TritNet

Optimising matrix multiply computation through scaling bit-wise logical operations

Ryan Cherian

August 2024

Contents

1 Axioms of logical operations

1.1 Scalar logical algebra

Note that throughout this derivation, we abide the notation that Greek letters represent bits, and all other variables (incl. binary variables outside $\{0, 1\}$, e.g. in $\{-1, 1\}$) are Roman. The axioms in sum are:

$\forall \alpha, \beta \in \{0, 1\}$

Axiom 1.1 (AND): $\alpha \wedge \beta = \alpha\beta$

Axiom 1.2 (OR): $\alpha \vee \beta = \alpha + \beta - \alpha\beta$

Axiom 1.3 (NOT): $\bar{\alpha} = 1 - \alpha$

Table 1: Truth tables

α	β	$\boldsymbol{\alpha\beta}$
0	0	0
0	1	0
1	0	0
1	1	1

Table 1: AND

α	β	$\boldsymbol{\alpha + \beta - \alpha\beta}$
0	0	0
0	1	1
1	0	1
1	1	1

Table 2: OR

α	$\boldsymbol{1 - \alpha}$
0	1
1	0

Table 1: NOT

1.2 Multi-linear logical algebra

A population count (`_popc()`) is equivalent to a *full* tensor contraction (one form of multi-linear reduction) in integer or floating-point arithmetic - i.e. the sum of all elements of an n-rank tensor.

Axiom 1.4 (Population Count): $\texttt{popc}(\boldsymbol{\alpha}) := \sum_i \alpha_i$

Thus, if a tensor can be represented in first-rank (vector) form, and as a concatenation of strictly Boolean operations, it's full contraction can be represented as a pop. count. For instance:

$$\sum_{i,j} \alpha_{i,j} \vee \beta_{i,j} = \texttt{popc}(\boldsymbol{\alpha} \vee \boldsymbol{\beta})$$
$$\sum_{k} \alpha_{i,k} \wedge \beta_{k,j} = \texttt{popc}(\boldsymbol{\alpha}_i \wedge \boldsymbol{\beta}_j) \quad = \boldsymbol{\alpha}_i \cdot \boldsymbol{\beta}_j$$

Where the last equality represents summations of the dot product of Boolean vectors. In other words, the *matrix multiplication* of Boolean matrices.

2 Lemmas to represent multiplication of non-bit sets as logical operations

Lemma 1 (OR): $\alpha \vee \beta = \alpha\beta! + \beta$

Proof.

$$\begin{aligned} \alpha \vee \beta &= \alpha + \beta - \alpha\beta \\ &= \alpha(1-\beta) + \beta = \alpha\beta! + \beta \end{aligned}$$

$\square$

Lemma 2 (XOR): $\alpha \veebar \beta = \alpha\beta! + \beta\alpha! = \alpha + \beta - 2\alpha\beta$

Proof.

$$\begin{aligned} \alpha \veebar \beta &= \alpha\beta! + \beta\alpha! \qquad \text{(truth table)} \\ &= \alpha(1-\beta) + \beta(1-\alpha) = \alpha + \beta - 2\alpha\beta \end{aligned}$$

$\square$

Lemma 3 (Transform): $a \in \{-1,1\} \mapsto \alpha \in \{1,0\}$ through $\alpha = \frac{1-a}{2}$

Proof. Simple linear transformation:

$$\forall\ a \in \{-1,1\}\ \exists\ \alpha = \frac{1-a}{2} : \alpha \in \{1,0\}$$

$\square$

Lemmas 4.x Multiplication of binary variables:

Lemma 4.0 Binary multiplication in $\alpha, \beta \in \{0,1\}$: $\quad \alpha\beta = \alpha \wedge \beta$

(This is Axiom 1)

Lemma 4.1 Binary multiplication of $\forall a, b \in \{-1,1\}$: $ab = 1 - 2(\alpha \veebar \beta)$

Proof.

$$\begin{aligned} \textbf{Lemma 3} \implies \qquad & ab = (1-2\alpha)(1-2\beta) = 1 - 2\alpha - 2\beta + 4\alpha\beta \\ & \quad = 1 - 2(\alpha + \beta - 2\alpha\beta) \\ \textbf{Lemma 2} \implies \qquad & \quad = 1 - 2(\alpha \veebar \beta) \end{aligned}$$

$\square$

Lemma 4.2: Binary multiplication $\alpha \in \{0,1\}$ with $b \in \{-1,1\}$: $\alpha b = \alpha - 2\alpha\beta = \alpha \veebar \beta - \beta$

Proof.

$$\begin{aligned} \textbf{Lemma 3} \implies \qquad & \alpha b = \alpha(1-2\beta) \quad \forall \alpha \in \{0,1\},\ b \in \{-1,1\} \\ & \quad = \alpha - 2\alpha\beta \quad\ = \alpha - 2(\alpha \wedge \beta) \\ \textbf{Lemma 2} \implies \qquad & \quad = \alpha \veebar \beta - \beta \end{aligned}$$

$\square$

3 Axioms of matrix multiplication

General matrix multiplication $\mathbf{Z} = \mathbf{WA}$ is defined as follows:

$$\textbf{Axiom 2.1:} \qquad \exists\ \mathbf{Z} = \mathbf{WA} : Z_{i,j} = \sum_k W_{i,k} A_{k,j} \qquad \forall\ W_{i,k}, A_{k,j}, Z_{i,j} \in \mathbb{R}$$

If using NVIDIA Tensor Cores, we can define two new binary matrix multiplication types:

$$\textbf{Axiom 2.2:} \qquad \exists\ \mathbf{Z} = \omega * \alpha : Z_{i,j} = \sum_k (\omega_{i,k} \wedge \alpha_{k,j}) \qquad \forall\ \omega_{i,k}, \alpha_{k,j} \in \{0,1\},\ Z_{i,j} \in [0..\ ..k]$$

$$\textbf{Axiom 2.3:} \qquad \exists\ \mathbf{Z} = \omega \star \alpha : Z_{i,j} = \sum_k (\omega_{i,k} \veebar \alpha_{k,j}) \qquad \forall\ \omega_{i,k}, \alpha_{k,j} \in \{0,1\},\ Z_{i,j} \in [0..\ ..k]$$

Note that NVIDIA's cores each operate on a maximum of $(8, 128, 8)$ dimensions.

4 Matrix multiplication using only bitwise operations

4.1 Multiplication of binary matrices in $\{0,1\}$

$$Z_{i,j} = \sum_k \omega_{i,k}\alpha_{k,j} \qquad \forall\ \omega_{i,k}, \alpha_{k,j} \in \{0,1\}$$

Axiom 1.1 $\implies$
$$Z_{i,j} = \sum_k \omega_{i,k} \wedge \alpha_{k,j}$$

Axioms 2.1, 2.2 $\implies$
$$\therefore \mathbf{Z} = \boldsymbol{\omega}\boldsymbol{\alpha} = \boldsymbol{\omega} * \boldsymbol{\alpha}$$

4.2 Multiplication of binary matrices in $\{-1,1\}$

$$Z_{i,j} = \sum_k W_{i,k}A_{k,j} \qquad \forall\ W_{i,k}, A_{k,j} \in \{-1,1\},\ Z_{i,j} \in k \cdot [W \times A]$$

Lemma 4.1 $\implies$
$$= \sum_k 1 - 2(\omega_{i,k} \veebar \alpha_{k,j})$$
$$= k - 2\sum_k (\omega_{i,k} \veebar \alpha_{k,j})$$

Axioms 2.1,2.3 $\implies$
$$\therefore\ \mathbf{Z} = \mathbf{WA} = \mathbf{K} - 2(\boldsymbol{\omega} \star \boldsymbol{\alpha}) \qquad \forall\ K_{i,j} \in \{k\}$$

So for a (m,k,n) matrix multiply, calculated compactly through bitwise operations on N-bit words:

```
Z[i][j] += N - 2*_popc(W[i][k] ^ A[k][j]);

//or equivalentally
Z[i][j] += N - (_popc(W[i][k] ^ A[k][j])<<1);

//or when utilising NVIDIA Tensor Core capability:
Z[i][j] = n*k - 2*(bmma_PopXor(W,A))[i][j];        //_popc(W[i][k] ^ A[k][j])
```

4.3 Multiplication of trinary matrices in $\{-1,0,1\}$

$$W_{i,j}, A_{i,j} \in \{-1,0,1\}\ \forall\ i,j\ \in \mathbb{N} : \mathbf{Z} = \mathbf{WA}$$
$$\exists\ \omega^o_{i,j}, \alpha^o_{i,j} \in \{0,1\},\ W'_{i,j}, A'_{i,j} \in \{-1,1\} : \mathbf{W} = \boldsymbol{\omega}^o \odot \mathbf{W}',\ \mathbf{A} = \boldsymbol{\alpha}^o \odot \mathbf{A}'$$

$$\therefore \mathbf{Z} = (\boldsymbol{\omega}^o \odot \mathbf{W}')(\boldsymbol{\alpha}^o \odot \mathbf{A}')$$

Axiom 1 $\implies$
$$Z_{i,j} = \sum_k W_{i,k}A_{k,j} = \sum_k \omega^o_{i,k}W'_{i,k}\alpha^o_{k,j}A'_{k,j}$$
$$= \sum_k \omega^o_{i,k}\alpha^o_{k,j}W'_{i,k}A'_{k,j}$$

Lemma 4 $\implies$
$$= \sum_k \omega^o_{i,k}\alpha^o_{k,j}(1 - 2(\omega'_{i,k} \veebar \alpha'_{k,j}))$$
$$= \sum_k \omega^o_{i,k}\alpha^o_{k,j} - 2\sum_k \omega^o_{i,k}\alpha^o_{k,j}(\omega'_{i,k} \veebar \alpha'_{k,j})$$

This gives the calculation for each element. Thus you can stop here for the operation per kernel thread:

```
Z[i][j] += _popc(W0[i][k] & A0[k][j]) - 2*_popc(W0[i][k] & A0[k][j] & (W1[i][k] ^ A1[k][j]));
```

Or if utilising Tensor cores, continue by expanding the XOR operation using Lemma 2, then simplify:

$$\begin{aligned}\textbf{Lemma 2} \implies &= \sum_k \omega^o_{i,k}\alpha^o_{k,j} - 2\sum_k \omega^o_{i,k}\alpha^o_{k,j}(\overline{\omega'_{i,k}}\alpha'_{k,j} + \omega'_{i,k}\overline{\alpha'_{k,j}}) \\ &= \sum_k \omega^o_{i,k}\alpha^o_{k,j} - 2\sum_k \omega^o_{i,k}\alpha^o_{k,j}\overline{\omega'_{i,k}}\alpha'_{k,j} - 2\sum_k \omega^o_{i,k}\alpha^o_{k,j}\omega'_{i,k}\overline{\alpha'_{k,j}} \\ &= \sum_k \omega^o_{i,k}\alpha^o_{k,j} - 2\sum_k (\omega^o_{i,k}\overline{\omega'_{i,k}})(\alpha^o_{k,j}\alpha'_{k,j}) - 2\sum_k (\omega^o_{i,k}\omega'_{i,k})(\alpha^o_{k,j}\overline{\alpha'_{k,j}})\end{aligned}$$

$$\begin{aligned}\textbf{Axioms 1,2} \implies \therefore \mathbf{Z} = \mathbf{WA} &= \boldsymbol{\omega}^o\boldsymbol{\alpha}^o - 2(\boldsymbol{\omega}^o \odot \overline{\boldsymbol{\omega}'})(\boldsymbol{\alpha}^o \odot \boldsymbol{\alpha}') - 2(\boldsymbol{\omega}^o \odot \boldsymbol{\omega}')(\boldsymbol{\alpha}^o \odot \overline{\boldsymbol{\alpha}'}) \\ &= \boldsymbol{\omega}^o\boldsymbol{\alpha}^o - 2\mathbf{PQ} - 2\mathbf{RS}\end{aligned}$$

This translates to:

```
Z = bmma_PopAND(W0,A0) - 2*(bmma_PopAND(W0(!W1), A0A1)) - 2*(bmma_PopAND(W0W1, A0(!A1)));
```

4.4 Multiplication of trinary-binary matrices

$$W_{i,j} \in \{-1,0,1\}, A_{i,j} \in \{-1,1\} \;\forall\; i,j \;\in \mathbb{N} : \mathbf{Z} = \mathbf{WA}$$
$$\exists\; \omega^+_{i,j}, \omega^-_{i,j} \in \{0,1\} : \mathbf{W} = \boldsymbol{\omega}^+ - \boldsymbol{\omega}^-$$

$$\begin{aligned} &\therefore \mathbf{Z} = (\boldsymbol{\omega}^+ - \boldsymbol{\omega}^-)\mathbf{A} \\ \textbf{Axiom 1} \implies \quad & Z_{i,j} = \sum_k W_{i,k}A_{k,j} = \sum_k (\omega^+_{i,k} - \omega^-_{i,k})A_{k,j} \\ &\quad = \sum_k \omega^+_{i,k}A_{k,j} - \sum_k \omega^-_{i,k}A_{k,j} \\ \textbf{Lemma 4} \implies \quad &\quad = \sum_k (\omega^+_{i,k} \veebar \alpha_{k,j} - \alpha_{k,j}) - \sum_k (\omega^-_{i,k} \veebar \alpha_{k,j} - \alpha_{k,j}) \\ &\quad = \sum_k \omega^+_{i,k} \veebar \alpha_{k,j} - \sum_k \omega^-_{i,k} \veebar \alpha_{k,j} \\ \textbf{Axiom 2.3} \implies \quad & \therefore \mathbf{Z} = \mathbf{WA} = (\boldsymbol{\omega}^+ \star \boldsymbol{\alpha}) - (\boldsymbol{\omega}^- \star \boldsymbol{\alpha})\end{aligned}$$

Thus you can calculate each element using the following operation per kernel thread:

```
Z[i][j] += _popc(W+[i][k] ^ A[k][j]) - _popc(W-[i][k] ^ A[k][j]);
```

Or calculate the whole matrix using tensor cores, with the following:

```
Z = bmma_PopXOR(W+,A) - bmma_PopXOR(W-,A);
```

5 Extension

Note that this method is **extendable**. Consider the following split:

$$\forall \alpha \in \{2, 1, 0, -1\}$$
$$\exists\ abcd = \alpha \quad \forall a \in \{0, 1\}, b \in \{1, 1\}, c \in \{1, 2\}, d \in \{1, -1\}$$

With this we can then simplify the multiplication of the two quaternary variables like so:

$$abcd * pqrs = ap * bq * cr * ds = ...$$

However, we then require 4 bits to hold each quaternary variable - which could easily have been held in 2 bits. So this method is not feasible beyond multiplications of sets with 3 elements.