Flash-Cosmos
In-Flash Bulk Bitwise Operations
Using Inherent Computation Capability
of NAND Flash Memory

Jisung Park, Roknoddin Azizi, Geraldo F. Oliveira,
Mohammad Sadrosadati, Rakesh Nadig, David Novo, Juan Gómez Luna,
Myungsuk Kim, and Onur Mutlu

Published at MICRO 2022

Rakesh Nadig
11\textsuperscript{th} November 2022
Executive Summary

• **Background:** Bulk bitwise operations are widely used in many important data-intensive applications, e.g., databases, graph processing, cryptography etc.

• **Problem:**
  • Performance and energy efficiency of bulk bitwise operations are bottlenecked by
    1) data movement between storage and the compute unit in traditional systems and in-storage processing (ISP)
    2) data sensing (serial reading of operands) in prior in-flash processing (IFP) techniques
  • Prior IFP techniques provide low reliability during computation

• **Goal:** Improve performance, energy efficiency and reliability of bulk bitwise operations in in-flash processing

• **Key Idea:** Flash-Cosmos (Flash-Computation with One-Shot Multi-Operand Sensing) is an in-flash processing technique that is based on two key ideas:
  • Multi-Wordline Sensing (MWS): Enables multi-operand bulk bitwise operations with a single sensing (read) operation
  • Enhanced SLC-mode Programming (ESP): Increases the voltage margin between the erased and programmed states to provide higher reliability during in-flash computation

• **Key Results:** Flash-Cosmos is evaluated using 160 real 3D NAND flash chips and with a state-of-the-art SSD simulator on three real-world workloads
  • Flash-Cosmos improves the performance and energy efficiency by 3.5x and 3.3x over state-of-the-art IFP technique while providing high reliability during computation
Talk Outline

Motivation

Background

Flash-Cosmos

Evaluation

Summary
Bulk Bitwise Operations

- Hyper-dimensional Computing
- Cryptography
- Set Operations
- Graph Processing
- Genome Analysis
- Databases
- Web Search
Data movement between compute units and the memory hierarchy significantly affects the performance of bulk bitwise operations.
Data-Movement Bottleneck

- Conventional systems perform outside-storage processing (OSP) after moving the data to host CPU through the memory hierarchy.

**Diagram:**

- Host Processor (CPU, GPU)
- Main Memory
- Storage

**Data Movement Bottleneck**

External I/O bandwidth of storage systems is the main bottleneck for data movement in OSP.
NDP for Bulk Bitwise Operations

Our focus
Large data sets that do not fit in main memory

In-Flash
(e.g., ParaBit⁵)

In-Storage
(e.g., Biscuit⁴)

Near-Data Processing

Cache
(e.g., Compute Cache¹)

DRAM-based main memory
(e.g., Ambit²)

NVM-based main memory
(e.g., Pinatubo³)

In-Storage Processing (ISP)

- ISP performs computation using an in-storage computation unit (embedded cores or FPGA)
- ISP reduces external data movement by transferring only the computation results to the host
In-Storage Processing (ISP)

- ISP performs computation using the in-storage computation unit
- ISP reduces external data movement by transferring only the computation results to the host

**Storage internal I/O bandwidth** is the main bottleneck in ISP

- Memory Bandwidth: tens to hundreds of GB/s
- Storage External I/O Bandwidth: ~ 8 GB/s
- Storage Internal I/O Bandwidth: ~ 9.6 GB/s

Data Movement Bottleneck

(CPU, GPU) -> Memory -> Storage
In-Flash Processing (IFP)

- IFP performs **computation within the flash chips** as the data operands are being read **serially**
- IFP reduces the **internal data movement bottleneck in storage** by transferring only the computation results to the in-storage computation unit

![Diagram of IFP and data flow](Image)

- **Memory Bandwidth**
  - Tens to hundreds of GB/s
- **Storage External I/O Bandwidth**
  - ~ 8 GB/s
- **Storage Internal I/O Bandwidth**
  - ~ 9.6 GB/s

*SAFARI*
• State-of-the-art IFP technique \cite{1} performs bulk bitwise operations by controlling the latching circuit of the page buffer.

Data Sensing Bottleneck in IFP

- State-of-the-art IFP technique performs bulk bitwise operations by controlling the latching circuit of the page buffer.
Data Sensing Bottleneck in IFP

- State-of-the-art IFP technique performs bulk bitwise operations by controlling the latching circuit of the page buffer.
Data Sensing Bottleneck in IFP

- State-of-the-art IFP technique performs bulk bitwise operations by controlling the latching circuit of the page buffer.
Data Sensing Bottleneck in IFP

- State-of-the-art IFP technique performs bulk bitwise operations by controlling the latching circuit of the page buffer.

Serial data sensing is the bottleneck in in-flash processing techniques.
Reliability Issues in IFP

- NAND Flash memory suffers from high raw-bit error rate due to many sources of disturbances
- ECC and data-randomization techniques are used to improve the reliability of flash memory
- Prior IFP approaches cannot leverage ECC and data-randomization techniques as computation is performed within the flash chips during data sensing

Reliability Issues in IFP

- Prior IFP approaches suffer from erroneous computation results due to high raw-bit error rate of NAND flash memory.
Reliability Issues in IFP

- Prior IFP approaches suffer from erroneous computation results due to high raw-bit error rate of NAND flash memory.
Reliability Issues in IFP

- Prior IFP approaches suffer from erroneous computation results due to high raw-bit error rate of NAND flash memory.
Reliability Issues in IFP

- Prior IFP approaches suffer from erroneous computation results due to high raw-bit error rate of NAND flash memory.

Prior IFP techniques require the application to be highly error-tolerant.
Our Goal

Address the bottleneck of state-of-the-art IFP techniques (serial sensing of operands)

Make IFP reliable (provide accurate computation results)
Our Proposal

• Flash-Cosmos
  • Enables Computation on multiple operands using a single sensing operation
  • Provides high reliability during in-flash computation

![Diagram of NAND Flash Chip]

- NAND Flash Chip
- Page Buffer
- Simultaneous data Sensing
- A AND B AND C
A flash cell stores data by adjusting the amount of charge in the cell.

- Erased Cell (Low Charge Level): Operates as a resistor.
- Programmed Cell (High Charge Level): Operates as an open switch.
NAND Flash Basics: A NAND String

- A set of flash cells are **serially connected** to form a **NAND String**
NAND Flash Basics: Read Mechanism

• NAND flash memory reads data by **checking the bitline current**

![Diagram of NAND Flash Memory Read Mechanism]

**Non-Target Cells:**
Operate as *resistors* regardless of stored data

**Bitline (BL)**

1

NAND String
NAND flash memory reads data by **checking the bitline current**

- **Target Cells:**
  - Operate as *resistors* (1)
  - or *open switches* (0)

- **Non-Target Cells:**
  - Operate as *resistors* regardless of stored data
NAND Flash Basics: Read Mechanism

- NAND flash memory reads data by **checking the bitline current**

Target Cells:
*Operate as resistors (1) or open switches (0)*

Non-Target Cells:
*Operate as resistors regardless of stored data*

NAND String

BL$_i$

Reads as ‘1’ if BL current flows

BL$_j$

Reads as ‘0’ if BL current cannot flow
NAND strings connected to different bitlines comprise a NAND block.

A single wordline (WL) controls a large number of flash cells: High bit-level parallelism.
NAND Flash Basics: Block Organization

- A large number of blocks share the same bitlines
Similarity to Digital Logic Gates

- A large number of blocks share the same bitlines

Cells in the same block are connected serially:
Similar to digital AND

2-input AND

A • B
• A large number of blocks share the same bitlines.

Cells in the same block are connected serially:
Similar to digital AND

2-input AND

A\cdot B

Cells in different blocks are connected in parallel:
Similar to digital OR

2-input OR

A + B
Enables in-flash bulk bitwise operations on multiple operands with a single sensing operation using Multi-Wordline Sensing (MWS)
Multi-Wordline Sensing (MWS): Bitwise AND

- Intra-Block MWS: Simultaneously activates multiple WLs in the same block
  - Bitwise AND of the stored data in the WLs
Multi-Wordline Sensing (MWS): Bitwise AND

- **Intra-Block MWS**: Simultaneously activates multiple WLs in the same block
  - Bitwise AND of the stored data in the WLs

```plaintext
<table>
<thead>
<tr>
<th></th>
<th>BL1</th>
<th>BL2</th>
<th>BL3</th>
<th>BL4</th>
</tr>
</thead>
<tbody>
<tr>
<td>WL1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>WL2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>WL3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WL4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Non-Target Cells: *Operate as resistors*
Multi-Wordline Sensing (MWS): Bitwise AND

- **Intra-Block MWS**: Simultaneously activates multiple WLs in the same block
  - Bitwise AND of the stored data in the WLs

Target Cells: *Operate as resistors* (1) or *open switches* (0)

Non-Target Cells: *Operate as resistors*

Result: 0 0 0 1
Multi-Wordline Sensing (MWS): Bitwise AND

- **Intra-Block MWS**: Simultaneously activates multiple WLs in the same block
  - **Bitwise AND** of the stored data in the WLs

A bitline reads as ‘1’ only when all the target cells store ‘1’
→ Equivalent to the bitwise AND of all the target cells

**Target Cell**: Operate as a resistance

**Non-Target Cell**: Operate as an open switch

Result: 0 0 0 1
Multi-Wordline Sensing (MWS): Bitwise AND

- **Intra-Block MWS**: Simultaneously activates multiple WLs in the same block
  - **Bitwise AND** of the stored data in the WLs

![Diagram of Multi-Wordline Sensing](image)

**Target Cell**: Operate as a resistance (1) or an open switch (0)

**Result**: 0 0 0 0

**SAFARI**
Multi-Wordline Sensing (MWS): Bitwise AND

- **Intra-Block MWS**: Simultaneously activates multiple WLs in the same block
  - Bitwise AND of the stored data in the WLs

A bitline reads as ‘0’ even if one target cell stores ‘0’
→ Equivalent to the bitwise AND of all the target cells
Multi-Wordline Sensing (MWS): Bitwise AND

- Intra-Block MWS: Simultaneously activates multiple WLs in the same block
  - Bitwise AND of the stored data in the WLs

Flash-Cosmos (Intra-Block MWS) enables bitwise AND of multiple pages in the same block via a single sensing operation
Multi-Wordline Sensing (MWS): Bitwise OR

- **Inter-Block MWS**: Simultaneously activates multiple WLs in different blocks
  - Bitwise OR of the stored data in the WLs
Multi-Wordline Sensing (MWS): Bitwise OR

- **Inter-Block MWS**: Simultaneously activates multiple WLs in different blocks
  - Bitwise OR of the stored data in the WLs

![Diagram of Multi-Wordline Sensing](image)

Result: 1 1 1 0
Multi-Wordline Sensing (MWS): Bitwise OR

- **Inter-Block MWS**: Simultaneously activates multiple WLs in different blocks
  - Bitwise OR of the stored data in the WLs

A bitline reads as ‘0’ only when all the target cells store ‘0’
→ Equivalent to the bitwise OR of all the target cells
**Multi-Wordline Sensing (MWS): Bitwise OR**

- **Inter-Block MWS**: Simultaneously activates multiple WLs in different blocks
  - Bitwise OR of the stored data in the WLs

![Diagram of MWS](image)

- **WL\textsubscript{x} in Block\textsubscript{1}**
- **WL\textsubscript{y} in Block\textsubscript{i}**
- **WL\textsubscript{z} in Block\textsubscript{j}**

Result: 1

BL\textsubscript{1}  BL\textsubscript{2}  BL\textsubscript{3}  BL\textsubscript{4}
• **Inter-Block MWS**: Simultaneously activates multiple WLs in different blocks
  
  - Bitwise OR of the stored data in the WLs
  
  A bitline reads as ‘1’ only when one of the target cells store ‘1’
  → Equivalent to the bitwise OR of all the target cells
**Multi-Wordline Sensing (MWS): Bitwise OR**

- **Inter-Block MWS:** Simultaneously activates multiple WLs in different blocks

- Bitwise OR of the stored data in the WLs

Flash-Cosmos (Inter-Block MWS) enables bitwise OR of multiple pages in different blocks via a single sensing operation.
Supporting Other Bitwise Operations

**Bitwise NOT**

Exploit **Inverse Read**\(^1\) which is supported in modern NAND flash memory for copy-back operations.

**Bitwise NAND/ NOR**

Use Inverse Read operation with bitwise AND/OR operation.

**Bitwise XOR/XNOR**

Use XOR between the latches\(^2\) which is also supported in NAND flash memory.

---

[1] Lee+, “High-Performance 1-Gb-NAND Flash Memory with 0.12-µm Technology,” JSSC, 2002

Flash-Cosmos: Overview

Enables in-flash bulk bitwise operations on multiple operands with a *single* sensing operation using Multi-Wordline Sensing (MWS)

Increases the reliability of in-flash bulk bitwise operations by using Enhanced SLC-mode Programming (ESP)
Enhanced SLC-Mode Programming (ESP)

- SLC-mode programming provides a large voltage margin between the erased and programmed states.
- SLC-mode programming is still highly error-prone without the use of ECC and data-randomization.
Enhanced SLC-Mode Programming (ESP)

- ESP further increases the voltage margin between the erased and programmed states.
- A wider voltage margin between the two states improves reliability by making the cells less vulnerable to errors.
- Perform additional steps in the incremental step pulse programming (ISPP) scheme to increase the voltage margin.

![Graph showing the voltage margin increase in ESP]

1 Erased
0 Prog.
0 Prog.

Increased voltage margin in ESP
Enhanced SLC-Mode Programming (ESP)

- ESP increases the voltage margin between the erased and programmed states.
- A wider voltage margin between the two states improves reliability during data sensing by making the cells less vulnerable to errors.

ESP improves the reliability of in-flash computation without the use of ECC or data-randomization techniques.
Enhanced SLC-Mode Programming (ESP)

- ESP increases the voltage margin between the erased and programmed states.
- A wider voltage margin between the two states improves reliability during data sensing by making the cells less vulnerable to errors.

ESP can improve the reliability of prior in-flash processing techniques as well.
New Flash Commands

• Three new commands to support Flash-Cosmos

  - **MWS command** to be used for:
    - Intra- and inter-block MWS
    - Inverse read
    - Accumulation of results of all reads
  - **ISCM command** slot allows the flash controller to turn on/off four features,
    - Inverse-read mode (I)
    - Sensing-latch (S-latch) initialization (S)
    - Cache-latch (C-latch) initialization (C)
    - Move data from S-latch to C-latch (M)
New NAND Commands

• **Three new commands to support Flash-Cosmos**

  ![Diagram](image)

  - Inverse read / S-latch init. / C-latch init. / Move
  - Page Bitmap (PBM) to specify the WLs to be activated for MWS operations
  - Four address slots for inter-block MWS command

• **ESP command** works like a regular program command

• **XOR command** performs bitwise XOR between the sensing and cache latches and stores the result in C-latch
Operational example

\[
\{ A_1 + (B_1 \cdot B_2 \cdot B_3 \cdot B_4) \} \cdot (C_1 + C_3) \cdot (D_2 + D_4)
\]

1. \((C_1 + C_3) \cdot (D_2 + D_4) \equiv (\overline{C_1} \cdot C_3) + (D_2 \cdot D_4)\)

   - **ISCM**
   - **BLK**
   - **PBM**
   - **BLK**
   - **PBM**

   - **MWS**
   - Blk: 3 WLS: 1&3
   - Blk: 4 WLS: 2&4

2. \(A_1 + (B_1 \cdot B_2 \cdot B_3 \cdot B_4)\)

   - **MWS**
   - Blk: 1 WLS: 1
   - Blk: 2 WLS: all

**Flash-Cosmos NAND Flash Chip**

- **Plane#1**
- **Plane#2**

**Blk**

- **A**
- **B**
- **C**
- **D**

- **1**
- **2**
- **3**
- **4**

- **1**
- **2**
- **3**
- **4**

- **1**
- **2**
- **3**
- **4**

\(-\)**
### Operational example

\[
\{A_1 + (B_1 \cdot B_2 \cdot B_3 \cdot B_4)\} \cdot (C_1 + C_3) \cdot (D_2 + D_4)
\]

1. \( (C_1 + C_3) \cdot (D_2 + D_4) \equiv (\overline{C_1} \cdot \overline{C_3}) + (D_2 \cdot \overline{D_4}) \)

<table>
<thead>
<tr>
<th>ISCM</th>
<th>BLK</th>
<th>PBM</th>
<th>BLK</th>
<th>PBM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MWS</td>
<td>11110011</td>
<td>0101</td>
<td>0101</td>
<td>0100</td>
</tr>
</tbody>
</table>

- Blk: 3 WLS: 1&3
- Blk: 4 WLS: 2&4

2. \( A_1 + (B_1 \cdot B_2 \cdot B_3 \cdot B_4) \)

| MWS | 00110001 | 0001 | 0010 | 1111 | CONF |

- Blk: 1 WLS: 1
- Blk: 2 WLS: all

#### Flash-Cosmos NAND Flash Chip

- Plane#1
- Plane#2

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Blk_1
- Blk_2
- Blk_3
- Blk_4

---

Bit vectors \( C_i \) and \( D_i \) are programmed with their inverse data for the bitwise OR operation.
Operational example

\[ \{ A_1 + (B_1 \cdot B_2 \cdot B_3 \cdot B_4) \} \cdot (C_1 + C_3) \cdot (D_2 + D_4) \]

1. \((C_1 + C_3) \cdot (D_2 + D_4) \equiv (\overline{C_1 \cdot C_3} + \overline{D_2 \cdot D_4})\)

   - ISCM: iiii
   - BLK: i
   - PBM: 0011
   - CONT: 0101
   - BLK: 0
   - PBM: 1010
   - CONF: 0
   - Blk: 3
   - WLs: 1&3
   - Blk: 4
   - WLs: 2&4

2. \(A_1 + (B_1 \cdot B_2 \cdot B_3 \cdot B_4)\)

   - ISCM: 0001
   - BLK: 0
   - PBM: 0001
   - CONT: 0010
   - BLK: 1
   - PBM: 1111
   - CONF: 0
   - Blk: 1
   - WLs: 1
   - Blk: 2
   - WLs: all

Issue an MWS command for 
\((C_1 + C_3) \cdot (D_2 + D_4)\) by enabling inverse-read mode and initialization of sensing and cache latches.
Operational example

\[ \{ A_1 + (B_1 \cdot B_2 \cdot B_3 \cdot B_4) \} \cdot (C_1 + C_3) \cdot (D_2 + D_4) \]

\( (C_1 + C_3) \cdot (D_2 + D_4) \equiv (\overline{C_1} \cdot \overline{C_3}) + (\overline{D_2} \cdot \overline{D_4}) \)

\[ \{ A_1 + (B_1 \cdot B_2 \cdot B_3 \cdot B_4) \} \]

Issue an **MWS command** for
\[ \{ A_1 + (B_1 \cdot B_2 \cdot B_3 \cdot B_4) \} \] while disabling the inverse-read mode and initialization of both latches
Operational example

\[
\{A_1 + (B_1 \cdot B_2 \cdot B_3 \cdot B_4)\} \cdot (C_1 + C_3) \cdot (D_2 + D_4)
\]

By disabling the initialization of latches, the result of the two MWS commands are accumulated in sensing and cache latches.
Talk Outline

- Motivation
- Background
- Flash-Cosmos
- Evaluation
- Summary
Evaluation Methodology

• We evaluate Flash-Cosmos using

160 real state-of-the-art 3D NAND flash chips
Real Device Characterization

• We validate the **feasibility, performance, and reliability** of Flash-Cosmos

• **160 48-layer 3D TLC NAND flash chips**
  • 3,686,400 tested wordlines

• **Under worst-case operating conditions**
  • 1-year retention time at 10K P/E cycles
  • Worst-case data patterns
Results: Real-Device Characterization

Both intra- and inter-block MWS operations require no changes to the cell array of commodity NAND flash chips.

Both MWS operations can activate multiple WLs (intra: up to 48, inter: up to 4) at the same time with small increase in sensing latency (< 10%).

ESP significantly improves the reliability of computation results (zero bit error in the tested flash cells).
Evaluation Methodology

• We evaluate Flash-Cosmos using

160 real state-of-the-art 3D NAND flash chips

Three real-world applications that perform bulk bitwise operations
Evaluation with real-world workloads

• Simulation
  • MQSim [Tavakkol+, FAST’18] to model the performance of Flash-Cosmos and the baselines

• Workloads
  • Three real-world applications that heavily rely on bulk bitwise operations
  • Bitmap Indices (BMI): Bitwise AND of up to \(~1,000\) operands
  • Image Segmentation (IMS): Bitwise AND of 3 operands
  • \(k\)-clique star listing (KCS): Bitwise OR of up to 32 operands

• Baselines
  • Outside-Storage Processing (OSP): a multi-core CPU (Intel i7 11700K)
  • In-Storage Processing (ISP): an in-storage hardware accelerator
  • ParaBit [Gao+, MICRO’21]: the state-of-the-art in-flash processing (IFP) mechanism
Results: Performance & Energy

Flash-Cosmos provides **significant performance & energy benefits** over all the baselines.

Performance and energy benefits only increase with more number of operands.
Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory

Jisung Park$\textsuperscript{SV}$ Roknoddin Azizi$\textsuperscript{S}$ Geraldo F. Oliveira$\textsuperscript{S}$ Mohammad Sadrosadati$\textsuperscript{S}$ Rakesh Nadig$\textsuperscript{S}$ David Novo$\dagger$ Juan Gómez-Luna$\textsuperscript{S}$ Myungsuk Kim$\ddagger$ Onur Mutlu$\textsuperscript{S}$

$\textsuperscript{S}$ETH Zürich $\nabla$ POSTECH $\dagger$LIRMM, Univ. Montpellier, CNRS $\ddagger$Kyungpook National University

Talk Outline

Motivation

Background

Flash-Cosmos

Evaluation of Flash-Cosmos and Key Results

Summary
Flash-Cosmos: Summary

- First work to enable multi-operand bulk bitwise operations with a single sensing operation and high reliability
- Improves performance by 3.5x/25x/32x on average over ParaBit/ISP/OSP
- Improves energy efficiency by 3.3x/13.4x/95x on average over ParaBit/ISP/OSP
- Low-cost & requires no changes to flash cell arrays
Flash-Cosmos
In-Flash Bulk Bitwise Operations
Using Inherent Computation Capability
of NAND Flash Memory

Jisung Park, Roknoddin Azizi, Geraldo F. Oliveira,
Mohammad Sadrosadati, Rakesh Nadig, David Novo, Juan Gómez Luna,
Myungsuk Kim, and Onur Mutlu

Published at MICRO 2022

Rakesh Nadig
09th November 2022
BACKUP SLIDES
NAND Flash Programming

(a) SLC–mode programming

Retention loss Interference

Retention age [months]

(b) MLC–mode programming

PEC: 0K 1K 3K 2K 6K 10K

w/ data randomization

w/o data randomization

(a) SLC–mode programming
Outside-Storage Processing (OSP)

- Moves every operand from the storage to the compute unit (CPU/GPU) for computation
- Performs the computation and writes the results back to the SSD

OSP: Outside-Storage Processing
Outside-Storage Processing (OSP)

- Moves every operand from the storage to the compute unit (CPU/GPU) for computation.
- Performs the computation and writes the results back to the SSD.

**Outside-Storage Processing** is bottlenecked by data movement between the compute unit and SSD (SSD external bandwidth).
Outside-Storage Processing (OSP)

- Moves every operand from the storage to the compute unit (CPU/GPU) for computation
- Performs the computation and writes the results back to the SSD

Near-Data Processing (NDP)

✓ moves computation closer to where the data resides
✓ is a promising approach to mitigate data movement
In-Storage Processing (ISP)

- Reads the operands from the NAND flash chips to the SSD Controller in a serial manner
- Performs the computation in the SSD controller
- Moves the computation result to the host

**Diagram:**
- SSD Controller
  - Channels
  - NAND Flash Array
  - External BW: 8 GB/s
  - Internal BW: 9.6 (1.2×8) GB/s

**Graph:**
- Start computation & result transfer to host
- Bottleneck: Internal I/O

**Abbreviation:**
- ISP: In-Storage Processing
In-Storage Processing (ISP)

- Reads the operands from the NAND flash chips to the SSD Controller in a serial manner
- Performs the computation in the SSD controller
- Moves the computation result to the host

**In-Storage Processing** is bottlenecked by data movement between NAND flash memory and SSD controller (**SSD internal bandwidth**)
In-Flash Processing (IFP)

- Performs computation within the NAND flash chips
- Moves only the computation results from the NAND flash chips to the SSD controller and host

**Diagram:***

- **SSD Controller**
  - Channels
  - NAND Flash Array

- **Channels**
  - Channel#1
  - Channel#8
  - Bit-Vectors
  - Die#1
  - D#4
  - Plane
  - D#1
  - D#4

**External BW:** 8 GB/s

**Internal BW:** 9.6 (1.2×8) GB/s

**Start computation & result transfer to controller**

**Bottleneck:** Sensing

**IFP:** In-Flash Processing
**In-Flash Processing (IFP)**

- Performs computation within the NAND flash chips
- Moves only the computation results from the NAND flash chips to the SSD controller and host

**In-Flash Processing** significantly reduces data movement compared to OSP and ISP

**Bottleneck:** Sensing

**IFP:** In-Flash Processing
In-Flash Processing (IFP)

- Performs computation within the NAND flash chips
- Moves only the computation results from the NAND flash chips to the SSD controller and host

In-Flash Processing is bottlenecked by data sensing
Overview of three computation approaches

In-Flash processing significantly reduces data movement
Background

- NAND Flash Organization

NAND Flash Block

NAND Flash Package

Peripheral

Die#0

Die#1

⋯
Background

• NAND flash read mechanism consists of three steps:
  1) Precharge  2) Evaluation  3) Discharge
Enable precharge transistor $M_{PRE}$ to charge all target BLs and their sense-out capacitors ($C_{SO}$) to $V_{PRE}$. 

- **Precharge**
- **Evaluation**
- **Discharge**
• Evaluation
Disconnect the BLs from $V_{\text{PRE}}$ and enable the latching circuit.
If $V_{TH} \leq V_{REF}$, the charge in $C_{SO}$ quickly flows through the NAND string (Sensed as 1)
Background

• Evaluation

If $V_{TH} > V_{REF}$, the target cell blocks the BL discharge current (Sensed as 0)
Bitlines are discharged to return the NAND string to its initial state for future operations.
During precharge, the flash chip charges the BL, making \( V_{SO} = 1 \). Before evaluation, the chip initializes the latching circuit by activating only transistor \( M_1 \) while enabling \( M_2 \). Charge in \( C_{SO} \) quickly flows through the NAND string making \( V_{SO} = 0 \) and \( V_{OUT} = 1 \).
If $V_{TH} > V_{REF}$, $C_{SO}$ cannot discharge due to the target cell acting as an open switch, leading to $V_{SO} = 1$ and $V_{OUT} = 0$.
During precharge, the flash chip charges the BL, making $I_{SO} = 1$.

Before evaluation, the chip initializes the latching circuit by activating only transistor $M_2$.

Evaluation step disables $M_{PRE}$ and $M_2$ while enabling $M_1$. Charge in $C_{SO}$ quickly flows through the NAND string making $I_{SO} = 0$ and $I_{OUT} = 1$.

**Background**

- Latching circuit during **inverse read** operation

**Diagram**

Charge in $C_{SO}$ quickly flows through the NAND string making $I_{SO} = 0$ and $I_{OUT} = 1$.
If \( V_{TH} > V_{REF} \), \( C_{SO} \) cannot discharge due to the target cell acting as an open switch, leading to \( I_{SO} = 1 \) and \( I_{OUT} = 0 \).