Biscuit: A Framework for Near-Data Processing of Big Data Workloads

Boncheol Gu et al.
2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)

Jun Heo / Jonghyun Bae (ARC)
(j.heo@snu.ac.kr / jonghbae@snu.ac.kr)

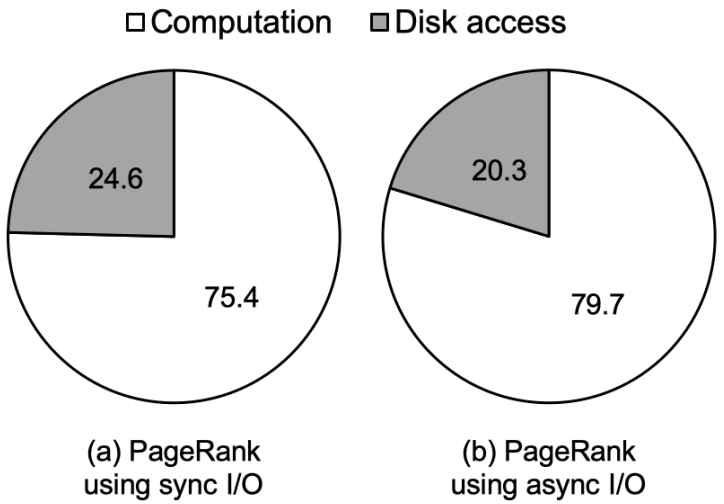
June 11th, 2019

Contents

- Background
- Biscuit
- Evaluation
- Related work
- Conclusion

Background: Data-intensive Applications

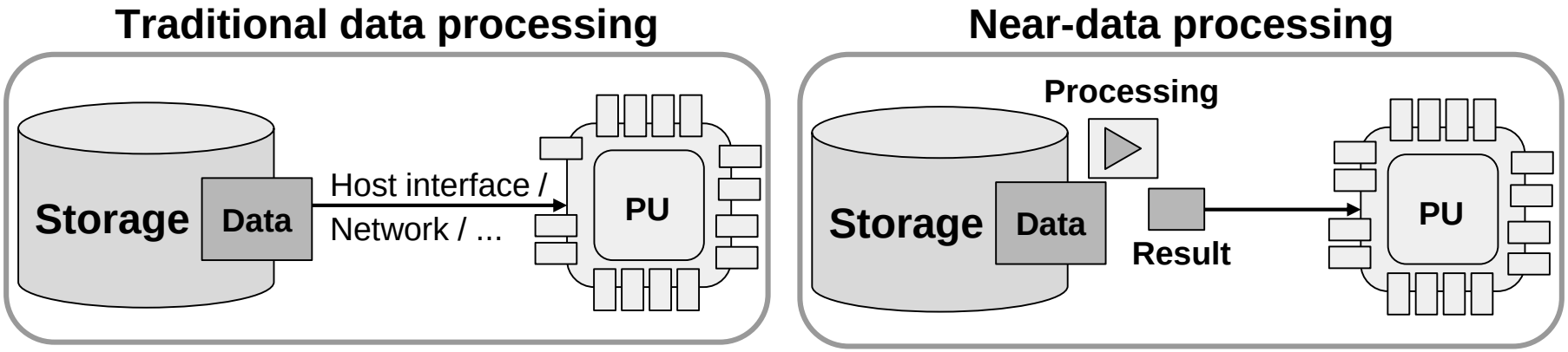
```
// Function call for reading data
for (Text value : values) {
   String content = value.toString();

   String[] split = content.split("\\t");

   double pageRank = Double.parseDouble(split[0]);
   int totalLinks = Integer.parseInt(split[1]);

   sumPageRanks += (pageRank / totalLinks);
}
double newRank = PageRank.DAMPING * sumPageRanks +
      (1 - PageRank.DAMPING);
// Function call for writing result ranks
```



- **Data communication between storage and CPU is enlarged**
  - e.g., PageRank on Hadoop MapReduce[1)] (24.6% by sync I/O, 20.3% by async I/O)

1) Daniele Pantaleone, Hadoop PageRank, “https://github.com/danielepantaleone/hadoop-pagerank”

Background: What is Near-Data Processing (NDP)?



- **Avoiding the need for costly chip-to-chip transfers, thus yielding massive parallel, high-performance, energy-efficient processing[1)]**

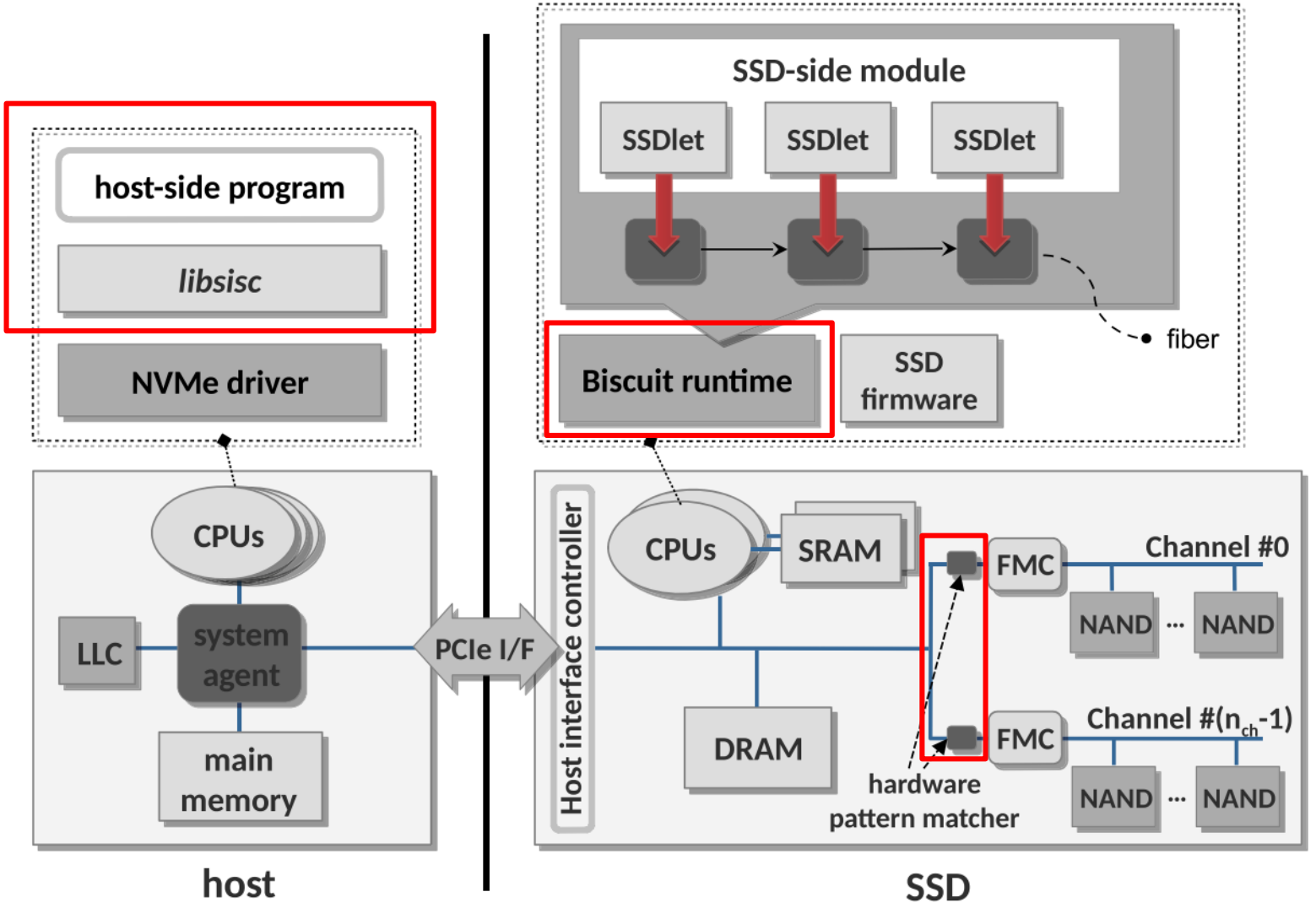
- **Various form of near-data processing system**
  - Processing-in-memory (PIM), in-storage processing (ISP), FPGA-accelerated SSD processing, ...

1) Rajeev Balasubramonian and Boris Grot, Near-data Processing, IEEE Micro, 2016

Background: Key Ingredients for NDP

- **Ability to run user-written code on a device**
- **Efficient communication between host and storage-side tasks**
- **Efficient resource utilization in runtime**
- **Intuitive, high-level programming**
- **Safety**

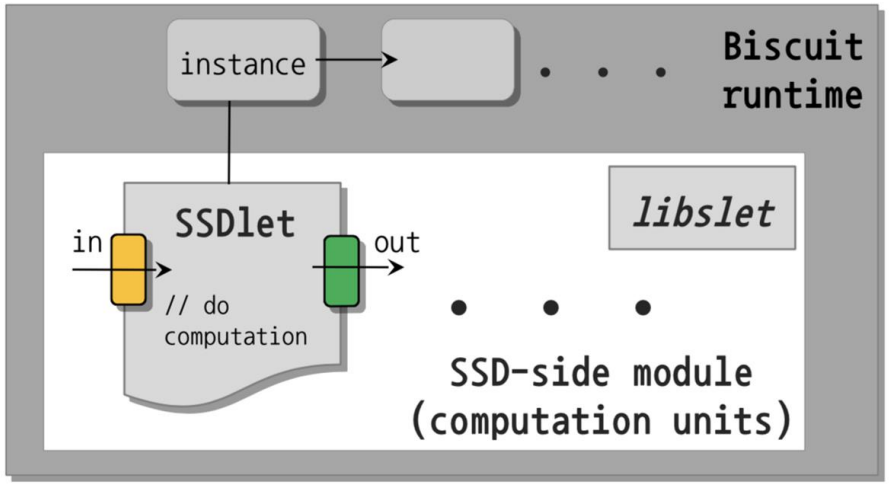
Biscuit: System Overview



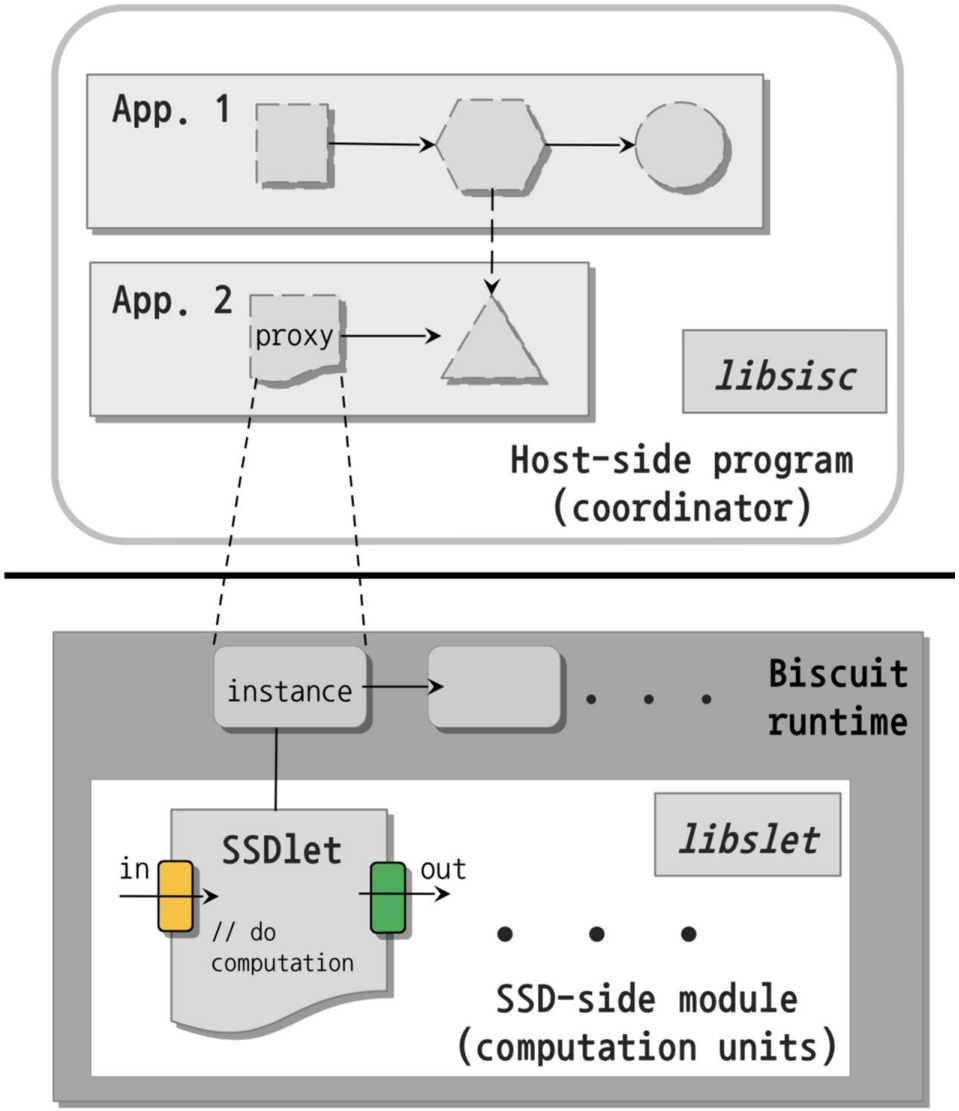
Biscuit: Programming Model

Computation model

- Specifying computation using its input and output data
- *Libslet* to define SSD-side tasks and perform file I/O



Biscuit: Programming Model



Computation model

- Specifying computation using its input and output data
- `Libslet` to define SSD-side tasks and perform file I/O

Coordination model

- Creating and managing tasks
- Establishing producer/consumer relationship
- `Libsisc` to invoke and coordinate execution of SSD-side tasks

Biscuit: SSDlet

- **A simple C++ program written with Biscuit APIs**
- **A unit of execution independently scheduled, represented by the SSDlet class**

```
class UserTask
    : public SSDLet <IN_TYPE<int32_t>,
                             OUT_TYPE<int32_t>,
                             ARG_TYPE<File>> {
public:
       void run() override {
              auto in = getInputPort<0>();
              auto out = getOutputPort<0>();
              auto& file = getArgument<0>();
              FileStream fs(std::move(file));

              // do some computation
}}
```

Biscuit: Wordcount Example (1)

- **Wordcount**
  - Count the frequency of each word in a given input file

Host-side Task

```
int main (int argc, char* *argv[]) {
  SSD ssd("/dev/nvme0n1");
  auto mid = ssd.loadModule(...);
  ...
  SSDLet mapper1(...);
  ...
  wc.connect(mapper1.out(0), shuf
  ...
  wc.start();
  wc.wait();
  ssd.unloadModule(mid);
  return 0;
}
```

SSD-side Task

```
class Mapper: public SSDLet …

public:
  void run () {
                    tArgument<0>();


    line.tokenize();
    …
    while (...) {
      if (!output.put(std::string(word), 1)) return;
}}}};
```

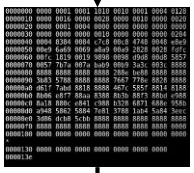
- specify the target device
- load module on SSD
- create app instance & proxy SSDLet instances
- establish connections

(1) Write the codes

- take a file
- pass tokenized words to Shuffler

(2) X86 compile

(3) ARM cross-compile

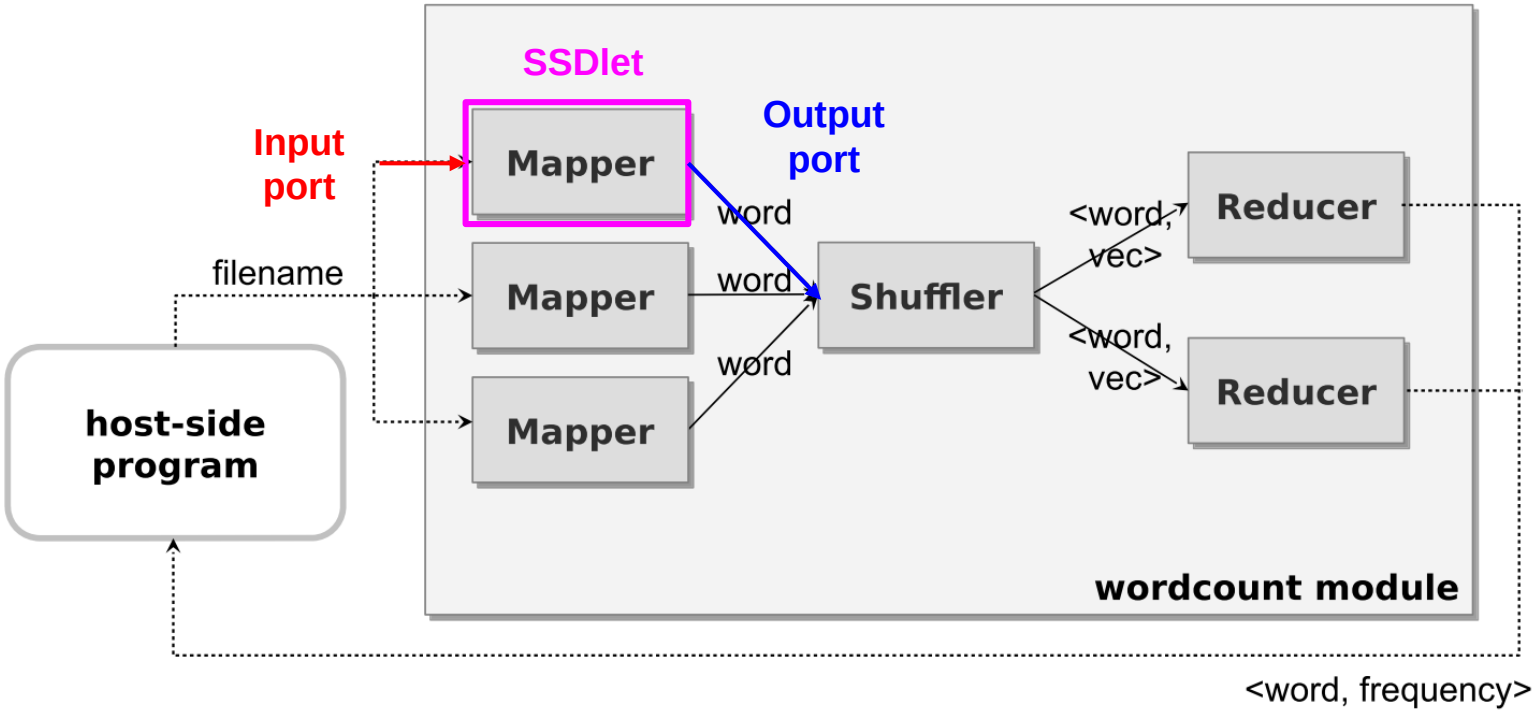


(5) Run the host program

(4) Copy the module into /var/isc/slets (/dev/nvme0n1)

Biscuit: Wordcount Example (2)

- **SSDLet**
  - Mapper: tokenization
  - Shuffler: data partitioning and transformation, merge
  - Reducer: summary



Biscuit: Implementation

- **Multithreading support - Cooperative Multithreading**
  - Low context switching overhead: explicit yield calls or blocking I/O function calls
  - Multi-core support: a unit of multi-core scheduling (= application)

- **Efficient data communication - I/O Ports as Bounded Queues**
  - Sending/receiving data = enqueue/dequeue operation
  - Channel manager

- **Dynamic module loading**
  - Function table: a collection of functions to perform task
  - Separate address space for each SSDlet instance

- **Dynamic memory allocation**
  - System memory allocator, user memory allocator

Evaluation: Methodology

- **Hardware setup**

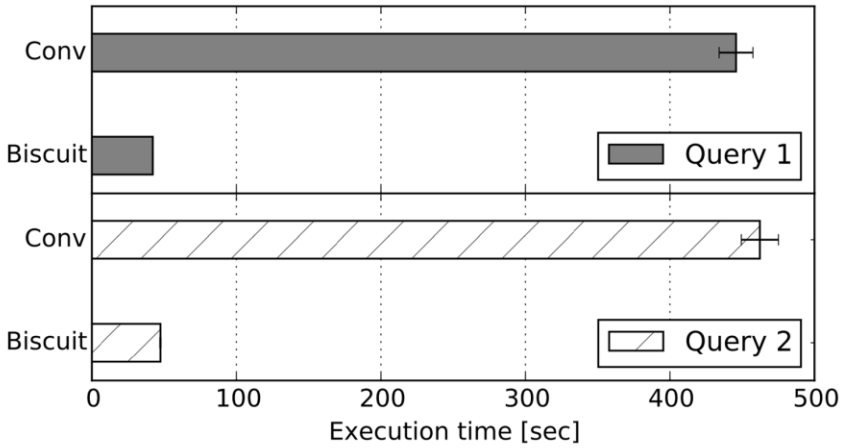
System	Dell PowerEdge R720 server
CPU	2 Intel Xeon(R) CPU E5-2640 (12 threads/socket) @ 2.50 GHz
Memory	64 GB DRAM

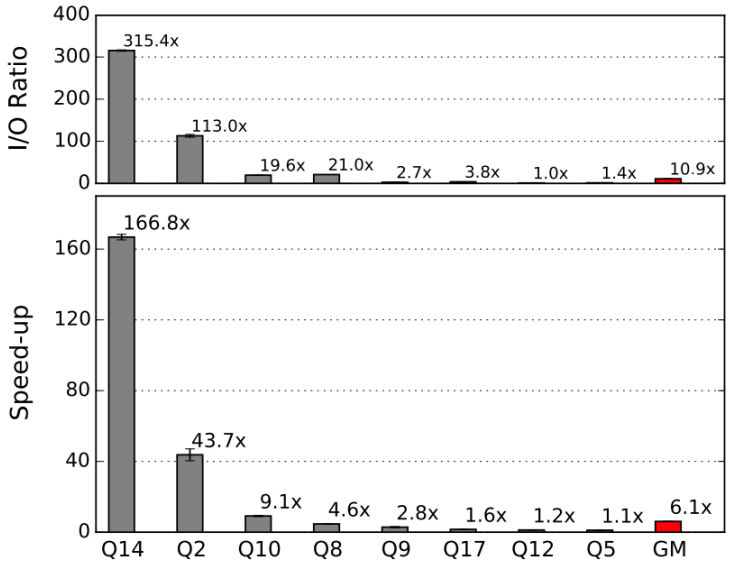
Item	Description
Host interface	PCIe Gen.3 ×4 (3.2GB/s max. throughput)
Protocol	NVMe 1.1
Device density	1 TB
SSD architecture	Multiple channels/ways/cores
Storage medium	Multi-bit NAND flash memory
Compute resources for Biscuit	Two ARM Cortex R7 cores @750MHz with L1 cache, no cache coherence
Hardware IP	Key-based pattern matcher per channel

- **Basic performance results**
  - Data read latency: 18% shorter latency (biscuit < conv)
  - Data read bandwidth: 1 GB/s larger bandwidth (biscuit >> conv)

- **Application level workloads**
  - Pointer chasing, string search, **DB scan/filtering, TPC-H (on MariaDB)**

Evaluation: Application-level

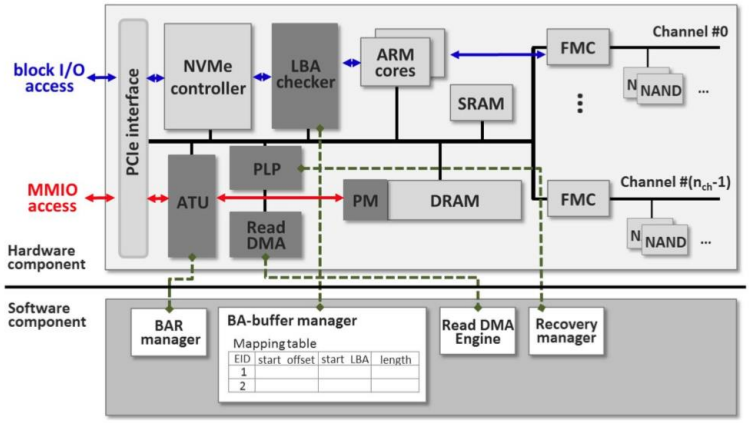




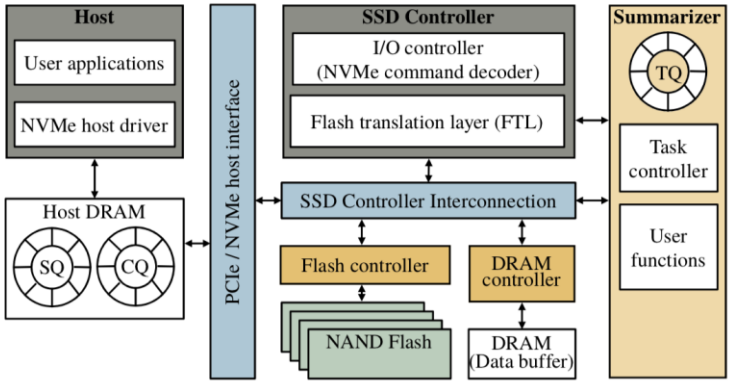
- **DB scan/filtering**
  - Simple (Q1) / complex (Q2) WHERE clause (filter condition)
  - Biscuit achieves speed-ups of about 11x (Q1) and 10x (Q2)
- **TPC-H queries**
  - The figure shows speed-ups correlated with the I/O reduction ratios shown together
  - Biscuit reduces the number of intermediate row sets that must be read from the SSD

Related Work

- **2B-SSD[1)]**
  - Byte- and block-addressable SSD
  - Achieves DRAM-like write latency on SSD



- **Summarizer[2)]**
  - Offload a data intensive task to the SSD processor (similar motivation as Biscuit)
  - SQL filtering/scanning acceleration by task queue/controller and user function stack

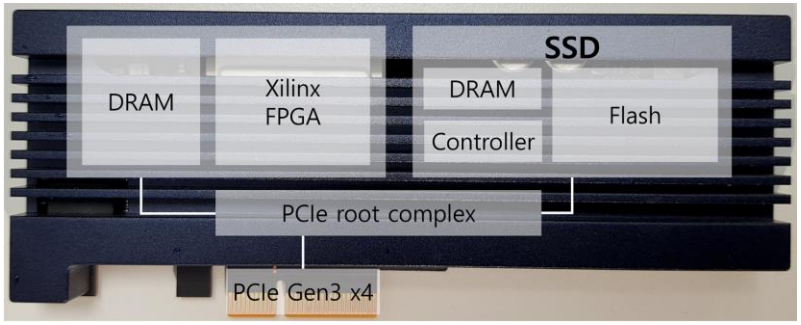


1) D. Bae *et al*., "2B-SSD: The Case for Dual, Byte- and Block-Addressable Solid-State Drives," *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018.

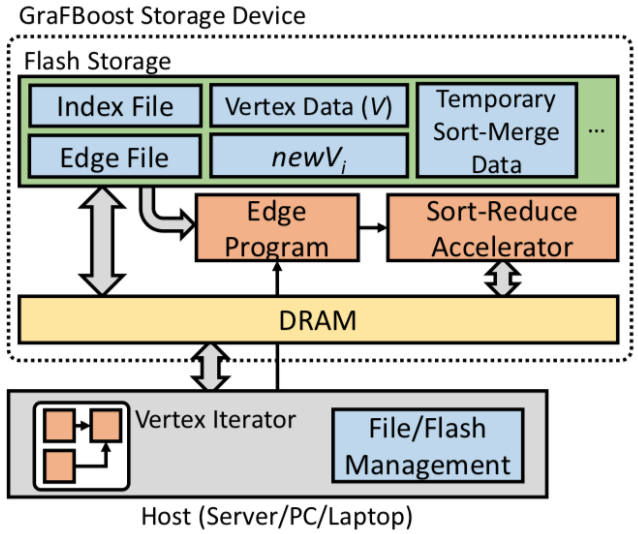
2) Gunjae Koo et al., "Summarizer: trading communication with computing near storage," *50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017.

Related Work

- **SmartSSD[1)]**
  - Xilinx-based SSD: high-performance accelerated computing closer to the data



- **GraFBoost[2)]**
  - Flash-based hardware acceleration for multi-terabyte graphs
  - Sort-reduce method of vertex updates



1) Samsung Tech Day 2018, "https://techday.samsungatfirst.com."

2) Sang-Woo Jun et al., "GraFBoost: Using Accelerated Flash Storage for External Graph Analytics," *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018.

Conclusion

- **Ability to run user-written code on a device**
  - Supporting C++11 features and standard libraries
- **Efficient communication between host and storage-side tasks**
  - I/O ports as bounded queues
- **Efficient resource utilization in runtime**
  - Hardware pattern matcher and lightweight multithreading
- **Intuitive, high-level programming**
  - SSDlet based on flow-based programming model
- **Safety**
  - Biscuit prohibits SSDlets from directly using low-level, logical block addresses