A New Computational Storage Devices Architecture to optimize HPC applications

Wadjom Eunice

To cite this version:
Wadjom Eunice. A New Computational Storage Devices Architecture to optimize HPC applications. 2021. hal-03494689

HAL Id: hal-03494689
https://hal.science/hal-03494689
Preprint submitted on 19 Dec 2021
A New Computational Storage Devices Architecture to Optimize HPC Applications

[Extended Abstract]

Wadjom Eunice
ISIMA, University of Clermont Auvergne
Clermont-Ferrand, France
eunice_guedaliah.wadjom_kammegne@etu.uca.fr

In this extended abstract, I will synthetize the work of Torabzadehkashi et al. (2019a).

I. INTRODUCTION

High-Performance Computing (HPC) consists of combining the power of several thousand processors to perform complex calculations and massive data processing in real time. It was adopted in 1960 and represents a good compromise between cost and computer power. But, the problem with HPC applications are bottlenecks during communication (Awerbuch et al. 1994) which can be resolve by improving interconnection systems.

In fact, 30% of the power consumption in demanding HPC applications is due to the high bandwidth interconnect system used to transfer data (Cicotti et al. 2016). Besides that, the storage system has a great influence on their performance when handling with large amounts of data.

Nowadays, data centers are favoring the use of Solid-State Drives (SSDs) because they are considered more efficient and higher performing than Hard Disk Drives (HDDs). For example, Figure 1 (Torabzadehkashi et al. 2019b) shows a large server storage system with 64 SSDs. Each SSD has about 8.5 GB/s internal data bandwidth, so it gives a total of 544 GB/s internal data bandwidth for all SSDs, which is far more than the one of the host machine (16 GB/s). This can cause a loss of performance for applications that use large amounts of data; a problem that can be mitigated by computational storages by bringing treatments to data.

The problem here is then to design a solution based on Computational Storage Devices (CSDs) to optimize the data movement cost in order to reduce energy consumption and enhance the efficiency of HPC applications. Several researches have been done on how to process user tasks directly in the storage device and hence the transformation of SSDs to CSDs.

This extended abstract will first focus on the study of the Catalina CSD implemented by Torabzadehkashi et al. (2019a) to show that it is able to optimize the power and energy consumption of HPC applications, by avoiding data movement. Then, on the study its architecture in detail and on their implementation of a server with six Catalina CSDs to run HPC applications, in order to show the efficiency of their solution.

II. PRIOR RELATED WORK

CSDs allow to optimize the communication speed as they bring the processing and the data closer together, and therefore eliminate the data transfer cost (Torabzadehkashi et al. 2018). There are three main CSD architectures which use three different tools to process user application: existing cores, Field Programmable Gate Arrays (FPGA) and a dedicated processing engine.

Today's SSDs have powerful real-time processing cores that can be used to run user data but not efficiently. Hence the development of Biscuit (Gu et al. 2016), to run applications on both the storage device and the host machine.

To improve performance, some computational tasks must be offloaded to the SSD. But, it is not known what effect a host Input/Output (I/O) request and user data execution in the storage device at the same time has on system performance. Furthermore, a flow-based tool to dynamically transfer tasks to the Biscuit's embedded processing unit had been proposed, but it is time-consuming.

FPGA is efficient for parallelism, it's used in BlueDBM and makes it 10x faster. The matter is that, to alter a FPGA, you have to reconfigure the Register Transfer Language (RTL) and regenerate the bitstream (Rezaei, Kim, and Bozorgzadeh 2018). Moreover, FPGA is not adapted for sequential tasks. In this paper, Torabzadehkashi et al. (2019a), prove that processing...
engines deployed on Application-Specific Integrated Circuits (ASIC) achieve better performance of HPC applications than FPGA-based speeders.

The other type of CSD architecture is very flexible because it uses a dedicated multi-purpose system to process tasks in-place, but it is not really performant due to embedded processors in storage devices. Implementing CSDs requires hard work that must be rewarded by optimizing performance and energy consumption.

The CSD architecture proposed by Torabzadehkashi et al. (2019a) in this paper to maximize performance, combines both the flexibility of general-purpose processors and the power of hard Intellectual Property (IP) cores like NEON Single Instruction Multiple Data (SIMD) to run HPC applications.

III. CATALINA ARCHITECTURE

In this section, Torabzadehkashi et al. (2019a) present the elements that allow Catalina to run HPC applications in-place.

A. Hardware Architecture

The Catalina architecture as shown in Figure 2 has a CSD controller which is divided in two parts: Programmable Logic (PL) and Processing System (PS).

The PS contains the dedicated processor ARM Cortex-A53 for the execution of user applications in-place, and two ARM Cortex-R5 real-time processing units to manage the Flash Translation Layer (FTL) and the Nonvolatile Memory express (NVMe)/PCIe protocol to talk to the host.

The PL block is in fact an FPGA and implements usual SSD functions such as communication with the host via NVMe protocol, communication with remote flash memory components via the flash memory control unit and error handling via the Error Correcting Code (ECC) unit.

The strength of Catalina is the power of the quad-core ARM Cortex-A53 processor dedicated to run user applications in-place: it has direct access to FTL routines and flash data which significantly reduces power consumption. Moreover, it optimizes performance thanks to the NEON SIMD units. ARM Cortex-A53 with its embedded Linux operating system, uses the Transfer Control Protocol / Internet Protocol (TCP/IP) tunnel built with the NVMe protocol to receive user requests and return the results to the host via the PCIe link.

B. Software Stack

By combining a host machine with several Catalina CSDs, it is possible to perform parallel programming via tools like Message Passing Interface (MPI) tool. Since CSDs improve the processing speed of the host, the software stack shown in Figure 3 is intended to increase the processing power of the CSDs relative to the server processing unit.

As you can see in Figure 3, the MPI coordinator manages multiple MPI workers on both the host machine and the Catalina CSDs. While the workers running on the host have access to the data on the Catalina CSDs via the NVMe protocol, the workers running on the Catalina CSDs only have access to their own data. The block device driver was designed to optimize access to flash memory, it sends commands directly to the Cortex-R5 real-time processing unit.

C. Catalina Prototype

Figure 4 shows a Catalina prototype built by Torabzadehkashi et al. (2019a) using Xilinx Zynq Ultrascale plus MPSoC (Boppana et al. 2015) and its specifications are given in TABLE I.

![Figure 2. Catalina architecture](image)

![Figure 3. Catalina software stack](image)

![Figure 4. Catalina Prototype](image)

<table>
<thead>
<tr>
<th>TABLE I. CATALINA HARDWARE SPECIFICATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processor</strong></td>
</tr>
<tr>
<td><strong>DRAM memory Size</strong></td>
</tr>
<tr>
<td><strong>Cache size</strong></td>
</tr>
<tr>
<td><strong>Other processing engines</strong></td>
</tr>
</tbody>
</table>
### TABLE II. DEVELOPED SERVER SPECIFICATIONS

<table>
<thead>
<tr>
<th>CPU</th>
<th>Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz (8 cores - 16 threads)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM</td>
<td>DDR4 32GB</td>
</tr>
<tr>
<td>Storage</td>
<td>Six Catalina CSDs connected via PCIe</td>
</tr>
<tr>
<td>Operating system</td>
<td>Ubuntu 18.04 LTS</td>
</tr>
</tbody>
</table>

### IV. EXPERIMENTAL RESULTS

Here, Torabzadehkashi et al. (2019a) present the server they built to run experiments and the result of the comparative study between their CSD architecture and the common one.

#### A. Experimental Setup

TABLE II shows the specifications of the server built by Torabzadehkashi et al. (2019a) to compare their CSD architecture to the usual one, and Figure 5 is an illustration of that server. To perform the distributed execution of the benchmark on MPI, they used a TCP/IP based network.

#### B. Benchmarking and Dataset

Among all benchmarks, Fast Fourier Transform (FFT) is very interesting, as it is an important numerical algorithm (Sabharwal et al. 2008). Its discrete version Discrete Fourier Transform (DFT), is obtained from the following equations:

\[ Y_k = \sum_{n=0}^{N-1} X_n e^{\frac{2\pi i kn}{N}} \]  

and:

\[ y_{k_1,\ldots,k_d} = \sum_{n_1=0}^{N_1-1} \left( \sum_{n_2=0}^{N_2-1} \left( \sum_{n_d=0}^{N_d-1} \alpha_{n_1}^{n_1k_1} \alpha_{n_2}^{n_2k_2} \ldots \alpha_{n_d}^{n_dk_d} \cdot x_{n_1,n_2,\ldots,n_d} \right) \right) \]  

\[ \alpha_{n_l} = \exp(-2\pi/N_l) \]

X: finite sequence of samples of a signal.

Y: result of the DFT function.

To execute this algorithm, Torabzadehkashi et al. (2019a) used the multi-threaded FFT in the West (FFTW) library which implements the SIMD ARM NEON engines, parallel processing and the MPI tool.

The DFT functions executed on the built server use 1D, 2D and 3D datasets whose specifications are given in TABLE III.

#### C. Result

In this sub-section, Torabzadehkashi et al. (2019a) present the results of the DFT algorithms on the different datasets with various numbers of Catalina CSDs, in order to better evaluate the impact of CSDs on energy consumption and performance. The performance is the number of DFT operations per second and the energy was measured with a power meter.

Figure 6 shows the performance obtained by applying the FFT algorithm on each type of data and varying each time the number of Catalina CSDs used. It can be seen that the performance of the system increases with the number of CSDs and has doubled with the use of six CSDs to perform the DFT algorithm.

Through Figure 7, you can see that the energy consumption decreases considerably when more Catalina CSDs are used and it was reduced by more than half when using six CSDs.

### TABLE III. DATASET SPECIFICATIONS

<table>
<thead>
<tr>
<th>Description</th>
<th>Dimensions</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 Million heart Electocardiography (ECG) signals</td>
<td>N = 180</td>
<td>288 GB</td>
</tr>
<tr>
<td>14.4 Million synthetic 2D matrices</td>
<td>N_1 = 50</td>
<td>288 GB</td>
</tr>
<tr>
<td>288,000 synthetic 3D matrices</td>
<td>N_1 = 50</td>
<td>288 GB</td>
</tr>
</tbody>
</table>
V. CONCLUSION

The work of Torabzadehkashi et al. (2019a) in this paper aims at optimizing HPC applications with Computational Storage Devices. To achieve this goal, they first presented the Catalina CSD architecture with its strengths, and then proposed an architecture with six Catalina CSDs that successfully executed DFT computations by doubling the performance and reducing the energy consumption by 54% compared to the conventional execution by the host machine.

Their proposed architecture is the proof that the use of CSDs allows to accelerate HPC applications while reducing their energy consumption. Nevertheless, this architecture can still be improved by replacing the Catalina CSDs with ASIC-based CSDs which are intended to be even more efficient and with lower power consumption.

REFERENCES


