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

Dušan B. Gajić¹, Radomir S. Stanković²

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

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

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

$$S_f = T^{-1}F,$$

- Matrix with basis functions as columns

$F$ - Functional vector for $f$

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

$$S_f = [s_f(0), s_f(1),..., s_f(p^n - 1)]^T$$

$$S_f = \begin{bmatrix} \text{transform} \\ \text{matrix} \end{bmatrix} \cdot F \Rightarrow O(N^2)$$

Function is reconstructed from the spectrum as: $F = TS_f$

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

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

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

$\phi_i$ - basis functions (products of powers of variables)

$$F = [f(0), f(1), ..., f(4^n - 1)]^T$$

$$S_{f,4GF} = G_{4GF}(n)F$$

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

\[
G_{4GF}(1) = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 3 & 2 \\
0 & 1 & 2 & 3 \\
1 & 1 & 1 & 1 \\
\end{bmatrix}
\]

Cooley-Tukey factorization:

\[
C_1 = G_{4GF}(1) \otimes I \\
C_2 = I \otimes G_{4GF}(1)
\]
Example:
GF(4)

$n = 2$
Reed-Muller-Fourier (RMF) Transform for Quaternary Logic Functions

Polynomial expressions for a quaternary logic function of \( n \) variables

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

\( g_i \in \{0, 1, 2, 3\} \)

\( \phi_i \) - basis functions (products of powers of variables)

\[
\begin{align*}
F &= [f(0), f(1), \ldots, f(4^n - 1)]^T \\
S_{f,4\text{RMF}} &= R_{4\text{RMF}}(n)F \\
R_{4\text{RMF}}(n) &= \bigotimes_{i=1}^{n} R_{4\text{RMF}}(1),
R_{4\text{RMF}}(1) &= 3
\end{align*}
\]
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):

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

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

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 opposite to the design of CPUs (throughput vs latency) different programming philosophy.
CPU and GPU Throughput

![Graph showing CPU and GPU Throughput from 2006 to 2014. The graph compares the throughput in GFLOPS for each year, with CPU and GPU performances indicated by different bar colors. The data shows a significant increase in throughput over the years, particularly for GPUs.](image-url)
CPU and GPU Bandwidth

![Bar chart showing CPU and GPU bandwidth over the years from 2006 to 2014. The chart compares the bandwidth in GB/s for CPUs and GPUs, with a notable increase in bandwidth for GPUs particularly from 2010 onwards.](chart.png)
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

1. RAM
   - input
   - output

2. Host CPU
   - L2 cache
   - Control Unit
   - L1
   - ALU

3. Device GPU
   - SM
   - Instruction unit
   - Register field
   - Constant cache
   - SP
   - Thread Control

4. Memory
   - Input buffer
   - Output buffer

GPU executes kernels with high parallelism

Different programming philosophy for GPUs
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

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

![Graph showing computational time vs. number of variables (n) for different platforms and transforms.]

- **CPU GF**
- **CPU RMF**
- **GPU GF**
- **GPU RMF**

Number of variables (n) vs. Computing time in milliseconds (ms).
Experimental Results – Platform 1 (Desktop)

<table>
<thead>
<tr>
<th>$n$</th>
<th>Processing time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU/C++</td>
</tr>
<tr>
<td></td>
<td>GF</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>47</td>
</tr>
<tr>
<td>11</td>
<td>210</td>
</tr>
<tr>
<td>12</td>
<td>917</td>
</tr>
<tr>
<td>13</td>
<td>3994</td>
</tr>
<tr>
<td>14</td>
<td>12119</td>
</tr>
</tbody>
</table>

On the CPU, RMF is from $1.3 \times$ to $2 \times$ faster than GF

On the GPU, RMF is from $4 \times$ to $6 \times$ faster than GF

Computing on GPUs is from $10 \times$ to $33 \times$ faster than on CPUs
Computational Efficiency of the GF and the RMF Transforms for Quaternary Logic Functions on CPUs and GPUs

Experimental Results – Platform 2 (Workstation)

![Graph showing computational time vs. number of variables for different platforms and transforms.]

- **CPU GF**
- **CPU RMF**
- **GPU GF**
- **GPU RMF**

The graph depicts the computing time in milliseconds (ms) for varying numbers of variables ($n$) on different platforms and transforms.
### Experimental Results – Platform 2 (Workstation)

<table>
<thead>
<tr>
<th>n</th>
<th>CPU/C++</th>
<th>GPU/CUDA</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>GF</td>
<td>RMF</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>1.3</td>
<td>0.3</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>5.3</td>
<td>2.7</td>
</tr>
<tr>
<td>11</td>
<td>87</td>
<td>22.0</td>
<td>12.2</td>
</tr>
<tr>
<td>12</td>
<td>371</td>
<td>91.6</td>
<td>54.3</td>
</tr>
<tr>
<td>13</td>
<td>1643</td>
<td>402.0</td>
<td>233.1</td>
</tr>
<tr>
<td>14</td>
<td>7032</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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

Dušan B. Gajić¹, Radomir S. Stanković²

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

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

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