Performance Portability Study of Linear Algebra Kernels in OpenCL

Karl Rupp
Institute for Microelectronics, TU Wien
Gußhausstr. 27-29/E360
A-1040 Wien, Austria
rupp@iue.tuwien.ac.at

Philippe Tillet
Institute for Microelectronics, TU Wien
Gußhausstr. 27-29/E360
A-1040 Wien, Austria
phil.tillet@gmail.com

Florian Rudolf
Institute for Microelectronics, TU Wien
Gußhausstr. 27-29/E360
A-1040 Wien, Austria
rudolf@iue.tuwien.ac.at

Josef Weinbub
Institute for Microelectronics, TU Wien
Gußhausstr. 27-29/E360
A-1040 Wien, Austria
weinbub@iue.tuwien.ac.at

Tibor Grasser
Institute for Microelectronics, TU Wien
Gußhausstr. 27-29/E360
A-1040 Wien, Austria
grasser@iue.tuwien.ac.at

Ansgar Jüngel
Institute for Analysis and Scientific Computing, TU Wien
Wiedner Hauptstr. 8-10/E101
A-1040 Wien, Austria
juengel@asc.tuwien.ac.at

ABSTRACT

The performance portability of OpenCL kernel implementations for common memory bandwidth limited linear algebra operations across different hardware generations of the same vendor as well as across vendors is studied. Certain combinations of kernel implementations and work sizes are found to exhibit good performance across compute kernels, hardware generations, and, to a lesser degree, vendors. As a consequence, it is demonstrated that the optimization of a single kernel is often sufficient to obtain good performance for a large class of more complicated operations.

Keywords

OpenCL, Performance, Portability, Linear Algebra

1. INTRODUCTION

With the introduction of OpenCL [10], a unified application programming interface for massively parallel hardware became available, simplifying the implementation of portable software. However, the portability of code does not automatically imply the portability of performance [8, 11, 12, 14, 18], neither across vendors nor within different hardware generations of the same vendor. To obtain best performance for a given device, autotuning approaches with different degrees of sophistication were proposed, particularly for the optimization of dense matrix-matrix multiplications [3, 5, 7]. These approaches perform an exhaustive search in a large space of *kernel configurations*, which we define as the triple consisting of parameterized source code, local work size, and global work size as specified in the OpenCL standard.

IWOCL '14, May 12 - 13 2014, Bristol, United Kingdom
 http://dx.doi.org/10.1145/2664666.2664674.

Despite the success of autotuning approaches for finding the best kernel configuration for a given device, they also entail excessive execution times. While these execution times are often acceptable for software developers and scientific investigations, they represent a considerable burden when exposed to users through software libraries. An example is the linear algebra library ATLAS [6, 17], which executes the autotuning process during installation on the target machine, eventually taking hours to complete and effectively blocking the machine for that period. With further increases in the number of cores and more complicated cache hierarchies, such autotuning approaches face severe practical limitations stemming from an explosion of the search space.

This work is motivated by previous work on portable performance for linear algebra operations through a kernel generation approach [15] implemented in the free open-source library ViennaCL [13, 16]. Due to the just-in-time compilation capabilities of OpenCL, such a kernel generation can be completely embedded into a software library. At the same time, the performance is competitive with vendor-tuned libraries on devices from AMD, Intel, and NVIDIA. The portable performance reported in [15] was obtained from in-house autotuning runs for each of the kernels on the respective target hardware. While this enables optimized kernel configurations for selected hardware, the approach does not scale well to the full range of hardware available on the market: First, the procedure has to be repeated for each new hardware generation, which is both very time consuming and does not provide a deeper understanding of the underlying hardware. Second, with the broader support of OpenCL by hardware vendors it becomes practically impossible to repeat the tuning procedure for each device generation available. To overcome these limitations, it is necessary to better understand the influences of each of the parameters available inside the kernel configurations.

The purpose of this work is to find a strategy for obtaining portable performance across a wide range of possibly unknown target hardware. To achieve this we systematically study the performance variations obtained for linear algebra kernels simpler than those for the aforementioned matrix-matrix multiplications. We consider vector

and dense matrix-vector operations, from which the basic linear algebra subprograms (BLAS) operations at level 1 and 2 are composed. More specifically, we consider the vector copy operation ($x \leftarrow y$), a scaled vector addition ($x \leftarrow \alpha y + \beta z$), the inner product of two vectors ($x \leftarrow \langle y, z \rangle$), and the matrix-vector product $x \leftarrow Ay$. Since all operations are limited by the available memory bandwidth, one may expect that it is sufficient to tune just one of these operations and deduce fast implementations for the others. As will be shown in the remainder of this work, this is indeed possible: The best implementations for the copy operation serve as archetypes for obtaining good performance for the other operations.

To stay within a reasonable scope, no operations related to sparse matrices are considered in this work. Their optimal data structure depends on the underlying sparsity pattern [1, 2], which would add another degree of freedom to the variability of performance due to the underlying hardware and the OpenCL implementation. Also, only real-valued arithmetic is considered, as OpenCL does not yet define a native type for complex arithmetic and the emulation of complex-valued arithmetic would hence introduce another unnecessary source of variation.

We conduct our study in a hierarchical manner: First, all kernel configurations for a given operation (vector copy, vector addition, inner product, or dense matrix-vector product) are investigated. Second, we correlate the performance of kernel configurations across different operations on the same hardware and it is shown that high performance for all kernels can be obtained by running a full optimization for only one of the kernels. Third, performances across hardware of the same type and vendor, e.g. GPUs from NVIDIA, are compared, where it is found that good kernel configurations for older hardware often provides good performance on newer devices. Finally, we compare the performance of kernel configurations across different hardware types and vendors and show that newer hardware generations show better performance portability when compared to older generations.

2. KERNEL PARAMETERIZATION

In a massively parallel setting, even simple operations such as the vector copy operation

$$\mathbf{x} \leftarrow \mathbf{y}$$

allow for many different ways of assigning the computational work to the individual work items. These work items are logical entities and the actual mapping to the underlying hardware is left to the OpenCL implementation. One implementation of the kernel body for this operation with vectors of size N is

```
for (size_t i = get_global_id(0);
2            i < N;
             i+= get_global_size(0))
4  x[i] = y[i];
```

with **double**-arrays x and y, where the vector entries are processed in blocks of the global work size given by get_global_size(0). A drawback of the kernel above is that the increment of the index i by the global work size is detrimental to caching. As an alternative, one can rewrite the kernel in a more cache-friendly way such that each workgroup operates on consecutive memory:

```
for (size_t i = group_start + get_local_id(0);
2            i < group_end;
             i+= get_local_size(0))
4  x[i] = y[i];
```

Here, group_start and group_end represent suitably chosen group boundaries, for example multiples of the local work size in order to preserve aligned memory accesses.

In addition to the two kernel skeletons presented above, the native OpenCL vector data types such as double2, double4, double8, and double16 increase the number of possible implementations even further. We include these vector data types in our study even though they are less suited if strided vector operations are required. Finally, each OpenCL kernel launch requires the specification of local and global work sizes. We restrict the choice of local work sizes to powers of two up to a value of 512, because other workgroup sizes are either not well-suited for parallel reduction operations such as inner products, or exhaust the available local memory. The number of workgroups considered is empirically chosen as powers of two up to 1024 supplemented by 48, 80, 96, 112, 160, 192, 224, and 384 in order to cover additional multiples of 16 as recommended in vendor optimization guides. Although one may obtain negligible performance improvements with an even higher number of workgroups, we refrained from including this scenario in our benchmarks, as this would have resulted in too many configurations using large vector data types and large workgroup sizes with no work assigned.

In summary, with the parameters outlined above one obtains $2 \times 5 \times 10 \times 19 = 1900$ different kernel configurations. The same parameterization and thus the same number of kernel configurations are obtained for the scaled vector addition $\mathbf{x} \leftarrow \alpha\mathbf{y} + \beta\mathbf{z}$ with $\alpha \neq 0$ and $\beta \neq 0$, the inner product $\alpha \leftarrow \langle \mathbf{x}, \mathbf{y} \rangle$, and the dense matrix-vector product $\mathbf{x} \leftarrow \mathbf{A}\mathbf{y}$. Although the inner product and the matrix-vector multiplication use reductions in local memory and are therefore of slightly different nature than the vector copy and addition operations, the same set of parameters can be used. Matrix-vector products are implemented such that each workgroup computes one entry in the result vector: Workgroup i thus computes the inner product of row i with the right hand side vector. Other blocking strategies are possible for matrix-vector products, but as will be shown later, high performance can already be obtained in this reduced setting.

A similar parameterization is also applicable to matrix-matrix multiplications, where several additional parameters enter the optimization of (sub-)block sizes [3, 4, 5, 7, 9, 15]. Since a detailed understanding of these additional parameters is not necessary for an understanding of the results in Section 3, we refer the interested reader to the literature at this point.

3. RESULTS

The 1900 kernel configurations for each of the four operations above was compared on different hardware from AMD, INTEL, and NVIDIA, using the OpenCL implementation of the respective vendor. We selected both older and newer graphics processing units (GPUs) from AMD and NVIDIA in order to quantify the influence of more recent hardware on performance portability. Both the AMD Radeon HD 5850 (with Catalyst 14.1 on Windows 7) and the NVIDIA GeForce GTX 285 (with CUDA 5.0) were introduced in 2009, while the AMD FirePro W9000 (with Catalyst Pro

13.251.1) and the NVIDIA Tesla K20m (with CUDA 6.0) were released as high-end workstation GPUs in 2012 and 2013, respectively. Furthermore, results for the central processing unit (CPU) of an AMD A10-5800K accelerated processing unit (with AMD APP SDK 2.8 on Windows 7), a dual-socket INTEL Xeon E5-2670 machine, and an INTEL Xeon Phi are presented (both with INTEL OpenCL SDK 2013 XE 3.0). The local workgroup size was limited to 256 work items on AMD hardware, because larger sizes are not supported. As we were unable to obtain satisfactory performance for memory bandwidth limited operations using OpenCL on the INTEL Xeon Phi, we mostly plot results for the CPUs and GPUs in our comparison. This performance regression has been confirmed by INTEL engineers in direct communication for the case of vector addition and has also been reported by other colleagues in private communication.

Double precision arithmetic was used for all benchmarks, since this is the standard for linear algebra operations. Because all operations considered are limited by memory bandwidth, all results are presented relative to the theoretical peak memory bandwidth of the respective device to simplify comparisons. For matrix-vector products we assume perfect caching of the right hand side vector and only counted the minimum number of bytes necessary to compute the results. With the large caches on CPUs and broadcast memory transfers on GPUs, this simplification is justified and results in consistent performance results on most hardware considered. To ensure that the PCI-Express latency of up to 10 microseconds is negligible, that full data caching is not possible, and that the benchmark can also be run on GPUs with only 512 MB of main memory, vectors for the copy, addition, and inner product operations are chosen to consist of two million entries each. Similarly, the row and column counts of the matrix for the matrix-vector product are 2048.

3.1 Isolated Parameter Variation

We first consider benchmark results obtained by looking at the distribution of performances for each of the four parameters identified in the previous section:

- **Increment Type**: Whether the loop index is incremented by the local work size (*local*) or by the global work size (*global*).
- **Vector Length**: Whether to use the scalar data type **double** (vector length 1), or native vector types such as double16.
- **Local Work Size**: The number of work items per OpenCL workgroup.
- **Workgroups**: Total number of OpenCL workgroups.

The benchmark results for the four operations considered are qualitatively similar, with differences being more pronounced for the inner product and the matrix-vector product kernels. Consequently, we only depict results for the dot product in the following.

Figure 1 shows frequency plots with respect to the increment type in the **for**-loop. While the increment type has only a very mild influence on GPUs, there is a clear preference of increments by the local work size on CPUs, which is attributed to memory pages loaded from main memory. Therefore, if no information over the target device is available other than the device type, which can be obtained from the OpenCL stack at runtime, an increment by the global work size should be used for GPUs, and an increment by the local work size for CPUs.

A comparison of benchmark results for the different vector data types is given in Figure 2. The HD 5850 prefers longer vector types, particularly double8. Conversely, vector data types result in poor performance on the GTX 285, which prefers the plain **double** scalar type for its scalar architecture. On the W9000 and K20m one can obtain good performance with both **double**, double2 and to a lesser extent double4, while double8 and double16 result in poor performance on the K20m. CPUs are less sensitive to the different vector types, with double4 being slightly preferred on the Xeon E5-2670, whereas **double** is not recommended on the A10-5800K CPU.

Figure 3 depicts the benchmark results obtained for different local workgroup sizes. Both AMD and NVIDIA GPUs show good performance for 128 and 256 work items per workgroup. In contrast, good performance on the CPU is predominantly obtained with only one or two work items per workgroup. The same trend is observed for matrix-vector products, while large workgroup sizes yield better results for copy and addition (plots not shown). The INTEL CPUs are found to work slightly better with large workgroup sizes than the AMD CPU, which barely reaches more than half of the theoretical peak memory bandwidth. We suspect that this is caused by the use of only a single memory channel by the OpenCL implementation.

Finally, Figure 4 shows the benchmark results with respect to different numbers of workgroups, where only powers of two are plotted for the sake of conciseness. Good performance on AMD GPUs can be obtained already with 32 workgroups, whereas the results for NVIDIA GPUs as well as the CPUs show a clear preference for a high number of workgroups. We conclude that the number of workgroups should generally be chosen as high as reasonably possible particularly on NVIDIA GPUs and no detrimental effects on performance are to be expected by this choice.

3.2 Portability Within Each Device

Figure 5 and Figure 6 illustrate the change in performance of each kernel configuration for the four operations, always taking the performance obtained for the copy kernel as reference on the abscissa. If the performance obtained for the copy operation were the same for the other three operations, then all configurations would be aligned along the diagonal. The best configurations are located in the upper right corners, indicating that they result in high performance for both operations compared.

NVIDIA GPUs show a strong correlation of performance, where a good performance for the copy operation also results in good performance for the addition, the inner product, and the matrix-vector product, cf. Figure 5. The overall performance drop for matrix-vector products on the GTX 285 can be attributed to the absence of caching for the right hand side vector. Good performance correlation is also obtained on the W9000, even though there is a higher performance fluctuation when comparing the copy operation with the matrix-vector product. Nevertheless, by considering only configurations achieving more than 75 percent of the theoretical peak bandwidth for the copy kernel, configurations offering more then 75 percent of theoretical peak bandwidth for the matrix-vector product are included. The weakest

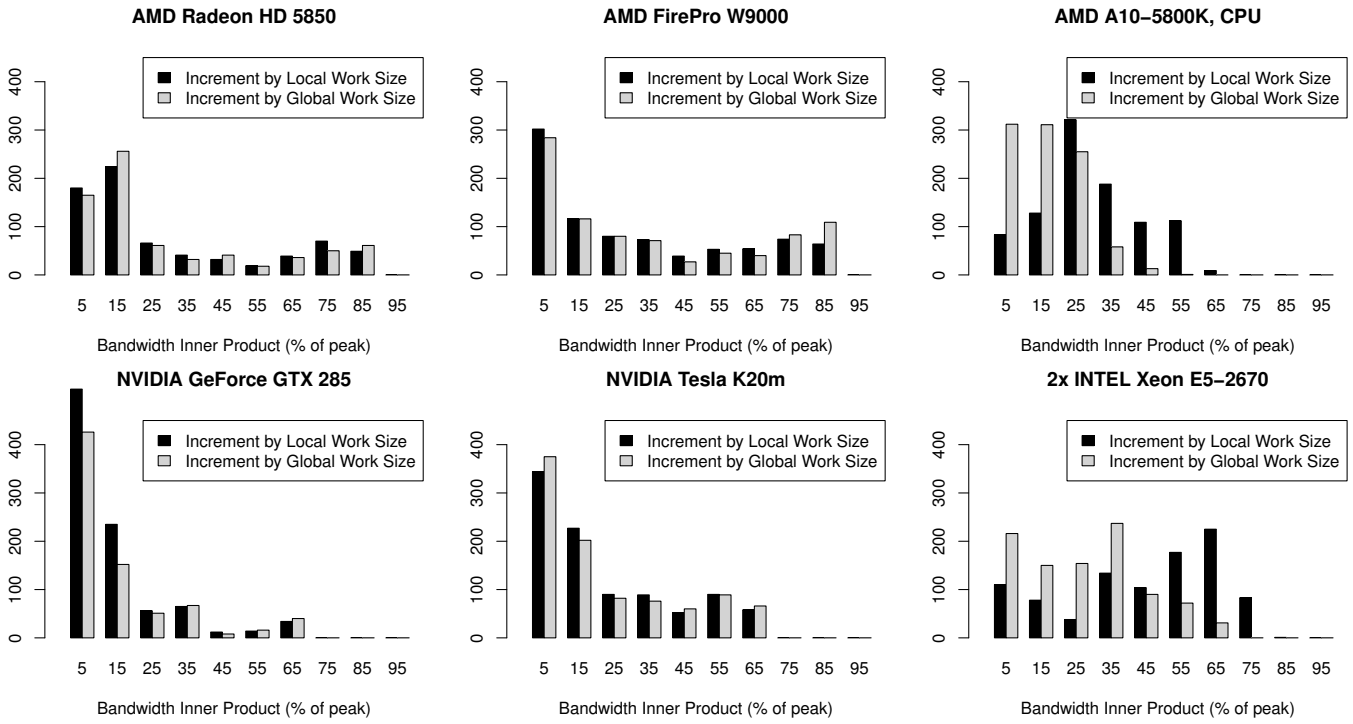

Figure 1: Frequency plots of the 1900 kernel configurations for the inner product operation. An increment by the local workgroup size is favorable over an increment by the global workgroup size for CPUs, here shown for a dual-socket INTEL Xeon E5-2670 machine. On GPUs an increment by the global workgroup size results in slightly better performance.

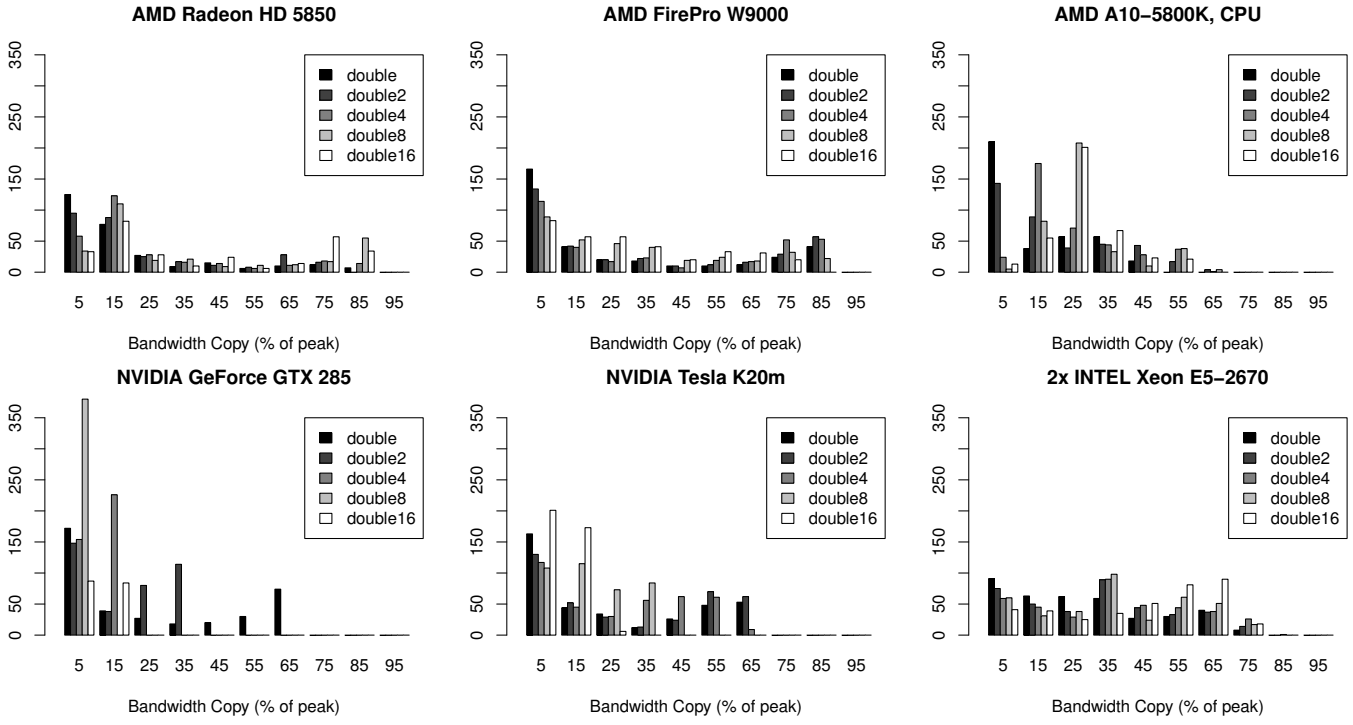

Figure 2: Frequency plots of the 1900 kernel configurations for the inner product operation investigating different vector data types. Although vector data types other than `double` occasionally result in higher performance, `double` shows the highest overall performance portability.

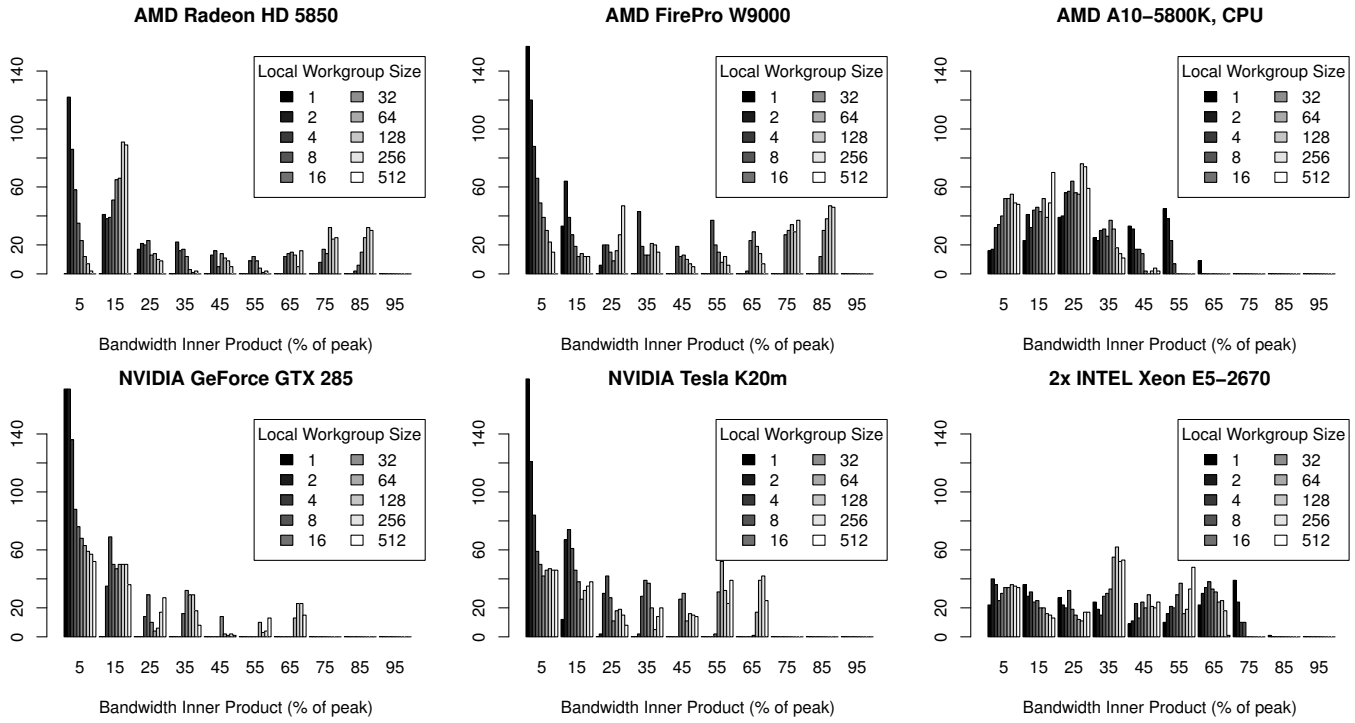

Figure 3: Frequency plots of the 1900 kernel configurations for the inner product operation investigating different local workgroup sizes. Local workgroup sizes of 128 and 256 provide the best overall performance on GPUs, while a local workgroup size of 1 is best for CPUs.

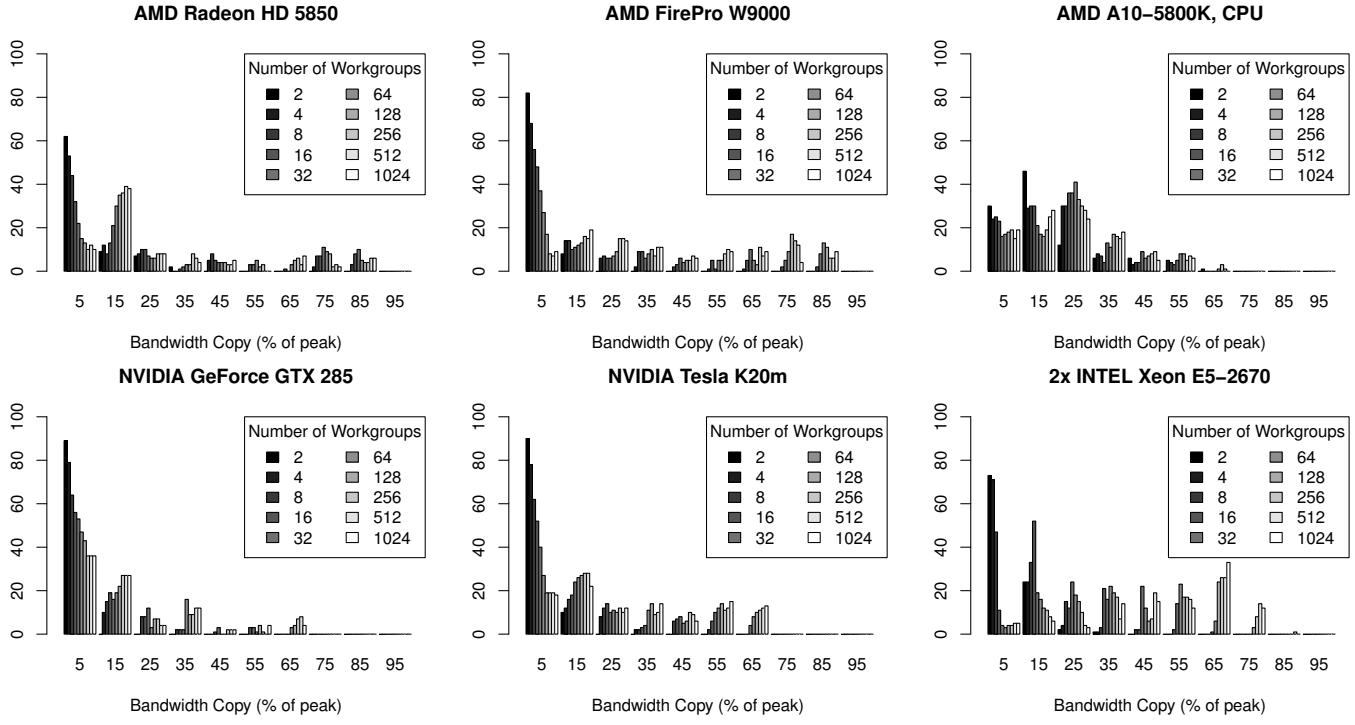

Figure 4: Frequency plots of the 1900 kernel configurations for the inner product operation investigating different number of workgroups. In contrast to other device types and GPUs from NVIDIA, GPUs from AMD do not necessarily favor a higher number of workgroups.

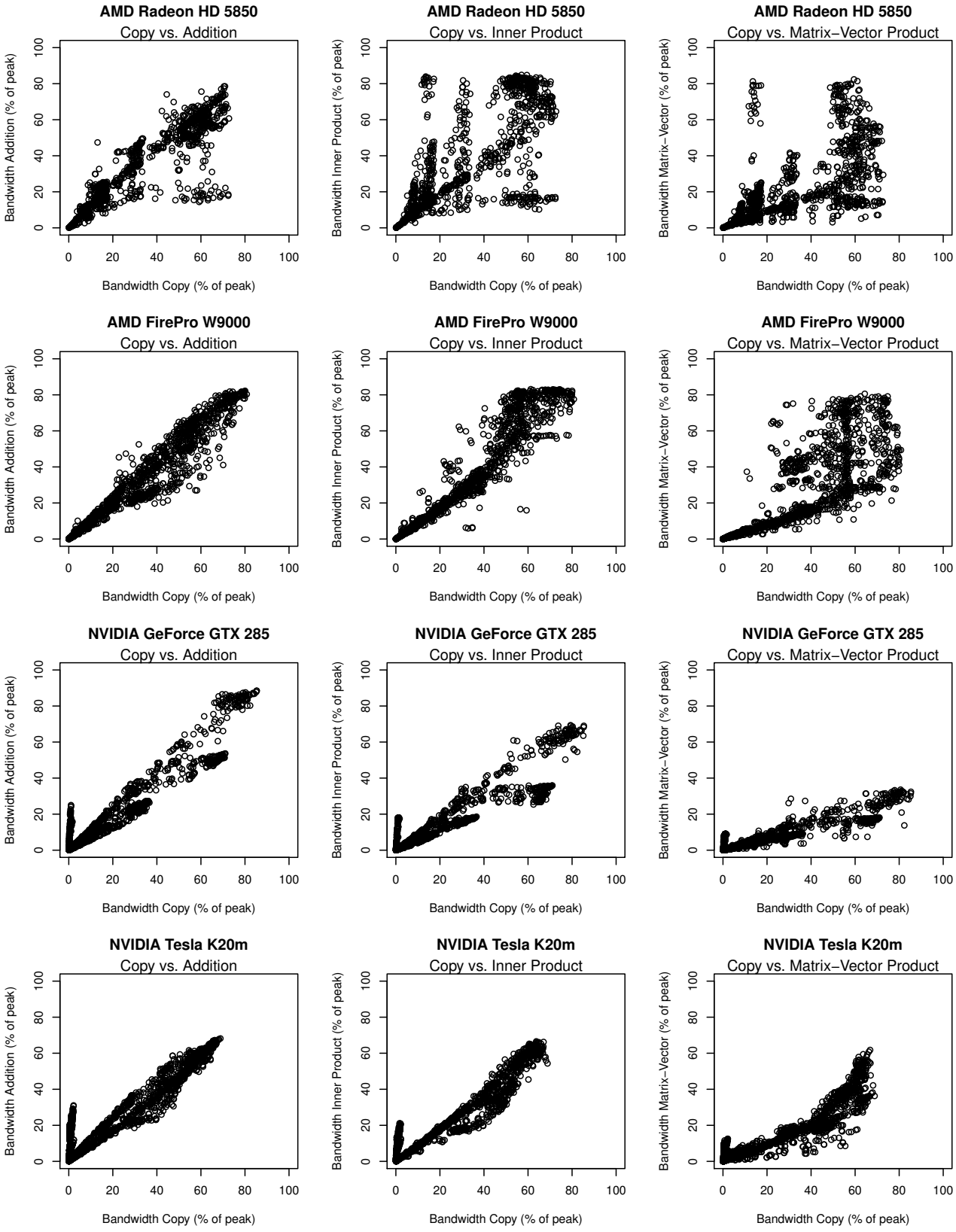

Figure 5: Comparison of kernel performances for GPUs from AMD and NVIDIA. Good performance for the copy kernel usually implies high performance for the other kernels on NVIDIA GPUs. Kernel configurations with high performance for the copy kernel are candidates for good performance of matrix-vector products on AMD GPUs.

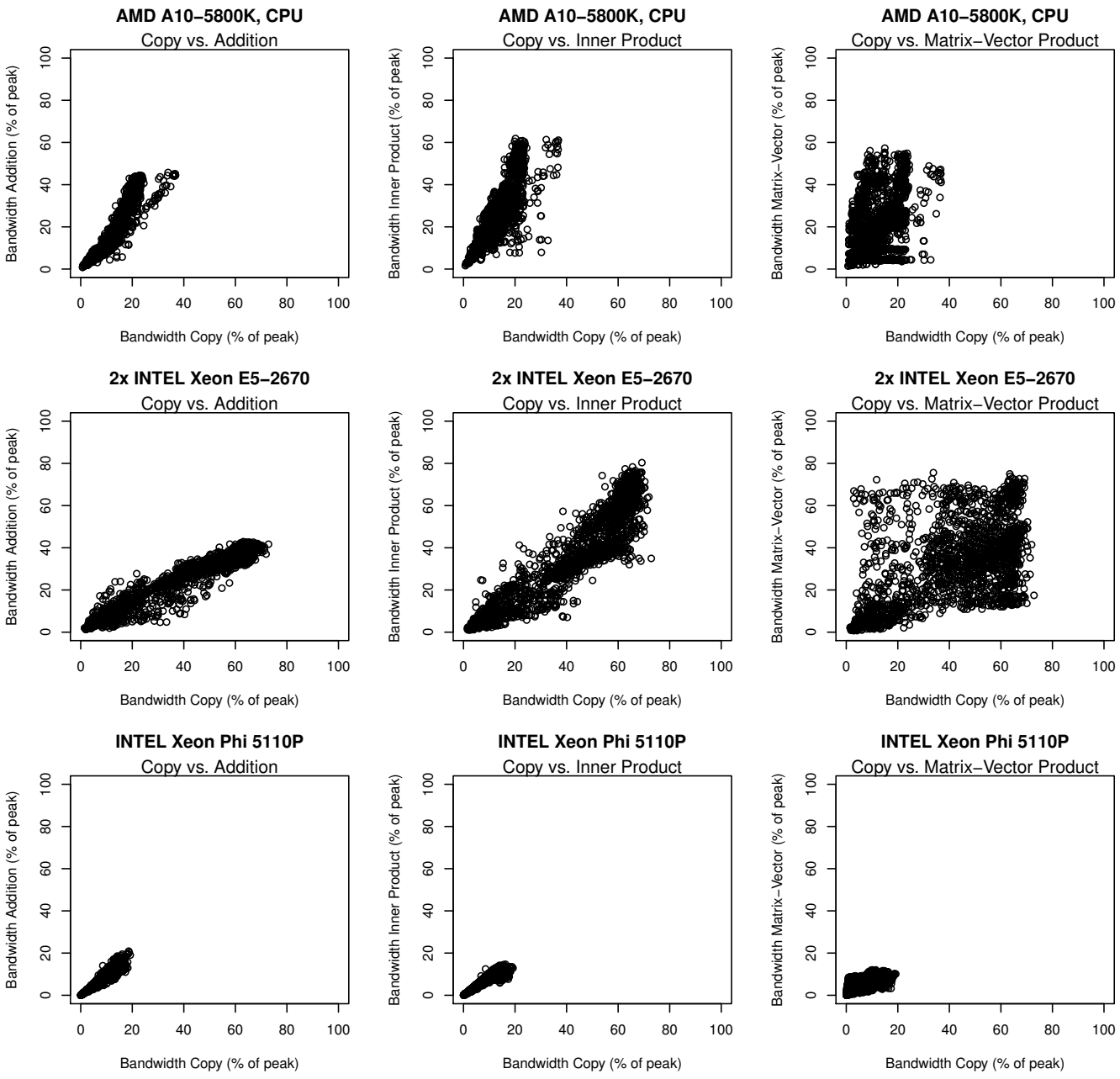

Figure 6: Comparison of kernel performances on an AMD A10-5800K CPU, a dual-socket INTEL Xeon E5-2670 machine, and the INTEL Xeon Phi. On AMD GPUs and the Intel CPUs one can use the copy kernel to obtain candidate kernel configurations for other operations. The performance correlation is smaller than for GPUs. Performance on the INTEL Xeon Phi when using OpenCL remains poor even after extensive autotuning.

performance correlation among the GPUs was found for the HD 5850: For example, configurations exist which obtain 75 percent of peak memory bandwidth for the copy operation, but only 20 percent of peak memory bandwidth for the addition despite of the strong similarity of the two operations. Regardless, by considering only configurations with more than 60 percent of peak memory bandwidth for the copy operation, high performance can also be obtained for the other operations.

Figure 6 show that a similar pattern holds true for CPUs, yet the correlation of performances for the different operations is particularly weak for the matrix-vector product. Still, the search space for tuning these operations can be substantially reduced to only those configurations providing good bandwidth for the copy operation. A similar correlation for the performance of the various configurations is obtained for the Xeon Phi, but the practical relevance is limited due to the low overall performance when using OpenCL.

3.3 Intra- and Inter-Vendor Portability

After clear performance correlations among the operations considered on the same device have been identified, the next step is to study the performance correlation across different devices. Figure 7 compares the benchmark results for the copy kernel for devices of the same vendor. The plot comparing the two AMD GPUs (HD 5850 vs. W9000) not only shows that multiple configurations exist which are fast on both devices, but also shows many more configurations in the upper left half than in the lower right half. This means that many configurations, which are slow on the older HD 5850, are significantly faster on the W9000, indicating that the newer hardware is more versatile. A similar picture is obtained when comparing the two NVIDIA GPUs (GTX 285 vs. K20m): The newer K20m often provides good performance for configurations which do not perform as well on the older GTX 285. The comparison of the INTEL CPU and the Xeon Phi even shows that the best configurations for one device are also among the best on the other device and vice versa.

An overview of the best configuration obtained on each of the devices and compared on all other devices considered in this benchmark is given in Table 1. The numbers demonstrate that it is not enough to tune on only one device, unless a performance penalty of at least $5 - 15\%$ on other devices is acceptable. Moreover, on closer inspection one can identify two groups, within which better performance portability is obtained: One group consists of the CPUs (A10-5800K, E5-2670) and the Xeon Phi. The second group contains the GPUs, within which better performance portability for hardware of the same vendor is observed.

While Figure 7 only compares benchmark results for the copy operation across different devices, Figure 8 compares the benchmark results of the copy operation on one device with the addition, inner product, and matrix-vector product on another device. About 40 configurations which are fast on both devices are found when comparing the HD 5850 and the GTX 285. A comparison of the more recent W9000 and K20m exhibits a much more pronounced correlation: Most configurations show about the same performance on both devices. However, the correlation is weaker when comparing the copy operation on the W9000 with the matrix-vector product on the K20m, yet a significant share of fast configurations for the copy operation on the W9000 is among the best configurations for the matrix-vector product on the K20m. A comparison of the A10-5800K CPU with the E5-2690 CPU shows that most configurations perform significantly better on the E5-2670 than on the A10-5800K. Again, it is possible to find configurations which are fast both for the copy operation on the A10-5800K and the addition, inner product, and matrix-vector product on the E5-2670.

3.4 Best Configurations

So far we have shown that among the best configurations for the copy kernel one can always find configurations with high performance for other operations. As is shown in Table 2, one may even pick a single configuration for each device, which then achieves almost optimal performance for all four operations. On all devices considered the performance is within 15 percent of the peak bandwidth for the respective operation. On NVIDIA and INTEL devices, the best configuration even results in performance within only five percent of the respective peak value. We also note that several configurations with almost identical performance are obtained for each device. For GPUs this also includes configurations without using any vector data types, which provide higher flexibility than vector data types.

4. SUMMARY & CONCLUSION

We investigated the performance of linear algebra operations on CPUs and GPUs of different vendors and found that kernel configurations, which work well for a simple vector copy operation, are also preferable for more complicated kernels. Moreover, we have shown that it is sufficient to only consider kernel configurations within 15 percent of the achievable peak performance for the copy operation in order to deduce fast configurations for more complicated operations such as inner products or even matrix-vector products. This strategy can even be applied across hardware from different vendors. Also, it has been observed that performance portability is better on newer hardware, which indicates that initial performance portability issues with OpenCL were also caused by immature hardware - and possibly software and drivers.

In summary, our study provides the following guidelines for OpenCL developers who aim to run their OpenCL-enabled software on a broad range of consumer hardware: For best performance, CPUs and GPUs should be considered separately. On CPUs, each work item should operate on large chunks of consecutive data and either very small or very large workgroup sizes should be considered. On GPUs, workgroup sizes of 128 and 256 are preferred. A larger number of workgroups is advisable on NVIDIA GPUs, whereas a number of about 128 already shows good performance on AMD GPUs. Vector data types for operations limited by memory bandwidth are not required on GPUs, but may provide mild benefits for CPUs. If, however, a performance reduction of about ten percent is acceptable, one may also use the same kernel structure for both CPUs and GPUs, reducing the overall implementation effort. Our benchmark results suggest that the best trade-off between performance and simplicity for memory bandwidth limited operations is to let each work item operate on consecutive data, to use 128 or 256 work items per workgroup, to execute the kernel with least 128 workgroups, and to ignore vector data types.

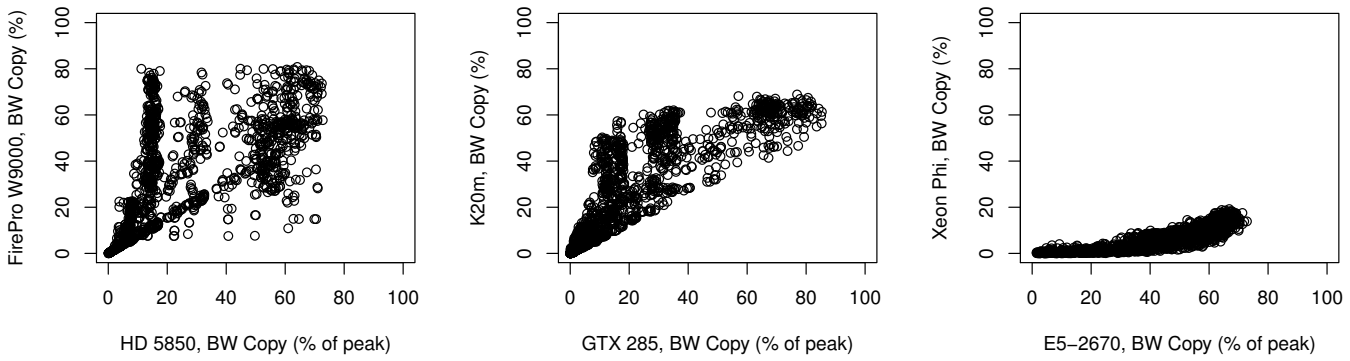

Figure 7: Comparison for the relative bandwidth (BW) of the copy kernel on hardware of the same vendor. A strong correlation is observed for devices from NVIDIA and INTEL, while the correlation on AMD hardware is weaker. In particular, good performance for the copy operation on one device has a high probability for good performance on the other device.

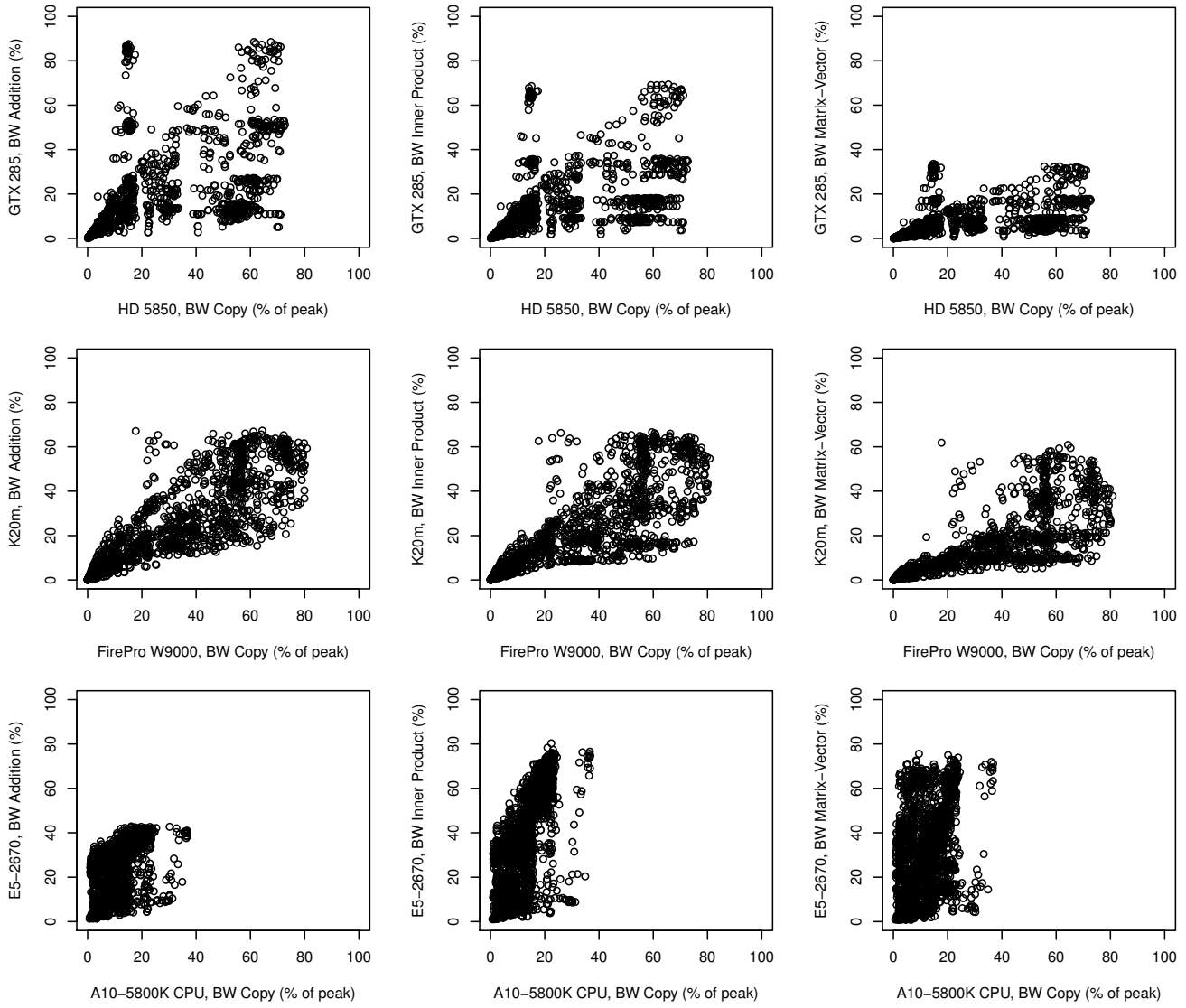

Figure 8: Correlations of the relative bandwidth (BW) for kernel configurations of the copy kernel on GPUs and CPUs from AMD compared with the vector addition, inner product, and matrix-vector product kernels on NVIDIA GPUs and INTEL E5-2670 CPUs. Some of the fastest kernel configurations for the copy kernel also provide close-to-peak performance on the respective other device for all three operations.

	Best Kernel Configuration for						
	A10-5800K	HD 5850	W9000	GTX 285	Tesla K20m	E5-2670	Xeon Phi
AMD A10-5800K CPU	**36.8** %	4.3 %	2.7 %	2.3 %	12.7 %	18.0 %	23.0 %
AMD Radeon HD 5850	0.1 %	**72.7** %	64.1 %	61.3 %	55.5 %	14.1 %	4.2 %
AMD FirePro W9000	4.4 %	73.1 %	**80.1** %	77.7 %	46.2 %	30.3 %	1.2 %
NVIDIA GeForce GTX 285	2.8 %	67.9 %	73.3 %	**85.3** %	76.7 %	14.2 %	0.2 %
NVIDIA Tesla K20m	9.5 %	47.1 %	50.5 %	61.3 %	**68.8** %	33.1 %	1.6 %
INTEL Xeon E5-2670 (dual)	65.6 %	38.1 %	41.5 %	37.1 %	58.6 %	**72.9** %	66.8 %
INTEL Xeon Phi	14.1 %	3.2 %	7.7 %	4.6 %	8.6 %	13.9 %	**19.2** %

Table 1: Memory bandwidth relative to the theoretical peak bandwidth for the vector copy operation. Performance penalties among the four GPUs (HD 5850, W9000, GTX 285, K20m) in the order of 5-15% relative to peak are observed if the best configuration for one device is used on the other. The penalty is about the same for the two CPUs (A10-5800K, E5-2670) and the Xeon Phi.

	A10-5800K	HD 5850	W9000	GTX 285	Tesla K20m	E5-2670	Xeon Phi
Increment Type	*local*	*global*	*global*	*global*	*global*	*local*	*local*
Vector Length	2	8	4	1	2	4	16
Local Work Size	1	128	64	128	256	1	1
Workgroups	256	1024	160	80	1024	512	512
BW Copy	36.7 (36.8)	59.5 (72.7)	73.9 (80.8)	85.3 (85.3)	66.8 (68.9)	69.5 (73.1)	18.8 (19.2)
BW Addition	45.2 (45.8)	61.5 (78.5)	77.5 (82.2)	88.5 (88.5)	67.1 (68.2)	40.1 (43.0)	20.3 (20.9)
BW Inner Product	60.3 (61.9)	84.8 (84.9)	82.7 (83.1)	69.0 (69.4)	62.6 (66.6)	80.6 (80.6)	13.5 (14.7)
BW Matrix-Vector	47.2 (57.3)	82.4 (82.4)	77.3 (80.6)	32.4 (33.6)	61.8 (61.8)	73.0 (75.9)	10.3 (12.0)

Table 2: Summary of the best configurations in terms of average memory bandwidth for each device. The memory bandwidth (BW) reported for each operations is given in percent and relative to the theoretical peak. Values in parentheses denote the best bandwidth among the 1900 configurations. The bandwidth obtained is in all cases within 15 percent of the achievable peak value, for NVIDIA and INTEL devices even within five percent.

Acknowledgments

K. Rupp, T. Grasser, and A. Jüngel gratefully acknowledge support from the Austrian Science Fund (FWF), grant P23598. F. Rudolf and J. Weinbub have been supported by the European Research Council (ERC) through the grant # 247056 MOSILSPIN. The authors thank Joachim Schöberl (TU Wien) for providing access to a machine equipped with NVIDIA Tesla K20m GPUs. Moreover, the authors are grateful to AMD for providing a FirePro W9000.

5. REFERENCES

[1] M. M. Baskaran and R. Bordawekar. Optimizing Sparse Matrix-Vector Multiplication on GPUs. *IBM RC24704*, 2008.

[2] N. Bell and M. Garland. Implementing Sparse Matrix-vector Multiplication on Throughput-oriented Processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 18:1–18:11, New York, NY, USA, 2009. ACM.

[3] P. Du, R. Weber, P. Luszczek, S. Tomov, G. Peterson, and J. Dongarra. From CUDA to OpenCL: Towards a Performance-Portable Solution for Multi-Platform GPU Programming. *Parallel Computing*, 38(8):391 – 407, 2012.

[4] K. Goto and R. A. van de Geijn. Anatomy of High-Performance Matrix Multiplication. *ACM Transactions on Mathematical Software*, 34(3):1–25, 2008.

[5] J. Kurzak, S. Tomov, and J. Dongarra. Autotuning GEMM Kernels for the Fermi GPU. *IEEE Transactions on Parallel and Distributed Systems*, 23(11):2045–2057, 2012.

[6] ATLAS Library. `http://math-atlas.sourceforge.net/`.

[7] K. Matsumoto, N. Nakasato, and S. G. Sedukhin. Implementing a Code Generator for Fast Matrix Multiplication in OpenCL on the GPU. In *6th IEEE International Symposium on Embedded Multicore SoCs (MCSoC-12), 2012*, pages 198–204, 2012.

[8] S. McIntosh-Smith, J. Price, R. B. Sessions, and A. A. Ibarra. High Performance in Silico Virtual Drug Screening on Many-Core Processors. *International Journal of High Performance Computing Applications*, 2014. In press.

[9] R. Nath, S. Tomov, and J. Dongarra. An Improved MAGMA GEMM For Fermi Graphics Processing Units. *International Journal of High Performance Computing Applications*, 24(4):511–515, 2010.

[10] OpenCL. `http://www.khronos.org/opencl/`.

[11] S. J. Pennycook, S. D. Hammond, S. A. Wright, J. A. Herdman, I. Miller, and S. A. Jarvis. An Investigation of the Performance Portability of OpenCL. *Journal of Parallel and Distributed Computing*, 73(11):1439 – 1450, 2013.

[12] S. Rul, H. Vandierendonck, J. D'Haene, and K. De Bosschere. An Experimental Study on Performance Portability of OpenCL Kernels. In *Application Accelerators in High Performance Computing*, page 3, 2010.

[13] K. Rupp, F. Rudolf, and J. Weinbub. ViennaCL - A High Level Linear Algebra Library for GPUs and Multi-Core CPUs. In *Proceedings of the International Workshop on GPUs and Scientific Applications (GPUScA 2010)*, pages 51–56, 2010.

[14] P. Thoman, K. Kofler, H. Studt, J. Thomson, and T. Fahringer. Automatic OpenCL Device Characterization: Guiding Optimized Kernel Design. In Emmanuel Jeannot, Raymond Namyst, and Jean Roman, editors, *Euro-Par 2011 Parallel Processing*, volume 6853 of *Lecture Notes in Computer Science*, pages 438–452. Springer Berlin Heidelberg, 2011.

[15] Ph. Tillet, K. Rupp, S. Selberherr, and C.-T. Lin. Towards Performance-Portable, Scalable, and Convenient Linear Algebra. In *5th USENIX Workshop on Hot Topics in Parallelism (HotPar'13)*, 2013.

[16] Vienna Computing Library (ViennaCL). `http://viennacl.sourceforge.net/`.

[17] R. C. Whaley, A. Petitet, and J. Dongarra. Automated Empirical Optimizations of Software and the ATLAS Project. *Parallel Computing*, 27(1-2):3–35, 2001. New Trends in High Performance Computing.

[18] Y. Zhang, M. Sinclair, and A. Chien. Improving Performance Portability in OpenCL Programs. In J. M. Kunkel, T. Ludwig, and H.-W. Meuer, editors, *Supercomputing*, volume 7905 of *Lecture Notes in Computer Science*, pages 136–150. Springer Berlin Heidelberg, 2013.