Exploiting Heterogeneous Computing Platforms By Cataloging Best Solutions For Resource Intensive Seismic Applications

Thomas Grosser, Alexandros Gremm, Sebastian Veith,
Gerald Heim, and Wolfgang Rosenstiel
University of Tübingen, Germany
{*tgrosser,gremm,veith,heim,rosenstiel*}*@informatik.uni-tuebingen.de*

Victor Medeiros and
Manoel Eusebio de Lima
Federal University of Pernambuco, Brazil
{*vwcm,mel*}*@cin.ufpe.br*

***Abstract*—Large heterogeneous data centers of today lack methods to appraise the best fitting solutions regarding, among others, hardware acquisition cost, development time, and performance. Especially resource intensive applications benefit from increased data center utilization to leverage heterogeneous resources and accelerators. In this paper, we implement various methods to accelerate a seismic modeling application, which is available for CPU, GPU, and FPGA. With the underlying heterogeneous environment, the current programming standard OpenCL is examined regarding CPUs and GPUs, and compared to traditional acceleration approaches in order to evaluate sets of platforms. Based on the variety of available versions, a flow is introduced, which allows to catalog best solutions by experimenting with different implementations for available hardware platforms. We encourage to derive indicators as hints for data center operators with respect to finding a cost-benefit trade-off, which must also be observed over time. The results highlight the GPU and FPGA implementations, and correlate performance optimizations with development time, regarding the seismic application and the underlying hardware platforms.**

***Keywords*-heterogeneous computing platform, accelerator, seismic exploration, OpenCL, CPU, GPU, FPGA**

I. Introduction

Large data centers consist of heterogeneous combinations of hardware resources, accelerators, operating systems, compilers, software libraries, and APIs. In this paper, we explore a multiplicity of heterogeneous resources guided by a workflow to track and evaluate the best solution for a given application. This enables exploration of varying sets of hardware platforms, as well as different implementations and optimizations regarding hardware accelerators. We choose a resource-intensive seismic modeling application, which is both compute and data-intensive. The parallelization of seismic modeling can be implemented on many different types of machines like CPUs, GPUs, and FPGAs [1]. This is ideal for a mixed heterogeneous data center as each architecture exhibits facilities to parallelize stencil operations regarding various multi-core architectures [2]. In the course of exploring heterogeneous computing platforms, we are especially interested in the OpenCL [3], [4] compute standard, as it promises to provide an abstraction from the underlying hardware. Our intention is to exploit and evaluate heterogeneous data centers with a set of available implementations, rather than implementing novel optimization schemes for seismic processing. To appraise the best fitting platform for the given application, a flow is introduced that allows to catalog the manifoldness of available solutions along with additional information about the underlying hardware platform and operating experiences. Based on our experiments, we exemplify how to derive indicators that correlate achieved performance with development time. The application of the proposed catalog allows to reuse programming and operating experiences, and also to correlate several parameters for multiple platforms in order to support future decision making processes for data center operators.

The rest of the paper is structured as follows: Section II reviews heterogeneous resources with respect to programming seismic modeling. In the following Section III, we demonstrate the intensiveness of the seismic application, and delve into the implementation on various platforms in Section IV. After describing the implementation and optimization for specific platforms, we introduce the workflow in Section V, which is used to build the catalog incorporating various platforms and versions. Section VI presents the results regarding performance and discusses the development time of particular implementations. We close this work by providing an outlook to future work, followed by concluding remarks in Sections VII and VIII, respectively.

II. State of the Art

As technology of processors is scaling down, manufacturers are moving from high-frequency designs to multi-core chips, instead of improving single-threaded performance. Recent processors, like IBM's Power7 or Intel's Core i7, implement four up to eight cores per processor, whereas each core may provide multiple threads in hardware. Besides programming CPUs, general-purpose computation on graphics processing units (GPGPUs) has become increasingly popular as a flexible, cost-competitive alternative [5]. Other studies show that state of the art multi-core CPUs are able to compete with GPUs, by exploiting single instruction multiple data processing (SIMD), multi-threading, and cache blocking techniques. For example, by particularly tuning a convolution algorithm on a CPU, it was shown that the processing is only about 2.8 times slower than on a GPU [6].

The recent heterogeneous compute standard OpenCL [3], [4] is up-and-coming to exploit heterogeneous platforms and promises the development of compute kernels independent from the underlying hardware. As an open standard for heterogeneous computing, we investigate OpenCL for both GPUs and CPUs. Different vendors meanwhile offer OpenCL implementations, e.g., Nvidia [7], AMD/ATI [8], IBM [9], and recently Intel [10], targeting GPGPUs, CPUs, and the combination thereof. Unlike software, fine-grained arrays, such as FPGAs, allow to implement custom pipelines that makes them extremely efficient and also hard to program on the other hand. Programming FPGAs using OpenCL is a matter of research today, i.e., research activities exist to use OpenCL as a high-level abstraction for FPGA accelerators [11]. Meanwhile, the hardware designer has to design and implement a hardware architecture specific to the algorithm, which is more costly in terms of development time as compared to software engineering. On the other hand, FPGAs are a powerful alternative because of low power consumption for certain applications [12], as they are running at moderate frequencies.

The seismic modeling algorithm is an embarrassingly parallel problem, which means that it can easily be divided into subproblems. Thus, the algorithm can be efficiently implemented on various platforms like CPUs, GPUs, FPGAs [1], and Cell/B.E. [13]. Even more exploitation of parallelism is also possible by leveraging a cluster of GPUs [14]. So, due to the paradigm shift from high-speed sequential to massively parallel processing, the acceleration of seismic modeling fits for many recent multi-core and many-core platforms. In this context, the best performance can be achieved if an appropriate accelerator is provided, whereas finding the best *solution*, with regard to a cost-benefit trade-off, requires consideration of additional parameters, e.g., hardware acquisition cost, development time, and power consumption.

III. Seismic Exploration Application

Seismic exploration of oil can be divided into three areas: data acquisition, data processing, and data interpretation. The acquisition is responsible for capturing seismic traces by geophones, through the injection of an excitation source (seismic pulse) into the Earth. The processing step includes various algorithms such as Kirchhoff [15] and reverse time migration (RTM) [1], which enables to extract the hidden information obtained from seismograms. This step essentially comprises the seismic modeling and migration stages that operate on a previously developed Earth model, i.e., an acoustic velocity model. In the interpretation stage an image, which represents several geological layers, is finally analyzed by experts. The development of this work focuses on the processing stage, particularly the seismic modeling, which will be referred to as *forward propagation* throughout the rest of this paper.

A. Intensity

In our research of the data intensiveness, typical operations on cubical Earth models with the dimensions of $1250 \times 250 \times 2500$ points results in $781,250,000$ points to be processed for each time step during the RTM algorithm execution. As the RTM algorithm execution consists of the forward and reverse propagation this number has to be doubled, which results in $1,562,500,000$ points.

In the field, typically $100,000$ shots are recorded by multiple receivers per survey. For each shot, the RTM algorithm runs for at least $10,000$ time-steps and all points must be calculated individually in each time step. In this scenario we conclude that 10^9 operations for each point are necessary. For simplicity, we omit that there may be multiple refinement steps performed by the geologist, which repeats the computation of the seismic exploration.

Assuming that the calculation of a single point requires about 37 floating point operations (FLOPs), it becomes clear that this results in $57,812,500$ TFLOPs for the cubical input data. Regarding data storage, the total amount of raw data of the cubical input, with respect to 4 bytes per float value, results in 3.125 GB. As the algorithm requires an additional cubical data set as temporary data buffer, two cubes have to be stored for each shot resulting in a total amount of 625 TB data for one survey. To partition the intensiveness of this application, the cubical input data can be decomposed into several subcubes in order to be processed on multiple heterogeneous machines of the data center simultaneously.

IV. Implementation on Various Architectures

The forward propagation algorithm is essentially an imaging algorithm that moves a stencil operator over a matrix. This algorithm exhibits regular memory access patterns and thus allows to be performed in SIMD fashion. As the input data can be split into several blocks, the algorithm can easily be executed by multiple threads, processing a number of subproblems in parallel. Considering the FPGA, there is the possibility to improve the throughput by implementing a pipeline of processing elements, which realizes increased parallelization in the time domain of the forward propagation.

A. CPUs

The basic calculation of the forward propagation on 2-dimensional data is shown in Figure 1. The main computation is contained in two nested loops processing all entries of the input matrices. These two nested loops are referred to as the spatial loop, which is working on 2-dimensional data (the operation can be extended to operate on 3-dimensional data [1], [2]). In order to simulate the excitation source, a 1-dimensional array is used (`seismicPulseVector`).

```
// time loop
for (t=0; t < timeSteps; t++) {
  // inject seismic pulse into 'actual pressure field'
  APF[spPosX][spPosY] += seismicPulseVector[t];
  // 2D spatial loop, moving stencil
  for (i=2; i < (dimX-2); i++) {
    for (j=2; j < (dimY-2); j++) {
      // compute 'next pressure field'
      NPF[i][j] = apply_stencil(i, j, APF, PPF, VEL);
    }
  }
  // switch pointers to buffers
  PPF = APF, APF = NPF, NPF = PPF;
}
```

Figure 1. Pseudo code implementing the 2-dimensional forward propagation.

The spatial loop itself is nested inside the time loop, which performs the wave propagation over time. For simplicity, the calculation of the wave propagation equation is performed by the function `apply_stencil`. This function uses the previous pressure field matrix `PPF` and the actual pressure field matrix `APF` to calculate the next pressure field matrix `NPF` using the Earth model, which represents the wave propagation velocity `VEL` of different layers of soil, e.g., sand or stone. Once one time step is done the pointers to the pressure field's matrices are switched to omit unnecessary copying.

1) Pthreads: By partitioning the pressure fields into squares, the data is shared among multiple threads as depicted in Figure 2(a). As threads operate on shared memory, there is no additional memory transfer overhead when operating on overlapped regions. When moving the stencil, the operating thread for a specific square should be scheduled to the same core to prevent cache misses. For this purpose, the pthreads library offers functions that enable the exploitation of data locality by pinning threads to a specific core.

The main thread of the application calculates essential offsets for each thread, in order to access the pressure field matrices. That way, each thread reads valid memory locations from `PPF` and `APF` and writes the results into the `NPF` without the need of extra synchronization. Once one time step is calculated, two barriers are needed to safely switch the pointers to the pressure fields `PPF`, `APF`, and `NPF`.

2) Single Instruction Multiple Data: As the memory access pattern of the seismic algorithm is regular, the stencil uses data from nearest neighbors, i.e., there are no scatter/gather operations needed. Therefore, the algorithm exhibits ideal prerequisites to a straight-forward SIMD implementation. So, the stencil is extended to compute four points in parallel, as shown in Figure 2(b). This is achieved by loading four consecutive float values into 128-bit wide vector registers to enable SIMD processing. While AltiVec on POWER machines provides intrinsics for aligned loads, which results in shorter loading time as data is aligned to 16-byte boundaries, SSE on x86 machines provides intrinsics

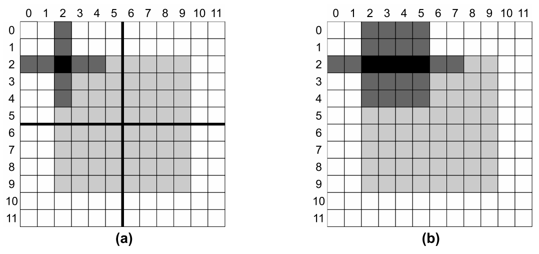

Figure 2. (a) Single stencil operator on decomposed matrix. (b) 4-way stencil operator.

for both aligned and unaligned loads. For performance and comparability reasons, we always implement the stencil operation using aligned loads. Reconsidering the 4-way stencil, an alignment offset has to be introduced as the algorithm starts accessing the array at the third element, depicted in Figure 2(b). We extend the input data by two entries, which does not affect the calculation and effectively enables padding, so aligned loads become possible. As a side effect of SIMDizing the algorithm, the widths of the pressure fields must be divisible by 4.

B. OpenCL on CPUs

An OpenCL system consists of the host (CPU) and multiple devices, e.g., a GPU or the CPU itself. In turn, one device itself contains multiple compute units (CU), that execute one or more work-groups. The host is responsible to launch compute kernels, which are mapped to work-items that are moreover arranged as groups to allow scheduling blocks of threads. Regarding CPUs, OpenCL aims to provide a portable vector abstraction by introducing data types like `float4`. That is, the mapping of data to vectorized registers is subject to the vendor's compiler. By using these data types, the OpenCL compiler for CPUs should be able to exploit SIMD for individual work-items and map multiple work-groups to threads. So, our approach to use OpenCL on CPUs is to develop a kernel that is suited for CPU devices.

In the OpenCL memory hierarchy, there is the notion of global and local device memory. In case of the CPU, global and local device memory reside in the CPU's main memory, hence using global memory accesses inside the kernel are sufficient on CPUs. Loading data into local memory first is counterproductive as this may result in hidden memory operations. Considering a quad-core CPU, we launch four work-groups each containing a single work-item that processes a subset of the data in a loop.

C. GPGPU Implementation

In contrast to the CPU implementation, the computations are performed by a multitude of work-items, which map to the stream processors of the GPU. Due to the architecture of the GPU, there are typically hundreds of threads to exploit massive parallelism, and unlike the CPU implementation, there are typically no loops in a GPU kernel. When launching a kernel, the host determines the number of work-groups according to the dimension of the input data, while a single work-group contains 16×16 work-items that operate in parallel. One work-group is then mapped to one compute unit of the OpenCL device.

With regard to the OpenCL memory hierarchy, the local memory is shared for a work-group, i.e., all work-items have fastest access to. As a rule of thumb, access times for the GPU's local and global memory are of similar order of magnitude as compared to access times of the CPU's cache and memory, respectively.

Under control of the OpenCL runtime, work-groups are scheduled on the CUs. If there are fewer groups than CUs, the device may not be utilized completely. Therefore, the number of work-groups must be maximized for one OpenCL device. By launching as many work-groups as possible, the OpenCL runtime schedules threads independently on the CUs, which in effect hides memory latencies. According to the OpenCL standard, the processing order of neither work-items nor work-groups can be influenced by the programmer directly. Work-groups must be synchronized by the host for each time step in our implementation. So, each time step the host invokes the kernel, which calculates the spatial domain in parallel. This applies to both CPUs and GPUs. In the following, we describe two different compute kernels, as we are also exploiting special hardware facilities common to GPUs.

1) Default Kernel: As the default implementation of a GPU-aware kernel, data values are loaded from GPU global memory to the CU's local memory at first. Each work-item and work-group has unique ids to compute offsets to global and local memory buffers, which are given as argument to the kernel. After staging global memory to local memory, work-items operate on the low-latency local memory for all operations performing the wave propagation. To avoid unnecessary memory accesses between host and device memory, temporary results remain in the GPU's global memory buffers. The host performs the outer loop in the time domain, switches pointers to memory buffers, and sets kernel arguments accordingly before initiating the compute kernel for the next time step.

2) Image Kernel: Another approach to operate on data is to use image objects provided by the OpenCL API. On a GPU device these images reside in the texture memory, which corresponds to a specialized cache optimized for spatial locality access. Images must be declared read-only and write-only, which applies to the input buffers (`PPF` and `APF`) and output buffer (`NPF`) respectively. As kernel arguments, two image buffers for the input and one for the output are used, which is different from using simple array-like buffers as in the default kernel. From the programmers point of view, the kernel code is more obvious, because the calculation of required addresses and offsets to read data from is done by the OpenCL runtime using a so called *sampler*. The sampler specifies how to access the data inside images, also specifying how to behave at borders. The GPU hardware has natural limits here, e.g., our Tesla system allows images to a maximum size of 8192×8192. However, this was not exceeded in our experiments so far. Once the problem is getting bigger, the algorithm has to be refined by splitting data accordingly.

The functions to read images return data directly into a `float4` variable. So, the kernel code using images is similar to the SIMD implementation, as vectors containing four values are processed by one work-item, incorporating fastest loading times due to image objects. As GPUs are designed to process data this way, we expect the image kernel to yield even more performance that the default kernel.

D. Implementation on FPGAs

The current design of the FPGA implementation is a pipeline-based stream processing architecture composed by a set of processing elements (PEs) that implement the wave propagation equation. These processing elements operate on single precision floating point data. Due to memory bandwidth and FPGA internal resource constraints, four PEs are instantiated inside the processing core. The solution can be optimized when using a more powerful FPGA as the number of PEs in the architecture can be increased easily. However, due to the great amount of data in the algorithm, the memory bandwidth is already a bottleneck regarding four PEs. So, it is not possible to increase the number of PEs exploring the spatial domain parallelization, but it is possible to allocate more PEs to explore the time domain parallelization, i.e., processing more time steps concurrently. This approach allows increasing the computational power without the requirement of a larger memory bandwidth. Other possible optimizations, like data compression techniques or different floating point precisions, can also be explored. This is a great advantage over other platforms like CPUs and GPUs, which do not feature such comparable customizations at the lower-level architecture.

In our current approach, we focus on modeling optimizations to trade-off performance benefits before actually implementing them. In order to validate further refinements of the architectural design, a software model of the FPGA architecture is developed, which estimates the system performance when implementing several improvements. Experimental results show that the model yields over 99% accuracy, and is therefore also considered in the course of improved heterogeneous data center exploitation.

V. Cataloging Best Solutions

Evaluation of all implementations on all available machines results in an ample amount of potential solutions. A solution consists of a specific machine with associated versions of the implementation. In order to evaluate scalability scenarios, for example by increasing the input data, series of experiments have to be performed, which are stored in a set of configurations. Due to the manifoldness of implemented versions and configurability options, we propose the elementary flow shown in Figure 3, which manages the complexity of different sets of machines, available versions, and applied configurations. Since the catalog is intended to enable inspection of multiple parameters, we focus on a correlation of different hardware platforms with development time in this experiment. Considering future applications, the catalog is intended to add additional dimensions, i.e., correlating hardware platforms with different classes of parallel problems besides the seismic application.

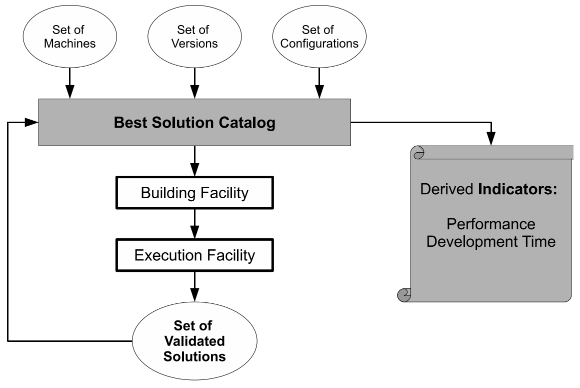

Figure 3. Flow to automate exploration of heterogeneous data centers by building a catalog containing best solutions.

A. Building and Execution Facilities

The building facility is aware of the underlying machine, its capabilities, and associated versions. Additional hosts can be added easily as build rules are generalized for each version. According to the available host and its capabilities, we are able to run all of the following:

- sequential C code,
- multi-threaded code,
- SIMD intrinsics (both SSE[1] and AltiVec[2]),
- threads and SIMD combined,
- OpenCL on CPUs[3],
- OpenCL on GPGPUs[4],
- FPGA[5] through a host CPU.

Before execution, the facility checks for available builds and executes those that are listed in the specified scenario, including a set of configuration files. As part of the execution facility, a shared library is implemented to read configuration files and set parameters in the application accordingly. This allows to observe implications of parameter changes, e.g., increased input sizes or number of threads.

B. Catalog with Indicators

After executing the scenario on a certain machine, all versions are summarized into a CSV file. This allows to manage different scenarios on various machines, comparing each other. The current functionality locates the most-optimal solution for a given host and problem size, and adds these to a set of validated solutions.

Relevant information about the underlying machine and additional operating experiences are stored in the catalog. So, the catalog allows to archive snapshots of different solution to recapitulate which specific implementation performed best on a given accelerator. This enables guidance for data center operators when deploying specific implementations to a machine. In our experiments, we consider the development time of specific solutions in correlation with the achieved performance.

VI. Results

In the current state of our work, we perform multiple runs based on specific configurations and extract runtime information to find the best solution regarding performance at first. This includes scaling input sizes from 600×748 to 2300×748 points for all versions. Multi-threaded versions are executed multiple times with a varying number of threads. After finding the best solutions with respect to performance, we are able to correlate that with development time and feed back this information into the catalog.

A. Results on CPUs

On the CPUs in our data center, we evaluate the achieved speed-up of SIMD-enabled and multi-threaded versions compared to the single-threaded sequential version. When OpenCL is available, we compare this to the version running SIMD and threads combined.

1) SIMD: As the SIMD processing implements the 4-way stencil (shown in Figure 2(b)), the theoretical speed-up is limited to 4. On the Intel Xeon E5405, we observed that SIMD-enabled processing achieves an average speed-up of 2.12, while on the JS21 blade an average speed-up of 2.35 is achieved. For both machines, the maximum speed-up of 2.5 is achieved with the largest input data set of 2300×748.

2) Threads: The evaluated Intel Xeon E5405 and JS21 blade hosts exhibit four cores, thus there is also a theoretical speed-up limited to 4. As the computation is not bandwidth-bound, we observe that launching more threads than cores can slightly increase performance for certain input sizes. That is, the maximum achieved speed-up is 3.38 on the Intel Xeon E5405 machine with the input image dimension of 1600×748.

[1]Intel Xeon E5405 and AMD Athlon 64 X2 6000+
[2]IBM PPC970MP (JS21 blade)
[3]AMD Athlon 64 X2 6000+
[4]Nvidia Tesla T10p
[5]Altera Stratix III 80E

3) Combinations: Upon evaluating all input sizes stated above, we observe no additional significant performance gains using the combined versions (threads/SIMD) on the Intel Xeon E5405 machine. However, on the JS21 blade the threads/SIMD solution achieves a total speed-up of 3.38 using four threads and even 4.83 using 12 threads, as compared to the plain SIMD or multi-threaded versions, which only yield an average speed-up of 2.38.

B. OpenCL on CPUs

The AMD Athlon 64 X2 6000+ machine is the only machine equipped with an OpenCL runtime environment to run kernels on the CPU. Hence, we compare execution times of the combined threads/SIMD version to the OpenCL kernel, which is effectively mapped to SIMD intrinsics and threads by the OpenCL compiler. In our evaluation, the OpenCL-enabled solution runs even faster than the traditional approach. When comparing the threads/SIMD version with four threads, the OpenCL equivalent code runs 1.31 times faster. With an overcommitment of 12 threads, the OpenCL solution is 1.23 times faster.

C. Results on GPGPUs

When evaluating the two implemented GPU kernels to the sequential code, we observe speed-ups of 16.81 and 31.23 regarding the default kernel and image kernel, respectively. As it is intend to locate the best solution, the comparison of the single-threaded sequential CPU code to the highly parallel GPU version is not legitimate. So, we also compare the two GPU kernels to the best solution of the Intel Xeon E5405 machine, which is the combined usage of threads/SIMD. In this case, the speed-ups are 6.63 and 12.31, regarding the default and image kernel, respectively.

D. FPGA and GPGPU

When comparing the GPU with the FPGA, it must be stated that the comparison is to handle with care, as architecture-specific optimizations cannot be compared easily. However, with the intention to derive indicators to find the best fitting solution, the comparison becomes legitimate henceforth as other requirements can also be incorporated, e.g., power consumption. It is also conceivable that comparisons are less appropriate, as changes to the implementation on a specific platform result in different parallelization strategies. Therefore, we argue that the catalog becomes even more important, as it allows to evaluate different platforms and workloads with respect to gathered operating experiences.

The results, depicted in Figure 4, show that the initial FPGA design is up to 5.63 times slower than the default kernel, while the improved FPGA model is 1.14 times faster than the best GPU solution, which is using the image kernel. The FPGA model considers two main optimizations when compared to the hardware version. The first is the usage of three available memory banks, instead of one. This change allows us to increase the system clock frequency from 50 MHz to 150 MHz. The second optimization is the time domain parallelization. All of these optimizations are completely feasible and are currently being implemented in real hardware.

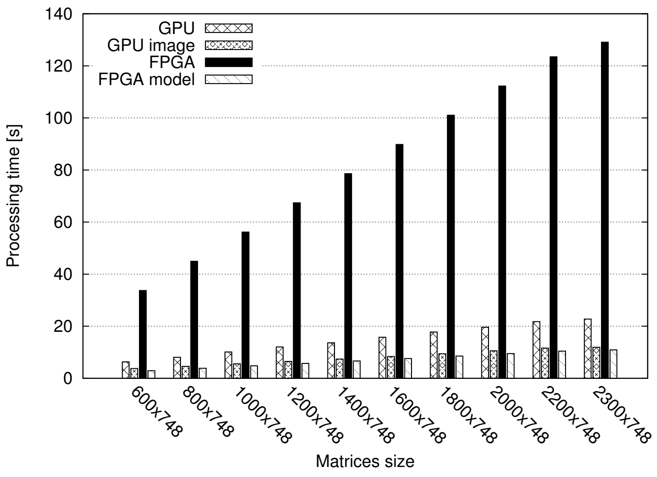

Figure 4. Results comparing available FPGA and GPGPU implementations.

E. Development Time

The most-promising platforms for our seismic application are the GPU and the FPGA, as both achieve the best performance compared to the CPU implementations. The performance results reveal that the GPU optimization of the image kernel over the default kernel enables an additional speedup of 1.86, while the modeled FPGA optimization achieves an additional speedup of 11.7, compared to the initial FPGA implementation. Based on our experiments, the initial development of the FPGA architecture took roughly 8 months with four engineers working, while both the GPU prototype and the optimization could be developed each within only 1.5 months with a single programmer.

To summarize this, the GPU allows to start quickly and promises good performance results. On the other hand, the GPU has natural limits regarding available compute cores, so further speed-ups may not be expected. Considering the FPGA, the development time is much longer and more intensive in terms of acquisition cost and development time. On the other hand, the FPGA is likely to enable very specialized optimizations. We estimate that the development of a pipelined architecture inside the FPGA would require another 3 months for one engineer. So, in this specific experiment, the GPU is the best solution regarding achieved performance and development time.

VII. Outlook

When storing additional information inside the catalog, it is possible to evaluate multiple parameters for given platforms in heterogeneous data centers. For instance, the FPGA could evolve as better solution with respect to performance and power consumption over time. To find a cost-benefit trade-off for a given time period, hardware acquisition cost can also be considered. We believe that more beneficial indicators can be extracted out of the catalog, which enables an added value to heterogeneous data centers.

VIII. Conclusion

In this paper we explore a seismic modeling application on a set of heterogeneous machines, including CPUs, GPUs, and FPGAs, guided by a workflow to manage the manifoldness of heterogeneous resources. The proposed flow allows to explore different hardware platforms and versions of the application, which leads to locating and cataloging the best fitting solutions regarding performance. We elaborated on implementation details in order to gather operating experiences of different parallelization approaches. This includes multi-threaded, SIMD, and OpenCL processing on CPUs and GPUs. The promise that OpenCL kernels will run on each architecture is to handle with care: in our operating experience, OpenCL compute kernels exploit more performance when still being aware of the underlying hardware's capabilities. In the results section, we show that the best solutions are accomplished using GPUs and FPGAs. We also discuss the development time of GPU and FPGA-specific optimizations and correlate that with the achieved performance, which reveals that in the GPU is the best fitting solution in our experiment. The overall benefit of the proposed heterogeneous platform exploration flow is to support decision making processes for choosing the best fitting solution with regard to resource intensive seismic applications.

References

[1] R. G. Clapp, H. Fu, and O. Lindtjorn, "Selecting the right hardware for reverse time migration (in High-performance computing)," in *Leading Edge*, Tulsa, OK, 2010, pp. 48–58.

[2] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker *et al.*, "Stencil computation optimization and autotuning on state-of-the-art multicore architectures," in *In (submitted to) Proc. SC2008: High performance computing, networking, and storage conference*, 2008.

[3] Khronos Group, "OpenCL - The open standard for parallel programming of heterogeneous systems," 2011/01, URL: http://www.khronos.org/opencl/.

[4] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in Science and Engineering*, vol. 12, pp. 66–73, 2010.

[5] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007. [Online]. Available: http://www.blackwell-synergy.com/doi/pdf/10.1111/j.1467-8659.2007.01012.x

[6] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen *et al.*, "Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU," in *Proceedings of the 37th annual international symposium on Computer architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 451–460. [Online]. Available: http://doi.acm.org/10.1145/1815961.1816021

[7] NVIDIA, "Developer Zone – OpenCL," 2010/11, URL: http://developer.nvidia.com/object/opencl.html.

[8] AMD, "AMD Accelerated Parallel Processing SDK," 2010/11, URL: http://developer.amd.com/gpu/AMDAPPSDK/Pages/default.aspx.

[9] IBM, "OpenCL Development Kit for Linux on Power," 2011/01, URL: http://www.alphaworks.ibm.com/tech/opencl. [Online]. Available: http://www.alphaworks.ibm.com/tech/opencl

[10] Intel, "Intel OpenCL SDK," 2011/01, URL: http://software.intel.com/en-us/articles/intel-opencl-sdk/.

[11] D. Singh, "Higher Level Programming Abstractions for FPGAs using OpenCL." Presented at the FPGA 2011 Pre-Conference Workshop: The Role of FPGAs in a Converged Future with Heterogeneous Programmable Processors, Monterey, CA, 2011. [Online]. Available: http://www.eecg.toronto.edu/~jayar/fpga11/Singh_Altera_OpenCL_FPGA11.pdf

[12] D. B. Thomas, L. Howes, and W. Luk, "A comparison of CPUs, GPUs, FPGAs, and massively parallel processor arrays for random number generation," in *FPGA '09: Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*. New York, NY, USA: ACM, 2009, pp. 63–72.

[13] M. Perrone, "Finding Oil with Cells: Seismic Imaging Using a Cluster of Cell Processors," 2009, URL: https://www.sharcnet.ca/my/documents/show/44.

[14] R. Abdelkhalek, H. Calendra, O. Coulaud, J. Roman, and G. Latu, "Fast Seismic Modeling and Reverse Time Migration on a GPU Cluster," in *The 2009 High Performance Computing & Simulation - HPCS'09*, Leipzig Germany, 2009, Best Paper Award at HPCS'09 Total. [Online]. Available: http://hal.inria.fr/inria-00403933/en/

[15] Ö. Yilmaz, *Seismic Data Analysis*. Tulsa, OK: Society of Exploration Geophysicists, 2001. [Online]. Available: http://link.aip.org/link/doi/10.1190/1.9781560801580