Analyzing the Energy-Efficiency of Sparse Matrix Multiplication on Heterogeneous Systems: A Comparative Study of GPU, Xeon Phi and FPGA

Heiner Giefers, Peter Staar, Costas Bekas, Christoph Hagleitner
IBM Research – Zurich
Saumerstrasse 4
CH-8803 Ruschlikon, Switzerland
{hgi,taa,bek,hle}@zurich.ibm.com

***Abstract*—Hardware accelerators have evolved as the most prominent vehicle to meet the demanding performance and energy-efficiency constraints of modern computer systems. The prevalent type of hardware accelerators in the high-performance computing domain are PCIe attached co-processors to which the CPU can offload compute intensive tasks. In this paper, we analyze the performance, power, and energy-efficiency of such accelerators for sparse matrix multiplication kernels. Improving the efficiency for sparse matrix operations is of eminent importance since they work at the core of graph analytics algorithms which are in turn key to many big data knowledge discovery workloads. Our study involves GPU, Xeon Phi, and FPGA co-processors to embrace the vast majority of hardware accelerators applied in modern HPC systems. In order to compare the devices on the same level of implementation quality we apply vendor optimized libraries for which published results exist. From our experiments we deduce that none of the compared devices generally dominates in terms of energy-efficiency and that the optimal solutions depends on the actual sparse matrix data, data transfer requirements and on the applied efficiency metric. We also show that a combined use of multiple accelerators can further improve the system's performance and efficiency by up to 11% and 18%, respectively.**

I. Introduction

The aim for energy-efficient computer systems has spurred the trend towards heterogeneous computing and massively parallel architectures. Heterogeneous systems use GPUs and other types of co-processors to accelerate application hotspots. The first 32 systems on the June 2015 release of the Green500 list apply a combination of Intel Xeon CPUs with accelerators, either GPUs (NVIDIA Kepler and AMD FirePro) or many-core co-processors (Intel Xeon Phi or PEZY-SC) [1] and empirical studies substantiate the suggestion that accelerators provide much better energy-efficiency compared to traditional CPUs [2]. However, SIMD-based accelerators are not suited to handle tasks that do not expose a fair amount of well-structured data-level parallelism. Field Programmable Gate Arrays (FPGAs), in contrast, offer very different opportunities to make use of low-level parallelism. By customizing a data path to the requirements of a specific algorithm, FPGAs can exploit temporal and spatial locality much more directly than instruction-based processors which is particularly favorable in terms of power consumption. While it has been shown that FPGAs can yield tremendous gains in performance and, especially energy-efficiency [3] [4] [5] [6], their adoption in data centers and HPC facilities has been limited in the past, mainly due to the lack of high-level language programming tools. In recent years, significant attention has been paid on design productivity for FPGAs and modern high-level synthesis (HLS) tools now enable the automatic generation of hardware accelerators from C-level languages. Companies like Microsoft and Baidu started to use FPGAs in their production data centers [7] [8] and leading server chip providers integrate FPGAs with their high-performance CPUs via cache-coherent, low-latency interconnects [9] [10].

Among the key workloads that could profit from hardware acceleration is graph-based data mining. Many real world big data sets have a graph structure (e.g., the Internet structure, social interactions, or scientific data) which can itself be represented as sparse matrix. Sparse matrix-vector (SpMV) and sparse matrix-matrix (SpMM) multiplication are the workhorse in most algorithms that operate on sparse data sets, such as iterative methods for linear systems solving, eigenvalue problems, and graph traversal. In addition to the important role in many applications, SpMV is is also widely known for its poor performance on modern multi-core CPUs. A vast numbers of data formats and algorithmic improvements have been investigated in order to cope with the difficulties posed by the SpMV kernel.

In this paper we study the performance and energy-efficiency of SpMV and SpMM kernels on CPU and various compute accelerators. Our benchmarks embrace Intel Xeon CPUs as well as Intel Xeon Phi, NVIDIA Tesla K20, and Nallatech PCIe-385N co-processors and we apply vendor optimized libraries in order to compare the devices at a consistent implementation quality. The experiments show that none of the compared alternatives is generally superior in terms of energy-efficiency and that it always depends on the actual matrix structure, data transfer requirements and on the

978-1-5090-1953-3/16/$31.00 ©2016 IEEE

applied efficiency metric which device is the best. We find that the FPGA is most efficient for the actual computation but also provides the lowest average performance which is detrimental for the efficiency of the overall system. Our main contributions can be summarized as follows:

- We provide the first extensive study on the energy-efficiency of sparse matrix-vector/-matrix multiplication kernels on GPU, Xeon Phi and FPGA.
- Instead of selecting a sparse matrix format and implementation that would favor a particular hardware architecture we apply the prevalent CSR format and use routines from vendor optimized math libraries.
- We employ four different metrics to examine the energy-efficiency of the kernels under various use-case scenarios. We also include a substantially larger set of test matrices for our benchmarks compared to previous studies in order to not bias the results.

II. Related Work

While the energy-efficiency of heterogeneous computing has been demonstrated in numerous cases, the comprehensive and comparative analysis of accelerator devices is an important matter of ongoing research. Due to the great variety of accelerator architectures, parallel programming models and languages, an absolutely fair comparison of different platforms becomes virtually impossible to manage. Researchers proposed benchmarking frameworks based on computational kernels such as the Berkeley dwarfs [11] or the SHOC Benchmark Suite [12], but they rather focus on software-programmed accelerators and take FPGAs into account only marginally [13] [14] [15]. Some initial work analyzes the efficiency of accelerators targeting specific applications [3] [4] [16] or hardware architectures [17] [18] [19].

The performance of sparse matrix multiplication on accelerators has been analyzed in several studies. For a C2015 GPU, NVIDIA reports an average performance for single precision SpMM of 16 Gflops[1] [20] for a selection of 18 sparse matrices taken from the University of Florida collection [21].

A joint study from the Georgia Institute of Technology and the Intel Parallel Computing Lab compares the performance of SpMV for different matrix representations on a 61-core pre-production part of the Knights Corner architecture [22]. Performance results are based on a subset of 16 sparse matrices each of which larger than 30MB. For the CSR matrix format, the mean performance of the SpMV operation is 14 Gflops[1]. For a similar set of matrices, the MKL CSR SpMV kernel we use provides an average performance of 15.6 Gflops on the Xeon Phi 5110P which is 11% faster. However, the authors also investigate a specialized sparse matrix representation format that better facilitates SIMD efficiency and data access locality. Using this custom format, the performance of their SpMV implementation could be improved by 1.85x on average.

A performance analysis for SpMV and SpMM on the Xeon Phi and the NVIDIA K20 GPU is provided by Saule *et al.* [23]. The paper uses 22 sparse matrices (partly taken from [21]) of which a subset of nine matrices is also covered in this work. For the respective subset of matrices our SpMM implementation on the K20 GPU on average shows a 51% better performance[1].

Zhang *et al.* present a hardware implementation of the SpMV kernel on FPGA and evaluate the performance against an NVIDIA GeForce GTX 260 GPU [24]. For a collection of 9 sparse matrices the mean performance for the GPU and the FPGA is 6.2 Gflops and 1.4 Gflops, respectively. Our SpMV implementation is automatically generated from a high-level specification an achieves 1.3 Gflops on average.

An SpMV implementation for the Convey HC-1 reconfigurable computer [25] delivers 2.6 Gflops [26]. The HC-1 has a very specific memory architecture and includes four high-end Xilinx FPGAs that are available to the programmer to implement a custom hardware accelerator. Thus, the performance results are not directly comparable to work that applies a single device only.

A universal architecture for accelerating sparse and dense matrix-vector multiplication on FPGA is presented by Kestur *et al.* [27]. The implementation is able to handle different sparse matrix formats, but only achieves a rather limited average performance of 140 Mflops for SpMV.

III. Energy-Efficiency Metrics

Selecting proper metrics is essential for assessing the energy-efficiency of a computation executed on a specific system. An efficiency metric provides a meaningful interpretation of the measured performance and power consumption and should allow for comparing design alternatives against each other.

Performance Measurements: A very simple, yet effective, tool to obtain performance results from an application is to use the OS's capabilities to measure the elapsed time between the start and end of a code region. However, even for such basic operation, many alternative implementations with different semantics and implications are possible. For our benchmarks, we use the `gettime` function with `CLOCK_MONOTONIC` clock source which provides a most accurate measure of real time consumed by a process in POSIX-compliant OS's.

Power Measurements: Determining the power consumption of computer systems is a more involved task because the ability to measure power or current is not necessarily built into the system, mainly because the required sensors are not available. Many modern CPUs and co-processor devices, however, have built-in power sensors [28] [29] or performance counter based models to estimate the actual power consumption [30] but these sensors typically only comprise

[1]Performance numbers were obtained from a figure in the paper and should be considered as approximations.

a specific power domain, such as the CPU socket power or the power consumed by a specific PCIe card. In order to get reliable power usage information for an entire compute node, external monitoring tools (e.g., Watts up? Pro [31]) can be applied.

Having the ability to trace power at multiple levels leads to interesting insights. An example is the Xeon Phi co-processor card: On our test system, Intel's `micsmc` tool reports 97W of for the Phi when no computation is offloaded to the device. An external power meter shows 326W for the whole system in that situation. When we stop polling the Phi for power, the measured system power reduces to 271W. Hence, the activation of the Phi card for polling the power sensor consumes 55W and the actual idle power for a non-active Phi card can be estimated to 42W. Having only the reading from the card would result in overestimating the static power and underestimating the dynamic power consumption of the device. A similar behavior can be seen for GPUs.

We analyze the power consumption of the test system with an external monitoring instrument custom-built following the guidelines of the SPEC Power and Performance Committee [32]. An AC current transducer is hooked into the primary supply cable and provides a 0-5V DC signal proportional to current flow through the cable. Using an external I/O board, the analog signal is transformed into a 10-bit digital value that is sent via USB to the monitoring PC. The sampling rate of the current monitor is configurable up to 1kHz. To be able to control the monitoring process from within the application, we read out the current sensor directly on the experimental platform and set the sampling rate to moderate 125Hz in order to keep the overhead small.

Power Measures: The power consumption of a computing system can be generally divided in two parts: dynamic power and static power. On the system level, static power denotes the idle power consumption (P^{idle}) of the system when no active computation is taking place and the dynamic power (P^{com}) is the part of the total power (P^{sys}) that is spend atop of the idle power during active processing. We use the terms *idle* and *com*pute at the system level because the entities do not exactly correspond to the dynamic and static power consumption at the circuit level. For our measurements, we use a test system with all three accelerator types installed and, because of the static power of the individual co-processor cards, the idle power of the node is relatively high (246W). As we always use only a single accelerator type in our benchmarks P^{idle} would overrate the actual idle power and thus we subtract the static power of the unused devices to determine an individual P^{idle} for each configuration. Equation 1 describes how the compute power for a specific processing task τ is assessed. We trace the actual power $P^{sys}(t)$ at time t with a sample rate of 125Hz for the execution time T_τ in order to compute an average for P^{com}.

$$P_\tau^{com} = 1/k \sum_{k=1}^{k \leq T_\tau} P^{sys}(k) - P^{idle} \quad (1)$$

Energy Measures: To evaluate the energy consumption of the system various measures are possible, each of which leading to different efficiency model. The *system-wide energy* (E^{sys}) is the most common measure and applies the average total power P^{sys} consumed by the processing node during a certain computation. Using E^{sys} is most useful when the benchmarks reflect the workload for which the actual system is being deployed. Micro-benchmarks, however, are designed to measure a very specific aspect of a program or a system and thus, do not generate a typical workload situation. An efficiency metric based on E^{sys} for our benchmarks overrates the energy consumption and therefore delivers biased results.

One way to tackle this issue is to only factor in the *device energy* as measured on the co-processor card (or the CPU socket on the host). This model includes the static and dynamic power consumption of the processing device but completely disregards energy consumed for memory, data transfer, and control operations on the host. In contrast, the *compute energy* E^{com} involves all types of dynamic energy and is used as an alternative energy measure in this work.

Efficiency Metrics: Energy-efficiency can be defined as the ratio of useful work performed by a system over a certain time and the energy consumed by the system during that time. For scientific computations, Gflop is established as a meaningful work metric because the number of floating-point calculations for a non-iterative numerical operation is rather fixed. For a sparse matrix-vector multiplication $y = Ab$, for example, every non-zero element of A is multiplied with a respective entry in x and the result is accumulated to get the final result entry in y. Thus, the number of FLOPs corresponds directly to the doubled number of non-zero elements in A.

Two energy-efficiency metrics that are very common in domain of VLSI design are the power-delay product (PDP) and the energy-delay product (EDP) [34]. PDP is essentially a measure of energy consumption and should be applied when energy is the primary design constraint for the system. The EDP metric puts higher emphasis on performance and was introduced by Horowitz *et al.* as a more useful efficiency metric for workstation and server-class processors [36].

The widely applied Gflops/W measure is an inverse PDP metric and as such favors low-power architectures. In this paper we use Gflops/W and the analogous inverse EDP metric Gflops^2/W with both, the *system-wide* and *compute energy* measures.

IV. Computing Platforms

In this section, we present the hardware architectures and programming frameworks used to benchmark the sparse matrix kernels for performance and energy. We apply representatives for the most commonly used hardware accelerators today.

TABLE I: Overview and performance characteristics of the applied hardware platforms.

Vendor	Intel	Intel	NVIDIA	Nallatech
Device	Xeon	Xeon Phi	TESLA K20	PCIe-385N_A7
Chip	E5-2630 v2	5110P	GK110	Altera Stratix-V
Reference	[41]	[37]	[38]	[39] [42]
f_{core}	2.60 GHz	1.053 GHz	706 MHz	100-450 MHz
cores	6 (SMT2)	60 (SMT4)	2496	N/A
$LL	15MB L3	30MB L2	1.5 MB L2	7.1 MB RAM
Memory	32GB DDR3	8GB GDDR5	5GB GDDR5	8GB DDR3
BW_{mem}	51.2 GB/s	320 GB/s	208 GB/s	25.6 GB/s
PCIe	v3.0 40 lanes	v2.0 x16	v2.0 x16	v3.0 x8
BW_{IO}	38.5 GB/s	15.4 GB/s	7.8 GB/s	7.7 GB/s
TDP	80W	225W	225W	25W
Static Pwr.	NA	ca. 42W	ca. 18W	ca. 7W
Techn. node	22nm Intel	22nm Intel	28nm TSMC	28nm TSMC
Price	ca. $600	ca. $2500	ca. $2700	ca. $5000
P_{float}	250 Gflops	2 Tflops	3.52 Tflops	1.25 Tflops
P_{double}	125 Gflops	1 Tflops	1.17 Tflops	N/A
SpM Impl.	MKL 11.2	MKL 11.2	cuSPARSE 7.0	OpenCL
Compiler	icc 15.0.0	icc 15.0.0	icc 15.0.0	Quartus 15.0, icc

A. Hardware Platforms

Our baseline platform is a standard server node running a single Xeon E5-2630 v2 CPU and 32 GB of DDR3 main memory. To the host system, we attach three different accelerator cards as peripheral devices via the PCIe bus.

The Intel Xeon Phi co-processor card integrates 60 in-order cores each of which extended by a wide 512-bit vector unit to deliver high floating-point performance [37]. The cores support 4-way simultaneous multi-threading (SMT) and have 32 kB L1 data cache and 512 kB L2 cache. The L2 caches are kept coherent with a directory-based MESI protocol and are connected via a bidirectional ring bus to 8GB of GDDR5 memory. A Xeon Phi runs a Linux micro-kernel allowing the co-processor to appear as a separate compute node and run native applications.

The GPU co-processor applied in this study is an NVIDIA Tesla K20 [38]. The Kepler GK110 device has 13 SMX (Streaming Multiprocessor eXtreme) clusters, each of which comprises 6 blocks of 32-wide SIMD units, resulting in a total number of 2496 floating point ALU per devices.

The Nallatech PCIe-385N accelerator card employs an Altera Stratix-V A7 FPGA device [39] and comprises two independent DDR3 SDRAM banks with up to 8 GB of memory.The PCIe-385N is supported by the Altera OpenCL framework that allows for compiling OpenCL kernel specifications to Verilog hardware descriptions that are further synthesized into an FPGA bitstream [40].

Table I presents a comparison of various features for the CPU and the three accelerator devices. The numbers for bandwidth (BW) and floating point performance (P) are theoretical peak values and are taken from the device vendor data sheets. All of the accelerator cards were released around the same time and are fabricated in comparable CMOS technology (22nm/28nm).

The PCIe bandwidth of the co-processors should be almost at par because the MIC and the GPU, that are build to the PCIe Gen2 standard use a 16-lane interface and the FPGA card features an 8-lanes Gen3 connection. However, the OpenCL endpoint on the card uses a Gen2 PCIe core because the Gen3 IP does not support the FPGA reconfiguration via the PCIe bus. Due to this limitation, the sustainable bandwidth to the FPGA card is drastically reduced. Figure 1 shows a comparison for the measured PCIe bandwidth of all three co-processor devices for different transfer sizes.

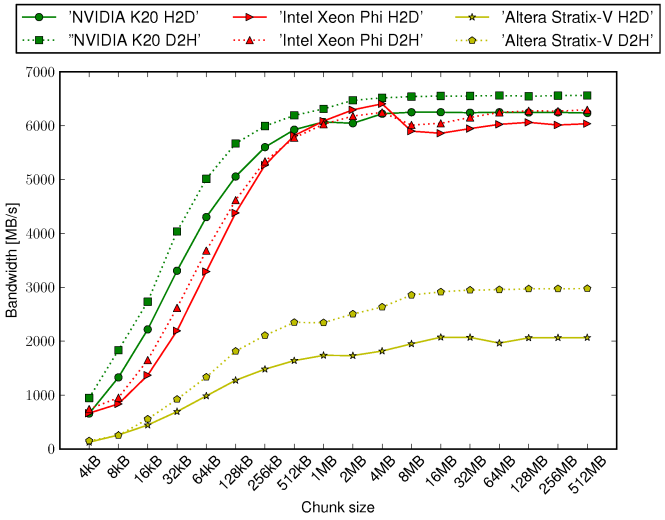


Fig. 1: Measured PCIe host-to-device (H2D) and device-to-host (D2H) bandwidth for the applied co-processor cards.

B. Software Frameworks

The choice of a proper programming model and runtime library has a mature impact on the performance for a specific application. On the CPU and the Xeon Phi, we apply the Intel Math Kernel Library (MKL), a software library of math functions highly optimized for Intel's line of microprocessors. The sparse matrix kernels for the Tesla K20 are taken from the CUDA 7.0 cuSPARSE [43] library. All C/C++ codes for the host CPU were compiled with icc 15.0.0 using the "-O3" flag.

For the PCIe-385N board, we use Altera's OpenCL high-level synthesis (HLS) framework and the Quartus-II 15.0 backend tools to generate FPGA configurations for the compute kernels. 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 endpoint 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.

V. Sparse Matrix Multiplication

The sparse matrix-vector multiplication (SpMV) is an example for a computational kernel with very low arithmetic intensity [44]. The workload is bounded by memory transfer and typically achieves only a fraction of the CPU's theoretical peak performance [45]. An SpMV operation solves $y = Ax$, where x and y are (normally) dense vectors and A is a sparse matrix of size $M \times K$. The SpMV kernel appears as a computational bottleneck in many applications from various domains such as scientific computing, engineering, financial modeling, and information retrieval [46].

A. Sparse Matrix Formats

The storage format used to compress the sparse matrix determines the implementation of the multiplication algorithm and significantly impacts performance. The most widely applied formats to store sparse matrices are coordinate (COO) and compressed sparse row (CSR). Both formats store the NNZ nonzero elements consecutively in a field A and use 2 additional fields to encode the position of these elements. The COO format simply stores the absolute x and y indices of the nonzero elements in two arrays of length NNZ. The CSR format replaces the x-index field by a pointer-array I of size $M+1$ maintaining the first nonzero element positions of the rows in A.

Among other traditional sparse matrix formats are the Diagonal (DIAG), Modified sparse row (MSR), ELLPACK-/ITPACK (ELL), Jagged diagonal (JAD), Skyline (SKY), Block compressed stripe (BCSR), and Variable block row (VBR) format [47]. In his extensive experimentally study on sparse matrix formats, Vuduc concludes that the CSR format is the most reasonable default format for CPUs because it provides best performance on a wide class of matrices and on a variety of superscalar architectures [46]. Several new sparse matrix formats are proposed that enhance the suitability for SIMD vectorization which is particularly advantageous for hardware accelerators. Some examples for such formats are BCCOO [48], Cocktail [49], SELL-C-σ [50], BCSR, and BELLPACK [51]. The drawback of many of those new formats is that they are tailored to a specific hardware architecture and/or a specific class of sparse matrices. Despite the multitude of sparse matrix formats, CSR is presumably the most common format and is supported by all major sparse matrix libraries, e.g., [47] [52] [53] [43]. In recent work [54] an SpMV implementation on GPU based on the CSR format has been shown to outperform other formats such as ELL and Cocktail, while also avoiding the overhead of transforming the sparse matrix to those formats.

B. Algorithmic Challenges

The motivation for the development and optimization of a multitude of matrix formats and implementations lies in the inherent challenges posed by the sparse matrix multiplication operation. One main problem is the very low arithmetic intensity caused by the lack of temporal locality in the accesses to the sparse matrix and only marginal spatial locality in the accesses to the right-hand side (RHS) x. In the SpMV algorithm, each value in A is used only once in the computation and, if the sparse matrix is not structured or blocked, most of the entries in a cache line fetched to get an element in x remain unused. Both characteristics lead to cache pollution and to a high memory overhead per floating point operation.

Another problem is the indirect and irregular memory access to the RHS data. The elements of x are accessed based on the column IDs of the values in A and thus, strongly dependent on the sparsity structure of the matrix. Such data dependencies are generally detrimental to static compiler optimization (e.g., loop unrolling and vectorization) because there is – if not enforced by a special matrix format – no information about consecutive column indices at compile time.

The sparse matrix-matrix multiplication (SpMM) is a generalization of SpMV in which a sparse matrix A is multiplied by a typically tall and narrow dense matrix X. Many applications that originally rely on SpMV can be transformed to use SpMM, for example by combining multiple independent SpMV operations into one SpMM operation [55].

SpMM mitigates some of the aforementioned problems of SpMV. The arithmetic intensity increases because the values of A are now multiplied with an entire row of the RHS X and the spatial locality enables vectorization. However, SpMM like SpMV is a highly data-dependent and memory intense algorithm and both are significantly less efficient compared to dense linear algebra codes.

C. Benchmarks

Our test matrices are obtained from the University of Florida Sparse Matrix Collection [21]. For our benchmarks, we selected all real (non-complex), non-binary, square matrices that have more than 20k rows and more than 2M nonzeroes. Using these constraints resulted in a subset of 181 matrices all with a memory footprint larger than 8MB.

Although often criticized because it does not exactly reflect the behavior of real-world applications, micro-benchmarking is useful for comparing the behavior of low-level primitives [56]. We carefully define a test set-up to fairly compare the SpMV and SpMM kernels on the different compute platforms. In order to obtain stable power and performance data we run all the kernel benchmarks for (at least) 5 seconds.

The data transfer between the PCIe-attached accelerators and the host, as thoroughly analyzed by Gregg and Hazelwood [57], can have a tremendous impact on the performance. To address this issue, we study two different scenarios. In one case, the right-hand side matrix or vector is transferred to the accelerator memory before each SpMM or SpMV operation, respectively. After completing the call, the result Y is copied back to host memory. The matrix data always resides in the device memory. In the second case, we

```
/*  Warmup device: Copy problem data to device memory and
    run one iteration of the kernel */
profile_start();
do{
  if(io) copy_rhs_to_device()
  execute_kernel();
  if(io) copy_result_to_host()
  iterations++;
}while(!profile_stop(&runtime, &power));
```

Listing 1: Benchmarking loop

do not move data among the host and co-processor memory. While the latter use-case reflects the behavior of iterative methods, the first scenarios entails the highest communication overhead.

Listing 1 depicts the general benchmarking loop. Before the actual code segment is entered we execute one iteration of the kernel to warm up the device and afterwards reset the power and execution time counters. We synchronize with the co-processor, after each kernel iteration and use the `profile_stop()` function to test if the loop is running for more than 5 seconds. The device synchronization involves an overhead, which is, however, limited as we have eliminated small matrices (<8MB or <20k rows) from the test data set. Our experiments show that without synchronization the average performance improvement is 8% for the GPU. We did not further investigate asynchronous offload for the MKL-MIC and the OpenCL-FPGA implementations because synchronizing with the host is a common use-case and we also expect that all devices would profit equally.

D. Implementations

Our implementations encode the dense matrices X and Y in a storage order so that the respective library can handle the call without additional matrix transpositions. We use single precision floating point data and align memory addresses for matrices and vectors are on 64-byte cache line boundaries.

Although various publications have shown that vendor library versions of SpMV and SpMM can be improved, they are the most commonly used implementations and as such an appropriate choice for comparing the efficiency of the processing devices We apply the `mkl_scsrmv` and `mkl_scsrmm` sparse BLAS routines for the Intel Xeon CPU and use the compiler assisted offload (CAO) capabilities of the Intel tool chain to execute the same function calls to the Xeon Phi. In the CAO mode, a code segment can be explicitly offload to the MIC using the `#pragma offload target(mic)` directive. The data transfer among the host and the co-processor is controlled by *in/out/inout* clauses and memory allocation is controlled by annotating the flags `alloc_if` and `free_if` to this clauses. For example, the clause `in(X: length(k*n) alloc_if(0) free_if(0))` copies $k*n$ elements of the input pointer X to the devices. The X-pointer on the device memory is reused from a previous operation and is neither allocated or freed at start or end the offload section, respectively. By setting `length(0)` the compiler can be instructed to not copy any

```
__kernel
void spmm(
  __global float * restrict Y,
  __global uint * restrict IA,
  __global uint * restrict JA,
  __global const float * restrict X,
  __global float * restrict A
)
{
  float sum[N];
  uint i, k;
  size_t gid = get_global_id(0);

  #pragma unroll
  for(k = 0; k < N; k++){
    sum[k] = 0.0;
  }
  for(i=IA[gid];i<IA[gid+1];i++){
    #pragma unroll
    for(k = 0; k < N; k++){
      sum[k] += X[(JA[i]<<LOGN)+k] * A[i];
    }
  }
  #pragma unroll
  for(k = 0; k < N; k++){
    Y[(gid<<LOGN)+k] += sum[k];
  }
}
```

Listing 2: Single precision SpMM in OpenCL.

data so that the current buffer content is reused. For the GPU, we use the `cusparseScsrmv` and `cusparseScsrmm2` functions from the extended (cusparse_v2) cuSPARSE API of the CUDA v7.0 toolkit.

To the best of our knowledge, there is no FPGA-optimized sparse linear algebra library available. FPGA vendors typically provide implementations for their devices in form of configurable IP cores, such as PCIe interfaces [58], external memory controllers [59], and floating-point operator libraries [60] [61]. These components allow FPGA developers to design hardware accelerators without rewriting standard IP modules from scratch. HLS compilers enable designers to work at a higher-level of abstraction by using a software program to specify the hardware functionality [62]. This approach is particularly appropriate for compute accelerators that implement software-compliant kernel operations. The recently available OpenCL frameworks for FPGAs expand the HLS approach to an industry standard parallel programming model and provide a rich runtime API to control and execute HLS-generated kernels from the host [40] [63]. A comparison of an OpenCL design flow with traditional low-level HDL methods for various image-processing kernels showed that, albeit the HDL designs achieved a much more efficient use of resources (up to 70% less logic utilization on FPGA device), the end-to-end performance of both design methodologies is almost identical (2-10% difference in frames-per-second) [64].

We have implemented custom SpMV/SpMM kernels in OpenCL and used the Altera OpenCL compiler to generate the hardware accelerator. The implementation of the SpMM kernel is presented in Listing 2. The width N of the RHS matrix X and $LOGN$ are compile time parameters and used for unrolling inner loops.

To experience the potential of optimization, we have imple-

TABLE II: FPGA resource utilization and clock frequency.

	SpMV	SpMM B=32	SpMM B=64	SpMM B=128
Logic utilization [%]	37	50	54	52
DSP blocks [%]	6	38	50	50
RAM blocks [%]	61	38	53	48
f **[MHz]**	240	214	198	185

mented several variants of the FPGA accelerator in OpenCL. We found, that the baseline kernel performed well in average, while optimizations favor specific input matrices. Using a global cache for the RHS data (induced with the `__global const` qualifier) improves the performance for denser matrices (e.g., a speedup of 1.6 for TSOPF_RS_b2383), but is worse for very sparse matrices where the average access latency to X is increased due to frequent cache misses. As the use of a cache for X improved the performance for the greater part of test matrices we implemented an 32kB cache for the SpMM kernels and a 64kB cache for the SpMV kernel. Table II shows the FPGA resource utilization and the achieved maximum clock frequency for four configurations. The utilization is relatively moderate which helps the hardware synthesis tools to generate efficient implementations with clock rates of 185–240MHz.

VI. Results

We execute single precision SpMV and SpMM kernels on the four test platforms using a large subset of 181 matrices from the UF sparse matrix collection (cf. Section V-C). Figure 2 shows the performance distribution for the experiments. The horizontal line in the boxes marks the median and the boxes span the first and third quartile of the measurements. Vertical lines issuing from the box extend to the minimum and maximum; outliers are drawn as separate points. The left part of the figure depicts measurements that include memory transfers in each iteration, for the measurements on the right part of the figure the data resides in accelerator memory.

The performance of SpMV on all applied platforms is below 20 Gflops for 97% of the test cases. If the data movement over the PCIe bus is considered, bandwidth obviously becomes a bottleneck and the CPU outperforms the co-processors in the average case. If the data transfer is excluded, the performance for the accelerators greatly improves for SpMM. The different accelerators achieve their respective peak at different block size. cuSPARSE performs well from N=32 (which matches the warp size of the K20 GPU), whereas the MKL SpMM function on the Phi requires a wider right-hand side matrix. For the FPGA a block size 64 is optimal for the average case.

Figure 3 presents an evaluation of the energy-efficiency of matrix multiplication kernels according to the four different efficiency metrics presented in Section III. In Figures 3a and 3b the PDP metric Gflops/W is applied and 3c and 3d uses the EDP metric Gflops^2/W. While the total energy of the node

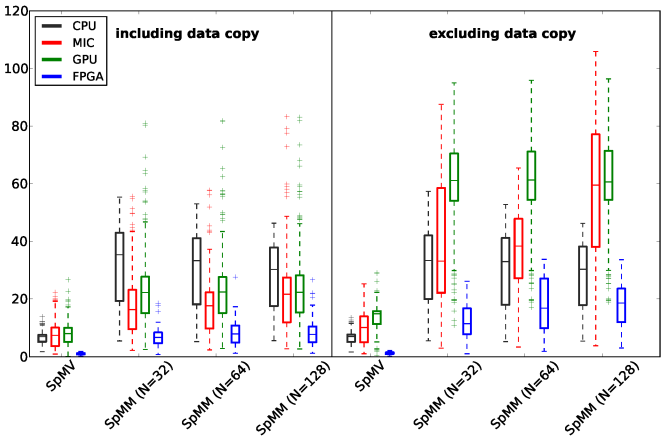


Fig. 2: Measured Performance in Gflops for SpMV and SpMM with different block size.

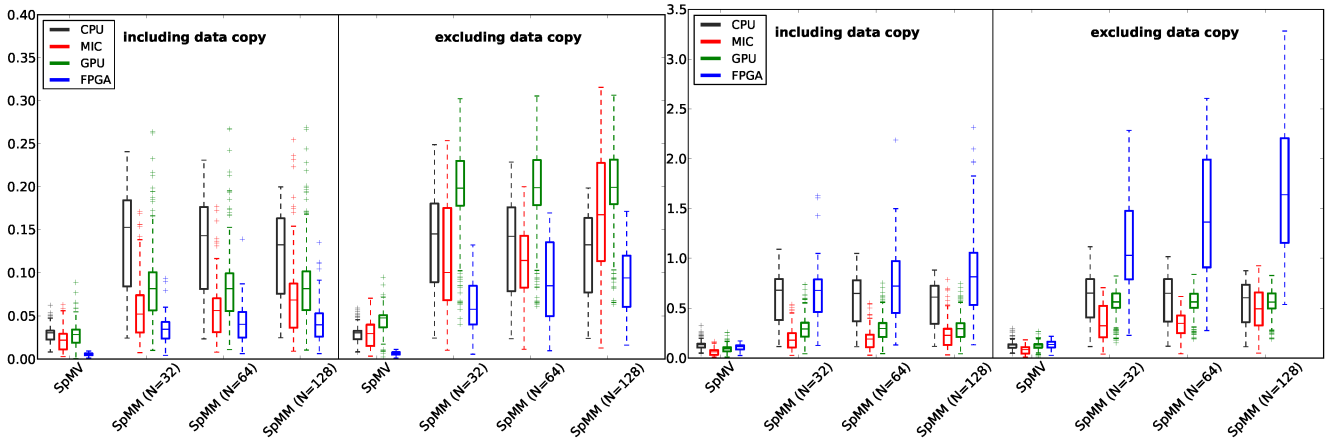
E^{sys} is applied in 3a and 3c, Figures 3b and 3d involve only the dynamic energy consumption E^{com}.

As the power consumption is relatively stable across all test matrices the results under the traditional Gflops/W metric are proportional to the performance results in Figure 2. If we disregard idle power, the FPGA implementation is about 3x more efficient when the data resides in accelerator memory and even outperforms the CPU when data movement is considered (Fig. 3b).

The Gflops^2/W metric naturally has a bias towards high performance (Fig. 3c). In this case, the GPU and the MIC provide similar efficiency. However, if we apply the E^{com} energy measure the lower-performance FPGA version is able to compete with the other co-processors. Considering that we apply an automatically generated, non-optimized accelerator architecture on the FPGA these results are especially remarkable.

Figure 4 presents detailed performance and energy-efficiency results for one of the test cases and a randomly selected subset of 50 sparse matrices ordered with respect to the absolute memory footprint. The applied SpMM kernel does not consider data movement and uses matrices of width N=128 as RHS. The FPGA is the slowest alternative for 90% of the matrices, but also provides the most stable performance among all devices. When the total energy consumption E^{sys} is considered, the GPU is the most efficient accelerator for smaller size matrices. For larger inputs, the MIC frequently surpasses the GPU. If E^{com} is used as energy measure in a PDP metric, the FPGA dominates the other devices for all inputs.

Figure 4 illustrates how strongly the performance of SpMM varies depending on the sparse matrix data. The most important factor for performance are the number of non-zero elements per row and the overall distribution of these elements. Figure 5 shows sparsity plots for selected test matrices. Matrix (a) ecology2 is obtained from a 2D mesh problem and has a diagonal structure with an average number



(a) Energy-efficiency in Gflops/W using E^{sys}

(b) Energy-efficiency in Gflops/W using E^{com}

(c) Energy-efficiency in Gflops^2/W using E^{sys}

(d) Energy-efficiency in Gflops^2/W using E^{com}

Fig. 3: Energy-efficiency of sparse matrix operations on CPU, GPU and FPGA.

of 5 non-zero elements per row. The multi-banded matrix (b) kkt_power is a data set from a nonlinear optimization problem and is also very sparse with about 6 non-zero elements per row. Compared to ecology2 the processing performance is lower on the MIC and the GPU due to the more irregular structure of kkt_power. Matrix (c) nd24k is taken from a set of 3D mesh problem and, with almost 400, has a significantly higher number of non-zero elements per row. Although the non-zero values are even more distributed than in (b), the higher density leads to a better performance on all four platforms.

The results as presented in Figures 3 and 4 show that none of the analyzed platforms is universally energy-optimal for sparse matrix multiplication. A heterogeneous system with multiple co-processors attached can improve the efficiency if operations are mapped to the device that is optimal for the actual problem case. In order to identify the potential savings possible through an optimal mapping technique our obtained benchmark result can be used. We compute the average efficiency over all matrices under the assumption that always the best device is used and compare the obtained optimal efficiency with the average efficiency of the best device.

Figure 6 visualizes the results for every metric and use case; labels on bars exhibit the on-average best device for the specific case. An optimization potential of 0% indicates that the best average device delivers the best efficiency (or performance) for all the considered input matrices. For the Gflops/W (E^{sys}) metric the FPGA is on average the most efficient platform (except for non-iterative SpMV). For the other metrics, the GPU is on average the best platform but an optimal device selection policy could even improve the efficiency by up to 30% and the performance by up to 11%.

VII. Concluding Remarks

In this study we compare the performance and energy-efficiency of SpMV and SpMM kernels on Intel Xeon, Xeon Phi, NVIDIA Tesla K20 and Nallatech PCIe-385N (hosting an Altera Stratix-V A7 FPGA) using widely applied vendor library implementations for the de-facto standard CSR

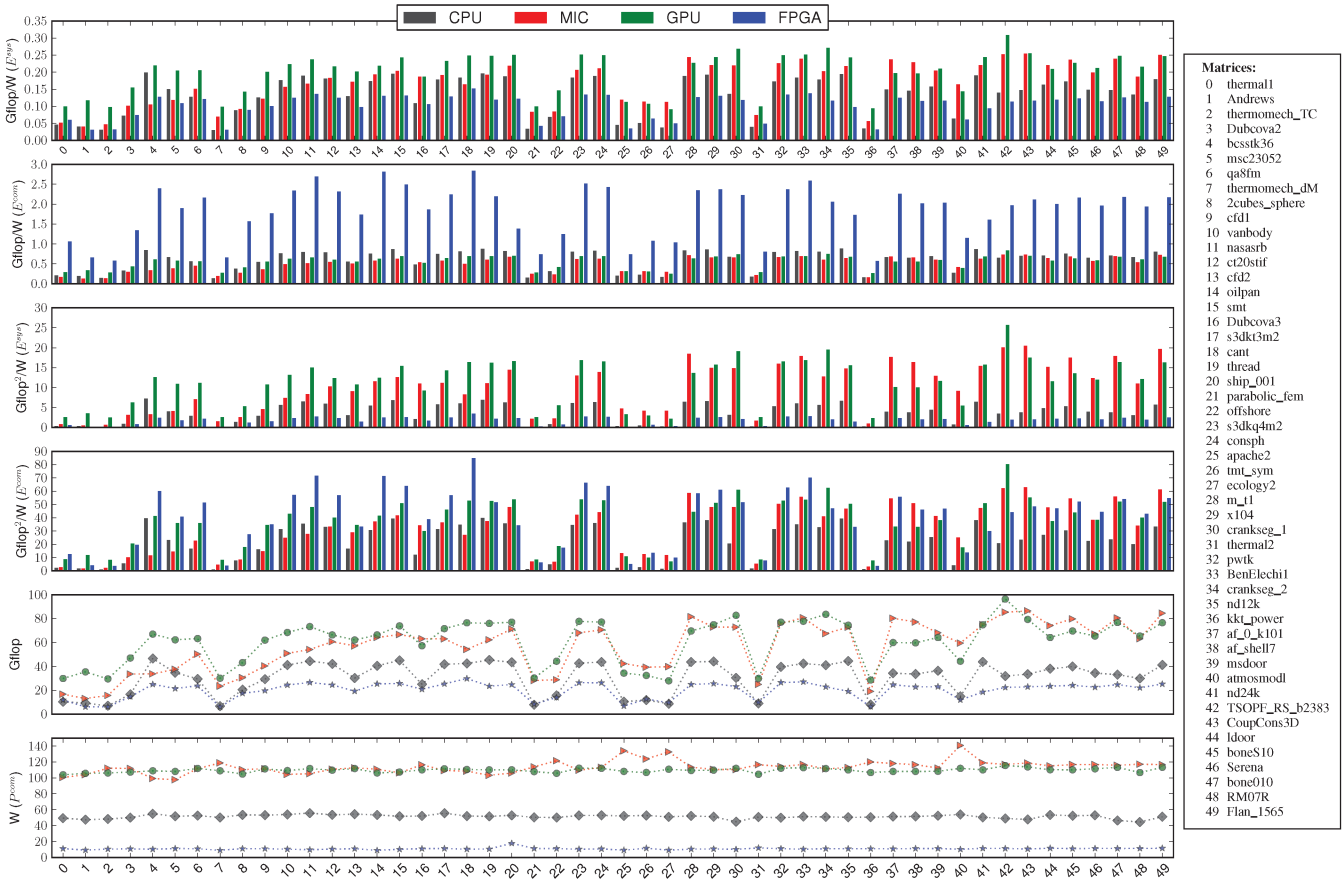
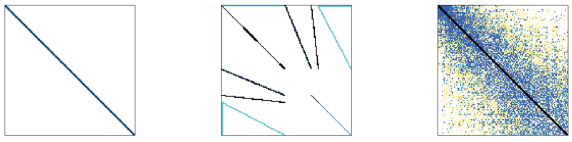


Fig. 4: Detailed performance and energy-efficiency results for a selected subset of 50 sparse matrices. The considered SpMM kernel operates on a RHS matrix of width 128 and excludes data movement from or to the host.



(a) ecology2 (#27) (b) kkt_power (#36) (c) nd24k (#41)

Fig. 5: Sparsity plots for selected test matrices.

sparse matrix format. Although the results significantly vary depending on the applied metric and input data, we can draw several general conclusions.

(i) The FPGA is remarkably efficient under E^{com} metrics: We believe that E^{com} is the most insightful energy measure for energy proportional computing [65]; in fact, E^{com} equals E^{sys} on an ideal energy proportional system. Over the last years, the Power Usage Effectiveness (PUE) has been dramatically improved by eliminating major inefficiencies in power delivery and cooling. Server consolidation [66] [67] and small form factor packaging [68] [69] can significantly reduce power overheads on the node level. The next step towards energy efficient computing is to reduce the power consumption of individual tasks and functions. We demonstrate that accelerators, particularly FPGAs can accomplish this goal for sparse matrix kernels.

(ii) The efficiency gain from using co-processors will increase as I/O bandwidth approaches memory bandwidth: The I/O bottleneck is the main reason why the CPU dominates in terms of performance and efficiency for use-case that involve memory transfers. However, network and I/O bandwidth is growing much faster than the off-chip memory bandwidth which facilitates a very tight integration of co-processors and CPU's in the future. Hardware-managed coherency enables efficient offloading of fine-grained task and will alleviate the memory transfer barrier.

(iii) High-level synthesis tools bridge the productivity gap for FPGAs: Although we do not apply an optimized, RTL accelerator core, a competitive performance is achieved by the FPGA. Available HLS tools, produce efficient hardware and make FPGA technology accessible to software developers.

Recent studies show that HLS-generated designs on average achieve 96% [64] and 82% [70] the performance of optimized RTL designs for various computational kernels

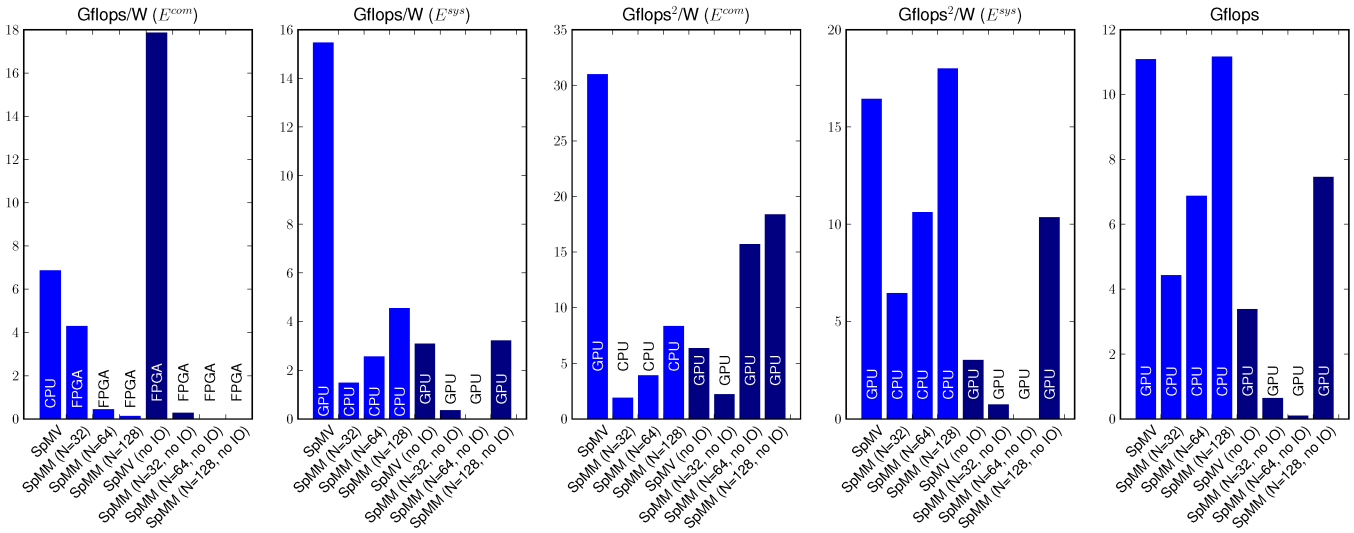


Fig. 6: Potential energy and performance gains (in %) using an optimally device selection policy for SpMV/SpMM.

relevant in high-performance computing. As the data path of SpMV and SpMM kernels is straightforward and can be equally generated by HLS tools, the key for the accelerator is to efficiently exploit the memory subsystem. The OpenCL interface logic is designed to provide high memory bandwidth and includes IP to optimize memory transfers (such as a coalescing unit and data caches). We use this features in our accelerators and measure a sustainable memory bandwidth of more than 13 GB/s for selected matrices. With that we achieve more than 50% the peak bandwidth of the applied FPGA board, which is a good result with regard to the random memory accesses requirements of the sparse matrix operations. As the kernel is memory-bounded, we can conclude that the performance of our CSR based SpMM accelerator is in the range of 50-100% of the theoretic peak of the particular FPGA board and that a highly optimized RTL design would not outperform our accelerator by more than 2x.

In ongoing work, we focus on the question how the system can exploit heterogeneity in form of multiple co-processors. Power profiling provides insights about the efficiency of computations on different devices. An energy-efficient systems should profit from this knowledge by scheduling tasks to the most appropriate resource. In order to accomplish this goal, we need cross-accelerator compatible APIs that allow for load balancing function calls among several platforms and to make the presence of accelerators transparent to the programmer.

Acknowledgment

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant number 610509 (Nanostreams).

References

[1] "The Green500 List - June 2015," http://www.green500.org/lists/green201506, accessed: 2015-09-04.

[2] B. Subramaniam, W. Saunders, T. Scogland, and W.-C. Feng, "Trends in energy-efficient computing: A perspective from the Green500," in *Green Computing Conference (IGCC)*, 2013.

[3] 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)*, 2012.

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

[5] C. W. Fletcher, I. A. Lebedev, N. B. Asadi, D. R. Burke, and J. Wawrzynek, "Bridging the GPGPU-FPGA Efficiency Gap," in *Field Programmable Gate Arrays (FPGA)*, 2011.

[6] B. Cope, P. Cheung, W. Luk, and L. Howes, "Performance Comparison of Graphics Processors to Reconfigurable Logic: A Case Study," *IEEE Trans. on Computers*, vol. 59, no. 4, 2010.

[7] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. 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. Xiao, and D. Burger, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," in *Int. Symp. on Computer Architecture (ISCA)*, 2014.

[8] J. Ouyang, S. Lin, W. Qi, Y. Wang, B. Yu, and S. Jiang, "SDA: Software-Defined Accelerator for Large-Scale DNN Systems," in *HotChips26*, 2014.

[9] P. K. Gupta, "Intel Xeon+FPGA Platform for the Data Center," in *Field Programmable Logic and Applications (FPL), Workshop on Reconfigurable Computing for the Masses*, 2014.

[10] J. Stuecheli, B. Blaner, C. Johns, and M. Siegel, "CAPI: A Coherent Accelerator Processor Interface," *IBM Journal of Research and Development*, vol. 59, no. 1, 2015.

[11] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The Landscape of Parallel Computing Research: A View from Berkeley," EECS Department, University of California, Berkeley, Tech. Rep., 2006.

[12] A. Danalis, G. Marin, C. McCurdy, J. Meredith, P. Roth, K. Spafford, V. Tipparaju, and J. Vetter, "The Scalable HeterOgeneous Computing (SHOC) Benchmark Suite," in *3rd Workshop on General-Purpose Computation on Graphics Processors (GPGPU)*, 2010.

[13] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Int. Symp. on Workload Characterization (IISWC)*, 2009.

[14] W.-c. Feng, H. Lin, T. Scogland, and J. Zhang, "OpenCL and the 13 Dwarfs: A Work in Progress," in *Int. Conf. on Performance Engineering (ICPE)*, 2012.

[15] A. Guerre, J.-T. Acquaviva, and Y. Lhuillier, "A unified methodology for a fast benchmarking of parallel architecture," in *Design, Automation and Test in Europe (DATE)*, 2014.

[16] 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)*, 2014.

[17] B. Li, H.-C. Chang, S. L. Song, C.-Y. Su, T. Meyer, J. Mooring, and K. Cameron, "The Power-Performance Tradeoffs of the Intel Xeon Phi on HPC Applications," in *Int. Workshop on Large Scale Parallel Processing (LSPP)*, 2014.

[18] J. Sim, A. Dasgupta, H. Kim, and R. Vuduc, "A performance analysis framework for identifying potential benefits in GPGPU applications," in *Principles and Practice of Parallel Programming (PPoPP)*, 2012.

[19] M. S.-A. H. Wong, M.-M. Papadopoulou and A. Moshovos, "Demystifying GPU microarchitecture through microbenchmarking," in *Performance Analysis of Systems Software (ISPASS)*, 2010.

[20] M. Naumov, L. S. Chien, P. Vandermersch, and U. Kapasi, "CUSPARSE Library: A Set of Basic Linear AlgebraSubroutines for Sparse Matrices," in *GPU Technology Conference*, 2010.

[21] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, 2011.

[22] X. Liu, M. Smelyanskiy, E. Chow, and P. Dubey, "Efficient Sparse Matrix-vector Multiplication on x86-based Many-core Processors," in *Int. Conf. on Supercomputing (ISC)*, 2013.

[23] E. Saule, K. Kaya, and Ü. V. Çatalyürek, "Performance Evaluation of Sparse Matrix Multiplication Kernels on Intel Xeon Phi," in *Parallel Processing and Applied Mathematics (PPAM)*, 2013.

[24] Y. Zhang, Y. Shalabi, R. Jain, K. Nagar, and J. Bakos, "FPGA vs. GPU for sparse matrix vector multiply," in *Field-Programmable Technology (FPT)*, 2009.

[25] T. Brewer, "Instruction Set Innovations for the Convey HC-1 Computer," *Micro, IEEE*, vol. 30, no. 2, 2010.

[26] K. Nagar and J. Bakos, "A Sparse Matrix Personality for the Convey HC-1," in *Field-Programmable Custom Computing Machines (FCCM)*, 2011.

[27] S. K. amd John D. Davis and E. S. Chung, "Towards a Universal FPGA Matrix-Vector Multiplication Architecture," in *Field-Programmable Custom Computing Machines (FCCM)*, 2012.

[28] *NVML API REFERENCE MANUAL*, Version 4.304.55 ed., NVIDIA Corp., 2012.

[29] M. Burtscher, I. Zecena, and Z. Zong, "Measuring GPU Power with the K20 Built-in Sensor," in *Proceedings of Workshop on General Purpose Processing Using GPUs*, ser. GPGPU-7, 2014.

[30] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory Power Estimation and Capping," in *Int. Symp. on Low Power Electronics and Design (ISLPED)*, 2010.

[31] *Watts up? and Watts up? PRO Operators Manual*, https://www.wattsupmeters.com, Electronic Educational Devices, accessed: 2015-09-08.

[32] *Power and Performance Benchmark Methodology*, V2.1 ed., Standard Performance Evaluation Corporation (SPEC), SPEC Power and Performance Committee, 2012.

[33] V. G. Oklobdzija, *The Computer Engineering Handbook*, 2001.

[34] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. Cook, "Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors," *IEEE Micro*, vol. 20, no. 6, 2000.

[35] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *Solid-State Circuits*, vol. 31, no. 9, 1996.

[36] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-Power Digital Design," in *IEEE Symp. Low Power Electronics*, 1994.

[37] *Intel Xeon Phi Coprocessor. Datasheet*, Intel, 2014.

[38] *Tesla K20 GPU Accelerator, BD-06455-001_v07*, NVIDIA, 2013.

[39] *PCIe-385N Product Brief*, Nallatech, 2014.

[40] *Altera SDK for OpenCL. Programming Guide*, 13th ed., Altera Corp., Dec. 2013.

[41] *Intel Xeon Processor E5-2600 v2. Product Brief*, Intel, 2014.

[42] *Stratix V Device Handbook*, Altera, Jul. 2014.

[43] "NVIDIA CUDA Sparse Matrix library," https://developer.nvidia.com/cusparse, accessed: 2015-09-04.

[44] B. Dally, P. Hanrahan, and R. Fedkiw, "A Streaming Supercomputer. Stanford Computer Systems Laboratory White Paper," 2001.

[45] G. Goumas, K. Kourtis, N. Anastopoulos, V. Karakasis, and N. Koziris, "Performance evaluation of the sparse matrix-vector multiplication on modern architectures," *The Journal of Supercomputing*, vol. 50, no. 1, 2009.

[46] R. W. Vuduc, "Automatic Performance Tuning of Sparse Matrix Kernels," Ph.D. dissertation, University of California, Berkeley, CA, USA, 2004.

[47] Y. Saad, "SPARSKIT: a basic tool kit for sparse matrix computations," Tech. Rep., 1994, version 2.

[48] S. Yan, C. Li, Y. Zhang, and H. Zhou, "yaSpMV: Yet Another SpMV Framework on GPUs," in *Principles and Practice of Parallel Programming (PPoPP)*, 2014.

[49] B.-Y. Su and K. Keutzer, "clSpMV: A Cross-Platform OpenCL SpMV Framework on GPUs," in *Supercomputing (ISC)*, 2012.

[50] M. Kreutzer, G. Hager, G. Wellein, H. Fehske, and A. R. Bishop, "A unified sparse matrix data format for efficient general sparse matrix-vector multiply on modern processors with wide SIMD units," *SIAM Journal on Scientific Computing*, vol. 36, no. 5, 2014.

[51] J. W. Choi, A. Singh, and R. W. Vuduc, "Model-driven Autotuning of Sparse Matrix-vector Multiply on GPUs," in *Principles and Practice of Parallel Programming (PPoPP)*, 2010.

[52] R. Pozo and K. Remington, "SparseLib++ v1.5 – Sparse Matrix Class Library," Tech. Rep., 1996, reference Guide.

[53] T. A. Davis, "SuiteSparse: A Suite of Sparse Matrix Software," http://www.suitesparse.com, accessed: 2015-09-16.

[54] J. L. Greathouse and M. Daga, "Efficient sparse matrix-vector multiplication on gpus using the csr storage format," in *High Performance Computing, Networking, Storage and Analysis (SC)*, 2014.

[55] P. W. J. Staar, P. K. Barkoutsos, R. Istrate, A. C. I. Malossi, I. Tavernelli, N. Moll, H. Giefers, C. Hagleitner, C. Bekas, and A. Curioni, "Stochastic Matrix-Function Estimators: Scalable Big-Data Kernels with High Performance," in *Parallel and Distributed Processing Symposium (IPDPS)*, 2016.

[56] M. Seltzer, D. Krinsky, K. Smith, and X. Zhang, "The Case for Application-Specific Benchmarking," in *Hot Topics in Operating Systems (HOTOS)*, 1999.

[57] C. Gregg and K. Hazelwood, "Where is the Data? Why you Cannot Debate CPU vs. GPU Performance Without the Answer," in *Performance Analysis of Systems and Software (ISPASS)*, 2011.

[58] Altera Corp., "IP Compiler for PCI Express," 2013.

[59] ——, "External Memory Interface Handbook," 2015.

[60] ——, "Floating-Point Megafunctions," User Guide, 2013.

[61] Xilinx, Inc., "Floating-Point Operator v7.1," LogiCORE IP Product Guide, 2015.

[62] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, "A survey and evaluation of fpga high-level synthesis tools," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 2016, Preprint.

[63] Xilinx, Inc., "SDAccel Development Environment," User Guide, 2015.

[64] K. Hill, S. Craciun, A. George, and H. Lam, "Comparative analysis of OpenCL vs. HDL with image-processing kernels on Stratix-V FPGA," in *Application-specific Systems, Architectures and Processors (ASAP)*, 2015.

[65] L. A. Barroso and U. Hölzle, "The Case for Energy-Proportional Computing," *Computer*, vol. 40, no. 12, 2007.

[66] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy Aware Consolidation for Cloud Computing," in *HotPower*, 2008.

[67] A.-C. Orgerie, M. D. d. Assuncao, and L. Lefevre, "A Survey on Techniques for Improving the Energy Efficiency of Large-scale Distributed Systems," *ACM Comput. Surv.*, vol. 46, no. 4, 2014.

[68] *HP Moonshot System Family Guide*, Hewlett-Packard, 2014.

[69] R. Luijten, D. Pham, R. Clauberg, M. Cossale, H. Nguyen, and M. Pandya, "Energy-Efficient Microserver Based on a 12-Core 1.8GHz 188K-CoreMark 28nm Bulk CMOS 64b SoC for Big-Data Applications with 159GB/s/L Memory Bandwidth System Density," in *Solid-State Circuits Conference (ISSCC)*, 2015.

[70] G. Wang, H. Lam, A. George, and G. Edwards, "Performance and Productivity Evaluation of Hybrid-Threading HLS versus HDLs," in *High Performance Extreme Computing Conference (HPEC)*, 2015.