Optimized GPU-Based Matrix Inversion

Through The Use of Thread-Data Remapping

Samuel J Monson

Seattle Univerisity

2024-05-31

Introduction

Overview

- cpu-inverse
  - Basic algorithm implementation to prove validity.

Overview

- cpu-inverse
  - Basic algorithm implementation to prove validity.
- inverse
  - GPU implementation written in CUDA.
  - Based on the works of Sharma et al [1] and DasGupta [2].

Overview

- cpu-inverse
  - Basic algorithm implementation to prove validity.
- inverse
  - GPU implementation written in CUDA.
  - Based on the works of Sharma et al [1] and DasGupta [2].
- tdr-inverse
  - Utilizes Thread-Data Remapping (TDR) to more efficiently use the GPU.
  - Actual TDR implementation is Dr. Cuneo's Harmonize library [3].

Introduction to Inverses

Definition (Inverse)

The inverse of a is some value a^{-1} such that $a \star a^{-1} = \boldsymbol{i}$ where $\boldsymbol{i}$ is the identity of $\star$.

What Is An Inverse?

Definition (Inverse)

The inverse of a is some value a^{-1} such that $a \star a^{-1} = \boldsymbol{i}$ where $\boldsymbol{i}$ is the identity of $\star$.

Definition (Identity)

The identity of an operation $\star$ and set G is some value $\boldsymbol{i} \in G$ where for all $a \in G$, $\boldsymbol{i} \star a = a \star \boldsymbol{i} = a$.

Definition (Inverse)

The inverse of a is some value a^{-1} such that $a \star a^{-1} = \boldsymbol{i}$ where $\boldsymbol{i}$ is the identity of $\star$.

Definition (Identity)

The identity of an operation $\star$ and set G is some value $\boldsymbol{i} \in G$ where for all $a \in G$, $\boldsymbol{i} \star a = a \star \boldsymbol{i} = a$.

- For example, the set and operation $(\mathbb{R}, \times)$ has the identity $\boldsymbol{i} = 1$ since $1 \times x = x \times 1 = x$ for all $x \in \mathbb{R}$.
- Thus the inverse of a is $\frac{1}{a}$ since $a \times a^{-1} = 1 \rightarrow a = \frac{1}{a}$.
  - Note that this is only true because $a \times b = b \times a$ for all $a, b \in \mathbb{R}$.

$$3x + 2y = 2$$
$$-7x - 5y = 4$$
$$\begin{bmatrix} 3 & 2 \\ -7 & -5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$
$$\begin{bmatrix} 3 & 2 \\ -7 & -5 \end{bmatrix}^{-1} \begin{bmatrix} 3 & 2 \\ -7 & -5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ -7 & -5 \end{bmatrix}^{-1} \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$
$$\boldsymbol{I}_2 \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 5 & 2 \\ -7 & -3 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$
$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 18 \\ -26 \end{bmatrix}$$

Methods of Matrix Inversion

Definition (Matrix Inverse)

Let A be an $n \times n$ matrix,

$$AA^{-1} = \boldsymbol{I} = A^{-1}A$$

$$A^{-1} = \frac{1}{\det(A)}X$$

Definition (Matrix Inverse)

Let A be an $n \times n$ matrix,

$$AA^{-1} = \boldsymbol{I} = A^{-1}A$$

$$A^{-1} = \frac{1}{\det(A)}X$$

Determinate: $\det(\mathbb{R}^{n\times n}) \rightarrow \mathbb{R}$

$$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

Definition (Matrix Inverse)

Let A be an $n \times n$ matrix,

$$AA^{-1} = \boldsymbol{I} = A^{-1}A$$

$$A^{-1} = \frac{1}{\det(A)}X$$

Determinate: $\det(\mathbb{R}^{n\times n}) \to \mathbb{R}$

$$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

Value of X

$$X = \det(A) \cdot A^{-1}$$

Definition (Matrix Inverse)

Let A be an $n \times n$ matrix,

$$AA^{-1} = \boldsymbol{I} = A^{-1}A$$

$$A^{-1} = \frac{1}{\det(A)}X$$

Determinate: $\det(\mathbb{R}^{n\times n}) \rightarrow \mathbb{R}$

$$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

Value of X

$$X = \det(A) \cdot A^{-1}$$

$$A \cdot X = \det(A) \cdot \boldsymbol{I}$$

Definition (Matrix Inverse)

Let A be an $n \times n$ matrix,

$$AA^{-1} = \boldsymbol{I} = A^{-1}A$$

$$A^{-1} = \frac{1}{\det(A)}X$$

Determinate: $\det(\mathbb{R}^{n\times n}) \to \mathbb{R}$

$$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

Value of X

$$X = \det(A) \cdot A^{-1}$$

$$A \cdot X = \det(A) \cdot \boldsymbol{I}$$

$$= \begin{bmatrix} \det(A) & 0 \\ 0 & \det(A) \end{bmatrix}$$

Definition (Matrix Inverse)

Let A be an $n \times n$ matrix,

$$AA^{-1} = \mathbf{I} = A^{-1}A$$

$$A^{-1} = \frac{1}{\det(A)}X$$

Determinate: $\det(\mathbb{R}^{n\times n}) \to \mathbb{R}$

$$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

Value of X

$$X = \det(A) \cdot A^{-1}$$

$$A \cdot X = \det(A) \cdot \mathbf{I}$$

$$= \begin{bmatrix} \det(A) & 0 \\ 0 & \det(A) \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix} = \begin{bmatrix} ad - bc & 0 \\ 0 & ad - bc \end{bmatrix}$$

Definition (Matrix Inverse)

Let A be an $n \times n$ matrix,

$$AA^{-1} = \boldsymbol{I} = A^{-1}A$$

$$A^{-1} = \frac{1}{\det(A)}X$$

Determinate: $\det(\mathbb{R}^{n\times n}) \to \mathbb{R}$

$$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

Value of X

$$X = \det(A) \cdot A^{-1}$$

$$A \cdot X = \det(A) \cdot \boldsymbol{I}$$

$$= \begin{bmatrix} \det(A) & 0 \\ 0 & \det(A) \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix} = \begin{bmatrix} ad - bc & 0 \\ 0 & ad - bc \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix} = \begin{bmatrix} ad - bc & 0 \\ 0 & ad - bc \end{bmatrix}$$

Augmented Matrix

$$\operatorname{aug}(A) = A|\boldsymbol{I} = \left[\begin{array}{cc|cc} a & b & 1 & 0 \\ c & d & 0 & 1 \end{array}\right]$$

Augmented Matrix

$$\operatorname{aug}(A) = A|\boldsymbol{I} = \left[\begin{array}{cc|cc} a & b & 1 & 0 \\ c & d & 0 & 1 \end{array}\right]$$

Operations

1. Swap any two rows: $swap(R_i, R_j)$
2. Multiply any row by a non-zero value: $c \times R_i$
3. Add to any row a multiple of another row: $R_i + c \times R_j$

Augmented Matrix

$$\operatorname{aug}(A) = A|\boldsymbol{I} = \left[\begin{array}{cc|cc} a & b & 1 & 0 \\ c & d & 0 & 1 \end{array}\right]$$

Operations

1. Swap any two rows: $swap(R_i, R_j)$
2. Multiply any row by a non-zero value: $c \times R_i$
3. Add to any row a multiple of another row: $R_i + c \times R_j$

$$\left[\begin{array}{cc|cc} 3 & 2 & 1 & 0 \\ -7 & -5 & 0 & 1 \end{array}\right] \xrightarrow{R_1 \leftrightarrow R_2} \left[\begin{array}{cc|cc} -7 & -5 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{array}\right]$$

Augmented Matrix

$$\operatorname{aug}(A) = A|\boldsymbol{I} = \left[\begin{array}{cc|cc} a & b & 1 & 0 \\ c & d & 0 & 1 \end{array}\right]$$

Operations

1. Swap any two rows: $swap(R_i, R_j)$
2. Multiply any row by a non-zero value: $c \times R_i$
3. Add to any row a multiple of another row: $R_i + c \times R_j$

$$\left[\begin{array}{cc|cc} 3 & 2 & 1 & 0 \\ -7 & -5 & 0 & 1 \end{array}\right] \xrightarrow{R_1 \leftrightarrow R_2} \left[\begin{array}{cc|cc} -7 & -5 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{array}\right] \xrightarrow{R_2 \times 2} \left[\begin{array}{cc|cc} -7 & -5 & 0 & 1 \\ 6 & 4 & 2 & 0 \end{array}\right]$$

Augmented Matrix

$$\operatorname{aug}(A) = A|\boldsymbol{I} = \left[\begin{array}{cc|cc} a & b & 1 & 0 \\ c & d & 0 & 1 \end{array}\right]$$

Operations

1. Swap any two rows: $swap(R_i, R_j)$
2. Multiply any row by a non-zero value: $c \times R_i$
3. Add to any row a multiple of another row: $R_i + c \times R_j$

$$\left[\begin{array}{cc|cc} 3 & 2 & 1 & 0 \\ -7 & -5 & 0 & 1 \end{array}\right] \xrightarrow{R_1 \leftrightarrow R_2} \left[\begin{array}{cc|cc} -7 & -5 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{array}\right] \xrightarrow{R_2 \times 2} \left[\begin{array}{cc|cc} -7 & -5 & 0 & 1 \\ 6 & 4 & 2 & 0 \end{array}\right] \xrightarrow{R_1 + 1/2 R_2} \left[\begin{array}{cc|cc} -4 & -3 & -1 & 1 \\ 6 & 4 & 2 & 0 \end{array}\right]$$

Data: M is a matrix with N rows

foreach *row* M_i **do**

  $M_i \leftarrow M_i / M_{ii}$;

  foreach *row* M_j *in* M *where* $j \neq i$ **do**

    $M_j \leftarrow M_j - M_{ji} \times M_i$

  end

end

Gauss-Jordan Algorithm

Data: M is a matrix with N rows
foreach *row* M_i **do**
  $M_i \leftarrow M_i / M_{ii}$;
  foreach *row* M_j *in* M *where* $j \neq i$ **do**
    $M_j \leftarrow M_j - M_{ji} \times M_i$
  end
end

$$\left[\begin{array}{cc|cc} 3 & 2 & 1 & 0 \\ -7 & -5 & 0 & 1 \end{array}\right]$$

Data: M is a matrix with N rows
foreach *row* M_i **do**
    $M_i \leftarrow M_i / M_{ii}$;
    foreach *row* M_j *in* M *where* $j \neq i$ **do**
        $M_j \leftarrow M_j - M_{ji} \times M_i$
    end
end

$$\left[\begin{array}{cc|cc} 3 & 2 & 1 & 0 \\ -7 & -5 & 0 & 1 \end{array}\right] \xrightarrow{R_1/3} \left[\begin{array}{cc|cc} \mathbf{1} & \mathbf{2/3} & \mathbf{1/3} & \mathbf{0} \\ -7 & -5 & 0 & 1 \end{array}\right]$$

Data: M is a matrix with N rows
foreach *row* M_i **do**
  $M_i \leftarrow M_i / M_{ii}$;
  foreach *row* M_j *in* M *where* $j \neq i$ **do**
    $M_j \leftarrow M_j - M_{ji} \times M_i$
  end
end

$$\left[\begin{array}{cc|cc} 3 & 2 & 1 & 0 \\ -7 & -5 & 0 & 1 \end{array}\right] \xrightarrow{R_1/3} \left[\begin{array}{cc|cc} \mathbf{1} & \mathbf{2/3} & \mathbf{1/3} & \mathbf{0} \\ -7 & -5 & 0 & 1 \end{array}\right] \xrightarrow{R_2-(-7)R_1} \left[\begin{array}{cc|cc} 1 & 2/3 & 1/3 & 0 \\ \mathbf{0} & \mathbf{-1/3} & \mathbf{7/3} & \mathbf{1} \end{array}\right]$$

Gauss-Jordan Algorithm

Data: M is a matrix with N rows

foreach *row* M_i **do**

  $M_i \leftarrow M_i / M_{ii}$;

  foreach *row* M_j *in* M *where* $j \neq i$ **do**

    $M_j \leftarrow M_j - M_{ji} \times M_i$

  end

end

$$\left[\begin{array}{cc|cc} 3 & 2 & 1 & 0 \\ -7 & -5 & 0 & 1 \end{array}\right] \xrightarrow{R_1/3} \left[\begin{array}{cc|cc} \mathbf{1} & \mathbf{2/3} & \mathbf{1/3} & \mathbf{0} \\ -7 & -5 & 0 & 1 \end{array}\right] \xrightarrow{R_2-(-7)R_1} \left[\begin{array}{cc|cc} 1 & 2/3 & 1/3 & 0 \\ \mathbf{0} & \mathbf{-1/3} & \mathbf{7/3} & \mathbf{1} \end{array}\right]$$

$$\xrightarrow{R_2/-1/3} \left[\begin{array}{cc|cc} 1 & 2/3 & 1/3 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{-7} & \mathbf{-3} \end{array}\right]$$

Data: M is a matrix with N rows

foreach *row* M_i **do**

  $M_i \leftarrow M_i / M_{ii}$;

  foreach *row* M_j *in* M *where* $j \neq i$ **do**

    $M_j \leftarrow M_j - M_{ji} \times M_i$

  end

end

$$\left[\begin{array}{cc|cc} 3 & 2 & 1 & 0 \\ -7 & -5 & 0 & 1 \end{array}\right] \xrightarrow{R_1/3} \left[\begin{array}{cc|cc} \mathbf{1} & \mathbf{2/3} & \mathbf{1/3} & \mathbf{0} \\ -7 & -5 & 0 & 1 \end{array}\right] \xrightarrow{R_2-(-7)R_1} \left[\begin{array}{cc|cc} 1 & 2/3 & 1/3 & 0 \\ \mathbf{0} & \mathbf{-1/3} & \mathbf{7/3} & \mathbf{1} \end{array}\right]$$

$$\xrightarrow{R_2/-1/3} \left[\begin{array}{cc|cc} 1 & 2/3 & 1/3 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{-7} & \mathbf{-3} \end{array}\right] \xrightarrow{R_1-2/3R_2} \left[\begin{array}{cc|cc} \mathbf{1} & \mathbf{0} & \mathbf{5} & \mathbf{2} \\ 0 & 1 & -7 & -3 \end{array}\right]$$

Class Activity

Matrix Example

$$M = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Solution

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

Solution

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \xrightarrow{R_1/1} \left[\begin{array}{ccc|ccc} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

Solution

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \xrightarrow{R_1/1} \left[\begin{array}{ccc|ccc} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \xrightarrow[R_3-1R_1]{R_2-0R_1} \left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{-1} & \mathbf{0} & \mathbf{1} \end{array}\right]$$

Solution

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \xrightarrow{R_1/1} \left[\begin{array}{ccc|ccc} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \xrightarrow[R_3-1R_1]{R_2-0R_1} \left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{-1} & \mathbf{0} & \mathbf{1} \end{array}\right]$$

$$\xrightarrow{R_2/2} \left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{1/2} & \mathbf{0} & \mathbf{1/2} & \mathbf{0} \\ 0 & 1 & 0 & -1 & 0 & 1 \end{array}\right]$$

Solution

$$
\left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right]
\xrightarrow{R_1/1}
\left[\begin{array}{ccc|ccc} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right]
\xrightarrow[R_3-1R_1]{R_2-0R_1}
\left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{-1} & \mathbf{0} & \mathbf{1} \end{array}\right]
$$

$$
\xrightarrow{R_2/2}
\left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{1/2} & \mathbf{0} & \mathbf{1/2} & \mathbf{0} \\ 0 & 1 & 0 & -1 & 0 & 1 \end{array}\right]
\xrightarrow[R_3-1R_2]{R_1-0R_2}
\left[\begin{array}{ccc|ccc} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 0 & 1 & 1/2 & 0 & 1/2 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{-1/2} & \mathbf{-1} & \mathbf{-1/2} & \mathbf{1} \end{array}\right]
$$

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \xrightarrow{R_1/1} \left[\begin{array}{ccc|ccc} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \xrightarrow[R_3-1R_1]{R_2-0R_1} \left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{-1} & \mathbf{0} & \mathbf{1} \end{array}\right]$$

$$\xrightarrow{R_2/2} \left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{1/2} & \mathbf{0} & \mathbf{1/2} & \mathbf{0} \\ 0 & 1 & 0 & -1 & 0 & 1 \end{array}\right] \xrightarrow[R_3-1R_2]{R_1-0R_2} \left[\begin{array}{ccc|ccc} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 0 & 1 & 1/2 & 0 & 1/2 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{-1/2} & \mathbf{-1} & \mathbf{-1/2} & \mathbf{1} \end{array}\right]$$

$$\xrightarrow{R_3/-1/2} \left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1/2 & 0 & 1/2 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{1} & \mathbf{-2} \end{array}\right]$$

Solution

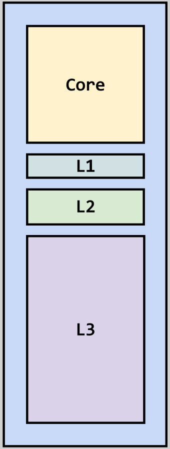
$$\left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \xrightarrow{R_1/1} \left[\begin{array}{ccc|ccc} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \xrightarrow[R_3-1R_1]{R_2-0R_1} \left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{-1} & \mathbf{0} & \mathbf{1} \end{array}\right]$$

$$\xrightarrow{R_2/2} \left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{1/2} & \mathbf{0} & \mathbf{1/2} & \mathbf{0} \\ 0 & 1 & 0 & -1 & 0 & 1 \end{array}\right] \xrightarrow[R_3-1R_2]{R_1-0R_2} \left[\begin{array}{ccc|ccc} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 0 & 1 & 1/2 & 0 & 1/2 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{-1/2} & \mathbf{-1} & \mathbf{-1/2} & \mathbf{1} \end{array}\right]$$

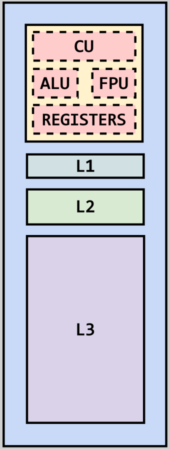
$$\xrightarrow{R_3/-1/2} \left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1/2 & 0 & 1/2 & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{1} & \mathbf{-2} \end{array}\right] \xrightarrow[R_2-1/2R_3]{R_1-1R_3} \left[\begin{array}{ccc|ccc} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{-1} & \mathbf{-1} & \mathbf{2} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{-1} & \mathbf{0} & \mathbf{1} \\ 0 & 0 & 1 & 2 & 1 & -2 \end{array}\right]$$

GPU Programming

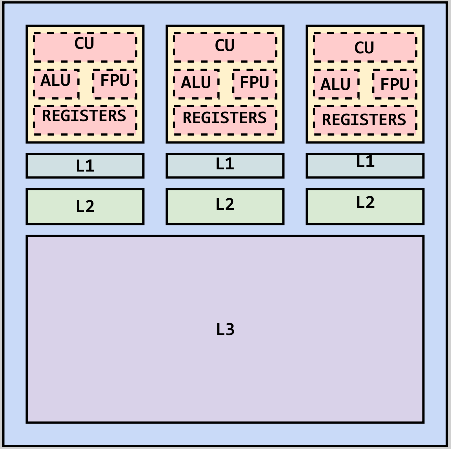
CPU Architecture



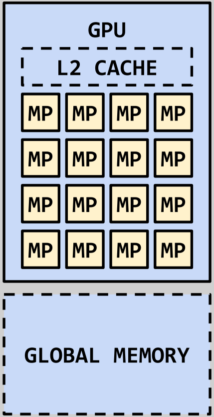
- Core
  - Processing unit
- L1 L2 L3
  - Caches for storing work



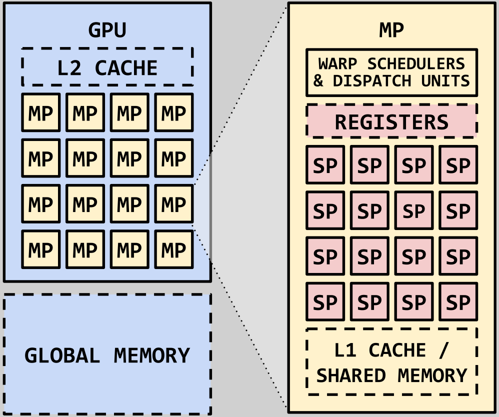
- Registers
  - Store localized data including the fetched instruction
- Control Unit (CU)
  - Decodes instruction and sends to appropriate logic unit
- Arithmetic Logic Unit (ALU) / Floating Point Unit (FPU)
  - Executes given instruction



- Registers
  - Store localized data including the fetched instruction
- Control Unit (CU)
  - Decodes instruction and sends to appropriate logic unit
- Arithmetic Logic Unit (ALU) / Floating Point Unit (FPU)
  - Executes given instruction
- Each core has its own registers and control logic

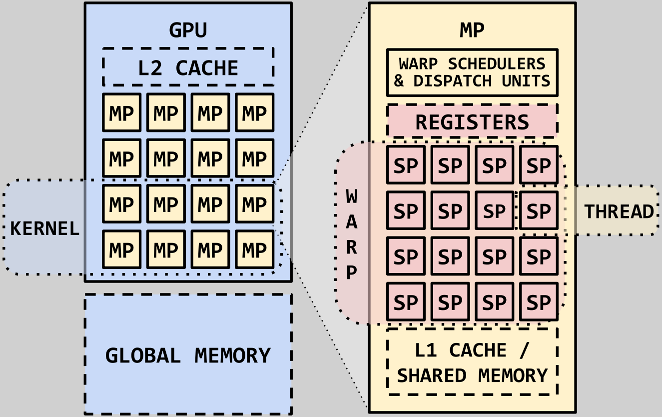


- Multiprocessor (MP)
  - A ”core” that handles simultaneous execution of a vector of tasks



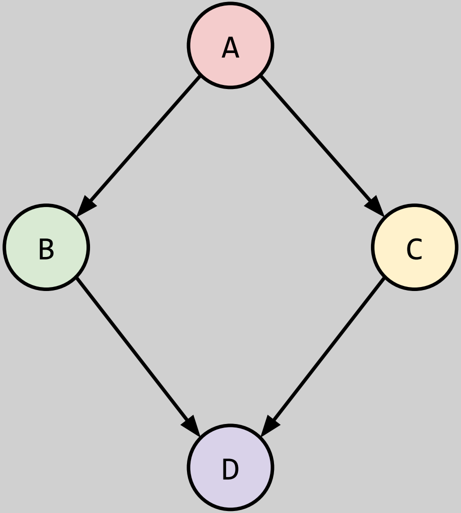
- Multiprocessor (MP)
  - A "core" that handles simultaneous execution of a vector of tasks
- Scalar processor (SP)
  - Executes a single scalar component

SIMT Architecture

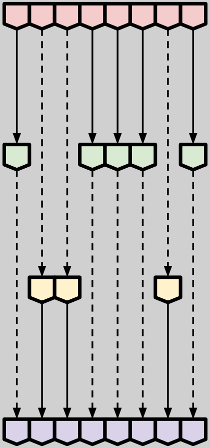


- Warp
  - Group of SPs in a MP that execute in lockstep
  - Warps share registers; including the program counter
- Kernel
  - Group of warps that operate on the same method

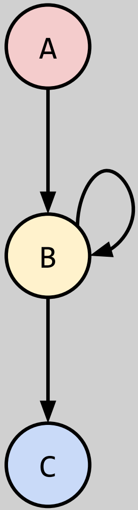
Branch Divergence



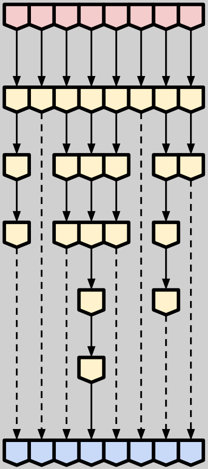

```
bool cond = A(x)

if( cond ){

    B(x);

} else {

    C(x);

}

D(x);
```



Loop Divergence




```
A(x)

while(B1(x)){

        B2(x)

}

C(x);
```



Basic CUDA Program

```
// A special method invoked by the
// CPU to launch a GPU kernel
__global__ print(string message) {
    int idx = threadIdx.x;
    int jdx = blockIdx.x;
    printf("%s from (%d, %d)\n",
           message, jdx, idx);
}

// Standard C main
int main() {
    // Call kernel launcher
    print<<<2, 4>>>("Hello World");
    // Wait for GPU to finish
    cudaDeviceSynchronize();
}
```

Output

Hello World from (0, 0)
Hello World from (0, 3)
Hello World from (1, 2)
Hello World from (1, 0)
Hello World from (0, 2)
Hello World from (1, 1)
Hello World from (1, 3)
Hello World from (0, 1)

CUDA Matrix Inversion: CPU Loop

```
for (size_t j = 0; j < rows; j++) {
  fixRow<<<1, cols>>>(data_gpu, cols, j);
  auto_throw(cudaDeviceSynchronize());

  fixColumn<<<rows, cols>>>(data_gpu, cols, j);
  auto_throw(cudaDeviceSynchronize());
}
```

```
__global__ void fixRow(
        float *matrix, int size, int rowId) {
    // the ith row of the matrix
    __shared__ float Ri[MAX_BLOCK_SIZE];
    // The diagonal element for ith row
    __shared__ float Aii;
    int colId = threadIdx.x;
    Ri[colId] = matrix[size * rowId + colId];
    Aii = matrix[size * rowId + rowId];

    __syncthreads();
    // Divide the whole row by the diagonal
    Ri[colId] = Ri[colId] / Aii;
    matrix[size * rowId + colId] = Ri[colId];
}
```

Example

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

CUDA Matrix Inversion: FixColumn

```
__global__ void fixColumn(
      float *matrix, int size, int colId) {
    int i = threadIdx.x, j = blockIdx.x;
    // The colId column
    __shared__ float col[MAX_BLOCK_SIZE];
    // The jth element of the colId row
    __shared__ float AColIdj;
    // The jth column
    __shared__ float colj[MAX_BLOCK_SIZE];
    col[i] = matrix[i * size + colId];
    __syncthreads();
    colj[i] = matrix[i * size + j];
    AColIdj = matrix[colId * size + j];
    if (i != colId) {
      colj[i] = colj[i] - AColIdj * col[i];
    }
    matrix[i * size + j] = colj[i];
}
```

Example

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

In-Place Optimization

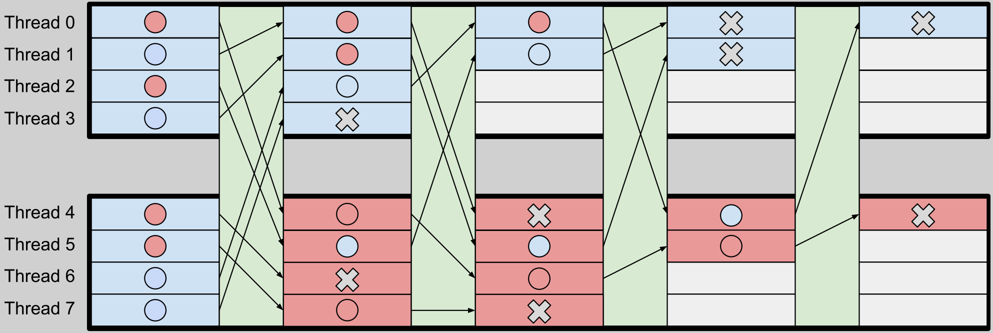
$$\left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & -1/2 & -1 & -1/2 & 1 \end{array}\right]$$

In-Place Optimization

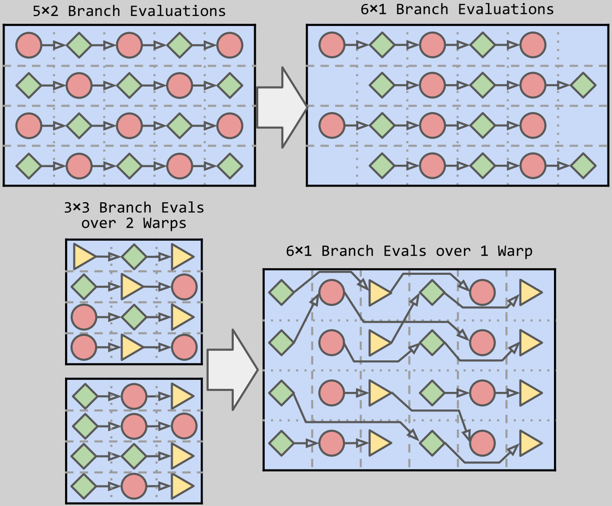
$$\left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & -1/2 & -1 & -1/2 & 1 \end{array}\right]$$

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & -1/2 & -1 & -1/2 & 1 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1/2 & 0 & 1/2 \\ -1 & -1/2 & 1 & -1/2 \end{array}\right]$$

Thread-Data Remapping



Thread-Data Remapping



TDR Matrix Inversion: FixRow

```
struct FixRow {
  using Type = void(*)(index_t rowId, size_t colId);

  template<typename PROGRAM>
  __device__ static void eval(PROGRAM prog, index_t rowId, size_t colId) {
    index_t size = prog.device.size.row;
    matrix_t Ri  = prog.device.matrix[size*rowId + colId];
    matrix_t Aii = prog.device.Aij[rowId];

    Ri /= Aii;
    prog.device.matrix[size*rowId + colId] = Ri;

    if( Ri != 0.0 ) {
      prog.template async<SplitCol>(rowId, 0, prog.device.size.col-1, colId);
    }
  }
};
```

TDR Matrix Inversion: FixColumn

```
struct FixCol {
  using Type = void(*)(index_t colId, index_t i_start,
                       index_t i_end, index_t j);
  template<typename PROGRAM>
  __device__ static void eval(
      PROGRAM prog, index_t colId,
      index_t i_start, index_t i_end, index_t j) {
    index_t size = prog.device.size.row;
    for (index_t i = i_start; i <= i_end; i++) {
      matrix_t col = prog.device.Aij[i];

      if (col != 0) {
        matrix_t colj    = prog.device.matrix[    i*size + j];
        matrix_t AColIdj = prog.device.matrix[colId*size + j];
        if (i != colId) {
          colj -= AColIdj * col;
          prog.device.matrix[i*size + j] = colj;
}}}};
```

Results

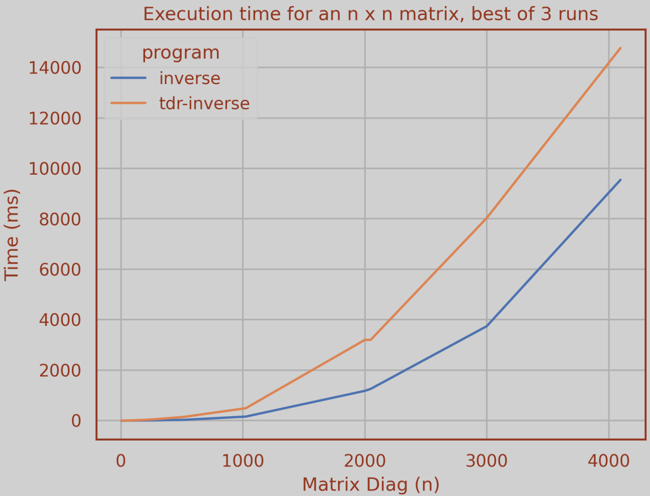
- cpu-inverse

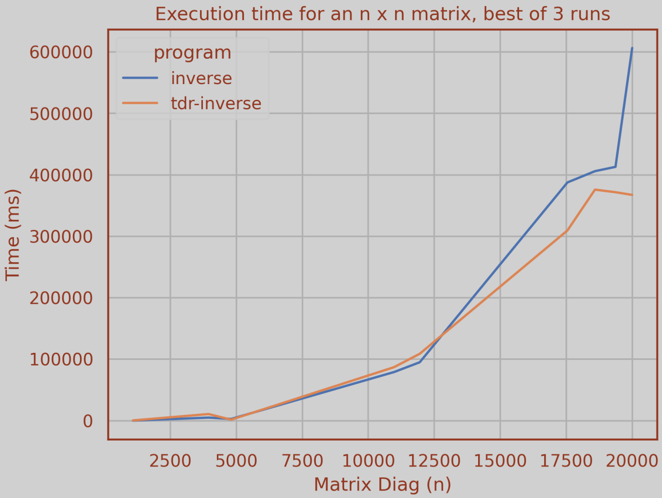
- cpu-inverse
- inverse

Comparison

- cpu-inverse
- inverse
- tdr-inverse

Random Matrices




Execution time for an n x n matrix, best of 3 runs
program
inverse
tdr-inverse
Time (ms)
600000
500000
400000
300000
200000
100000
0
2500
5000
7500
10000
12500
15000
17500
20000
Matrix Diag (n)

- The use of TDR can offer fairly significant gains when applied to sparse matrix inversion.

- The use of TDR can offer fairly significant gains when applied to sparse matrix inversion.
- Possible next step is to implement Strassen Inversion [4].

References I

[1] G. Sharma, A. Agarwala, and B. Bhattacharya, **"A fast parallel Gauss Jordan algorithm for matrix inversion using CUDA,"** *Computers & Structures*, vol. 128, pp. 31–37, 2013, ISSN: 0045-7949. DOI: 10.1016/j.compstruc.2013.06.015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045794913002095 (cit. on pp. 3–5).

[2] D. DasGupta, **"In-place matrix inversion by modified Gauss-Jordan algorithm,"** *Applied Mathematics*, vol. 4, no. 10, pp. 1392–1396, Sep. 2013. DOI: 10.4236/am.2013.410188 (cit. on pp. 3–5).

[3] B. Cuneo and M. Bailey, **"Divergence reduction in Monte Carlo neutron transport with on-GPU asynchronous scheduling,"** *ACM Trans. Model. Comput. Simul.*, vol. 34, no. 1, Jan. 2024, ISSN: 1049-3301. DOI: `10.1145/3626957`. [Online]. Available: `https://dl.acm.org/doi/10.1145/3626957` (cit. on pp. 3–5).

[4] V. Strassen, **"Gaussian elimination is not optimal,"** *Numerische Mathematik*, vol. 13, pp. 354–356, Aug. 1969. DOI: `10.1007/BF02165411` (cit. on pp. 65, 66).

[5] J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson, **"Gpytorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration,"** in *Advances in Neural Information Processing Systems*, 2018. DOI: `10.48550/arXiv.1809.11165`. [Online]. Available: `https://arxiv.org/abs/1809.11165`.

References III

[6] D. Bailey and H. Ferguson, "**A strassen-newton algorithm for high-speed parallelizable matrix inversion**," in *Supercomputing '88:Proceedings of the 1988 ACM/IEEE Conference on Supercomputing, Vol. I*, 1988, pp. 419–424. DOI: 10.1109/SUPERC.1988.44680.

[7] J. Xuebin, C. Yewang, F. Wentao, Z. Yong, and D. Jixiang, "**Fast algorithm for parallel solving inversion of large scale small matrices based on GPU**," *The Journal of Supercomputing*, vol. 79, no. 16, pp. 18 313–18 339, May 2023, ISSN: 1573-0484. DOI: 10.1007/s11227-023-05336-7. [Online]. Available: https://link.springer.com/article/10.1007/s11227-023-05336-7.

References IV

[8] S. C. Althoen and R. McLaughlin, "**Gauss-Jordan reduction: A brief history**," *American Mathematical Monthly*, vol. 94, no. 2, pp. 130–142, Feb. 1987, ISSN: 0002-9890. DOI: 10.2307/2322413. [Online]. Available: https://www.jstor.org/stable/2322413.

[9] H. Cohn, R. Kleinberg, B. Szegedy, and C. Umans, "**Group-theoretic algorithms for matrix multiplication**," in *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, 2005, pp. 379–388. DOI: 10.1109/SFCS.2005.39.

[10] D. Coppersmith and S. Winograd, "**On the asymptotic complexity of matrix multiplication**," in *22nd Annual Symposium on Foundations of Computer Science (sfcs 1981)*, 1981, pp. 82–90. DOI: 10.1109/SFCS.1981.27.

[11] I. Dimov, T. Dimov, and T. Gurov, **“A new iterative Monte Carlo approach for inverse matrix problem,”** *Journal of Computational and Applied Mathematics*, vol. 92, no. 1, pp. 15–35, 1998, ISSN: 0377-0427. DOI: 10.1016/S0377-0427(98)00043-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377042798000430.

[12] C. Misra, S. Haldar, S. Bhattacharya, and S. K. Ghosh, **“SPIN: A fast and scalable matrix inversion method in Apache Spark,”** in *Proceedings of the 19th International Conference on Distributed Computing and Networking*, ser. ICDCN ’18, Varanasi, India: Association for Computing Machinery, 2018, ISBN: 9781450363723. DOI: 10.1145/3154273.3154300.

[13] E. Agullo, H. Bouwmeester, J. Dongarra, J. Kurzak, J. Langou, and L. Rosenberg, **"Towards an efficient tile matrix inversion of symmetric positive definite matrices on multicore architectures,"** in *High Performance Computing for Computational Science – VECPAR 2010*, J. M. L. M. Palma, M. Daydé, O. Marques, and J. C. Lopes, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 129–138, isbn: 978-3-642-19328-6. doi: 10.1007/978-3-642-19328-6_14.

[14] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, **"Solution of linear algebraic equations,"** in *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. 32 Avenue of the Americas, New York, NY 10013-2473, USA: Cambridge University Press, 2007, ch. 2, pp. 37–109, isbn: 978-0-521-88068-8.

[15] H. Anton and C. Rosses, "**Applications of linear algebra,**" in *Elementary Linear Algebra*, 11th ed. 111 River Street, Hoboken, NJ 07030-5774, USA: John Wiley & Sons, Inc., 2014, ch. 10, pp. 527–713, isbn: 978-1-118-43441-3.

[16] H. Nguyen, *Gpu gems 3*, First. Addison-Wesley Professional, 2007, isbn: 9780321545428.

QUESTIONS?

https://github.com/scrufulufugus/tdr-inverse-materials