# Efficient Storage for Small Finite Fields

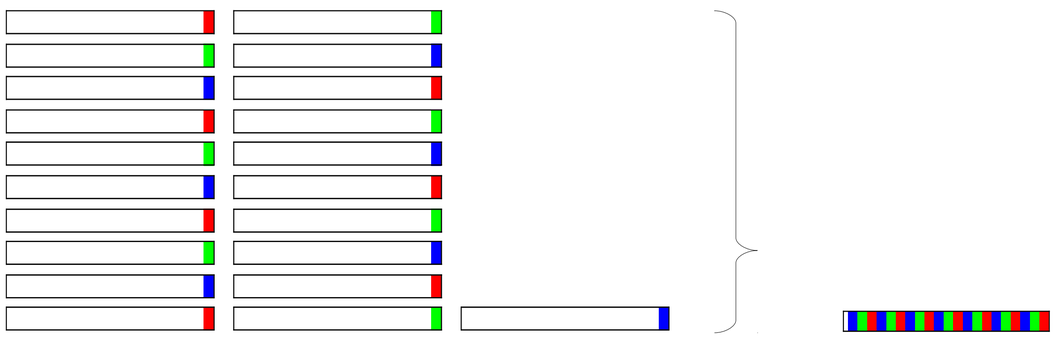
## Reducing memory usage and computation time on large matrices

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

Thanks to NSF Grant CCR-0515197

## *Bit Packing*

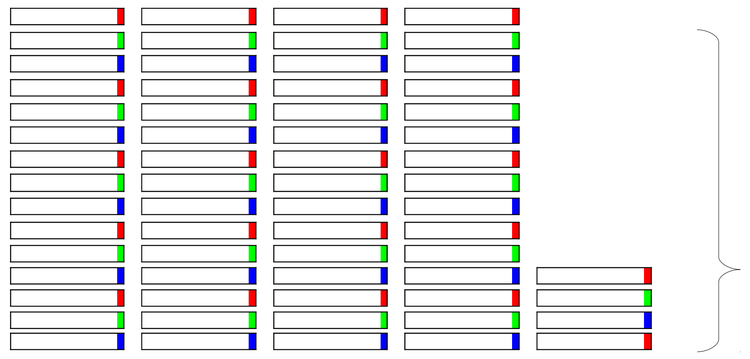
Multiple field elements are packed into few bits and aligned side-by-side to fit within a single machine word:



Example of the tightest bit-packing for the field of integers mod 3, using three bits as the packing-factor (pf). Although two bits can store field elements, one bit is needed as a carry bit following arithmetic operations. Assuming 64-bit machine words, this is a twenty-one to one compression ratio.

## *Bit Slicing*

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



Only two bits are needed to represent elements in the field of integers mod 3. No carry bit(s) necessary due to specialized bit operations on the bit-vectors to accomplish field arithmetic. On 64-bit machines, compression ratio is thirty-two to one.

## *Example*

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 |

| 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} | | | | |

| 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 |

This method can be generically applied across all small finite fields using an arbitrary number of bits to store values. Fewer bits means more frequent normalization during arithmetic. Pre-tuning can be done to determine an optimal packing-factor, in terms of computation-time, for a given field.

This method *could* be generically applied to other small fields by adding more "slices" of bit-vectors. However, bit packing is better for any field other than F3. Bit slicing wins here due to the highly specialized arithmetic enabling the use of the best possible compression ratio: only two bits used to store each element.

## *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, we have the concept of **Semi-normalization:** Resetting each value's MSB to zero while maintaining that value's integrity. Once the MSB is "open", field arithmetic can be performed freely on the packed values.

| Value (binary) | SemiNormalized |
|---|---|
| ۰۰۰ | ۰۰۰ |
| ۰۰۱ | ۰۰۱ |
| ۰۱۰ | ۰۱۰ |
| ۰۱۱ | ۰۱۱ |
| ۱۰۰ | ۰۰۱ |
| ۱۰۱ | ۰۱۰ |
| ۱۱۰ | ۰۱۱ |

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, which is an enormous benefit.

```
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]

## *Application & Data*

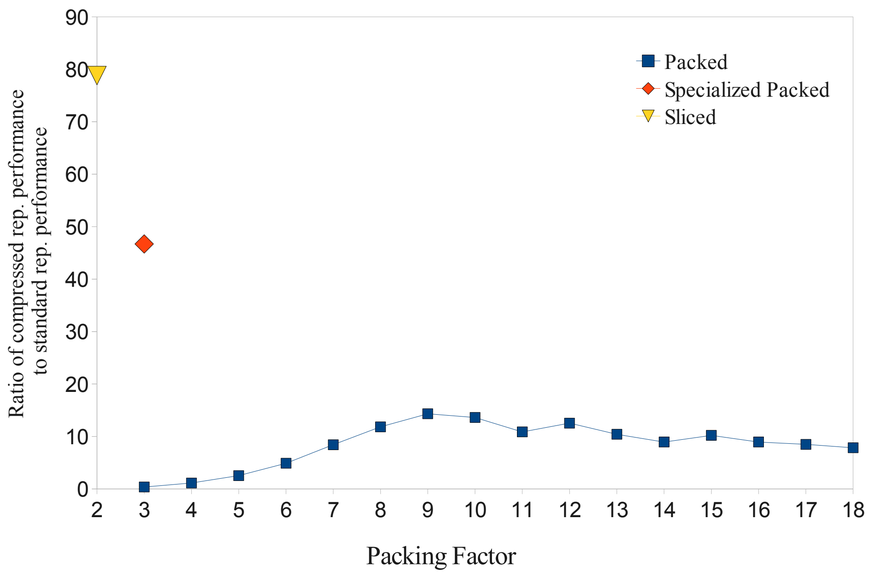
Our aim is to implement a complete arithmetic suite for both packing and slicing. The immediate application is computing the rank of large order matrices [2]. This calls for vector add, mul, and their combination (axpy).

| Memory Comparison for Matrix order 50,000 | | | |
|---|---|---|---|
| | **pf** | **Approx. Memory Usage, GB** | Notes |
| **Standard Representation** | - | 20 | |
| **Bit Slicing** | - | 0.63 | |
| **Bit Packing** | 3 | 0.95 | |
| | 4 | 1.25 | |
| | 5 | 1.66 | F5 slower than standard* |
| | 6 | 2 | F7 slower than standard* |
| | 8 | 2.5 | |
| | 9 | 2.86 | Optimal** for generic F3 |
| | 10 | 3.33 | Optimal** for F5 |
| | 12 | 4 | Optimal** for F7 |
| * Due to having to normalize too often<br>** Fastest running times on average across add, mul, axpy | | | |

| Arithmetic performance across fields, measured in MegaFFOPS (Finite Field Operations/Second) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | $F_3$ | | | $F_5$ | | | $F_7$ | | |
| | add | mul | axpy | add | mul | axpy | add | mul | axpy |
| Standard Representation | 166 | 136 | 99 | 157 | 139 | 98 | 152 | 137 | 98 |
| Sliced | 5747 | 52521 | 7847 | | | | | | |
| Spec Packed | 3321 | 20000 | 4596 | | | | | | |
| Packed Rep (PF) | | | | | | | | | |
| 3 | 41 | **25458** | 37 | | | | | | |
| 4 | 120 | 20161 | 111 | | | | | | |
| 5 | 257 | 13889 | 251 | 117 | **9615** | | | | |
| 6 | 492 | 11905 | 485 | 254 | 9615 | 121 | 173 | 5435 | 48 |
| 7 | 804 | 12077 | 836 | 442 | 7576 | 244 | 347 | 6579 | 116 |
| 8 | 1101 | 10870 | 1174 | 607 | 7813 | 419 | 542 | 5952 | 221 |
| 9 | **1238** | 8333 | **1420** | 1020 | 7143 | 696 | 710 | **6944** | 368 |
| 10 | 1142 | 8065 | 1351 | **1208** | 5208 | 859 | 1046 | 4902 | 579 |
| 11 | 862 | 6410 | 1073 | 1050 | 4902 | 836 | **1106** | 4237 | 689 |
| 12 | 1042 | 6944 | 1244 | 874 | 4717 | 753 | 1096 | 4902 | 772 |
| 13 | 842 | 5556 | 1033 | 1033 | 4167 | 836 | 996 | 3731 | 714 |
| 14 | 714 | 5682 | 883 | 919 | 4032 | 776 | 1004 | 3906 | 749 |
| 15 | 853 | 5319 | 1020 | 984 | 4032 | 794 | 1055 | 3676 | 772 |
| 16 | 708 | 5682 | 887 | 1042 | 3968 | **862** | 1046 | 3623 | **784** |

**Left** – data from application matrix detailing the memory efficiency of these methods.
**Above** – chart indicating relative performance boost using packing & slicing
**Right** – graph to visualize the third column of above chart

Normalized axpy gains: alternate representations for F3




[1] Tomas J. Boothby and Robert W. Bradshaw. Bitslicing and the method of four russians over larger finite fields. CoRR, 2009.
[2] B. D. Saunders and B. Youse. Large matrix, small rank. In Proc. of ISSAC'09, pages 317–324. ACM Press, 2009.