NETRONOME

WHITE PAPER

Efficient Data Movement in Modern SoC Designs – Why It Matters

COPROCESSORS OFFLOAD AND ACCELERATE SPECIFIC WORKLOADS, HOWEVER DATA MOVEMENT EFFICIENCY ACROSS THE PROCESSING CORES AND MEMORY IN SUCH SOC DESIGNS IS BECOMING PARAMOUNT.

CONTENTS

INTRODUCTION 1

EXAMPLE: NETWORKING AND STORAGE COPROCESSORS 2

EXAMPLE: NETWORKING AND INFERENCING COPROCESSORS 4

INTRODUCTION

The death of Moore's Law, coupled with increasing data traffic and deployment of processing-intensive workloads such as SDN, security and machine learning, has driven the rapid growth of the heterogeneous processing market. Application-specific offloads using coprocessors – implemented as a system on a chip (SoC) - deliver significantly better silicon utilization and overall productivity in servers. Coprocessors offload and accelerate specific workloads, however, data movement efficiency across the processing cores and memory in such SoC designs is becoming paramount. As a result, compute architectures in data center core and edge applications are evolving.

While considering data movement inside a traditional server, the assumption is that all processing will be carried out in the general-purpose host processor. In most cases, the host processor comprises of x86 CPU cores. Memory and storage accesses needed in the execution of compute tasks are carried out by the host processor. The networking silicon in the network interface card (NIC) installed in the server feeds data in and out of the server. All data is meant to go to a single destination – the host processor. In this environment, on the ingress path, data enters the server through the NIC port and traverses the PCIe bus into the host memory where the host processor can perform work on the data. As required by the application running in the host processor, the data is prepared and sent to the storage device (such as an SSD) available on the server. This is a very simplified scenario, as shown in Figure 1, but serves the purpose of this example. On the egress path, processed data is fed back out by the host CPU over the PCIe bus and back to the NIC, which then sends the data out to the network.

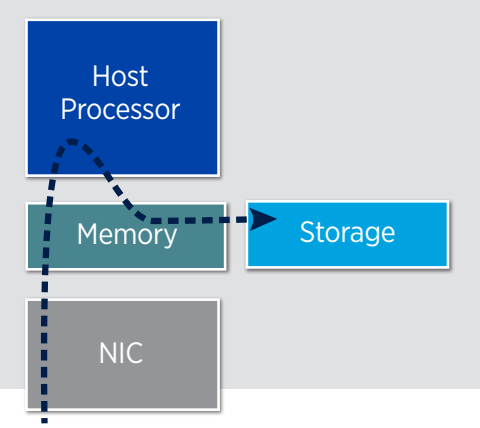

Figure 1. Simple data movement scenario on the ingress path in a traditional server

In modern servers built using a heterogeneous processing architecture, multiple processing elements exist. Besides the general-purpose host processor, the server may contain one or more of the following coprocessors - a network coprocessor, a security coprocessor, a graphics coprocessor, a storage workload coprocessor or a machine learning workload coprocessor. If data movement is conducted with similar assumptions as in traditional servers (i.e., most processing required and most memory and storage access functions needed in the execution of the compute tasks are carried out by the host processor), then significant resource utilization inefficiencies will be incurred, bringing down the productivity of the server. In fact, most of the benefits of heterogeneous processing could be lost as a result of poor data movement efficiency across the processing elements. The following describes some specific examples.

MOST OF THE BENEFITS OF HETEROGENEOUS PROCESSING COULD BE LOST AS A RESULT OF POOR DATA MOVEMENT EFFICIENCY ACROSS THE PROCESSING ELEMENTS

EXAMPLE: NETWORKING AND STORAGE COPROCESSORS

This example illustrates a heterogeneous server that includes co-processing for local storage access acceleration. The storage coprocessor could implement functions such as RAID, erasure coding, deduplication, encryption and compression. Data destined for a storage device (such as an SSD) on the server enters the server through the NIC. On the ingress path, the NIC sends the data to the host processor which then solicits the help of the storage coprocessor to accelerate the storage functions listed above. After this operation, once again facilitated by the host processor, the data is finally written to the storage device (Figure 2). The same steps in the reverse or egress direction are performed for a read operation when data is fetched from the storage device and then sent out of the server via the NIC port. Even though the storage coprocessor exists in the system, the host processor is still needed for many storage-related operations. The cycles spent for such processing become unavailable to revenue-generating applications, impacting the productivity of the server.

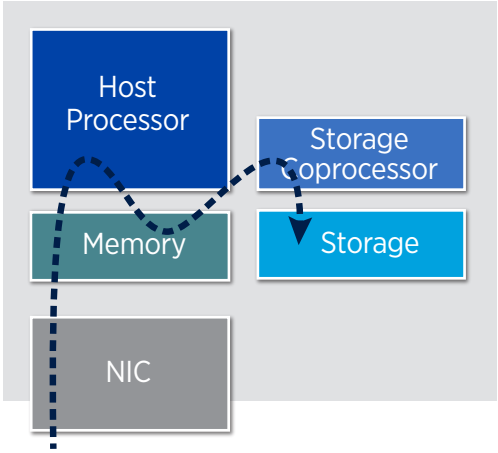

Figure 2. Host processor involved in storage processing functions on the ingress path

In an efficient data movement scenario applied in Figure 2, the NIC is replaced by an intelligent network coprocessor that acts as an efficient director of data to and from the storage coprocessor, bypassing the host processor. On the ingress path, data destined for a storage device on the server enters the server through the intelligent network coprocessor on a SmartNIC. Data in the first flow is sent to the host processor; once the flow is learned by the network coprocessor, it is able to send data in all subsequent known flows directly to the storage coprocessor. Once the storage acceleration functions are executed, the data can be directed to the storage device by the intelligent network coprocessor. The path to the storage device from the intelligent network coprocessor can be via PCIe, a network port or via the external memory interface. In a more cost-effective and streamlined implementation, the storage coprocessor functions may be implemented along with the network coprocessor functions in a single chip. In this approach, the host processor is freed up from executing storage-related operations, enabling the use of more precious cycles for revenue-generating applications, and improving the productivity of the server. The processing scenarios presented in this paragraph are depicted in Figures 3(a), (b) and (c).

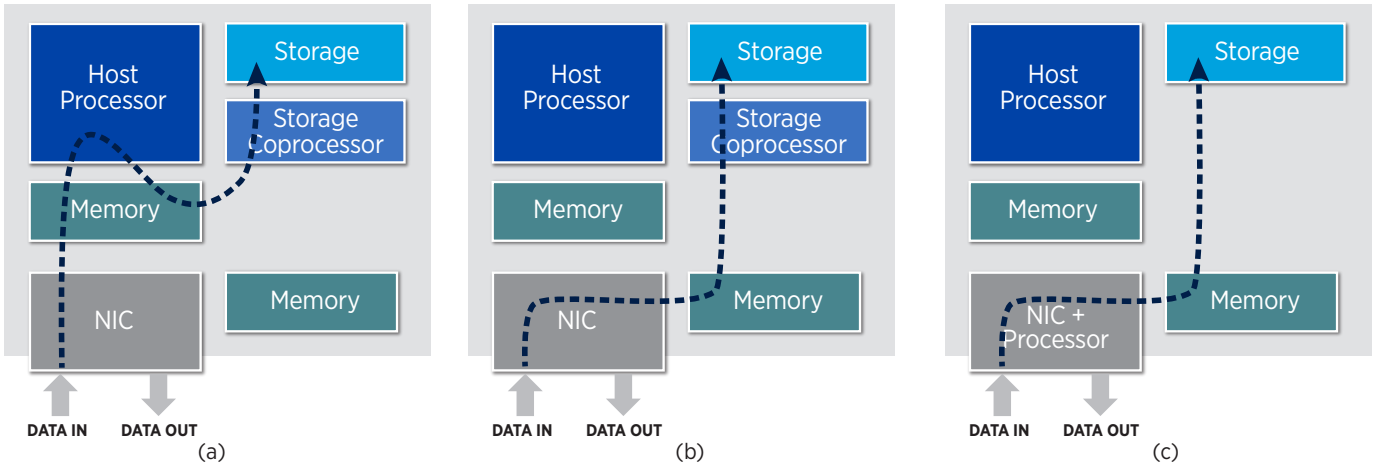

Figure 3. (a) First flow of data is processed by the host; (b) All subsequent known flows are cut-through by the network coprocessor, freeing up the Host Processor; (c) Example showing network and storage coprocessor functions on a single chip with streamlined data movement to storage device via the external memory interface

EXAMPLE: NETWORKING AND INFERENCING COPROCESSORS

The second example relates to coprocessing for an inferencing engine. The inferencing engine coprocessor has high-speed access to memory such as DDR4 or HBM that works in conjunction with a local and unified memory architecture. The coprocessor also implements compute functions such as a Matrix Multiply Unit and an Activation Pipeline, which perform the nonlinear function of the artificial neuron, with options for rectified linear unit, Sigmoid, etc. The NIC sends the input sensor data to the host processor, which then solicits the help of the inferencing engine coprocessor to process the sensor data and accelerate the inferencing functions listed above. The output actor data from the inferencing engine coprocessor is sent back to the host processor for further processing or sent to another processing node via the NIC. As is obvious, the host processor is involved in getting all very high volumes of sensor data flows in and actor data out of the inferencing engine coprocessor. The cycles spent for such processing become unavailable to revenue-generating applications, impacting the productivity of the server.

In an efficient data movement scenario applied to this example, the NIC is replaced by an intelligent network coprocessor that acts as an efficient director of data to and from the inferencing engine coprocessor, bypassing the host processor. Sensor data destined for the inferencing engine coprocessor on the server enters the server through the intelligent network coprocessor on a SmartNIC. Sensor data in the first flow is sent to the host processor; once the flow is learned by the network coprocessor, it is able to send sensor data in all subsequent known flows directly to the inferencing engine coprocessor. Once the inferencing and related acceleration functions are executed, the resulting actor data can be directed to the next processing node by the intelligent network coprocessor.

In a more cost-effective and streamlined implementation, the inferencing engine coprocessor functions may be implemented along with the network coprocessor functions in a single chip, to provide an inline inferencing function. In such a design, a common internal and external memory access mechanism can be implemented, with processing memory for functions such as hash lookup, atomic and bulk initialization and Matrix Multiply Unit operations. In this approach, the host processor is freed up from executing inferencing-related operations, enabling use of more precious cycles for revenue-generating applications, and improving the productivity of the server.

In the previous examples, the notion of combining the workload-specific coprocessors into a single chip design for higher levels of efficiency was discussed. Data movement efficiency in such heterogeneous architectures is critical. An inefficient design can expend significant on-chip area and power to provide memory coherence at low latency across all the programmable cores. The cost of consistency grows with the data rate, the target latency for consistency and the physical area over which consistency has to be achieved. In network processing and many other data-intensive applications, not all data processed by a system need be consistent at the same rate and with the same latency. Coherence for control data that specifies how packets are to be processed is more important than coherence for packet data. Data consistency or coherence requirements can be relaxed for a significant fraction of system data.

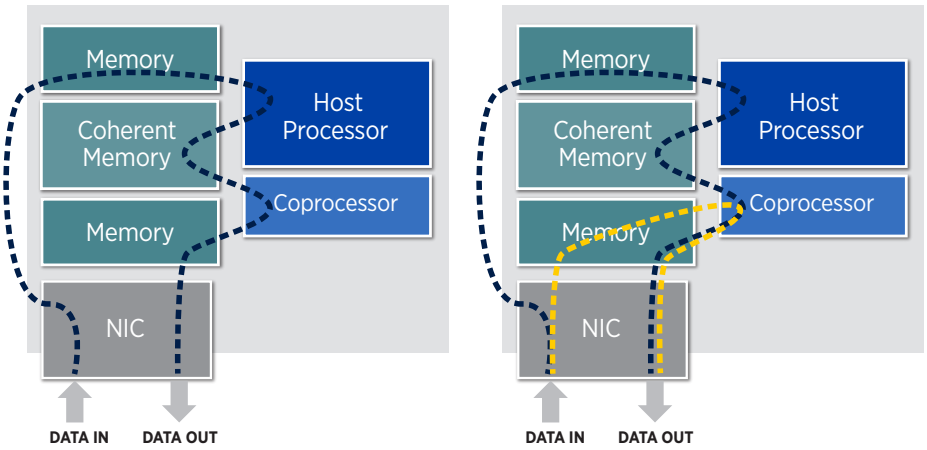

Figure 4. Example of cut-through processing with both coherent and non-coherent (or relaxed coherence) memory access

In heterogeneous processing architectures with efficient data movement, as highlighted in this document, many TCO level benefits can be obtained. For example, two Intel Xeon 8180 CPUs with a list price of $20K can be replaced with four Intel Xeon D-2173 for a list price of less than $5K. In addition to reducing latency of operations (such as memory access) and improving overall application performance, such efficient heterogeneous processing solutions enable the use of significantly less expensive host processors.

NETRONOME

Netronome Systems, Inc.
2903 Bunker Hill Lane, Suite 150 Santa Clara, CA 95054
Tel: 408.496.0022 | Fax: 408.586.0002
www.netronome.com