Accelerating arithmetic kernels with coherent attached FPGA coprocessors

Heiner Giefers, Raphael Polig, Christoph Hagleitner
IBM Research – Zurich
Email: {hgi,pol,hle}@zurich.ibm.com

***Abstract*—The energy efficiency of computer systems can be increased by migrating computational kernels that are known to under-utilize the CPU to an FPGA based coprocessor. In contrast to traditional I/O-based coprocessors that require explicit data movement, coherently attached accelerators can operate on the same virtual address space than the host CPU. A shared memory organization enables widely accepted programming models and helps to deploy energy efficient accelerators in general purpose computing systems. In this paper we study an FFT accelerator on FPGA attached via the Coherent Accelerator Processor Interface (CAPI) to a POWER8 processor. Our results show that the coherent attached accelerator outperforms device driver based approaches in terms of latency. Hardware acceleration delivers a 5× gain in energy efficiency compared to an optimized parallel software FFT running on a 12-core CPU and improves single thread performance by more than 2×. We conclude that the integration of CAPI into heterogeneous programming frameworks such as OpenCL will facilitate latency critical operations and will further enhance programmability of hybrid systems.**

I. Introduction

Gaining meaningful insights from big data requires deep analytics and cognitive computing tasks which are placing high demand on the processing resources. Commodity servers as used today by most organizations are built around conventional off-the-shelf processors and are getting more and more inadequate for emerging big data workloads. While in principle the massive amount of data-level parallelism accommodates scale-out architectures, the high static power consumption of large distributed systems has a negative impact on the energy costs. Resource consolidation across underutilized nodes in form of server virtualization and cloud computing is an effective method for saving power in a data center without compromising performance. However, this is only a one-time gain and also involves detrimental effects on the energy efficiency due to the resource management overhead and throughput reduction. A more progressive strategy to increase efficiency is the adaption of hardware resources to workload requirements by means of application-specific accelerators. Although it is generally accepted that accelerators can significantly improve energy efficiency, the use of hardware specialization in server environments has been declined in the past. Dedicated processing resources interfere with the virtualization concept and increase data canter management costs.

FPGA coprocessors enable specialization whilst providing flexibility through software reprogrammability and several studies have revealed the capabilities of reconfigurable hardware accelerators [11], [7], [17], [14]. However, the general acceptance of FPGA-based acceleration has been hampered by the lack of high-level programming models and standard hardware/software interfaces. Exploiting reconfigurable devices for compute acceleration is typically a complex and time-intensive hardware engineering task. In order to raise the abstraction level for FPGA development the major FPGA vendors recently released C-level programming frameworks that enable rapid development of FPGA based custom accelerators [4], [3]. Since driven by the chip vendors the quality of FPGA high-level synthesis (HLS) tools has made significant progress and the generated designs deliver a comparable or even better performance compared to manual RTL designs.

HLS tools particularly facilitate the generation of custom data-paths representing the core of the accelerator design. An even more challenging part, however, is the efficient design of a control and communication infrastructure. The standard use of I/O-attached coprocessor cards involves moving pinned pages to the on-card SDRAM with the help of device drivers in order to enable the functional units of the accelerator to access data at high bandwidth and low latency. For algorithms that expose a low arithmetic intensity, i.e., a low number of operations is applied per data item, this approach is sub-optimal because of the superfluous copy operations. Moreover, the handling of I/O interrupts and MMIO operations via the device driver offsets the possible speedup of the accelerator making it difficult to achieve a significant acceleration for the overall application.

The recently released IBM POWER8 processor includes a new Coherent Accelerator Processor Interface (CAPI), which provides cache-coherent access to DRAM for heterogeneous processors [15]. Using CAPI, peripheral devices can communicate directly with the CPU through shared memory and thus bypassing most of the operating system involvement and device driver overheads. The CAPI technology is made available through the OpenPower industry foundation and is now supported by Altera and Xilinx.

In this paper we present a CAPI-based hardware accelerator on FPGA for Fast Fourier Transformation (FFT). The design is evaluated against to a highly tuned FFT software library running on the POWER8 host system and an alternative FPGA-based hardware accelerator using the OpenCL framework. Our experimental results show that bypassing the device driver significantly reduces the communication and control overhead for a PCIe attached coprocessor. The FFT kernel on the FPGA outperforms a single-threaded software FFT by more than 2× and increases the energy efficiency of the system.

II. Kernel Acceleration Case Study: FFT

The discrete Fourier transform (DFT) is one the most widely used algorithms in signal processing and plays a key role

Acknowledgements: The authors would like to thank the IBM CAPI development team and Malcolm Ware for their help and support.

978-3-9815370-4-8/DATE15/©2015 EDAA

in many scientific and engineering applications. The fastest known methods for computing a DFT are referred to as fast Fourier transformations (FFTs). The most well-known FFT is the Cooley-Tukey algorithm [6] that uses a recursive divide & conquer approach to reduce the computational complexity of a DFT operation of N samples to $\mathcal{O}(N \log N)$. An overview of serial and parallel FFT algorithms is given in, e.g. [5].

DFTs are frequently applied in embedded systems as well as in applications from the HPC domain and thus, many highly optimized FFT implementations exist. Special purpose embedded systems use hardware implementations of the FFT to optimize for performance and energy efficiency. As hardware FFTs are often fixed to a specific data type or transformation size, they are normally not used in general purpose systems. Instead, optimized software implementations are applied. While the FFT method greatly reduces the operations required to compute a DFT, the all-to-all communication patters as required for the divide & conquer algorithm generally hampers performance. FPGA accelerators have the potential to bridge this performance gap, because they approach the efficiency of fixed hardware FFTs but are still fully programmable.

We study a one-dimensional FFT using complex single-precision floating-point data as an example for FPGA acceleration of a computational kernel. The applied co-processor is a Nallatech PCIe-385N card that employs an Altera Stratix-V FPGA device. The card comprises two independent DDR3 SDRAM banks with up to 4 GB of memory and two SFP+ interfaces to be used as 10 GbE ports. The PCIe-385N is supported by the Altera OpenCL framework as well as by the IBM Coherent Accelerator Processor Interface. The host is an IBM Power System S824 equipped with two 6-core POWER8 processor cards running at 4.2 GHz. The memory configuration consists of two modules each holding a memory hub chip (CDIMM) with 16 MB L4 cache and 64 GB of DDR3 memory allowing transfer rates of up to 192 GB/s per socket.

III. The Coherent Accelerator Processor Interface

In a shared-memory system multiple processing units communicate to a globally shared memory via the memory bus. POWER8 systems use a cache-coherent symmetric multiprocessor (SMP) design, in which all of the memory in the system is accessible to all of the processor cores. A POWER8 chip consists of up to twelve processor cores, each of which is equipped with L1 instruction and data-caches, a L2 cache, and a L3 cache, all of these caches are effectively shared.

The Coherent Accelerator Processor Interface (CAPI) allows for attaching co-processors or peripheral devices to a POWER8 systems in a cache coherent way [15]. CAPI removes the overhead and complexity of the I/O subsystem, allowing accelerators to operate as an integral part of a host application without the need for a device driver. The accelerator operates on the same virtual address space as the corresponding host process and has full access to it. It uses the processor's page table to handle address translations and page faults with the assistance of system software.

CAPI builds upon dedicated hardware support on the CPU and a service layer module that implements the endpoint on the coprocessor device. As depicted in Figure 1 the Coherent Attached Processor Proxy (CAPP) hooks into the snoop fabric on the POWER8 CPU and extends the coherency protocol

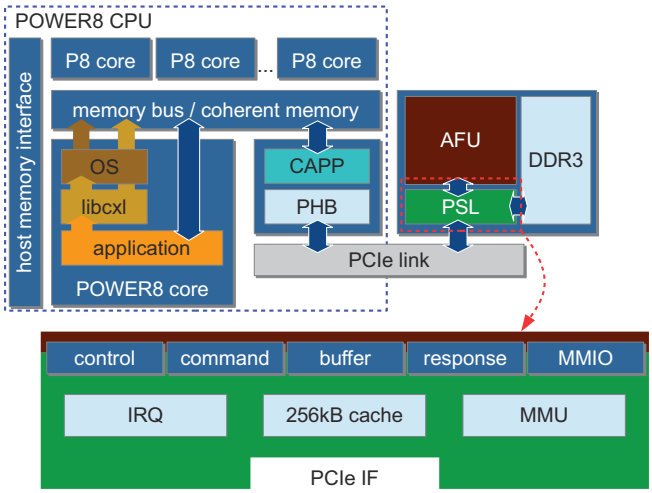


Fig. 1. CAPI Architecture and software stack

to the accelerator. The CAPP unit maintains a shadow cache directory on behalf of the accelerators and reacts on coherency messages without the need for going off-chip. Communication of data and control messages between the CPU and the accelerator is tunneled over a standard PCI-Express link. The PCIe Host Bridge (PHB) decodes CAPI protocol packets from the PSL and forwards them as memory bus data packets.

A CAPI accelerator configuration for the FPGA consists of the POWER Service Layer (PSL) and one or multiple Accelerator Function Units (AFUs). The PSL exposes a defined interface to the AFUs and provides services such as memory address translation and interrupt handling to the individual AFUs. Via the command interface, the AFUs issue memory operations very similar to a software thread running on a POWER8 core. Each read or write command is labeled with a tag number that can only be reused when the instruction is completed. The completion status of outstanding commands is indicated through the response interface. Data movement operations between the AFU and the PSL are handled via the buffer interface and carry 64 byte (a half of a cache line) per transaction. The actual data accesses are not strictly in order with the corresponding commands but can be identified using the tags. The MMIO interface can be used to read and write user-defined MMIO registers inside the AFU. Finally, the control interface is used to sense and control the state of the functional units. The flow of state transitions depends on the applied programming model of the AFU.

In order to cover different modes of operation, the CAPI architecture provides different programming models. In case of the *dedicated process programming model* the accelerator function is owned by a single process running on the host CPU. On behalf of a process, the hypervisor and the OS initialize the PSL and reset the AFU to an initial state that allows the user process to communicate with the accelerator via MMIO operations or through the shared address space. In the future, using the *PSL-controlled shared programming model* and *AFU-directed shared programming model* a single AFU can be shared among several processes and virtualized operating systems. To support a shared programming model, an AFU must either guarantee to complete in a specified amount of time or provide the ability to preempt the task that is

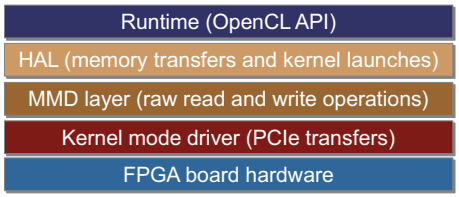


Fig. 2. OpenCL software stack (source: [3]).

currently running on the AFU hardware. For this, a mechanism to save and restore the context of the AFU is required. In case the PSL controls the sharing, a context switch is enforced by means of time-slicing. In the AFU-directed mode, the functional unit autonomously selects a process element out of a wait queue and must provide fairness between served processes.

When an application requests use of an shared AFU resource, a process element is added to the process-element linked list for the corresponding AFU type by the hypervisor. The process element also contains the work element descriptor (WED), an AFU-specific struct that is populated by the user application. A WED provides the AFU's function arguments and can contain pointers to shared memory or scalar parameters. The CAPI accelerator management library (libcxl) is a low-level management library that provides essential operations to the user.

IV. OpenCL Programming for FPGAs

While general C-to-gates compiler tools never archived widespread acceptance in the past, more comprehensive parallel programming frameworks like OpenCL ensure an appropriate degree of formalism to enable the high-level synthesis tools to generate efficient hardware but still let the programmer develop at a high level of abstraction. Recently, Altera and Xilinx released OpenCL software development kits (SDK) allowing FPGAs to be treated as OpenCL based accelerators [3], [16]. The Altera HLS compiler takes an OpenCL kernel and transforms it into a hardware description in Verilog. For a specific accelerator board, the compiler instantiates appropriate interface IP for the specific peripherals like the PCIe and the on-card SDRAM. The kernels of the OpenCL specification are getting transformed into custom processing pipelines implemented with the logic and memory resources in the reconfigurable fabric. The resulting pipeline-parallelism is inherently energy-efficient because intermediate results are directly passed to the next computing instance minimizing the costs for communication and caching, which contribute a large fraction of energy consumption in instruction-based processors. Run-time environments for OpenCL provide tools and device driver components that allow OpenCL programs to offload kernels to the targeted FPGA accelerator boards.

On our POWER8 host we use the 14.0 release of the Altera Runtime Environment for OpenCL which is available for x86, POWER (big endian and little endian), and ARM SoC platforms [2]. The OpenCL software stack involves multiple layers and is depicted in Figure 2. An OpenCL device is addressed from user programs via the platform independent OpenCL runtime library API. The hardware abstraction layer (HAL) provides platform dependent services such as device and kernel discovery, memory allocation, memory transfers, and kernel launch operations. Every FPGA board hardware requires a specific memory-mapped device (MMD) layer necessary for communication with the accelerator board. Finally, the communication is send over a PCIe link using a kernel mode driver. In particular, all communication between the host and the kernels goes through this layers and can cause a significant overhead.

The implementation of the OpenCL accelerator is based on an Altera design example [1] . The kernel uses radix-4 building blocks and is similar to the architecture described in [8]. As arguments to the OpenCL kernel, the user passes pointers to the source and target arrays of the FFT data and the number of FFTs to be computed in a row. For all benchmarks, we prepare data sets of more than 1 GB to avoid caching effects when invoking the kernels in an iterative fashion. Using the `clCreateBuffer` function from the OpenCL API, the programmer creates buffer objects and controls the allocation of memory for this buffers. When used with the `CL_MEM_ALLOC_HOST_PTR` no additional memory on the accelerator devices is allocated and the data is passed to (from) the kernel directly from (to) host memory. Normally, the input data set is copied to the accelerator memory before the kernel invocation in order to enable the accelerator to access data from local SDRAM at a high bandwidth. However, if the data items are not repeatedly accessed the copy operation can cause an overhead. In section VI we examine and compare both alternatives for the 1D FFT case study.

V. CAPI-based 1D FFT Accelerator

The FFT cores used by our CAPI accelerator were generated with the Spiral framework [13]. Spiral takes a problem specification as well as architectural parameters as input and generates an optimized data path for the given configuration. With the parameters, a user describes the algorithm's radix (basic block size), the number of input samples to the core and whether the pipeline is organized as an iterative or fully streamed architecture. Figure 3 shows a description of the Spiral-generated data path. Our accelerators make use of fully streamed architectures allowing to input a fixed amount of samples into the pipeleline in every clock cycle. For an N-point FFT, the architecture is constructed of $\mathcal{O}(\log N)$ cascaded stages, each of which consisting of a computation unit and a permutation unit. The computation unit implements a radix-2 butterfly scheme and the *twiddle factors* are stored in ROM units implemented in the embedded SRAM blocks on the FPGA. Local ROM look-ups avoid the need of streaming twiddle factors from the inputs throughout the pipeline.

The permutation of samples at the output of a processing stage is conducted with the help of RAM modules. Depending on the *stride* at a specific FFT stage, different size RAM modules get instantiated and controlled by appropriate address generators. For larger strides, deeper RAMs must be used to ensure that the data can be streamed in-order to the subsequent stage. The number of input samples to the FFT pipeline is chosen such that the bandwidth of the PSL interface can be fully utilized.

Figure 3 depicts how the FFT pipeline is embedded in a more general AFU pattern for streaming applications. The DMA wrapper can be configured to expose multiple put and get channels to host memory to the internal logic. Each channel handles the data read or write operations autonomously and incoming data is rearranged according to the order of requests.

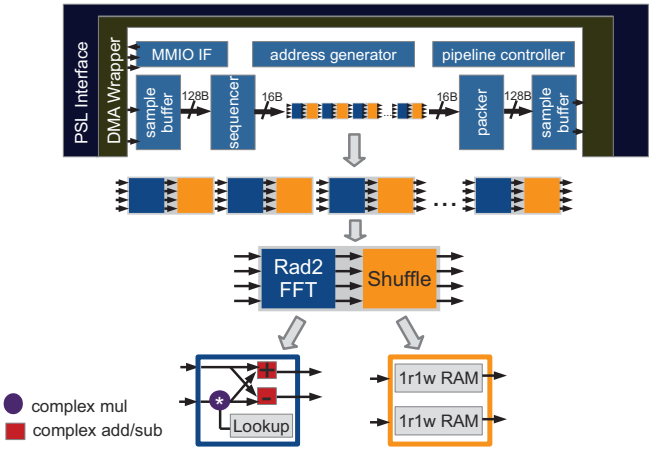


Fig. 3. FFT fully streamed processing pipeline with radix 4 based computation unit.

From the host, the user passes pointers to the source and target arrays of the FFT data as well as a scalar indicating the number of transformations in sequence. The parameters are wrapped into a work element descriptor struct (WED) and the address of the WED is send to the AFU by an MMIO operation. When the AFU recognizes the MMIO write of the WED pointer, it requests the WED data from host memory and uses the content to initialize address generators for read and write requests to the FFT data. The AFU keeps track of the consumed samples and labels the last write operation with an end flag. When the PSL acknowledges the write-back of the labeled data to DMA module, the AFU goes into an idle state and signals the termination to the host application. This can be done by raising a user-defined interrupt via the PSL command interface or by writing a status word to shared memory. Signaling and synchronization through the host memory does not require intervention by the OS or the hypervisor and thus reduces the latency of the accelerator functions at the cost of polling the status address from software.

VI. Comparison and Results

This section provides performance benchmarks and power measurements for 1D FFT kernels running in software on a POWER8 host system and on the FPGA coprocessor card.

Software In order to assess the performance of software based FFTs, we apply the FFT implementation from the IBM Engineering and Scientific Subroutine Library (ESSL) package [10]. Figure 4 presents performance results software FFTS. We vary the FFT size from 64 to 4M complex samples leading to input data sets of up to 32MB per FFT. For each problem size, we prepare batched data of 1 GB and compute an out-of-place FFT over the whole data array to avoid caching effects. We have studied multiple approaches to parallelize FFT computations in software and found that starting individual FFT threads for separate FFT inputs is generally preferable compared to running a multi-threaded FFT call on a single input. Figure 4 compares the performance of the ESSL FFT functions on POWER8 using 12 threads, each of which operating on different input data set. The results show an almost linear performance gain when moving from one to three threads and significant improvement for six and 12 threads. While the system is able to operate on up to 96 parallel threads in SMT mode, using more than 12 FFT threads

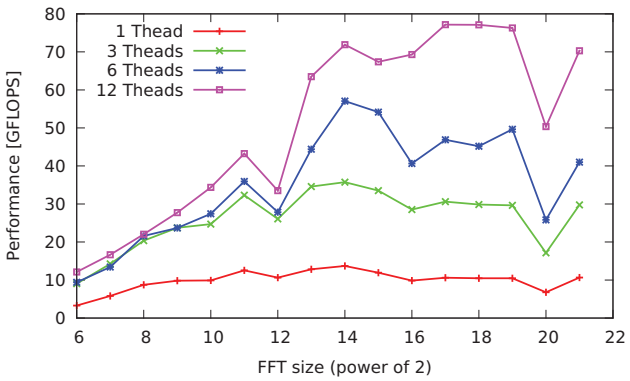


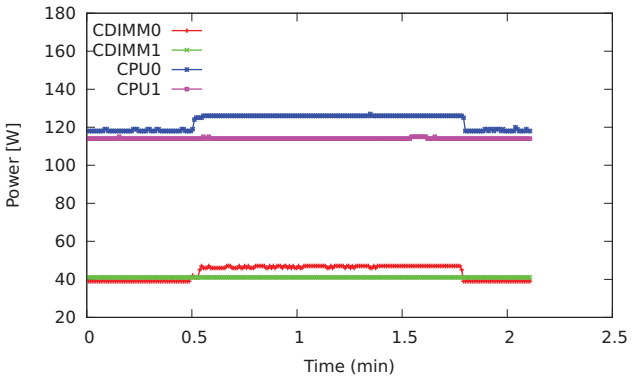
Fig. 4. Parallel FFTs on POWER8 using ESSL library

on the 12-core system only pays off for specific cases.

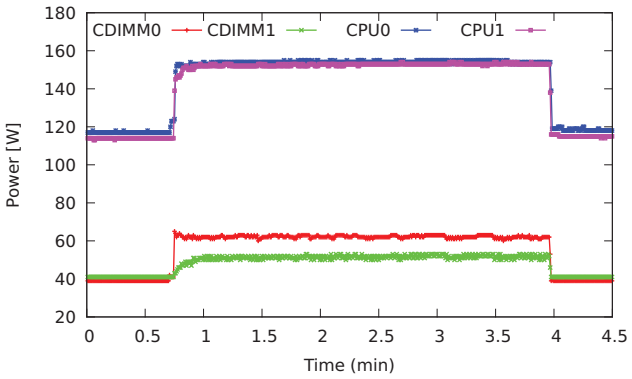
We use an IBM internal software tool to measure the power consumption of the POWER8 node. The tool is running on a remote computer and establishes a connection to a service processor on the measured system to record performance and power sensors out-of-band [9]. POWER servers comprise several current/voltage sensors allowing for measuring the power draw of individual domains, among them are the CPU sockets, memory DIMMs, cooling, and the I/O subsystem. Figure 5 presents power traces for software FFTs using the ESSL library. The power consumption of the system strongly depends on the applied CPU governor. When the cores constantly operate at their nominal clock frequency (performance governor), the idle power of the system is high and active workloads contribute only a relatively small dynamic power budget on top of the baseline power. Using a Dynamic Voltage Frequency Scaling (DVFS) governor, the power consumed in the CPU sockets is significantly lower at idle times and thus, a workload produces a higher dynamic power draw while the system-wide energy efficiency is typically superior.

The trace in 5(a) shows a sequential FFT using the ESSL library running on a 2-socket POWER8 server operating at the nominal frequency of 4.2 GHz. After an initial idle time of about 30 seconds, the FFT benchmark starts with the initialization of the FFT data which takes approximately 15 seconds. It can be seen that using a single core for the FFT computation, the power draw of the processors and memory modules is very uniform. For the experiment as depicted in Figure 5(b), the systems was running in DVFS mode and we used 12 FFT threads in parallel. Using additional cores allows better utilization of the memory bandwidth and leads to a higher power draw in CPUs and the memory modules.

OpenCL The OpenCL and CAPI accelerators are generated for specific FFT sizes and require a hardware synthesis step to generate an FPGA configurations file. We study both accelerators with FFTs using 1k and 4k input samples. In the host code of the OpenCL program we prepare an random input data set of 1 GB and call the OpenCL kernel on this input several times in a row. Afterwards, the test is repeated on a new input set but for the inverse FFT. When using a zero-copy mapping of device buffers the FFT data is instantly passed from the host to the FFT kernel implementation of the FPGA via the PCIe link. As shown by the results in Table I, the streaming of data from the host hampers performance so that the zero-copy version of the OpenCL program only achieves 3.7 and 4.4 GFLOPS for

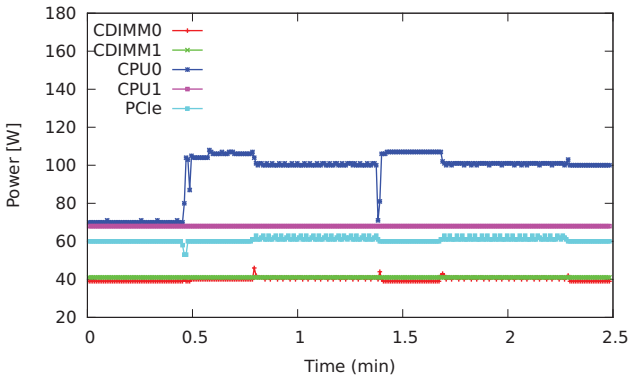


(a) Single-threaded FFT using ESSL. Performance mode.

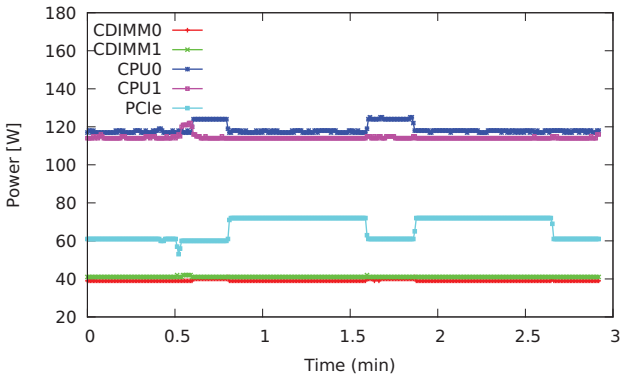


(b) Parallel ESSL FFT using 12 threads. DVFS mode

Fig. 5. Power traces for FFT threads on POWER8 captured out-of band with the Amester tool.



(a) Zero-copy OpenCL FFT. DVFS mode.



(b) OpenCL FFT using the accelerator memory. Performance mode

Fig. 6. Power traces for the FPGA OpenCL FFT on POWER8.

1k and 4k FFTs, respectively. When the kernel can read and write the data from local SDRAM the performance is much higher and 66.5 GFLOPS in case of a 1k FFT. If the application does allow to keep the data set in the accelerator memory the overhead caused by additional copy operations compromises performance.

Figure 6 show two power traces for the OpenCL FFT implementation. The PCIe power domain represents the the power draw of the I/O subsystem. In the zero-copy example the ondemand power governor was enabled on the host and the start of the workload leads to a major increase of the CPU power (cf. 6(a)). The actual computation on the FPGA consumes only limited dynamic power (about 2W) as can be seen from the PCIe trace. In between the kernel invocations, the host CPU does not throttle the speed so that the power consumption caused by the CPUs is significant. Figure 6(b) shows the power consumption of the OpenCL design when using local memory on the coprocessor card for buffering. Accessing the SDRAM and the higher processing speed cause an increase of about 12W on the PCIe power domain.

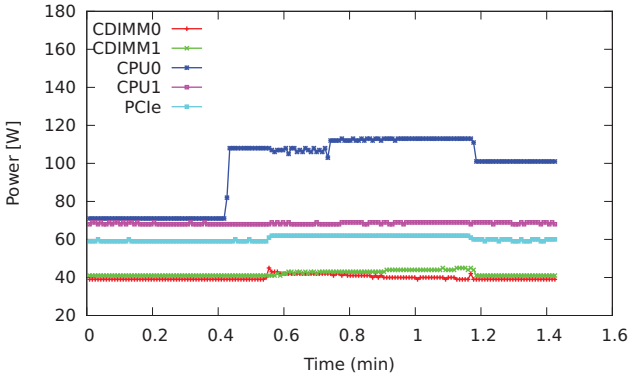
CAPI With our CAPI accelerator we achieve 21.7 and 22.8 GFLOPS for 1k and 4k FFTs, respectively. As the AFU receives and sends data through the PCIe link, the raw the performance is not as high as for the DDR OpenCL kernel but more than 5 $\times$ better than the zero-copy version of the OpenCL. An AFU can access the on-card SDRAM via the PSL and thus, the CAPI accelerator can be easily transformed to design similar to the OpenCL kernel. As expected, a single FFT AFU can not outperform the 12 cores of a POWER8 CPU, but is able to deliver 2.1 $\times$ higher performance compared to a single software thread.

Platform	Size	Gov.	Perf.	P_{tot}	P_{dyn}	E_{tot}	E_{dyn}
CPU 1T	4k	D	10.5	272	52.8	39	199
CPU 1T	4k	P	10.7	328	32.6	33	328
CPU 12T	4k	D	33.5	420	108	80	310
OpenCL DDR	4k	D	75.2	297	14	253	5371
OpenCL DDR	4k	P	76.1	388	11	196	6856
OpenCL 0-copy	4k	D	4.4	295	30	15	147
OpenCL 0-copy	4k	P	4.8	381	5	13	980
CAPI	4k	D	22.3	311	48	72	465
CAPI	4k	P	22.8	371	16	61	1425

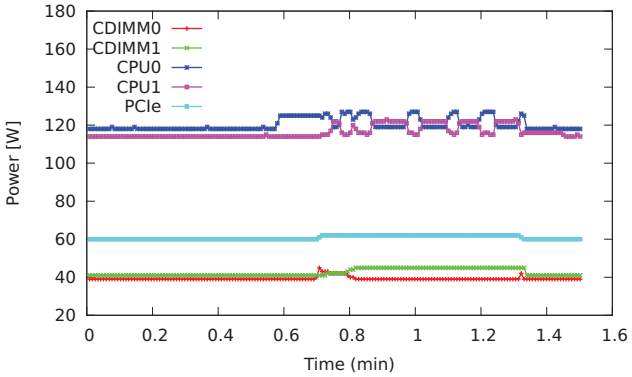
TABLE I. PERFORMANCE AND POWER RESULTS FOR CPU, OPENCL, AND CAPI IMPLEMENTATIONS OF THE 1D FFT.

Figure7(a) shows a benchmark of the CAPI 4k FFT AFU on a system in DVFS mode. Compared to the OpenCL accelerators the CAPI FFT induces slightly higher activity on the CPU. This is expected, because the cache coherent attachment of the accelerator via the CAPP unit as well as polling the status in shared memory from a host thread cause additional activity inside the CPU. The small increase of memory power consumption for the CAPI accelerator can be attributed to the higher bandwith compared to the OpenCL design. The FPGA power is comparable to the OpenCL kernel in zero-copy mode and about 2W.

The performance and power results for CPU, OpenCL, and CAPI implementations of the 1D FFT kernel are summarized



(a) CAPI FFT. DVFS mode.



(b) CAPI FFT. Performance mode.

Fig. 7. Power traces for the CAPI FFT on POWER8.

in Table I. D and P represent the ondemand and performance CPU governors, respectively. The total power P_{tot} comprises the memory and socket power; for the accelerators we also add the power draw from the I/O subsystem. P_{dyn} denotes the dynamic power consumption that is consumed on top of the idle power of the node as measured right before starting the workload. We use the two power measures to compute the energy efficiency E_{tot} and E_{dyn} in terms of MFLOPS/W.

A. Latency

To compare the latency of software and hardware accelerated FFT kernels we run isolated 4k FFT calls on the different platforms and measure the wall-time using the POSIX `clock_gettime` function with the `CLOCK_MONOTONIC` clock. We repeat the test 100 times and compute the average latency of a kernel invocation. In all cases, the input data is initialized on the host and resides in cache memory. On the CPU, the 4k FFT takes 47.5us on average. The zero-copy version of the OpenCL kernel has a rather poor performance and takes 539us. Copying the data to the accelerator memory improves the runtime and reduces the latency to 344us. Raw kernel operation without the data copy consumes 124us. The CAPI version of the 4k FFT accelerator takes 69us on average and with that is 5 $\times$ faster that the OpenCL runtime.

VII. Conclusion

High-level parallel programming models like OpenCL facilitate the programming of heterogeneous systems but normally suffer from overheads caused by system calls through a device driver stack. In this paper we demonstrate that using a globally shared virtual memory organization, as provided by the CAPI architecture on IBM's POWER platforms, can significantly reduce these overheads. In line with previous work on hardware accelerators we can show that offloading compute intensive kernels to an FPGA-based coprocessor can significantly increase the energy efficiency of a general purpose system. Although not necessarily required for the FFT case study, CAPI allows to couple hardware and software threads in a very fine-grained manner and we are investigating this feature for other applications in ongoing work. Shared-virtual-memory is a key innovation of the new OpenCL 2.0 standard and allows host and device platforms to operate on shared data structures using the same virtual address space [12]. CAPI is the enabling technology to implement this feature on POWER8 and with that, an important step towards unified hybrid computing systems.

References

[1] Altera Corp., "OpenCL Design Examples," http://www.altera.com/support/examples/opencl/opencl.html, accessed: 2014-11-25.

[2] *Altera RTE for OpenCL*, Altera Corp., 2014.

[3] *Altera SDK for OpenCL. Programming Guide*, Altera Corp., 2014.

[4] *Vivado Design Suite User Guide – High-Level Synthesis*, Ug902 (v2014.1) ed., Altera Corp., May 2014.

[5] E. Chu and A. George, *Inside the FFT Black Box. Serial and Parallel Fast Fourier Transform Algorithms*. CRC Press, 2000.

[6] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297–301, 1965.

[7] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A Performance and Energy Comparison of FPGAs, GPUs, and Multicores for Sliding-window Applications," in *Int. Symp. on Field-programmable Gate Arrays (FPGA)*. ACM, 2012.

[8] M. Garrido, J. Grajal, M. A. Sánchez, and O. Gustafsson, "Pipelined Radix-2K Feedforward FFT Architectures," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 21, no. 1, 2013.

[9] H. Giefers, R. Polig, and C. Hagleitner, "Analyzing the energy-efficiency of dense linear algebra kernels by power-profiling a hybrid CPU/FPGA system," in *Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2014.

[10] IBM Corp., *ESSL Guide and Reference*, 2012.

[11] S. Kestur, J. Davis, and O. Williams, "BLAS Comparison on FPGA, CPU and GPU," in *Annual Symposium on VLSI (ISVLSI)*. IEEE, 2010.

[12] Khronos OpenCL Working Group, "OpenCL 2.0 Specification," 2014.

[13] P. Milder, F. Franchetti, J. C. Hoe, and M. Püschel, "Computer Generation of Hardware for Linear Digital Signal Processing Transforms," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 17, no. 2, 2012.

[14] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," in *Int. Symp. on Computer Architecture (ISCA)*, 2014.

[15] B. Wile, "Coherent Accelerator Processor Interface (CAPI) for POWER8 Systems," IBM White Paper, Sep 2014.

[16] Xilinx, Inc., "The Xilinx SDAccel Development Environment," 2014.

[17] W. Zhang, V. Betz, and J. Rose, "Portable and Scalable FPGA-based Acceleration of a Direct Linear System Solver," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 5, no. 1, 2012.