A Vector Implementation of Gaussian Elimination over GF(2): Exploring the Design-Space of Strassen’s Algorithm as a Case Study

Enric Morancho

Departament d’Arquitectura de Computadors
Universitat Politècnica de Catalunya, BarcelonaTech
Barcelona, Spain
enricm@ac.upc.edu

23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing

Turku, Finland, March 4th – 6th, 2015
Outline

1. Introduction
2. Background
3. Vector implementation of Gaussian Elimination
4. Case study
5. Conclusions and Future work
Outline

1. Introduction
2. Background
3. Vector implementation of Gaussian Elimination
4. Case study
5. Conclusions and Future work
Gaussian Elimination (GE) is one of the key algorithms in linear algebra.

We discuss a vector implementation of GE over GF(2).

We apply this implementation to a case study:
- Enumerating all matrix-multiply algorithms over GF(2) similar to Strassen’s algorithm
  - Strassen’s algorithm: first known sub-cubic matrix-multiply algorithm
- The search engine relies on solving more than $10^{12}$ Gaussian Eliminations over GF(2) matrices.
Outline

1 Introduction

2 Background
   - Gaussian Elimination
   - Gaussian Elimination over GF(2)
   - Vector extensions

3 Vector implementation of Gaussian Elimination

4 Case study

5 Conclusions and Future work
1 Introduction

2 Background
   - Gaussian Elimination
     - Gaussian Elimination over GF(2)
     - Vector extensions

3 Vector implementation of Gaussian Elimination

4 Case study

5 Conclusions and Future work
Gaussian Elimination (GE)

- One of the key algorithms in linear algebra
  - Applications: Solving LSE’s, inverting nonsingular matrices,
- GE transforms a matrix into a matrix in row (column) echelon form
  - Forward elimination

\[
\begin{pmatrix}
2 & 1 & 4 & 3 \\
2 & 4 & 10 & 3 \\
6 & 6 & 18 & 9 \\
4 & 2 & 8 & 3
\end{pmatrix}
\rightarrow
\begin{pmatrix}
2 & 1 & 4 & 0 \\
0 & 3 & 6 & 3 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{pmatrix}
\]

- Gauss-Jordan Elimination (GJE) transforms a matrix into a matrix in reduced row (column) echelon form
  - Forward elimination + Back substitution

\[
\begin{pmatrix}
2 & 1 & 4 & 3 \\
2 & 4 & 10 & 3 \\
6 & 6 & 18 & 9 \\
4 & 2 & 8 & 3
\end{pmatrix}
\rightarrow
\begin{pmatrix}
2 & 1 & 4 & 0 \\
0 & 3 & 6 & 3 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{pmatrix}
\rightarrow
\begin{pmatrix}
1 & 0 & 1 & 0 \\
0 & 1 & 2 & 0 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0
\end{pmatrix}
\]
**Gaussian Elimination (GE)**

- GE iteratively applies three elementary transformations:
  - Swapping rows (columns)
  - Scaling rows (columns)
  - Adding to a row (column) a scalar multiple of another row (column)
- GE is defined over an algebraic field
  - Infinite fields: \( \mathbb{Q}, \mathbb{R}, \ldots \)
    - Computer arithmetic over infinite fields introduces round-off errors
    - GE implementations use pivoting techniques to minimize them
  - Finite fields: the Galois Field of two elements (GF(2)), \( \ldots \)
    - Computer arithmetic over finite fields is always exact
Outline

1. Introduction

2. Background
   - Gaussian Elimination
   - Gaussian Elimination over GF(2)
   - Vector extensions

3. Vector implementation of Gaussian Elimination

4. Case study

5. Conclusions and Future work
Gaussian Elimination over GF(2)

- GF(2) is the Galois field of two elements (aka $F_2$, binary field)
  - $GF(2) = \{0, 1\}$
  - addition $\equiv$ bitwise XOR
    - subtraction and addition are the same operation ($+1 = -1$)
  - multiplication $\equiv$ bitwise AND

- Implementation remarks
  - Gaussian Elimination can be specialized for GF(2)
    - The only element different from 0 is 1
  - Gauss-Jordan Elimination can be easily merged into GE
  - Pivoting is unnecessary
    - Computer arithmetic over GF(2) is always exact

- Applications
  - Factoring large integer numbers, cryptography, pattern matching,...
Outline

1. Introduction

2. Background
   - Gaussian Elimination
   - Gaussian Elimination over GF(2)
   - Vector extensions

3. Vector implementation of Gaussian Elimination

4. Case study

5. Conclusions and Future work
Vector extensions

- Most current processors implement vector instructions
  - x86 family supports several vector-instruction sets and lengths
    - MMX: 64-bit vector registers
    - SSE: 128-bit vector registers
    - AVX2: 256-bit vector registers
- Scalar processors may emulate vector instructions
  - SWAR: SIMD within a register
  - Interprets general-purpose registers as a bitvectors
  - Bitwise operations (AND, shifts,...) can be seen as SIMD operations
Outline

1. Introduction
2. Background
3. Vector implementation of Gaussian Elimination
   - Finding the pivot element
   - Row swapping
   - Forward elimination and Back substitution
4. Case study
5. Conclusions and Future work
Preliminaries

We focus in row echelon forms

Vector instructions allow us to exploit the parallelism available in the matrix transformations

We represent GF(2) matrices as vector registers
  - Both row-major and column-major layouts

Main steps of GE
  - For each column:
    - Finding the pivot
    - Row swapping
    - Forward elimination and Back substitution
Outline

1. Introduction

2. Background

3. Vector implementation of Gaussian Elimination
   - Finding the pivot element
   - Row swapping
   - Forward elimination and Back substitution

4. Case study

5. Conclusions and Future work
Finding the pivot element

- For each column, GE must locate a pivot (an element \( \neq 0 \))
  - First: column must be extracted
    - Depends on matrix layout
  - Second: pivot must be located
    - Naïve solution: iterating bit by bit
    - Optimized solution: using bit-scanning machine instructions
      - \texttt{tzcnt} (AVX2)
Outline

1. Introduction
2. Background
3. Vector implementation of Gaussian Elimination
   - Finding the pivot element
   - Row swapping
   - Forward elimination and Back substitution
4. Case study
5. Conclusions and Future work
Row swapping

- Depends on matrix layout
  - Row-major layout:
    - Vector instruction sets offer permute instructions
  - Column-major layout:
    - Several vector instructions are required
Row swapping

Example of swapping rows 3 and 5 on a $6 \times 4$ GF(2) matrix stored in column-major layout.
Forward elimination and Back substitution

- Add pivot row to the rows with non-zero entries in the pivot column
- We perform all additions at the same time
- Example on a $4 \times 6 \ GF(2)$ matrix stored in row-major layout
1. Introduction

2. Background

3. Vector implementation of Gaussian Elimination

4. Case study
   - Preliminaries
   - Adaptation to the case study
   - Experimental setup
   - Results

5. Conclusions and Future work
Block-recursive matrix-multiply algorithms

\[
A \cdot B = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = C
\]

\[
P_1 = A_{11} \cdot B_{11} \\
P_2 = A_{12} \cdot B_{21} \\
P_3 = A_{11} \cdot B_{12} \\
P_4 = A_{12} \cdot B_{22} \\
P_5 = A_{21} \cdot B_{11} \\
P_6 = A_{22} \cdot B_{21} \\
P_7 = A_{21} \cdot B_{12} \\
P_8 = A_{22} \cdot B_{22}
\]

\[
C_{11} = P_1 + P_2 \\
C_{12} = P_3 + P_4 \\
C_{21} = P_5 + P_6 \\
C_{22} = P_7 + P_8
\]

a) Conventional 
\[n^3\]
Block-recursive matrix-multiply algorithms

\[
A \cdot B = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = C
\]

\[
\begin{align*}
P_1 &= A_{11} \cdot B_{11} \\
P_2 &= A_{12} \cdot B_{21} \\
P_3 &= A_{11} \cdot B_{12} \\
P_4 &= A_{12} \cdot B_{22} \\
P_5 &= A_{21} \cdot B_{11} \\
P_6 &= A_{22} \cdot B_{21} \\
P_7 &= A_{21} \cdot B_{12} \\
P_8 &= A_{22} \cdot B_{22}
\end{align*}
\]

\[
\begin{align*}
P_1 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
P_2 &= (A_{21} + A_{22}) \cdot B_{11} \\
P_3 &= A_{11} \cdot (B_{12} - B_{22}) \\
P_4 &= A_{22} \cdot (\neg B_{11} + B_{21}) \\
P_5 &= (A_{11} + A_{12}) \cdot B_{22} \\
P_6 &= (-A_{11} + A_{21}) \cdot (B_{11} + B_{12}) \\
P_7 &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22})
\end{align*}
\]

\[
\begin{align*}
C_{11} &= P_1 + P_2 \\
C_{12} &= P_3 + P_4 \\
C_{21} &= P_5 + P_6 \\
C_{22} &= P_7 + P_8
\end{align*}
\]

\[
\begin{align*}
C_{11} &= P_1 + P_4 - P_5 + P_7 \\
C_{12} &= P_3 + P_5 \\
C_{21} &= P_2 + P_4 \\
C_{22} &= P_1 - P_2 + P_3 + P_6
\end{align*}
\]

\[
\begin{align*}
a) \text{ Conventional} \\
n^3
\end{align*}
\]

\[
\begin{align*}
b) \text{ Strassen’s algorithm} \\
n^{2.807}
\end{align*}
\]
Block-recursive matrix-multiply algorithms

\[ A \cdot B = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = C \]

\[ \begin{align*}
P_1 &= A_{11} \cdot B_{11} \\
P_2 &= A_{12} \cdot B_{21} \\
P_3 &= A_{11} \cdot B_{12} \\
P_4 &= A_{12} \cdot B_{22} \\
P_5 &= A_{21} \cdot B_{11} \\
P_6 &= A_{22} \cdot B_{21} \\
P_7 &= A_{21} \cdot B_{12} \\
P_8 &= A_{22} \cdot B_{22} \\
C_{11} &= P_1 + P_2 \\
C_{12} &= P_3 + P_4 \\
C_{21} &= P_5 + P_6 \\
C_{22} &= P_7 + P_8 \\
a) \text{ Conventional} & \quad n^3 \\
b) \text{ Strassen's algorithm} & \quad n^{2.807} \\
c) \text{ Winograd's variant} & \quad n^{2.807} \end{align*} \]
Several matrix-multiply algorithms with sub-cubic complexity
- $n^{2.81}$ [Strassen, 1969]
- $n^{2.79}$ [Pan, 1979]
- $n^{2.78}$ [Bini, 1979]
- $n^{2.55}$ [Schönhage, 1981]
- $n^{2.373}$ [Coppersmith and Winograd, 1987]
- $n^{2.37286}$ [LeGall, 2014]

But Strassen’s algorithm is optimal for $2 \times 2$ matrices

There are other algorithms similar to Strassen’s?
- Using a genetic algorithm, [Oh and Moon, 2010] searched for Strassen-like algorithms over $\mathbb{R}$
  - Their partial search discovered 608 Strassen-like algorithms
Enumerating all the Strassen-like algorithms for $2 \times 2$ GF(2) matrices
Strassen-like algorithms make 7 recursive calls

Each call computes a bilinear form

\[ P_i = (\alpha_{i1}A_{11} + \alpha_{i2}A_{12} + \alpha_{i3}A_{21} + \alpha_{i4}A_{22}) \]
\[ \cdot (\beta_{i1}B_{11} + \beta_{i2}B_{12} + \beta_{i3}B_{21} + \beta_{i4}B_{22}) \]
\[ \alpha_{ij}, \beta_{ij} \in GF(2) \]

There are \((2^4 - 1)^2 = 225\) possible \(P_i\)'s and \(\binom{225}{7} \approx 5.27 \times 10^{12}\) candidate algorithms

An algorithm is correct (computes \(A \cdot B\)) if there exist simultaneously next four linear combinations:

\[
\begin{align*}
C_{11} : & \sum_{k=1}^{7} \phi_{k1}P_k = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\
C_{12} : & \sum_{k=1}^{7} \phi_{k2}P_k = A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\
C_{21} : & \sum_{k=1}^{7} \phi_{k3}P_k = A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\
C_{22} : & \sum_{k=1}^{7} \phi_{k4}P_k = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}
\end{align*}
\]
\[ \phi_{ij} \in GF(2) \]
We represent each candidate algorithm as a $16 \times 11$ GF(2) matrix

$$\delta_{ijk} = 1 \iff \alpha_{ij} = \beta_{ik} = 1$$

$$\begin{pmatrix}
\delta_{1,1,1} & \ldots & \delta_{7,1,1} & 1 & 0 & 0 & 0 \\
\delta_{1,1,2} & \ldots & \delta_{7,1,2} & 0 & 1 & 0 & 0 \\
\delta_{1,1,3} & \ldots & \delta_{7,1,3} & 0 & 0 & 0 & 0 \\
\delta_{1,1,4} & \ldots & \delta_{7,1,4} & 0 & 0 & 0 & 0 \\
\delta_{1,2,1} & \ldots & \delta_{7,2,1} & 0 & 0 & 0 & 0 \\
\delta_{1,2,2} & \ldots & \delta_{7,2,2} & 0 & 0 & 0 & 0 \\
\delta_{1,2,3} & \ldots & \delta_{7,2,3} & 1 & 0 & 0 & 0 \\
\delta_{1,2,4} & \ldots & \delta_{7,2,4} & 0 & 1 & 0 & 0 \\
\delta_{1,3,1} & \ldots & \delta_{7,3,1} & 0 & 0 & 1 & 0 \\
\delta_{1,3,2} & \ldots & \delta_{7,3,2} & 0 & 0 & 0 & 1 \\
\delta_{1,3,3} & \ldots & \delta_{7,3,3} & 0 & 0 & 0 & 0 \\
\delta_{1,3,4} & \ldots & \delta_{7,3,4} & 0 & 0 & 0 & 0 \\
\delta_{1,4,1} & \ldots & \delta_{7,4,1} & 0 & 0 & 0 & 0 \\
\delta_{1,4,2} & \ldots & \delta_{7,4,2} & 0 & 0 & 0 & 0 \\
\delta_{1,4,3} & \ldots & \delta_{7,4,3} & 0 & 0 & 1 & 0 \\
\delta_{1,4,4} & \ldots & \delta_{7,4,4} & 0 & 0 & 0 & 1
\end{pmatrix}$$
We apply Gauss-Jordan elimination to the $16 \times 11$ matrix. If the form of the resulting matrix is:

\[
\begin{bmatrix}
\mathcal{I}_7 & \phi_{7\times4} \\
\mathcal{Z}_{9\times7} & \mathcal{Z}_{9\times4}
\end{bmatrix}
\]

then $\phi_{7\times4}$ are the coefficients of the linear combinations of $P_1 \ldots P_7$ that generate the $C_{ij}$'s of $A \cdot B$. 

- $\mathcal{I}_7$ $7 \times 7$ identity matrix
- $\mathcal{Z}_{9\times7}, \mathcal{Z}_{9\times4}$ $9 \times 7, 9 \times 4$ zero matrices
- $\phi_{7\times4}$ $7 \times 4$ arbitrary matrix
Search engines

- Search engine
  - Iterates through \( \binom{225}{7} \approx 5.2 \times 10^{12} \) candidate algorithms
  - Applies Gauss-Jordan elimination to the related \( 16 \times 11 \) GF(2) matrices
  - Checks the form of the resulting matrix
- Specializations
  - If a pivot is not found in anyone of the first seven columns, the algorithm is discarded
    - Redundant \( P_i \)'s
  - Gauss-Jordan is applied just on the first seven columns
  - Avoid explicitly swapping rows
  - Filtering-out algorithms that do not use all eight subproducts
    - \( A_{11} \cdot B_{11}, A_{12} \cdot B_{21}, A_{11} \cdot B_{12}, A_{12} \cdot B_{22}, A_{21} \cdot B_{11}, A_{22} \cdot B_{21}, A_{21} \cdot B_{12}, A_{22} \cdot B_{22} \)
Outline

1. Introduction
2. Background
3. Vector implementation of Gaussian Elimination
4. Case study
   - Preliminaries
   - Adaptation to the case study
   - Experimental setup
   - Results
5. Conclusions and Future work
Experimental setup

- Evaluated search engines
  - 6 scenarios:
    - AVX2, SSE, Scalar SWAR, Scalar no-SWAR, M4RI library (GE), M4RI library (GJE)
  - 2 search engines for each scenario
    - Generic: applies GJE/GE to all candidate algorithms
    - Specialized: implement specializations
      - In M4RI scenarios, specialization includes only the filtering mechanism
  - Search engines differ on the implementation of Gauss-Jordan elimination

- Column-major layout

- Evaluation platform
  - Intel Xeon CPU E3-1220 v3 (3.1 GHz, 4-core, Haswell)
    - Implements AVX2 vector extension (256-bit)
  - Search space is dynamically distributed among the 4 cores
1. Introduction

2. Background

3. Vector implementation of Gaussian Elimination

4. Case study
   - Preliminaries
   - Adaptation to the case study
   - Experimental setup
   - Results

5. Conclusions and Future work
### Performance results

<table>
<thead>
<tr>
<th></th>
<th>Generic</th>
<th>Special.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVX2 (256-bit vectors)</td>
<td>3.74</td>
<td>1.00</td>
</tr>
<tr>
<td>SSE (128-bit vectors)</td>
<td>5.01</td>
<td>1.28</td>
</tr>
<tr>
<td>Scalar SWAR (64-bit vectors)</td>
<td>7.43</td>
<td>1.92</td>
</tr>
<tr>
<td>Scalar no-SWAR (pure scalar)</td>
<td>21.13</td>
<td>6.88</td>
</tr>
<tr>
<td>M4RI (library GE)</td>
<td>11.30</td>
<td>5.96</td>
</tr>
<tr>
<td>M4RI (library GJE)</td>
<td>17.22</td>
<td>9.18</td>
</tr>
</tbody>
</table>

**Relative execution time of the search engines**

- With respect to specialized AVX2 engine
- The lower, the better
Performance results: vector vs. scalar

<table>
<thead>
<tr>
<th></th>
<th>Generic</th>
<th>Special.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVX2 (256-bit vectors)</td>
<td>3.74</td>
<td>1.00</td>
</tr>
<tr>
<td>SSE (128-bit vectors)</td>
<td>5.01</td>
<td>1.28</td>
</tr>
<tr>
<td>Scalar SWAR (64-bit vectors)</td>
<td>7.43</td>
<td>1.92</td>
</tr>
<tr>
<td>Scalar no-SWAR (pure scalar)</td>
<td>21.13</td>
<td>6.88</td>
</tr>
<tr>
<td>M4RI (library GE)</td>
<td>11.30</td>
<td>5.96</td>
</tr>
<tr>
<td>M4RI (library GJE)</td>
<td>17.22</td>
<td>9.18</td>
</tr>
</tbody>
</table>

- Speedup of AVX2 impl. wrt. vector and pure scalar impl.
  - AVX2 vs. SSE: ≈ 1.3X
  - AVX2 vs. Scalar SWAR: ≈ 1.9X
  - AVX2 vs. Scalar no-SWAR: ≈ 5.7X-6.8X

- Scalar implementations are not competitive
- The larger the vector-register length, the better the performance
### Performance results: vector vs. algebra library

<table>
<thead>
<tr>
<th></th>
<th>Generic</th>
<th>Special.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVX2 (256-bit vectors)</td>
<td>3.74</td>
<td>1.00</td>
</tr>
<tr>
<td>SSE (128-bit vectors)</td>
<td>5.01</td>
<td>1.28</td>
</tr>
<tr>
<td>Scalar SWAR (64-bit vectors)</td>
<td>7.43</td>
<td>1.92</td>
</tr>
<tr>
<td>Scalar no-SWAR (pure scalar)</td>
<td>21.13</td>
<td>6.88</td>
</tr>
<tr>
<td>M4RI (library GE)</td>
<td>11.30</td>
<td>5.96</td>
</tr>
<tr>
<td>M4RI (library GJE)</td>
<td>17.22</td>
<td>9.18</td>
</tr>
</tbody>
</table>

- In our case study, library implementations are not competitive
  - They are optimized for larger matrices
### Performance results: impact of code specializations

<table>
<thead>
<tr>
<th></th>
<th>Generic</th>
<th>Special.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVX2 (256-bit vectors)</td>
<td>3.74</td>
<td>1.00</td>
</tr>
<tr>
<td>SSE (128-bit vectors)</td>
<td>5.01</td>
<td>1.28</td>
</tr>
<tr>
<td>Scalar SWAR (64-bit vectors)</td>
<td>7.43</td>
<td>1.92</td>
</tr>
<tr>
<td>Scalar no-SWAR (pure scalar)</td>
<td>21.13</td>
<td>6.88</td>
</tr>
<tr>
<td>M4RI (library GE)</td>
<td>11.30</td>
<td>5.96</td>
</tr>
<tr>
<td>M4RI (library GJE)</td>
<td>17.22</td>
<td>9.18</td>
</tr>
</tbody>
</table>

- **Impact of code specialization: almost 4X**
  - Larger than doubling or even quadruplicating vector length
  - Impact break down:
    - Discarding algorithms if a pivot is not found: negligible
    - Applying elimination just on seven columns: 1.4X - 1.6X
    - Avoiding row interchange: 1.2X - 1.4X
    - Filtering-out algorithms without eight subproducts: 1.9X
Case-study results

- We have found 20 Strassen-like algorithms
  - Excluding permutations and symmetric versions
  - Detailed in our paper
- Results coherent with [Oh and Moon, 2010]
  - They found additional algorithms where coefficient 0.5 is involved
Outline

1. Introduction
2. Background
3. Vector implementation of Gaussian Elimination
4. Case study
5. Conclusions and Future work
Conclusions

- We have discussed a vector implementation of Gaussian Elimination.
- Our evaluation develops a case study.
  - Requires solving more than $10^{12}$ GE’s over $16 \times 11$ GF(2) matrices.
  - Vector implementations clearly outperform scalar implementation.
  - We point out the impact of code specialization for our case study.
    - Speedup: almost 4X.
    - Impact larger than doubling or quadruplicating vector-register length.
- Performance of analyzed algebra libraries is not competitive.
  - Libraries are optimized for larger matrices.
Future work

- Developing efficient implementations of Gaussian Elimination for larger matrices
A Vector Implementation of Gaussian Elimination over GF(2): Exploring the Design-Space of Strassen’s Algorithm as a Case Study

Enric Morancho

Departament d’Arquitectura de Computadors
Universitat Politècnica de Catalunya, BarcelonaTech
Barcelona, Spain
enricm@ac.upc.edu

23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing

Turku, Finland, March 4th – 6th, 2015