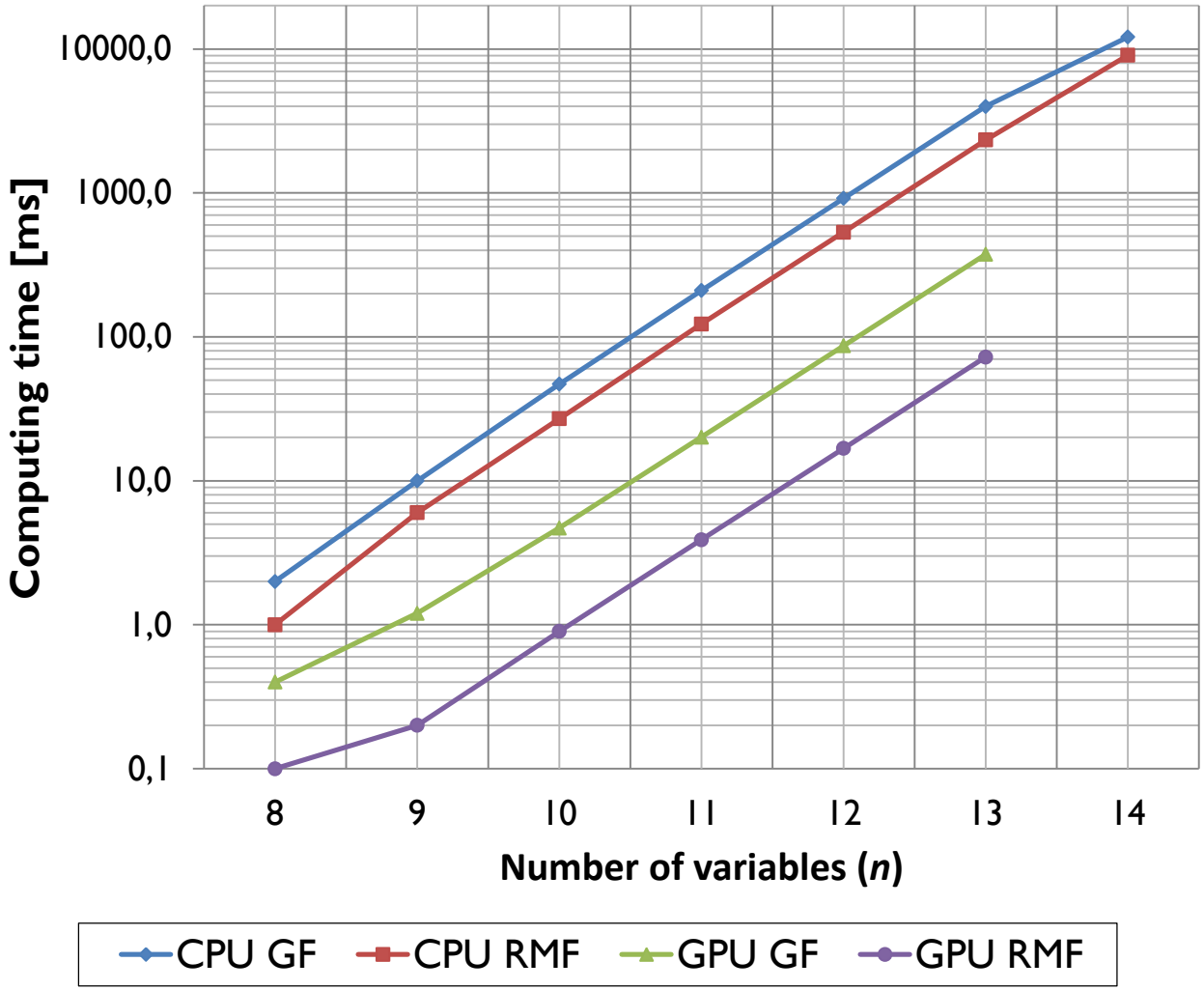
- **LUTs** for **GF(4)**
- **modulo arithmetic operator** % for **RMF(4)**

On GPUs there is additional time for memory transfers

Experimental Platforms

Component	Platform 1 (Desktop)	Platform 2 (Workstation)
CPU microarchitecture clock (GHz) processing power (GFLOPS) cores/threads	Intel Core i7-920 Bloomfield 2.66 28 4/8	Intel Xeon E5-1620 Haswell 3.5 122 4/8
RAM	12GB DDR3 2000 MHz	32GB DDR4 ECC 2133 MHz
GPU microarchitecture processing power (GFLOPS) cores memory type bandwidth (GB/s)	Nvidia GTX 560 Ti Fermi 1263 384 1 GB GDDR5 128 GB/s	Nvidia Quadro K620 Kepler 768 384 2 GB DDR3 28.8 GB/s
OS	Windows 7 64-bit	Windows 10 64-bit
GPU SDK	Nvidia GPU Computing 7.5	Nvidia GPU Computing 7.5

Experimental Results – Platform 1 (Desktop)



Experimental Results – Platform 1 (Desktop)

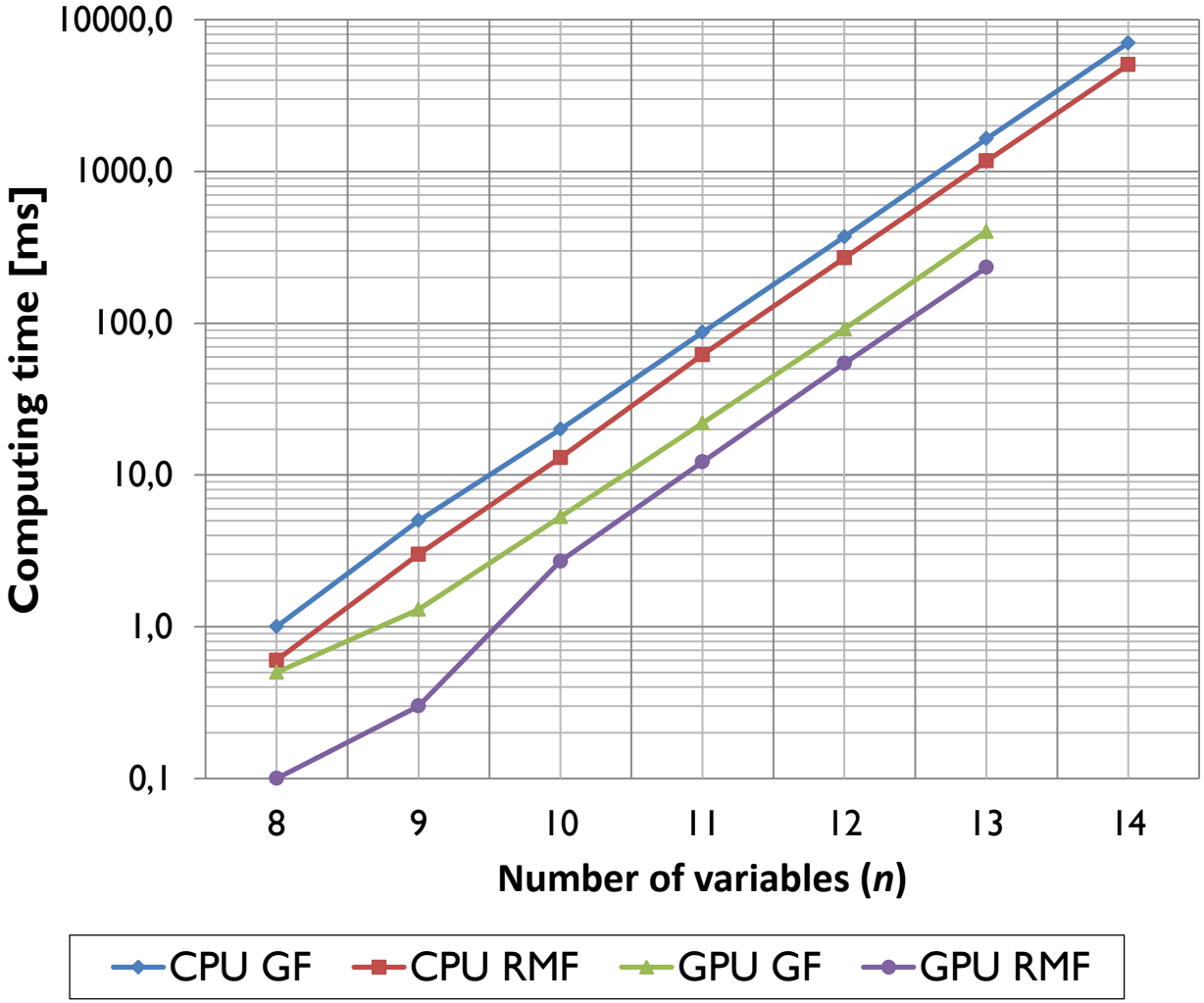
n	Processing time [ms]				
	CPU/C++		GPU/CUDA		
	GF	RMF	GF	RMF	Memory
8	2	1	0.4	0.1	0.1
9	10	6	1.2	0.2	0.4
10	47	27	4.7	0.9	1.5
11	210	123	20.1	3.9	5.9
12	917	534	86.8	16.8	23.6
13	3994	2337	374.8	72.4	94.0
14	12119	9057	-	-	-

On the CPU, RMF is from 1.3× to 2× faster than GF

On the GPU, RMF is from 4× to 6× faster than GF

Computing on GPUs is from 10× to 33× faster than on CPUs

Experimental Results – Platform 2 (Workstation)



Experimental Results – Platform 2 (Workstation)

n	Processing time [ms]				
	CPU/C++		GPU/CUDA		
	GF	RMF	GF	RMF	Memory
8	1	0.6	0.5	0.1	0.1
9	5	3	1.3	0.3	0.4
10	20	13	5.3	2.7	1.3
11	87	62	22.0	12.2	5.3
12	371	269	91.6	54.3	21.3
13	1643	1171	402.0	233.1	85.0
14	7032	5047	-	-	-

On the CPU, RMF is from 1.4× to 1.7× faster than GF

On the GPU, RMF is from 1.7× to 5× faster than GF

Computing on GPUs is from 2× to 5× faster than on CPUs

Closing Remarks

Performance comparison of computing the GF and the RMF transforms for quaternary logic functions on CPUs and GPUs

Modulo operators in RMF(4) outperform LUTs in GF(4) by 1.3× to 2× on CPUs

Modulo operators in RMF(4) outperform LUTs in GF(4) by 1.7× to 6× on GPUs

For considered tasks, GPUs are almost an order of magnitude faster than CPUs

The computational advantage of RMF over GF increases on novel computing architectures

5th International Conference on
Logic and Application – LAP 2016
Dubrovnik, Croatia, September 19-23, 2016

Computational Efficiency of the GF and the RMF Transforms for Quaternary Logic Functions on CPUs and GPUs

Dušan B. Gajić[1], Radomir S. Stanković[2]

[1]Dept. of Computing and Control, Faculty of Technical Sciences, University of Novi Sad
Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia

[2]Dept. of Computer Science, Faculty of Electronic Engineering, University of Niš
Aleksandra Medvedeva 14, 18000 Nis, Serbia

E-mail: [1]dusan.b.gajic@gmail.com, [2]radomir.stankovic@gmail.com