Linear Computation Coding: Exponential Search and Reduced-State Algorithms

Data Compression Conference (DCC) 2023, Snowbird, UT

Hans Rosenberger, Johanna S. Fröhlich, Ali Bereyhi, Ralf R. Müller

Institute for Digital Communications

Friedrich-Alexander-Universität Erlangen-Nürnberg

February 28, 2023

Introduction: Constant Matrix Vector Multiplication (CMVM)

Objective

Compute the multiplication of an arbitrary vector $\boldsymbol{x} \in \mathbb{R}^K$ with a known, but arbitrary matrix $\boldsymbol{A} \in \mathbb{R}^{N \times K}$:

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$$

with **minimum effort** given some desired **accuracy**.

- Ubiquitous tasked performed in various signal processing application
- A bulk of the computational burden of artificial neural networks (ANNs) in the inference phase consists of CMVMs

Introduction: Constant Matrix Vector Multiplication (CMVM)

Classical approach: Quantizing the entries of $\boldsymbol{A}$ independently:

$$\boldsymbol{A} = \begin{pmatrix} -0.1120 & -2.0713 \\ -0.4436 & 1.6139 \\ 1.2395 & -0.1762 \end{pmatrix} \approx \begin{pmatrix} -\frac{1}{8} & -2 \\ -\frac{1}{2} & 2 \\ 1 & -\frac{1}{4} \end{pmatrix}$$

Binary representation:

- Every additional bit improves the SQNR by a factor of 4 ($6\,\mathrm{dB}$).
- Every additional bit requires half of an addition per matrix entry on average.

Canonical signed digit (CSD) representation:

- Every additional signed digit improves the SQNR by a factor of 28 ($14.5\,\mathrm{dB}$).
- Every additional signed digit requires one addition/subtraction per matrix entry on average.

Introduction: Constant Matrix Vector Multiplication (CMVM)

Classical approach: Quantizing the entries of $\boldsymbol{A}$ independently:

$$\boldsymbol{A} = \begin{pmatrix} -0.1120 & -2.0713 \\ -0.4436 & 1.6139 \\ 1.2395 & -0.1762 \end{pmatrix} \approx \begin{pmatrix} -\frac{1}{8} & -2 \\ -\frac{1}{2} & 2 \\ 1 & -\frac{1}{4} \end{pmatrix} + \begin{pmatrix} \frac{1}{64} & -\frac{1}{16} \\ \frac{1}{16} & -\frac{1}{2} \\ \frac{1}{4} & \frac{1}{16} \end{pmatrix}$$

Binary representation:

- Every additional bit improves the SQNR by a factor of 4 ($6\,\mathrm{dB}$).
- Every additional bit requires half of an addition per matrix entry on average.

Canonical signed digit (CSD) representation:

- Every additional signed digit improves the SQNR by a factor of 28 ($14.5\,\mathrm{dB}$).
- Every additional signed digit requires one addition/subtraction per matrix entry on average.

Can we do better?

Linear Computation Coding: Multiplicative Decomposition

Idea: Approximate the target matrix $\boldsymbol{A}$ by a product of matrices

$$\boldsymbol{A} \approx \boldsymbol{F}_{\mathrm{Q}} \dots \boldsymbol{F}_2 \boldsymbol{F}_1$$

such that the product with a vector

$$\boldsymbol{A}\boldsymbol{x} \approx \boldsymbol{F}_{\mathrm{Q}} \dots (\boldsymbol{F}_2(\boldsymbol{F}_1\boldsymbol{x}))$$

can be efficiently computed.

R. Müller, B. Gäde, A. Bereyhi, 'Linear computation coding: A framework for joint quantization and computing', *Algorithms*, 2022

Multiplicative Decomposition: An Example

$$\boldsymbol{A} = \begin{pmatrix} -0.1120 & -2.0713 \\ -0.4436 & 1.6139 \\ 1.2395 & -0.1762 \end{pmatrix} \approx \overbrace{\begin{pmatrix} 1 & -\frac{1}{32} & 0 \\ 0 & 1-\frac{1}{4} & 0 \\ \frac{1}{16} & 0 & 1 \end{pmatrix}}^{\text{Wiring matrix } \boldsymbol{W_2}} \overbrace{\begin{pmatrix} -\frac{1}{8} & -2 \\ -\frac{1}{2} & 2 \\ 1+\frac{1}{4} & 0 \end{pmatrix}}^{\text{Codebook matrix } \boldsymbol{C_1}}$$

$$\approx \begin{pmatrix} -0.1094 & -2.0625 \\ -0.375 & 1.5 \\ 1.2422 & -0.125 \end{pmatrix}$$

- Multiplications only by signed powers of two $\Rightarrow$ Only bitshifts
- We only need one addition in forming a linear combination of two vectors, irrespective of the vector size.

Multiplicative Decomposition: An Example

$$\boldsymbol{A} = \begin{pmatrix} -0.1120 & -2.0713 \\ -0.4436 & 1.6139 \\ 1.2395 & -0.1762 \end{pmatrix} \approx \begin{pmatrix} 1+\frac{1}{256} & 0 & 0 \\ 0 & 1+\frac{1}{16} & 0 \\ \frac{1}{32} & 0 & 1 \end{pmatrix} \overbrace{\begin{pmatrix} 1 & -\frac{1}{32} & 0 \\ 0 & 1-\frac{1}{4} & 0 \\ \frac{1}{16} & 0 & 1 \end{pmatrix}}^{\text{Wiring matrix } \boldsymbol{W_2}} \overbrace{\begin{pmatrix} -\frac{1}{8} & -2 \\ -\frac{1}{2} & 2 \\ 1+\frac{1}{4} & 0 \end{pmatrix}}^{\text{Codebook matrix } \boldsymbol{C_1}}$$

$$\approx \begin{pmatrix} 1+\frac{1}{256} & 0 & 0 \\ 0 & 1+\frac{1}{16} & 0 \\ \frac{1}{32} & 0 & 1 \end{pmatrix} \underbrace{\begin{pmatrix} -0.1094 & -2.0625 \\ -0.375 & 1.5 \\ 1.2422 & -0.125 \end{pmatrix}}_{\text{Updated codebook matrix } \boldsymbol{C_2}=\boldsymbol{W_2}\boldsymbol{C_1}}$$

The approximation improves,

- the larger the matrix,
- the more matrix factors are used,
- the larger the number of codewords (ideally: $\#\text{rows} = 2^{\#\text{cols}}$).

Problem Statement

Given $\boldsymbol{A}$ and $\boldsymbol{C}$ we want to obtain $\boldsymbol{W}$, such that

$$\boldsymbol{A} \approx \boldsymbol{W}\boldsymbol{C}$$

Sparse Recovery Problem: Obtaining the wiring coefficients

Row-wise optimization problem, with $\boldsymbol{w}_n$ and $\boldsymbol{a}_n$ being the n-th row of $\boldsymbol{W}$ and $\boldsymbol{A}$, respectively:

$$\boldsymbol{w}_n = \underset{\boldsymbol{\omega} \in \mathcal{C}}{\operatorname{argmin}} \|\boldsymbol{a}_n - \boldsymbol{\omega}\boldsymbol{C}\|_2$$

with $$\underbrace{\mathcal{C} = \left\{ \boldsymbol{\omega} = \sum_{s=1}^{S} i_s \mathbf{1}_{j_s,N} : \; i_s \in \{0, \pm 2^{\mathbb{Z}}\},\, j_s \in \{1, ..., N\} \; \forall s \right\}}_{\text{Set of all vectors } \boldsymbol{\omega} \text{ containing at most } S \text{ non zero factors (signed powers of two).}}$$

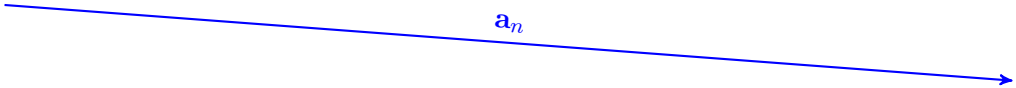
What are our options to solve this problem?

State of the Art: Discrete Matching Pursuit

Greedy, decision-directed algorithm based on the matching pursuit approach:

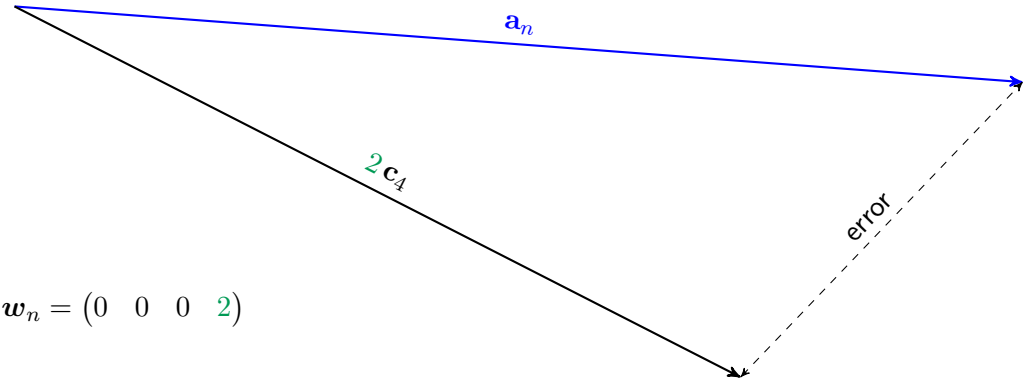
- Find the codeword with quantized scaling coefficient the minimizes the distance/error to the target vector.
- Perform iteratively S times.
- Time complexity: Cubic in N, the number of rows of $\boldsymbol{A}$.

Example: Discrete Matching Pursuit



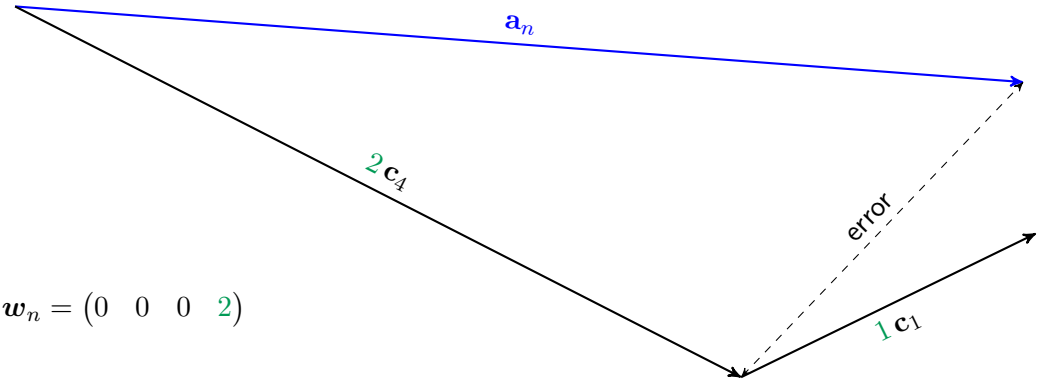
$$\boldsymbol{w}_n = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}$$

Example: Discrete Matching Pursuit



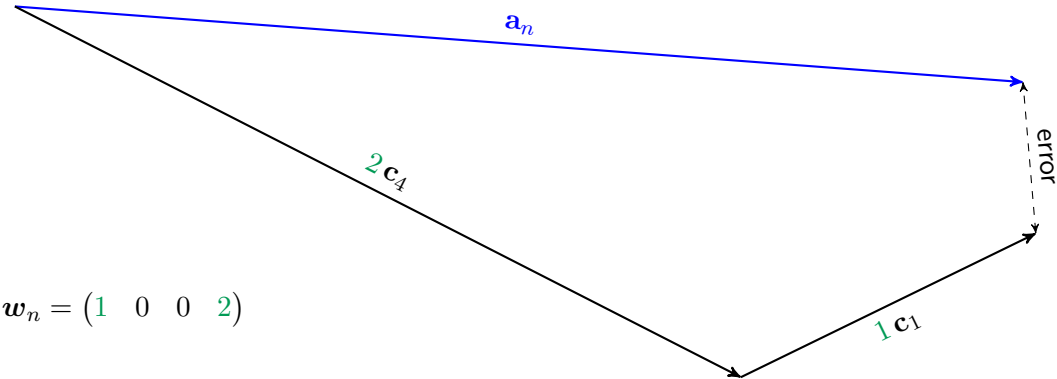
$$\boldsymbol{w}_n = \begin{pmatrix} 0 & 0 & 0 & 2 \end{pmatrix}$$

Example: Discrete Matching Pursuit



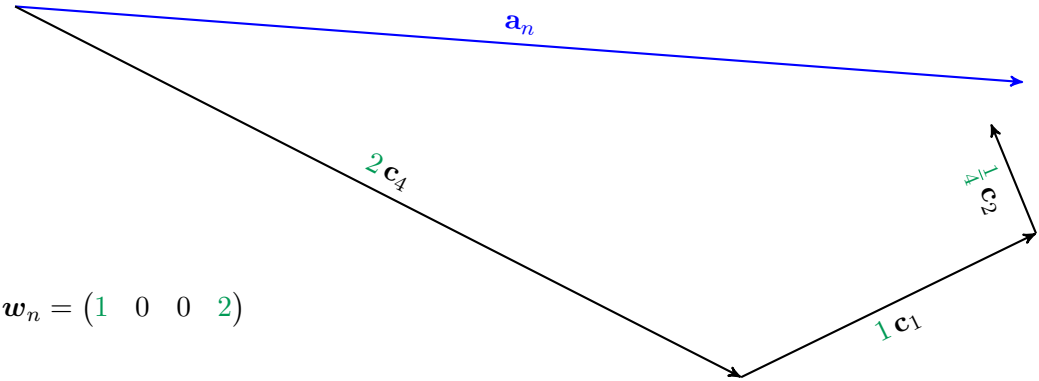
$$\boldsymbol{w}_n = \begin{pmatrix} 0 & 0 & 0 & 2 \end{pmatrix}$$

Example: Discrete Matching Pursuit



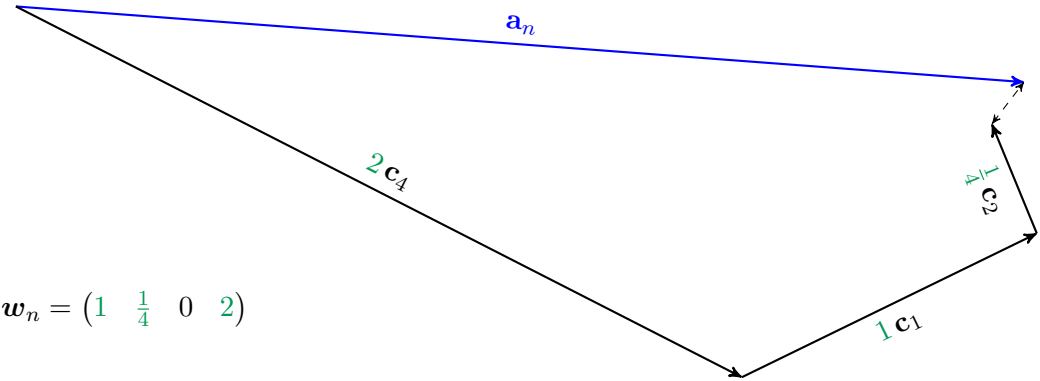
$$\boldsymbol{w}_n = \begin{pmatrix} 1 & 0 & 0 & 2 \end{pmatrix}$$

Example: Discrete Matching Pursuit



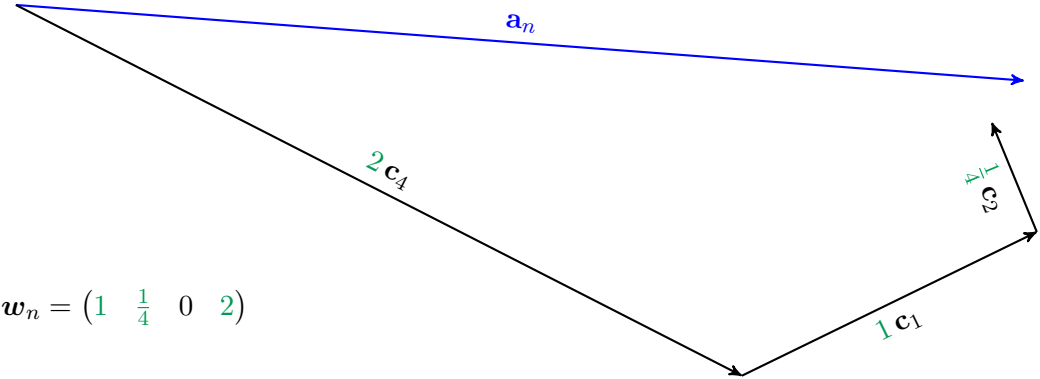
$$\boldsymbol{w}_n = \begin{pmatrix} 1 & 0 & 0 & 2 \end{pmatrix}$$

Example: Discrete Matching Pursuit



$$\boldsymbol{w}_n = \begin{pmatrix} 1 & \frac{1}{4} & 0 & 2 \end{pmatrix}$$

Example: Discrete Matching Pursuit



$\boldsymbol{w}_n = \begin{pmatrix} 1 & \frac{1}{4} & 0 & 2 \end{pmatrix}$

Approximation ($S = 3$): $\boldsymbol{a}_n \approx \boldsymbol{w}_n \boldsymbol{C} = 1\boldsymbol{c}_1 + \frac{1}{4}\boldsymbol{c}_2 + 2\boldsymbol{c}_4$

Exponential Search

To get a grasp on the optimum performance for solving our optimization problem:

- Exhaustive search to solve for $\boldsymbol{w}_n = \underset{\boldsymbol{\omega} \in \mathcal{C}}{\operatorname{argmin}} \|\boldsymbol{a}_n - \boldsymbol{\omega} \boldsymbol{C}\|_2$.
- We only use a finite subset of signed powers of two, to keep the computation tractable, i.e. $\mathcal{A}_{\text{exp}} \subset \{0, \pm 2^{\mathbb{Z}}\}$.
- Exponential time complexity both in N and $|\mathcal{A}_{\text{exp}}|$, i.e. $\mathcal{O}(N^S |\mathcal{A}_{\text{exp}}|^S)$.
- Computationally tractable only for small to medium matrix sizes and small S.

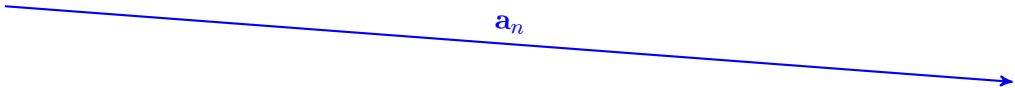
Reduced-State Approach

Middle ground between DMP and the exponential search algorithm:

- Don't update wiring coefficients in $\boldsymbol{w}_n$ in every iteration.
- Instead keep a list of M best vectors within each iteration.
- At termination select the vector with minimum error from the list.
- Retains cubic time complexity in N, only quadratic complexity in M.
- For $M = 1$ the algorithm reduces to DMP.

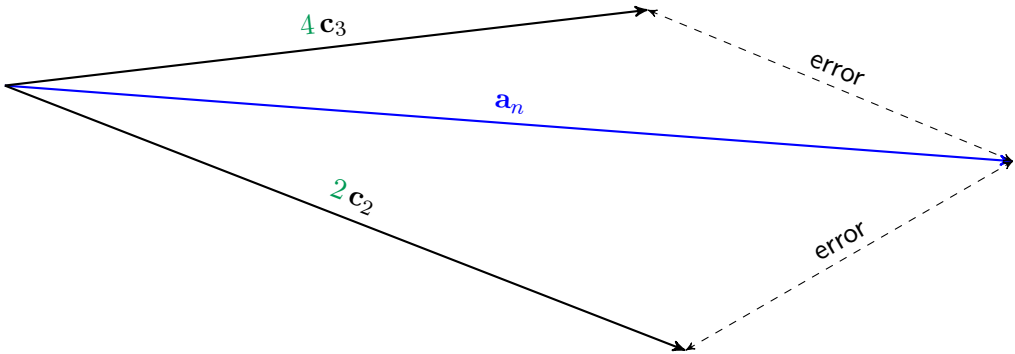
Example: Reduced-State Approach

Memory size: $M = 2$



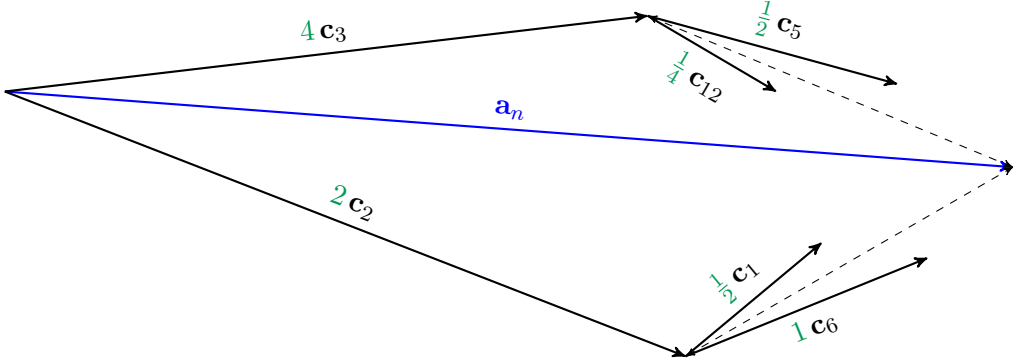
Example: Reduced-State Approach

Memory size: $M = 2$



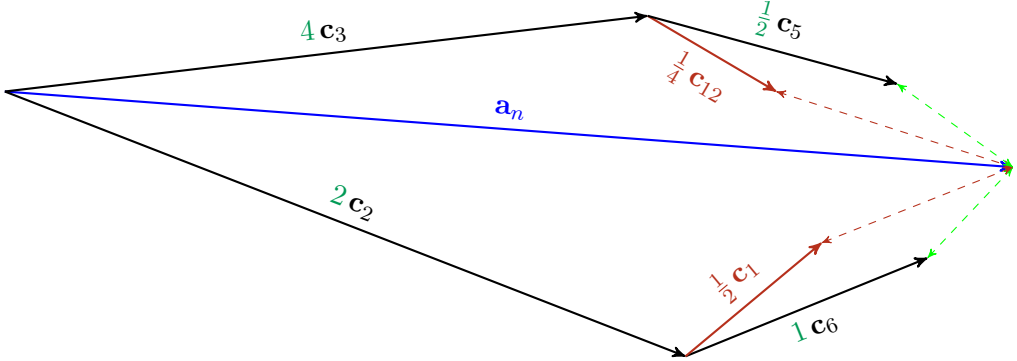
Example: Reduced-State Approach

Memory size: $M = 2$



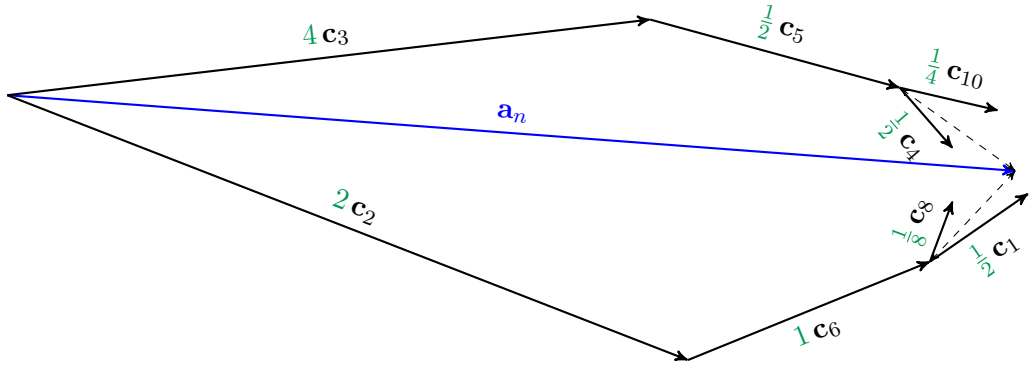
Example: Reduced-State Approach

Memory size: $M = 2$



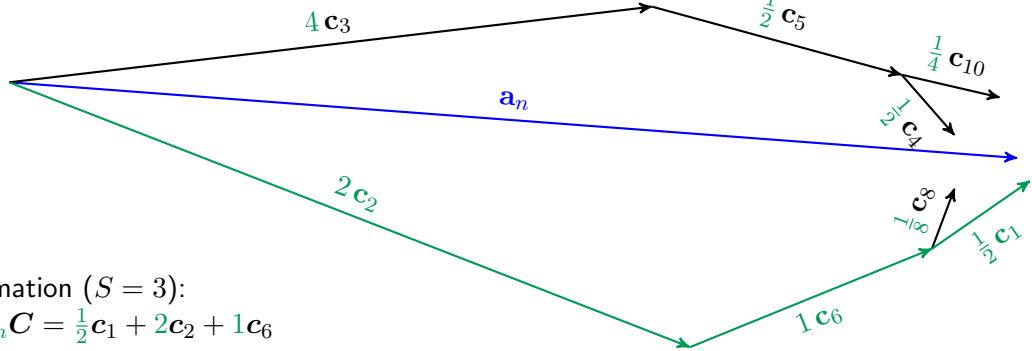
Example: Reduced-State Approach

Memory size: $M = 2$



Example: Reduced-State Approach

Memory size: $M = 2$



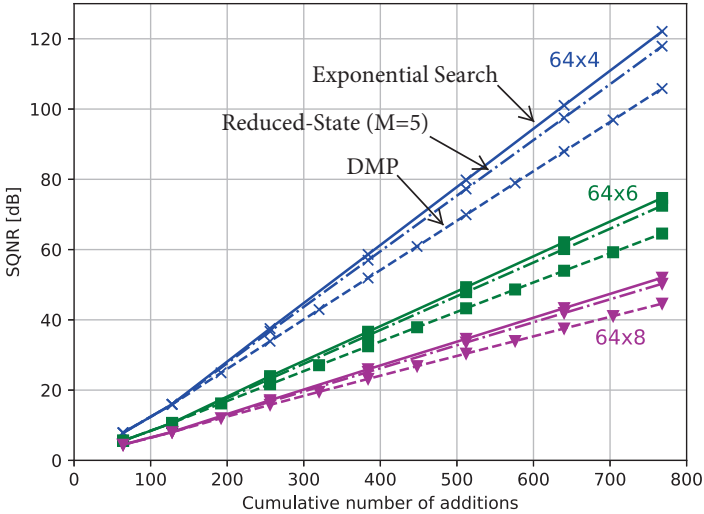
Approximation ($S = 3$):

$\boldsymbol{a}_n \approx \boldsymbol{w}_n \boldsymbol{C} = \frac{1}{2}\boldsymbol{c}_1 + 2\boldsymbol{c}_2 + 1\boldsymbol{c}_6$

Numerical Evaluation

- Target matrix entries drawn from an i.i.d. Gaussian distribution
- Averaged over 10^5 matrix entries (10^4 for exponential search algorithm)
- Performance metrics:
  - Computational Cost: Cumulative number of additions, i.e. the number of additions required for multiplying x to the multiplicative decomposition of $\boldsymbol{A}$
  - Accuracy/Distortion: Signal to Quantization Noise Ratio

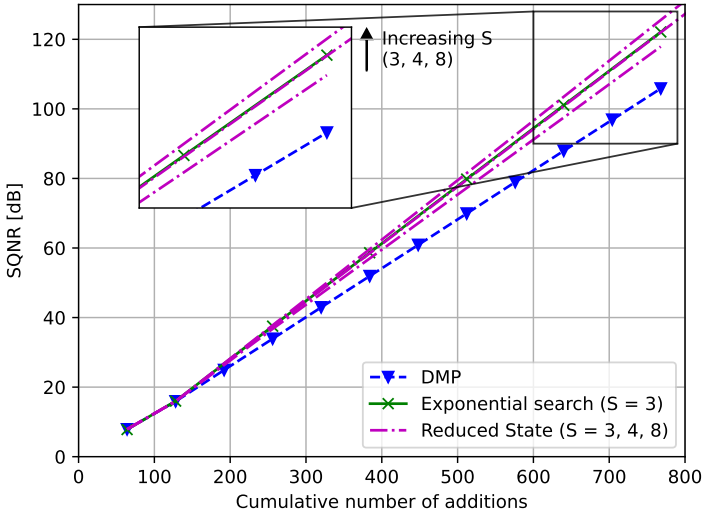
Performance Comparison



Parameters:

- DMP: $S = 2$
- Exponential search: $S = 3$, $\mathcal{A}_{\text{exp}} = \{\pm 2^{-40}, \pm 2^{+5}\}$
- Reduced-State: $S = 3$, $M = 5$

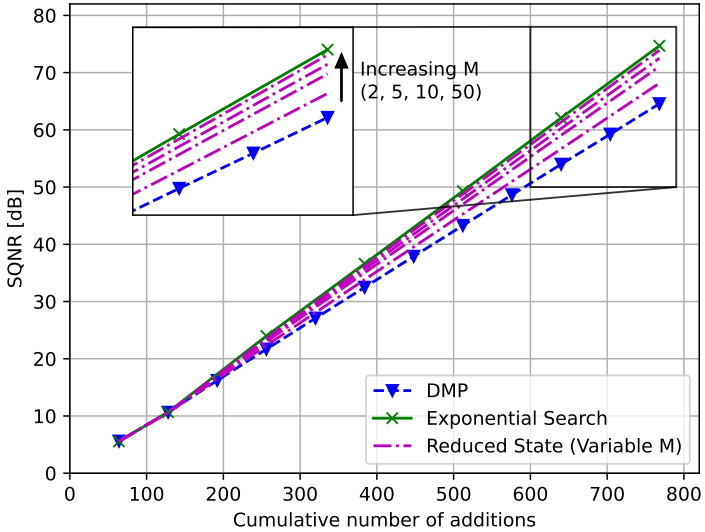
Performance Comparison: Varying S



Parameters:

- Dimension of $\boldsymbol{A}$: 64×4
- DMP: $S = 2$
- Exponential search: $S = 3$, $\mathcal{A}_{\text{exp}} = \{\pm 2^{-40}, \pm 2^{+5}\}$
- Reduced-State: Varying S, $M = 5$

Performance Comparison: Varying M



Parameters:

- Dimension of $\boldsymbol{A}$: 64×6
- DMP: $S = 2$
- Exponential search: $S = 3$, $\mathcal{A}_{\text{exp}} = \{\pm 2^{-40}, \pm 2^{+5}\}$
- Reduced-State: $S = 3$, Varying M

Performance Comparison: Different Matrix Sizes

Relative savings over benchmark DMP for a target precision of 8 bit integer arithmetic ($\mathrm{SQNR} \geq 47\,\mathrm{dB}$).

Matrix size	Exponential search $S=3$	Reduced State $S=3$ $M=5$	Reduced State $S=3$ $M=10$	Reduced State $S=4$ $M=5$	Reduced State $S=4$ $M=10$	Reduced State $S=8$ $M=5$	Reduced State $S=8$ $M=10$
16×2	$17.8\,\%$	$10.4\,\%$	$13.4\,\%$	$14.1\,\%$	$17.8\,\%$	$16.7\,\%$	$21.9\,\%$
16×4	$34.5\,\%$	$16.0\,\%$	$24.5\,\%$	$25.8\,\%$	$32.7\,\%$	$25.4\,\%$	$34.4\,\%$
32×4	$15.7\,\%$	$10.5\,\%$	$12.9\,\%$	$14.0\,\%$	$17.2\,\%$	$18.4\,\%$	$22.3\,\%$
32×6	$25.3\,\%$	$15.5\,\%$	$19.0\,\%$	$19.7\,\%$	$24.5\,\%$	$19.4\,\%$	$26.0\,\%$
64×4	$12.9\,\%$	$7.5\,\%$	$9.8\,\%$	$11.0\,\%$	$13.8\,\%$	$13.5\,\%$	$16.8\,\%$
64×6	$14.9\,\%$	$9.6\,\%$	$11.4\,\%$	$13.6\,\%$	$16.5\,\%$	$15.6\,\%$	$19.0\,\%$

Savings of $10\,\%$ and more over DMP.

Implementation: Practical Considerations

Linear computation coding is very efficient when implemented on reconfigurable hardware, such as FPGAs.

- Up to now $S = 2$: N adders required per wiring matrix performing independent computations
- Proposed algorithms harvest additional gains only for $S > 2$
- $S = 3$: Can be utilized efficiently due to the availability of efficient 3-input adders on modern FPGAs (Xilinx patent).
- $S = 4, 6, 8, \ldots$: Can be implemented efficiently using adder trees.

A. Lehnert, P. Holzinger, S. Pfenning, R. Müller, M. Reichenbach, 'Most ressource efficient matrix vector multiplication on FPGA', *IEEE Access*, 2023

Conclusion & Outlook

In summary:

- Linear computation coding reduces the computational effort required for constant matrix vector multiplication
- Proposed algorithms improve over benchmark DMP benchmark by $10\,\%$ and more.
- Proposed reduced state algorithm performs close to exponential search at a tractable computational complexity.
- Suitable for reconfigurable hardware due to efficient implementation of three input additions on modern FPGAs.

Outlook:

- Application to various types of neural networks.

Thank you for your attention!