Think Fast
A Tensor Streaming Processor (TSP) for Accelerating Deep Learning Workloads

Groq, Inc.

Dennis Abts, Jonathan Ross, Jonathan Sparling, Mark Wong-VanHaren, Max Baker, Tom Hawkins, Andrew Bell, John Thompson, et al. ...

ISCA 2020

Seminar in Computer Architecture [Spring 2023]
Presenter: Sven Pfiffner
Date: 06 April 2023
Executive Summary
Executive Summary

- Observation
- Key Idea and Insights
- Implementation
- Results and Takeaways
Executive Summary

- **Observation**
  - ML workloads exhibit **abundant data parallelism**, which can be readily mapped to **tensors in hardware**
  - A simple processor with producer-consumer stream programming model enables precise reasoning and control of hardware components which means **Good performance and power efficiency**

- **Key Idea and Insight**
  - Use **tensor streaming** to design a specialized hardware architecture that achieves significant speedup for **deep learning computations**.

- **Implementation**
  - Tensor computations are performed using a new deep learning processor architecture that features a dedicated **tensor streaming unit** and carefully optimized **memory hierarchy**.

- **Results and Takeaways**
  - Novel hardware-software approach to achieve **fast performance on ML workloads** e.g significant **improvement** for ResNet50 classification.
Outline
Outline

- Background
- Proposed Architecture
- Results
- Takeaway & Conclusion
Outline

- **Background**
  - Deep Learning and its Hardware Requirements
  - Limitations of existing hardware
  - Tensor Computations
- **Proposed Architecture**
- **Results**
- **Takeaway & Conclusion**
Background
Deep Learning and its Hardware Requirements

- **Training** and **deploying** neural networks
  - Example: Image recognition, natural language processing
  - Involves **processing large amounts of data**

- Traditional **GPUs** and **CPUs**
  - Are **not always efficient** for such large amounts of data
  - Development of specialized hardware:
    - ASICs: Application specific integrated circuit
    - FPGAs: Multipurpose microchip that is specialized but reprogrammable
  - Hardware must be optimized for **parallel processing** and **memory bandwidth**
Limitations of Existing Hardware

- **The CPU**
  - Not designed for *(massive)* parallel processing
  - Must compute many other things
  - *Limited handling* of large amounts of data in a timely manner

- **The GPU**
  - *Is designed* for parallelism, thus widely used for ML
    - But not for large scale tensor operations (e.g. convolutions, matrix multiplication,...)
      - Compute throughput -> Leads to batching
  - *Memory bandwidth* between storage and processing unit can impose bottleneck
Limitations of Existing Hardware

- Behavior not fully **deterministic**
  - Instruction Reordering
Limitations of Existing Hardware

- Behavior not fully deterministic
  - Instruction Reordering

(1) $r_1 \leftarrow r_5 \div r_4$
(2) $r_3 \leftarrow r_1 + r_8$
(3) $r_8 \leftarrow r_5 + 1$
(4) $r_3 \leftarrow r_7 - 2$
(5) $r_6 \leftarrow r_6 + r_7$
Limitations of Existing Hardware

- Behavior not fully **deterministic**
  - Instruction Reordering

(1) \( r_1 \leftarrow r_5 \div r_4 \)
(2) \( r_3 \leftarrow r_1 + r_8 \)
(3) \( r_8 \leftarrow r_5 + 1 \)
(4) \( r_3 \leftarrow r_7 - 2 \)
(5) \( r_6 \leftarrow r_6 + r_7 \)

Example Source: Computer Architecture 2010 – Out-Of-Order Execution; Lihu Rappoport & Adi Yoaz
Limitations of Existing Hardware

- Behavior not fully deterministic
  - Instruction Reordering
    
    
    (1) \( r_1 \leftarrow r_5 \div r_4 \)
    (2) \( r_3 \leftarrow r_1 + r_8 \)
    (3) \( r_8 \leftarrow r_5 + 1 \)
    (4) \( r_3 \leftarrow r_7 - 2 \)
    (5) \( r_6 \leftarrow r_6 + r_7 \)
Limitations of Existing Hardware

- Behavior not fully **deterministic**
  - Instruction Reordering

(1) \( r_1 \leftarrow r_5 \div r_4 \)
(2) \( r_3 \leftarrow r_1 + r_8 \)
(3) \( r_8 \leftarrow r_5 + 1 \)
(4) \( r_3 \leftarrow r_7 - 2 \)
(5) \( r_6 \leftarrow r_6 + r_7 \)
Limitations of Existing Hardware

- Behavior not fully **deterministic**
  - Instruction Reordering

```
(1) r_1 \leftarrow r_5 \div r_4 \\
(2) r_3 \leftarrow r_1 + r_8 \\
(3) r_8 \leftarrow r_5 + 1 \\
(4) r_3 \leftarrow r_7 - 2 \\
(5) r_6 \leftarrow r_6 + r_7
```
Limitations of Existing Hardware

- Behavior not fully \textit{deterministic}
  - Instruction Reordering

\begin{center}
\begin{tikzpicture}[node distance=2cm, thick, main node/.style={fill=black!10, draw, circle, minimum size=1cm}]
  \node (1) [main node] {1};
  \node (2) [main node] {2} [below of=1];
  \node (3) [main node] {3} [below of=2, xshift=-1cm];
  \node (4) [main node] {4} [below of=2, xshift=1cm];
  \node (5) [main node] {5} [right of=2];

  \draw [->] (1) -- (2);
  \draw [->] (2) -- (3);
  \draw [->] (2) -- (4);
  \draw [->] (5) -- (2);

\end{tikzpicture}
\end{center}

Example Source: Computer Architecture 2010 – Out-Of-Order Execution; Lihu Rappoport & Adi Yoaz
Tensor Computations

Tensor
- Multidimensional array
- **Convolutions** are operations on tensors

Challenges of tensor computations
- Tensors can be **big data objects** and require many memory accesses.
- Tensor operations involve strided access patterns.
  - E.g matrix stored in row-major but computation requires column-major
  - Access to non-sequential data leads to **Cache misses**
Outline

- Background
- **Proposed Architecture**
  - Overview of Think Fast
  - Functional Slicing
  - Parallel Lanes & Streams
  - Role of the Compiler
  - The TSP Architecture
- Results
- Takeaway & Conclusion
Overview of Think Fast

Streaming Model
- Data is processed as it *streams* through the processor
- Multiple processing elements
  - Can perform tensor computations in parallel

Performance Optimization
- Scratchpad Memory
- Low Latency Interconnect
- ...

High-performance / energy-efficient acceleration for deep learning workloads.
Functional Slicing

- Conventional Chip Multiprocessor
  - Independent cores interconnected using on-chip network for data exchange
  - Instruction execution over several stages
    - Instruction Fetch (IF)
    - Instruction decode (ID)
    - Execution on ALUs (EX)
    - Memory Access (MEM)
    - Writeback (WB)
Functional Slicing

- **TSP**
  - Reorganizes the homogeneous two-dimensional mesh of cores into a functionally sliced microarchitecture
  - Disaggregation of the basic elements of a core per their respective functions
    - Instruction Control and Dispatch (ICU)
    - Memory (MEM)
    - Vector-Matrix Multiplication (VXM)
    - Matrix-Matrix Multiplication (MXM)
    - Data Exchange Interface (C2C)

Note: Tiles of a slice execute the same instruction stream (SIMD)
Parallel Lanes & Streams

- **Instructions** flow northward from the **ICUs**
- **Data** flows East and West between **functional slices**
- **Inter-lane** data movement uses **SXM slice**
Parallel Lanes & Streams

Example: Element-wise multiplication
Example: Element-wise multiplication

\[
\begin{bmatrix}
1 \\
2 \\
3
\end{bmatrix} \times \begin{bmatrix}
6 \\
7 \\
8
\end{bmatrix}
\]
Parallel Lanes & Streams

Example: Element-wise multiplication

\[
\begin{bmatrix}
1 \\
2 \\
3
\end{bmatrix}
\times
\begin{bmatrix}
6 \\
7 \\
8
\end{bmatrix}
\]
Parallel Lanes & Streams

Example: Element-wise multiplication
Parallel Lanes & Streams

Example: Element-wise multiplication
Parallel Lanes & Streams

Example: Element-wise multiplication
Parallel Lanes & Streams

Example: Element-wise multiplication
Parallel Lanes & Streams

Example: Element-wise multiplication
Parallel Lanes & Streams

Example: Element-wise multiplication
Example: Element-wise multiplication

\[
\begin{bmatrix}
1 \\
2 \\
3
\end{bmatrix}
\times
\begin{bmatrix}
6 \\
7 \\
8
\end{bmatrix}
\]
Parallel Lanes & Streams

Comparison of vector addition between RISC and TSP – Stream

[1] RISC: reduced instruction set computer
Role of the Compiler

Scheduling is left to the compiler

- The compiler must schedule instructions to use hardware correctly and efficiently.
  - **Reducing** scheduling role of **ICU** allows it to be small (<3% of area)

Optimisation is mainly left to the compiler / programmer -> determinism
The TSP Architecture
The TSP Architecture

Each tile operates on 16 lines (SIMD)

Superlane
Thus, minimum vector length is 16
The TSP Architecture

Each tile operates on 16 lines (SIMD)
Superlane
Thus, minimum vector length is 16

20 superlanes
Maximum vector length is 320
The TSP Architecture

Each tile operates on 16 lines (SIMD)
Superlane
Thus, minimum vector length is 16

20 superlanes
Maximum vector length is 320

144 independent ICUs
The TSP Architecture

Each tile operates on 16 lines (SIMD)
Superlane
Thus, minimum vector length is 16

20 superlanes
Maximum vector length is 320

144 independent ICUs
64 logical streams per lane (32E & 32W)
The TSP Architecture
The TSP Architecture

VXM consists of 4x4 mesh of ALUs in each lane for point-wise arithmetic
The TSP Architecture

VXM consists of 4x4 mesh of ALUs in each lane for point-wise arithmetic.

MXM consists of 4 independent 2D arrays (operate on int8 or fp16).
The TSP Architecture

ICU provides:
- Instruction fetching (IFetch)
- Inter-slice synchronization (Synch, Notify)
- Inter-instruction delay (NoOp)

VXM consists of 4x4 mesh of ALUs in each lane for point-wise arithmetic

MXM consists of 4 independent 2D arrays (operate on int8 or fp16)
The TSP Architecture
The TSP Architecture

SXM for intra-superlane and inter-lane switching
The TSP Architecture

SXM for intra-superlane and inter-lane switching

East and West MEM composed of 44 parallel SRAM slices
- 13 bits of addressing per slice
- 16 byte words -> 1 per lane
Yields a total of 220 MiBytes on-chip SRAM
The TSP Architecture

- **SXM** for intra-superlane and inter-lane switching
- **East and West MEM** composed of 44 parallel SRAM slices
  - 13 bits of **addressing** per slice
  - 16 byte words -> 1 per lane
  - Yields a total of **220 MiBytes** on-chip SRAM
- **C2C** provide send and receive
  - To exchange 320-byte vectors between chips
Outline

- Background
- Proposed Architecture
- Results
- Takeaways & Conclusion
Results

- Mapping the ResNet50 v2 image classification model to the TSP
  - Inference query performed in < 43\(\mu s\)
  - Which equals 20.4K images per second (batch size 1)
  - 2.5x speedup relative to Google TPU v3 (large batch)

- Fast matrix operations
  - MEM slices read 409,600 weights from memory and install them into MXM arrays in less than 40 cycles
    - MEM slices deliver 32 1-byte stream operands for each of the 320 parallel lanes -> 10 TiB/s operand stream bandwidth into MXMs

- Good value per transistor
  - 30\(K\) deep learning Ops/sec/transistor
  - Volta 100 yields 6.2\(K\) Ops/sec/transistor
Results

- Slope indicates memory bandwidth bound
- Saturation point at “roofline peak”
  - Arithmetically limited
Results

Power ($W$) for layers 14 through 40 in ResNet50

- comb
- mem
- leak
- clock
- flop

conv2D layers
Outline

- Background
- Proposed Architecture
- Results
- Takeaways & Conclusion
Takeaways & Conclusion
Takeaways & Conclusion

- **Deep Learning** workloads tend to be highly tensor-oriented
  - Limited performance on CPU and GPU
  - Special hardware interesting

- **Data streaming** provides a good alternative to common architecture
  - Data flowing through functional slices
  - Computations and communication via streams
  - Leveraging SIMD

- Large **on-chip memory**
  - Limits slow access to off-chip memory

- Data streaming has the potential to **outperform** other state-of-the-art architectures and can be **energy efficient**
Questions
Critique
**Strengths**

- **Innovative**
  - The paper shows a new take on Deep Learning Hardware and introduces a novel architecture
    - Data Streaming

- **Effort in power-efficiency**
  - The presented design has power efficiency in mind
  - Environmental impact of AI
  - Cost reduction

- **Scalability**
  - The presented TSP offers C2C
    - Warehouse-scale computers (WSC)

- **Programming interface**
  - Developer friendly
  - Fine-grained parallelism
Weaknesses

- Most benefits limited to deep learning
  - Dense matrices in mind
  - Does not discuss computation on sparse matrices
  - Extensive discussion & focus on linear operations
    - Might not perform as desired on models that require non-linear operations
      - Classification Trees
      - Support Vector Machines (SVM)
      - K-Nearest Neighbors

- Some claims not fully explained?
  - E.g. 4x improvement compared to other [...] GPUs -> Which ones?

- Complicated and low-quality diagrams

- Limited testing
  - Not field-tested (at point of release)
  - Received silicone close to ISCA deadline
  - Focus on ResNet50
Discussion
How could the architecture be extended to improve sparse matrix performance?
Discussion

- **cuSPARSE**
  - Nvidia’s solution
  - Represent sparse matrices as **dense internally**

- **Related research:**
  - Carl Yang, Aydin Buluç, John D. Owens. Design Principles for Sparse Matrix Multiplication on the GPU. *University of California 2018*
Discussion

- **Compressed sparse row (CSR) support**
  - Consider CSR in memory
  - Maybe add specialized functional units
    - Sparse Matrix-Vector and sparse Matrix-Matrix (sVXM, sMXM)

```
Sparse matrix

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
<td></td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>e</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>g</td>
</tr>
</tbody>
</table>

Compressed Sparse Row (CSR)

- Row pointers: 0 3 4 6 7
- Column offsets: 0 2 3 1 2 3 3
- Data: a b c d e f g
```

- Related research:
  - Gengyu Rao, Jingji Chen, Jason Yik, and Xuehai Qian. SparseCore: stream ISA and processor specialization for sparse computation. **ASPLOS 2022**
Discussion

- How could the architecture be extended to improve non-linear operation performance?
Discussion

- Separate processing units for non-linear operations
  - Effectively perform non-linear transformation of input before passing it to the next layer in stream
  - Hardwire common activation functions like ReLu or sigmoid
- Potentially benefits CNNs\[1\] or RNNs\[2\]

<table>
<thead>
<tr>
<th>AF</th>
<th>Mathematical Description</th>
<th>Symmetry</th>
<th>Evaluation Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Sigmoid</td>
<td>( f(x) = \frac{1}{1+e^{-x}} )</td>
<td>( f(x) = \begin{cases} f(x) &amp; x \geq 0 \ 1 - f(</td>
<td>x</td>
</tr>
<tr>
<td>2. Tanh</td>
<td>( f(x) = \frac{e^x-e^{-x}}{e^x+e^{-x}} )</td>
<td>( f(x) = \begin{cases} f(x) &amp; x \geq 0 \ -f(</td>
<td>x</td>
</tr>
<tr>
<td>3. Gaussian</td>
<td>( f(x) = e^{-x^2} )</td>
<td>( f(x) = \begin{cases} f(x) &amp; x \geq 0 \ f(</td>
<td>x</td>
</tr>
<tr>
<td>4. SILU</td>
<td>( f(x) = \frac{x}{1+e^{-x}} )</td>
<td>-</td>
<td>(-8,8)</td>
</tr>
<tr>
<td>5. ELU</td>
<td>( f(x) = \begin{cases} a(e^x - 1) &amp; x \leq 0 \ x &amp; x &gt; 0 \end{cases} )</td>
<td>-</td>
<td>(-4,4)</td>
</tr>
<tr>
<td>6. Softplus</td>
<td>( f(x) = \ln(1 + e^x) )</td>
<td>-</td>
<td>(-4,4)</td>
</tr>
</tbody>
</table>

Related research:

- González-Díaz_Conti, G.; Vázquez-Castillo, J.; et. Al. Hardware-Based Activation Function-Core for Neural Network Implementations. Technological Institute of Sonora 2022

\[1\] CNN: Convolutional Neural Network; \[2\] RNN: Recurrent Neural Network
Discussion

- Separate **processing units for non-linear operations**
  - Effectively perform **non-linear transformation** of input before passing it to the **next layer** in stream
  - Hardwire common activation functions like **ReLu** or **sigmoid**

- Potentially benefits **CNNs**[1] or **RNNs**[2]

<table>
<thead>
<tr>
<th>AF</th>
<th>Mathematical Description</th>
<th>Symmetry</th>
<th>Evaluation Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Sigmoid</td>
<td>$f(x) = \frac{1}{1+e^{-x}}$</td>
<td>$f(x) = \begin{cases} f(x) &amp; x \geq 0 \ 1 - f(</td>
<td>x</td>
</tr>
<tr>
<td>2. Tanh</td>
<td>$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$</td>
<td>$f(x) = \begin{cases} f(x) &amp; x \geq 0 \ -f(</td>
<td>x</td>
</tr>
<tr>
<td>3. Gaussian</td>
<td>$f(x) = e^{-x^2}$</td>
<td>$f(x) = \begin{cases} f(x) &amp; x \geq 0 \ f(</td>
<td>x</td>
</tr>
<tr>
<td>4. SILU</td>
<td>$f(x) = \frac{x}{1+e^{-x}}$</td>
<td>-</td>
<td>$(-8, 8)$</td>
</tr>
<tr>
<td>5. ELU</td>
<td>$f(x) = \begin{cases} a(e^x - 1) &amp; x \leq 0 \ x &amp; x &gt; 0 \end{cases}$</td>
<td>-</td>
<td>$(-4, 4)$</td>
</tr>
<tr>
<td>6. Softplus</td>
<td>$f(x) = \ln(1 + e^x)$</td>
<td>-</td>
<td>$(-4, 4)$</td>
</tr>
</tbody>
</table>

**Related research:**
- González-Díaz_Conti, G.; Vázquez-Castillo, J.; et. Al. Hardware-Based Activation Function-Core for Neural Network Implementations. **Technological Institute of Sonora 2022**
- Ibrahim Ahmed, Sahil Parmar, et. al. **Groq Inc.** Answer Fast: Accelerating BERT on the Tensor Streaming Processor **2022**

Discussion

- **TransPimLib**
  - Software Approach
  - Transcendental Functions **through library**
    - Functions that do not satisfy a polynomial equation
  - Intended for PIM
  - Utilizes **CORDIC-based methods and Lookup-tables**
    - Iterative method that uses only bit-shifts, additions, and table lookups.

<table>
<thead>
<tr>
<th>Implementation Method</th>
<th>sin</th>
<th>cos</th>
<th>tan</th>
<th>sinh</th>
<th>cosh</th>
<th>tanh</th>
<th>exp</th>
<th>log</th>
<th>sqrt</th>
<th>GELU</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORDIC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>M-LUT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>M-LUT+Interpolation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>L-LUT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>L-LUT+Interpolation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>D-LUT+Interpolation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DL-LUT+Interpolation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CORDIC+LUT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- **Related research:**
  - Maurus Item, Juan Gómez-Luna, Yuxin Guo, Geraldo F. Oliveira, Mohammad Sadrosadati, Onur Mutlu
  - A Library for Efficient Transcendental Functions on Processing-in-Memory Systems. **ETH Zürich 2023**
Discussion

- How could this architecture be **scalable**?
  - Low Latency Interconnect (**C2C**)
How to prevent network *contention*?
Software Scheduled Network (SSN)
- Leaves data-movement decision-making to the compiler
- Possible due to determinism

More Information:
Groq Inc.
Synchronous Scalability for Real-time AI and HPC 2022
Are there potential security risks?

- Compiler must do a “good job”
- Potential for attacks
  - See e.g. Aimoniotis, Pavlos; Sjalander, Magnus; Kaxiras, Stefanos
    *Reorder Buffer Contention: A Forward Speculative Interference Attack for Speculation Invariant Instructions.*
    IEEE 2021

- Any other concerns?
Questions
1.0: A visualization of 0- to 3-dimensional tensors.
   - [22.03.2023] https://hkilter.com/index.php?title=What_is_Tensor%3F

2.0: The Groq TSP at a glance
   - [22.03.2023] https://www.forbes.com/sites/karlfreund/2021/02/25/the-cambrian-ai-landscape-groq/?sh=7d26c8b621bf

2.1: Conventional 2D mesh of cores reorganized into a functionally sliced arrangement of tiles
   - [PAPER]

2.2: The organization and dataflow within a row in the on-chip network.
   - [PAPER]

2.3: Conventional RISC execution contrasted with producer-consumer streams in the TSP.
   - [PAPER]

3.0: Die photo of 14nm ASIC implementation of the Groq TSP.
   - [PAPER]

3.1: Summary of Instructions for each functional slice.
   - [PAPER]

3.2: Roofline diagram showing arithmetic throughput (at 1 GHz core clock) varying with offered load.
   - [PAPER]

3.3: Power usage for ResNet50 layers.
   - [PAPER]

4.0: The Compressed Sparse Row (CSR) format for representing sparse matrices provides a compact representation, but requires index arrays to interpret the data.
   - Two Sparsities Are Better Than One: Unlocking the Performance Benefits of Sparse-Sparse Networks

4.1: Description of non-linear AFs for ANN.
   - Hardware-Based Activation Function-Core for Neural Network Implementations. *Electronics* 2022

4.2: A traditional non-deterministic network compared to a Software-scheduled Network.
   - Synchronous Scalability for Real-time AI and HPC 2022