# Efficient Storage for Small Finite Fields

## Analysis of two techniques to reduce memory requirements and computation time

Bryan Youse, Dr. B David Saunders: University of Delaware, Newark, Delaware

# [Concept]

Consider this set of 64 elements in $F_3$

| index | 0 | 1 | 2 | ... | 18 | 19 | 20 | 21 | 22 | 23 | ... | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| value | 1 | 2 | 0 | ... | 1 | 1 | 2 | 2 | 1 | 0 | ... | 0 | 1 | 2 |

## Bit Packing

Multiple field elements are packed into few bits and aligned side-by-side to fit within a single machine word:

| index | - | 0 | 1 | 2 | ... | 18 | 19 | 20 | - | 21 | 22 | 23 | ... | - | ... | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| value | - | 1 | 2 | 0 | ... | 1 | 1 | 2 | - | 2 | 1 | 0 | ... | - | ... | 0 | 1 | 2 |
| bits | 0 | 001 | 010 | 000 | ... | 001 | 001 | 010 | 0 | 010 | 001 | 000 | ... | 0 | ... | 000 | 001 | 002 |
| | machine word 1 {bits 0-63} | | | | | | | | machine word 2 {bits 64-127} | | | | | machine word 3 {bits 128-191} | | | | |

#bits/value is the "packing factor" (pf). Example is of pf = 3.

## Bit Slicing

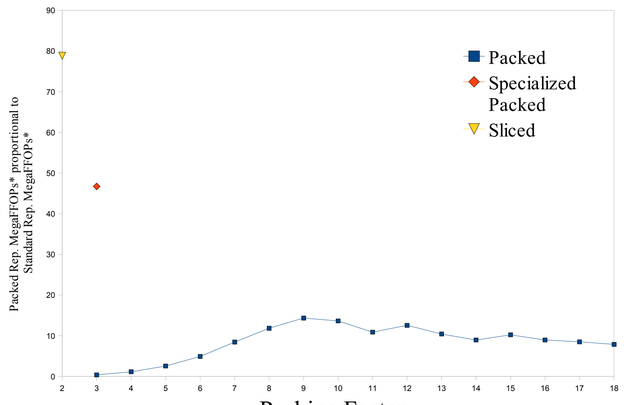
Field elements, represented by few bits, are striped across multiple machine words:

| index | 0 | 1 | 2 | ... | 18 | 19 | 20 | 21 | 22 | 23 | ... | 61 | 62 | 63 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| value | 1 | 2 | 0 | ... | 1 | 1 | 2 | 2 | 1 | 0 | ... | 0 | 1 | 2 | |
| bits ($2^0$) | 1 | 0 | 0 | ... | 1 | 1 | 0 | 0 | 1 | 0 | ... | 0 | 1 | 0 | machine word 1 |
| bits ($2^1$) | 0 | 1 | 0 | ... | 0 | 0 | 1 | 1 | 0 | 0 | ... | 0 | 0 | 1 | machine word 2 |

# [Application & Data]

Our eventual aim is to implement a complete arithmetic suite for both bit-packing and bit-slicing. Our immediate application, computing the rank of matrices of large order [2], calls for vector addition, multiplication, and axpy.

Normalized axpy gains: alternate representations for F3




*MegaFFOPs = Millions of Finite Field Operations/Second

Memory Comparison, 5000x5000 Matrix

| | pf | Memory Usage, MB (Approx.) | Computation Speed Notes per Field |
|---|---|---|---|
| **Standard Representation** | - | 255 | |
| **Bit Packing** | 3 | 18 | |
| | 4 | 23 | |
| | 5 | 29 | F5 slower than standard* |
| | 6 | 35 | F7 slower than standard* |
| | 8 | 42 | |
| | 9 | 48 | Optimal** for generic F3 |
| | 10 | 55 | Optimal** for F5 |
| | 12 | 66 | Optimal** for F7 |
| **Bit Slicing** | - | 15 | |

* Due to having to normalize too often
** Fastest running times on average across add, mul, axpy

# [Specializations]

Bit packing requires normalization after a number of arithmetic operations proportional to **pf** to avoid spilling carry bits.
For bit packing on $F_3$, with **pf** = 3:
concept of **semi-normalization:** Resetting the MSB (third bit) to zero. Once MSB is "open", field arithmetic can be performed freely on the packed values.

| Value (binary) | SemiNormalized |
|---|---|
| 000 | 000 |
| 001 | 001 |
| 010 | 010 |
| 011 | 011 |
| 100 | 001 |
| 101 | 010 |
| 110 | 011 |

Our implementation of bit slicing for $F_3$ actually uses bit pattern "11" to represent field element 2. Addition now consists of just six bit-operations. C-syntax pseudocode:

```
add(SlicedWord lhs, SlicedWord rhs)
   a = lhs.bit0 ^ rhs.bit1
   b = lhs.bit1 ^ rhs.bit0
   s = a ^ lhs.bit1
   t = b ^ rhs.bit1
   answer.bit1 = a & b
   answer.bit2 = s | t
```

[1]

# [References]

[1] T. J. Boothby and R. W. Bradshaw. Bitslicing and the method of four russians over larger finite fields, 2009. Preprint.

[2] B. David Saunders and Bryan Youse. Large Matrix, Small Rank, 2009. Preprint.

Thanks to NSF Grant CCR-0515197