Journal of Parallel and Distributed Computing 165 (2022) 120–129

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

Evaluation of Intel's DPC++ Compatibility Tool in heterogeneous computing

Germán Castaño [a], Youssef Faqir-Rhazoui [a], Carlos García [a,b,*], Manuel Prieto-Matías [a,b]

[a] Fac. Informática, Universidad Complutense Madrid, Spain
[b] Instituto de Tecnología del Conocimiento, Universidad Complutense de Madrid, Spain

ARTICLE INFO

Article history:
Received 13 July 2021
Received in revised form 25 February 2022
Accepted 27 March 2022
Available online 8 April 2022

Keywords:
DPC++
oneAPI
Rodinia
Intel DPCT
CUDA

ABSTRACT

The Intel DPC++ Compatibility Tool is a component of the Intel oneAPI Base Toolkit. This tool automatically transforms CUDA code into Data Parallel C++ (DPC++), thus assisting in the migration process. DPC++ is an implementation of the programming standard for heterogeneous computing known as SYCL, which unifies the development of parallel applications on CPUs, GPUs or even FPGAs.
This paper analyzes the DPC++ Compatibility Tool by considering the manual intervention required and the problems encountered while migrating the Rodinia benchmarks. For this suite, this tool achieves an impressive rate of almost 87% for code successfully migrated. Moreover, a comparative study of the performance obtained by the migrated code was carried out, showing a moderate overhead in most of the migrated examples. Finally, a performance comparison on different devices was also performed.

1. Introduction

Nowadays, heterogeneity is becoming increasingly common in both high-performance computing and consumer electronics. These systems add a multitude of co-processors or accelerators, such as Graphics Processing Units (GPUs), Tensor Processing Units (TPUs), and Field Programmable Gate-Arrays (FPGAs), to the traditional CPU. However, there is no simple, portable and efficient method to develop for these systems, although now Intel oneAPI aims to fulfill this role.

In recent years a clear tendency towards heterogeneity in computing has been observed, not only in high-performance computers, where NVIDIA GPUs are widely used as accelerators, but also in desktops and handheld devices. Most smartphones include a GPU and the latest processors such as the Apple M1 [4] system on chip (SOC) integrate ARM CPUs, a GPU and other accelerators such as a 16-core Neural Engine for artificial intelligence applications.

These heterogeneous SOCs, which are becoming more prominent, are clear indicators of this shift toward heterogeneity. A result of this is that it is essential to have a unified and straightforward way of developing for different heterogeneous architectures without depending on a specific vendor.

Heterogeneous systems not only improve performance but also focus on energy efficiency. One of the challenges that application and algorithm developers face is efficiently exploiting this large amount of computational resources. Although in heterogeneous programming, the arrival one decade ago of CUDA for NVIDIA's GPUs represented a great revolution that has allowed the development and adoption of this technology, its use has been limited to NVIDIA hardware. The OpenCL standard [22] has tried to solve this limitation, allowing the adoption of this heterogeneous programming paradigm by different vendors, not only on GPUs but also for other accelerators such as the discontinued Intel Xeon-Phi, Digital Signal Processors or even FPGAs. However, application developers' use of these programming paradigms remains an arduous task prone to errors and whose performance portability remains limited.

Among the most promising initiatives in the field of heterogeneous computing, we can highlight the SYCL standard [18], which has been promoted by the Khronos consortium. Although it was announced in 2014, this standard has been given a boost by the recent announcement of oneAPI [20] by Intel. Intel's oneAPI is a suite of programming tools divided commercially into Toolkits. The Intel Base Toolkit includes the Data Parallel C++ compiler (SYCL compiler denoted as DPC++ by Intel) and the Intel DPC++ Compatibility Tool (DPCT).

DPCT helps with the migration process by automatically transforming most CUDA code into DPC++, significantly increasing developer productivity. This paper aims to analyze the migration process from CUDA to DPC++ using DPCT and assess the perfor-

* Corresponding author at: Fac. Informática, Universidad Complutense Madrid, Spain.
E-mail address: garsanca@ucm.es (C. García).

https://doi.org/10.1016/j.jpdc.2022.03.017

mance obtained by the migrated DPC++ code on different GPU and CPU devices using the Rodinia benchmark suite as a workload.

The rest of the paper is organized as follows. Section 2 discusses related work and outlines some of the innovations introduced by Intel's oneAPI. Section 3 presents the methodology used in this study, and Section 4 discuss the results obtained in the migration process. Finally, Section 5 summarizes the main contributions and includes our remarks on the work.

2. Related work

Motivated by the popularity of the use of graphics processors with the appearance of CUDA [13], several heterogeneous programming paradigms have been proposed in the last few years. Among these, we can highlight OpenCL, or Open Computing Language, whose strength is its portability on different types of GPUs, CPUs or other devices in contrast to CUDA, which is only supported on NVIDIA graphics cards. Also, the OpenACC [21,12] and OpenMP [23] models allow the programmer to express heterogeneous parallelism with a higher level of abstraction by means of pragmas.

Recently, SYCL [19] has become one of the most promising alternatives. It is a cross-platform abstraction layer for heterogeneous programming using standard ISO C++ where host and kernel code can be included in a single source file. It uses generic programming with templates and lambda functions to enable higher-level abstractions. At the same time, developers still have access to lower-level code through seamless integration with the native acceleration API through the inter-operability mode, C/C++ libraries, and frameworks such as OpenCV or OpenMP.

There are multiple SYCL implementations available, with the DPC++ compiler [5] in the Intel oneAPI being one of the most promising initiatives. Intel's oneAPI is a cross-industry initiative that stands out for offering the following advantages: (1) it is open (the source code is freely available in a git repository[1]), and (2) it is based on the C++ programming standard, which facilitates its adoption. Its goal is to provide an efficient, high-performance programming model so that developers do not need to maintain a separate code base, multiple programming languages, and different tools and workflows for each architecture. The Intel oneAPI also supplements the SYCL and OpenCL standards with additional extensions, including optimized libraries compatible with DPC++ such as oneCCL, for scaling deep learning frameworks across multiple devices, oneDAL for data science, oneDNN for high performance implementations of primitives for deep learning frameworks, oneMKL for math routines, oneTBB to express thread-based parallelism for complex applications on multiprocessors, and oneVPL for scaling video processing applications.

SYCL is currently supported by four implementations: Codeplay's ComputeCpp [8], Intel's LLVM/clang implementation known as Intel's oneAPI DPC++/C++ compiler [20], triSYCL [24] led by Xilinx, and hipSYCL [3] led by Heidelberg University.

Furthermore, a debugger tool for SYCL programs that is based on GDB and which allows the offloading of kernels to CPU, GPU, or FPGA emulator devices has been developed [1]. Regarding oneAPI and DPC++ implementation, due to the recent launch of oneAPI in December 2020, there are hardly any studies on the use of the different oneAPI Toolkits. The early experience of porting a tsunami simulation code from CUDA to DPC++ is addressed in [7]. In [25], the authors developed a cross-architecture for a real-time medical ultrasound imaging application using oneAPI based on the open-source project SUPRA. Among the main contributions of [25] are the evaluation of the Intel's DPCT tool using different hardware from Intel and Nvidia. An interesting initiative can be found within the GROMACS project, a high-performance molecular dynamics package used worldwide (it is worth highlighting that GROMACS represents at least 5% of the worldwide consumption of HPC resources). It is also worth mentioning that there exist different implementations of GROMACS for heterogeneous computing, such as CUDA or OpenCL for AMD/Intel devices, which makes the evaluation more interesting. Moreover, the authors of [2] discuss the experiences and challenges of adding support for SYCL by means of the DPC++ compiler in GROMACS as well as the interaction process with OpenCL in parallel DAG-based scheduling. Other experiences related to the Intel's oneAPI compatibility Tool are described in [9], in which the authors conclude that some programmer adaptations are needed to complete the porting task in the test of matrix multiplication.

More recently, Nozal et al. [14] evaluated the performance and energy efficiency for a well-known set of regular and irregular HPC benchmarks using both integrated GPUs and CPUs. Among the main conclusions, the authors highlight that the use of oneAPI achieves competitive performance and co-execution on CPUs and GPUs improves efficiency even more when using unified shared memory.

2.1. The Intel DPC++ compiler

The DPC++ compiler included in the oneAPI suite allows the direct programming of devices such as CPUs or GPUs. DPC++ programs [17] are written in the ISO C++ standard and use the SYCL parallel programming model to distribute computation across processing elements in a device. DPC++ extends SYCL with features for performance and productivity.

DPC++ is single source, and device and host code can be included in the same source file. A DPC++ compiler generates code for both the host and device. Any C++ compiler can compile programs that only use the host subset of DPC++.

From an execution perspective, DPC++ and SYCL use selectors to define a queue to execute the kernel code. The most common selectors are *host_selector*, *gpu_selector* and *accelerator_selector*. All the context and states needed for kernel execution are encapsulated in a DPC++ queue. By default, a queue is created and associated with an accelerator. Kernels are enqueued to the queue and executed.

The memory model is based on the fact that device and host memories are separate. Buffers and accessors are used for the data transfer between host and device. Data declared on the host is wrapped in a buffer and transferred to the accelerators implicitly by the DPC++ runtime. The accelerators read or write to the buffer through an accessor. Unified Shared Memory (USM) is an alternative to buffers for managing and accessing memory from the host and device. We would like to point out that USM is now supported by the SYCL2020 specification. Explicit data movement with USM is accomplished, as in CUDA, with device allocations and a special *memcpy* found in the queue and handler classes. Data can be allocated for device, host or both (shared).

2.2. The Intel DPC++ compatibility tool

The DPCT [10] works by intercepting the application build process and replacing CUDA code with the oneAPI counterpart. Although DPCT automatically migrates most of the code, some manual work might be required for a complete migration. The tool outputs warnings to indicate how and where manual intervention is needed. These warnings have an assigned ID, of the form

[1] https://github.com/oneapi-src.

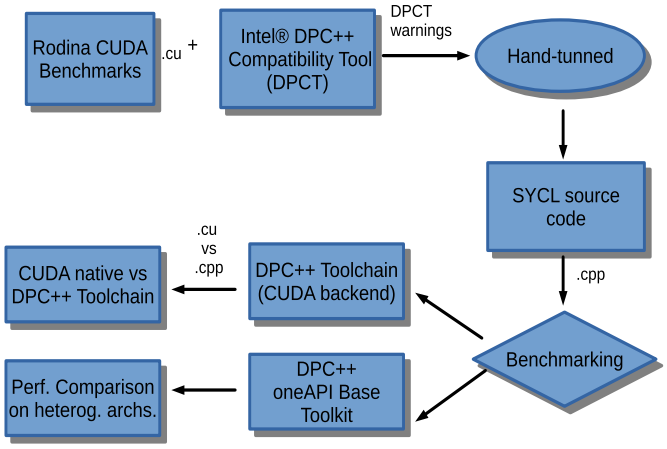

Fig. 1. Methodology workflow.

"DPCT10XX", that can be consulted in the Developer Guide and Reference.[2]

To migrate a project with several CUDA source files to DPC++, it is necessary to use DPCT multiple times. To simplify the migration process in this case, Intel provides the *intercept-build* script which tracks and saves the compilation commands, compilation flags, and options automatically in a JSON file. Then, the DPCT tool can be used to migrate the source files, and finally, the user should review the migration process, checking the DPCT warnings generated.

3. Benchmarking methodology

Fig. 1 shows the workflow used in this study. It involves two main phases: (1) the migration process by means of the DPCT tool, and (2) the benchmarking phase, where we evaluate the performance of the migrated code over the original CUDA code as well as the performance portability across different architectures.

As input for the DPCT tool we chose the well-known Rodinia [6] benchmark suite, whose main features are described in the next subsection.

3.1. Rodinia benchmark

The Rodinia Benchmark Suite[3] is implemented in CUDA, OpenMP and OpenCL, and includes applications from a wide range of fields such as medical imaging, fluid dynamics or image compression, among other. Table 1 summarizes the main features of the benchmarks and the problem size used in the experimentation phase.

3.2. Code instrumentation

Most of the Rodinia benchmarks did not output the time spent on the different execution stages, and the ones that did have their own output format. This is why all the benchmarks, the originals in CUDA and the migrated ones in DPC++, had to be adapted to ensure the same format of output measurements.

3.3. Benchmarking native CUDA vs DPC++ toolchain

Our first analysis of the migrated code compares its performance over the original CUDA code on an Nvidia GPU. We would like to point out that Intel's commercial DPC++ available in the oneAPI Base Toolkit does not currently support CUDA as a backend. However, as the oneAPI project is open, building the DPC++ Toolchain with Nvidia CUDA support is possible. For this research, the DPC++ compiler was built using experimental support for CUDA devices.[4]

It is also worth mentioning that this experimental backend is also available through a Docker Image [11] although it has not been updated since oneAPI 2021.2, while the latest version is 2021.4.

3.4. Performance comparison on other devices

One of the main advantages of DPC++ is the application portability it enables. The migrated Rodinia benchmark versions can run on different CPU or GPU-based systems. In this work, we have used a development sandbox, specifically the Intel DevCloud, which allows us to carry out projects and test performance on different hardware devices. Subsection 4.2.1 details the features of the systems selected for the experimentation.

4. Experiments

This section outlines the main issues found in the migration process of the Rodinia benchmarks and presents the main results obtained from the experiments. All the codes and the output logs generated in these experiments are publicly available in our Git repository.[5]

4.1. Migration phase

During the migration process, the DPCT generated a series of warnings indicating possible problems and the need for manual intervention by the user.

Across all the benchmarks, 99 files were processed by the DPCT, with a total of 43485 lines of code. It gave a total of 461 warnings, with an average of 4.65 warnings per file or a warning every 94.3 lines. Table 2 shows the warnings given by the DPCT tool.

These warnings can be grouped into the following categories:

- Error handling related warnings (DPCT1000, DPCT1001, DPCT1003, DPCT1009, DPCT1010, DPCT1024)
- Device information related warnings (DPCT1005, DPCT1019, DPCT1022, DPCT1051, DPCT1072)
- Kernel invocation warnings (DPCT1049)
- Time measurement warnings (DPCT1012)
- Warnings caused by the removal of unnecessary function calls (DPCT1026, DPCT1027)
- Warning generated because SYCL does not support something (DPCT1059)
- Performance improving suggestions (DPCT1065)
- Macro related warnings (DPCT1064, DPCT1077)
- Other (DPCT1004, DPCT1035, DPCT1039)

Fig. 2 summarizes the warnings generated by the compatibility tool. Most of them (57.3%) are caused by the fact that SYCL uses exceptions instead of error codes. Although it might be desirable in order to handle any error that might occur at runtime, no manual modifications were mandatory for these warnings as the tool modifies all error checks, so they always return a success.

[2] Diagnostics Reference of Intel® DPC++ Compatibility Tool available at: https://software.intel.com/content/www/us/en/develop/documentation/intel-dpcpp-compatibility-tool-user-guide/top/diagnostics-reference.html.

[3] Source code available in the repository: http://rodinia.cs.virginia.edu/doku.php.

[4] More information related with building a DPC++ toolchain at https://github.com/intel/llvm/blob/sycl/sycl/doc/GetStartedGuide.md.

[5] Rodinia Benchmarks for DPC++: https://github.com/artecs-group/rodinia-dpct-dpcpp.

Table 1
Rodinia suite benchmark with the problem size used.

Application	name	Dwarves	Domain	Problem Size
B+Tree	b+tree	Graph Traversal	Search	1M elements
Back Propagation	backprop	Unstructured Grid	Pattern Recognition	65536 input nodes
Breadth-1st Search	bfs	Graph Traversal	Graph Algorithms	1M vertices
CFD Solver	cfd	Unstructured Grid	Fluid Dynamics	0.2M elements
GPUDWT	dwt2d	Spectral Method	Image/Video Compr.	1024^2 images, forwd. 5/3
Gaussian Elimination	gaussian	Dense Linear Algebra	Linear Algebra	1024×1024 matrix
Heart Wall	heartwall	Structured Grid	Medical Imaging	104 frames
HotSpot	hotspot	Structured Grid	Physics Simulation	512x512 matrix
Hotspot3D	hotspot3D	Structured Grid	Physics Simulation	512x512 matrix
Huffman	huffman	Finite State Machine	Lossless data compr.	1 MB file
LavaMD2	lavaMD	N-Body	Molecular Dynamics	10 boxes
LU Decomposition	lud	Dense Linear Algebra	Linear Algebra	2048×2048 points
Myocyte	myocyte	Structured Grid	Biological Simulation	100 time steps
k-Nearest Neighbors	nn	Dense Linear Algebra	Data Mining	5 nearest neighbors
Needleman-Wunsch	nw	Dynamic Programming	Bioinformatics	12000x12000 points
Particle Filter	particlefilter	Structured Grid	Medical Imaging	1M points
PathFinder	pathfinder	Dynamic Programming	Grid Traversal	100000x1000 2D grid
SRAD	srad	Structured Grid	Image Processing	512x512 image
Streamcluster	streamcluster	Dense Linear Algebra	Data Mining	65536 points 256 dimensions

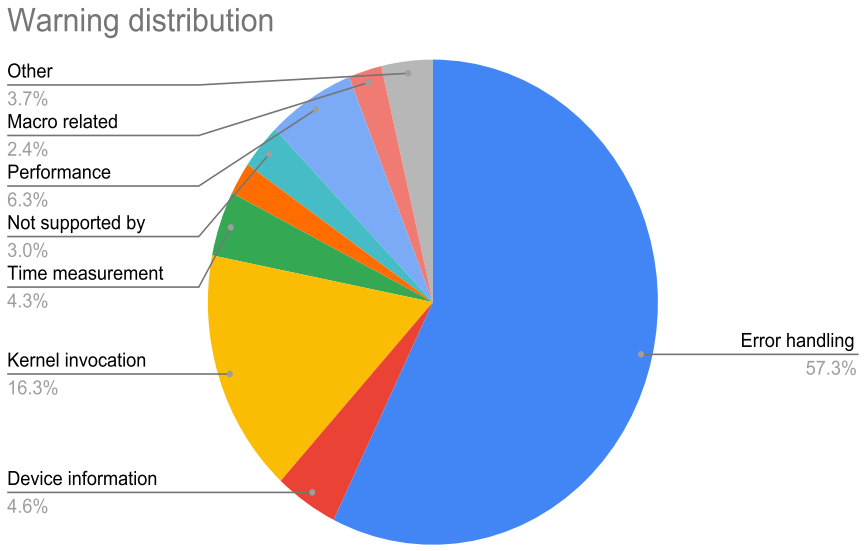

Fig. 2. Distribution of the warnings generated by DPCT.

Table 2
Warnings given by DPCT tool.

Warning code	Number of appearances
DPCT1000	4
DPCT1001	4
DPCT1003	171
DPCT1004	1
DPCT1005	15
DPCT1009	45
DPCT1010	38
DPCT1012	20
DPCT1019	1
DPCT1022	2
DPCT1024	2
DPCT1026	8
DPCT1027	2
DPCT1035	9
DPCT1039	7
DPCT1049	75
DPCT1051	2
DPCT1059	14
DPCT1064	2
DPCT1065	29
DPCT1072	1
DPCT1077	9
Total	**461**

The second group of warnings appears in every kernel invocation and simply reminds the user that the targeted device might have a smaller limit for the workgroup size. This is usually the case when moving to a less powerful GPU.

Of the rest, it should be noted that the warnings related to macros and device information are the ones that require the most manual intervention from the user. It is also worth mentioning that, as the migrated code is a benchmark, the number of time measurement related warnings is much higher than it would be in other codes.

In the end, twenty out of the twenty three benchmarks were successfully migrated without major manual intervention, resulting in a success rate of almost 87%. Table 3 shows the benchmarks that were successfully migrated with a moderate developer intervention.

After the automatic migration, some manual modifications of the code were necessary, always addressing the warning messages generated by the tool. As a summary, below we illustrate the main modifications performed:

- The workgroup size might need to be adjusted depending on the device used.

Table 3
DCPT migration of Rodinia benchmark.

Benchmark	Successful migration
b+tree	Yes
backdrop	Yes
bfs	Yes
cfd	Yes
dwt2d	Yes
gaussian	Yes
heartwall	Yes
hotspot	Yes
hotspot3D	Yes
huffman	Yes
hybridsort	**No**
kmeans	**No**
lavaMD	Yes
leukocyte	Yes
lud	Yes
mummergpu	**No**
myocyte	Yes
nn	Yes
nw	Yes
particlefilter	Yes
pathfinder	Yes
srad	Yes
streamcluster	Yes

- When the block size is specified with a macro and used to create a *sycl::range*, the expanded value should be changed back to the macro. When this is the case, the tool leaves the original macro commented.
- As SYCL uses exceptions instead of error codes, every check is modified by the tool in order to always succeed. Proper error checking might be added manually with try-catch constructions.
- The device selection logic must be manually reviewed as all DPC++ devices (not only the GPU) can be used to submit tasks.
- Many CUDA device properties do not have a SYCL equivalent, are slightly different or are not currently supported. On many occasions, this will cause the retrieval of incorrect values. For this reason, the user must manually review and correct the information queries to the device.
- SYCL only supports a 4-channel image format, so the code needs to be manually adjusted. An example of the modifications needed is given in the Intel DPC++ Compatibility Tool Developer Guide and Reference [10].
- When a function call is used in a macro definition, it might need to be migrated differently depending on how the macro is called. All uses of this macro must be reviewed.
- If a macro redefines a standard SYCL type, it may cause conflicts. The Developer Guide and Reference [10] suggests that the user renames the macro.

Other problems found that can be solved:

- *CL_INVALID_IMAGE_SIZE*: this is a known issue that occurs when the *info::device::image2d_max_width* value of the device is less than the width of the image passed into the kernel. There is no solution for GPUs yet, although it will be addressed soon for CPUs.[6]
- The latest CUDA version supported is 11.1. This causes minor problems with the *intercept-build* tool as it will not find certain libraries. Re-executing the command with the *–append* flag solves the error, effectively fixing the problem.
- A segmentation fault was observed on the particlefilter benchmark. This can be solved with a memory initialization.

4.2. Performance results

The results obtained during the benchmarking phase are discussed in this subsection. We analyze the performance of the native CUDA vs DPC++ Toolchain and perform a comparison across different hardware architectures via the DPC++ compiler available in Intel's Base Toolkit.

4.2.1. Experimental environment

The software used for the experimental phase is based on the latest commercial version of Intel's oneAPI that is currently available, namely 2021.4. Regarding CUDA, the Nvidia GPU device uses the 11.4 version.

Table 4 shows the main characteristics of the CPU devices we used. We chose two systems hosted in the Intel's Devcloud, which were based on Intel Core i9-10920X and Intel Xeon E-2176G processors. We would like to point out that in the third system we only evaluate the Nvidia GPU, so no performance measurement is presented for the CPU.

Regarding the GPU devices (see Table 5), we chose two of Intel's GPUs available in the Intel DevCloud (the integrated UHD P630 GPU and the DG1 connected via the PCIe bus), and a personal computer equipped with an Nvidia GT 1030 GPU.

4.2.2. Performance of the memory operations

In order to compare the performance of the memory operations we considered the overhead of memory management on the device and the cost of data transfer between the host and the device memories. This experiment evaluates the original CUDA version against the migrated DPC++ versions of the said benchmarks using the same GPU (Nvidia GT 1030).

Fig. 3 shows the average time spent (in seconds) on each memory operation across all the timed benchmarks. The comparison uses the average times for all the benchmarks. Although DPC++ achieves a similar performance on these operations with respect to CUDA, it introduces a series of CUDA API calls for context and event management (among others) that were not needed by the original CUDA program, and this introduces a slight overhead of up to a few milliseconds, depending on the specifics of the application.

4.2.3. Performance evaluation: native CUDA vs DPC++ on Nvidia GPUs

Figs. 4 and 5 show the time employed for native CUDA and DPC++ ported code on the Nvidia GT 1030. For the sake of visualization, we split the results into two charts. Fig. 4 presents the time employed in the tests that took less than a second, and Fig. 5 shows the same for those tests that took more than a second. In addition, both figures use a log representation on the time axis to facilitate the interpretation of the results.

While some of the migrated applications achieved very similar performance to the original CUDA, others display a considerable overhead, varying from 25% to 190%. However, considering that many of the tests finish in less than a second, we take a deeper look at the tests in Fig. 5, because the profiling tools usually introduce overhead in the execution time, and this could cause a significant variation in the short time tests. In those cases, we use the Nvidia Visual Profiler [15] to explore the difference between the CUDA and DPC++ versions.

Regarding the *cfd/euler3d* test, it shows a significant time degradation, performing 32% (CUDA 10.8 s, DPC++ 15.9 s) worse in DPC++. Using the profiler tool, we found that while the main kernel in CUDA spends 45% of the time on memory operations, its DPC++ counterpart spends 75% of the time on those operations.

[6] DPCT Key Features Releases: https://www.intel.com/content/www/us/en/developer/articles/release-notes/release-notes-for-intel-dpcpp-compatibility-tool-2021.html.

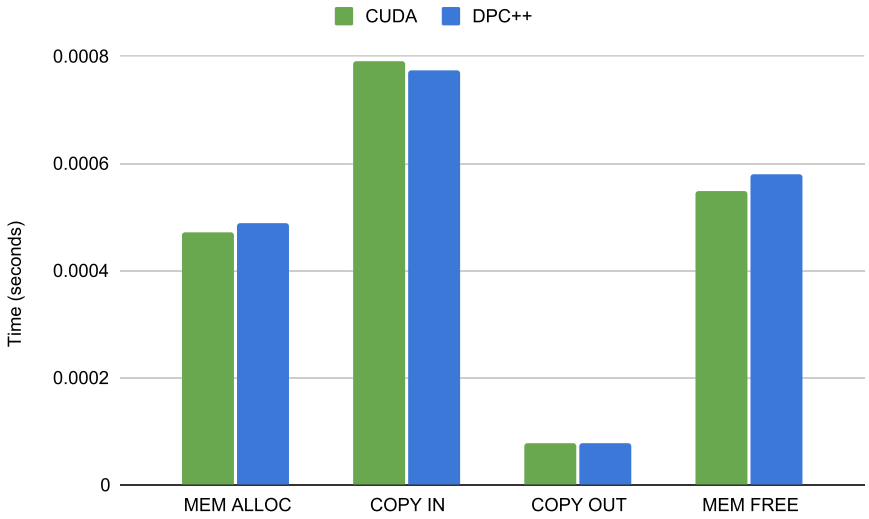

Fig. 3. The mean of the time (s) of all benchmarks involved on the memory operations.

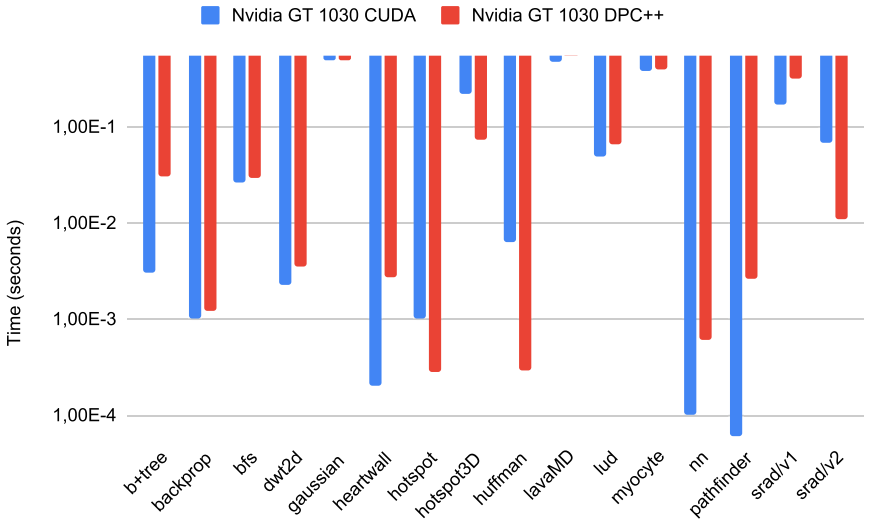

Fig. 4. Time comparison between CUDA and DPC++ in tests that account for less than one second.

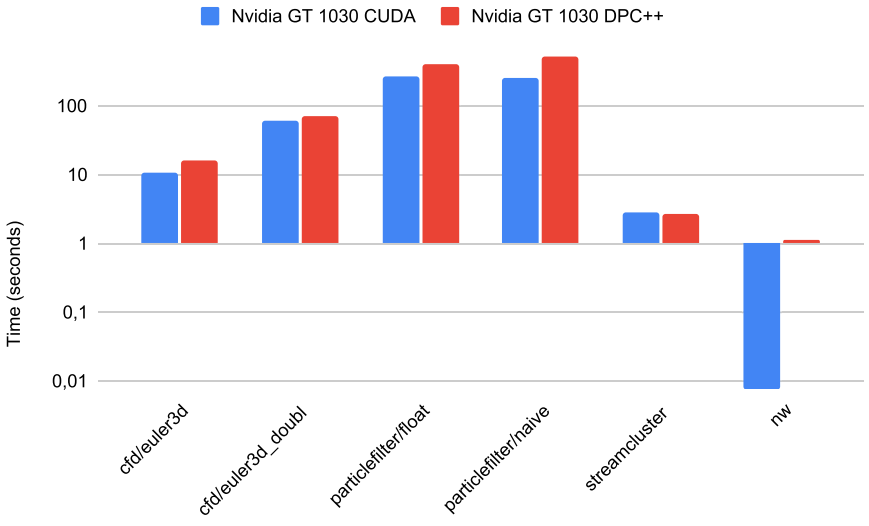

Fig. 5. Time comparison between CUDA and DPC++ in tests that account for more than one second.

Table 4
Specifications of the CPU devices used in the experimentation stage.

System n.	OS	CPU	Cores	Cache	Perf. Peak	RAM Memory BW
1	Ubuntu 18.04.3 LTS	Intel Core i9-10920X	12 (×2) HT	L1-64 KB per core L2-1 MB per core L3-19.25 MB shared	672 GFLOPS (FP32)	94 GB/s
2	Ubuntu 18.04.3 LTS	Intel Xeon E-2176G	6 (×2) HT	L1-384 KB per core L2-1.5 MB per core L3-12 MB shared	355.2 GFLOPS (FP32)	41.6 GB/s
3	Ubuntu 20.04 LTS	Intel Core i7-10700	8 (×2) HT	L1-64 KB per core L2-256 KB per core L3-16 MB shared	371.2 GFLOPS (FP32)	45.8 GB/s

Table 5
Specifications of the GPUs devices used in the experimentation stage.

System n.	SW	GPU	Cores	Driver	Perf. Peak	RAM Memory BW
1	oneAPI 2021.4	Intel UHD P630	24 EUs	i915 21.33.20678	480 GFLOPS (FP32)	41.6 GB/s
2	oneAPI 2021.4	Intel IRIS Xe MAX DG1	96 EUs	i915 21.33.20678	2534 GFLOPS (FP32)	62.26 GB/s
3	CUDA 11.4	NVIDIA GT 1030	384 CUDA cores	470.42.01	1127 GFLOPS (FP32)	48.0 GB/s

The memory profiler shows that the DPC++ kernel suffers from more write operations into the GPU main memory than the native CUDA version, causing higher contention on the memory interface.

With regards to the *particlefilter* tests, there was a ≈42% drop in performance in the migrated code (CUDA 268.2 s vs DPC++ 470.9 s). The Nvidia profiler reveals more instructions (2.3× more) in the DPC++ version, with a significant increment in the number of *miscellaneous* CUDA instructions.

In the case of the *nw* test, there was a 99.3% of performance drop in the DPC++ version (CUDA 7.7e-03 s vs DPC++ 1.15 s). The main cause is the under-utilization of the GPU resources. While the CUDA version creates warps of 32 threads, the DPC++ just uses 24. When focusing on the active threads, the tool reveals that CUDA uses 1024 threads vs 768 for the DPC++ version. The profiling report shows that the GPU utilization factor is only ≈36% in the DPC++ version. By contrast, the native CUDA code reaches a GPU utilization of up to ≈50%.

4.2.4. Performance of the kernel execution on CPUs

In this subsection we analyze the performance regarding portability of the Rodinia benchmark suite generated by Intel's DPCT compatibility tool in comparison with the parallelized benchmarks obtained with OpenMP [6]. We would like to point out that the latest (3.1) version of the OpenMP Rodinia benchmarks was used, and the source codes was instrumented to measure the kernel times.

Figs. 6 and 7 summarize the execution times of the Rodinia suite. The processors used in the comparison corresponded to a desktop computer based on the Intel Core i9-10920X processor and a server based on the Intel Xeon E-2176G processor.

Fig. 6 shows the performance results for the lightweight Rodinia's benchmarks that take less than one second in our platforms. OpenMP versions clearly outperform the DPC++ counterparts. A profilining with Intel vTune reveals huge overheads in the *backprop*, *myocyte*, *nw*, *srad_v1* and *srad_v2* benchmarks. Those overheads appear at the end of the *parallel_for* regions, where the DPC++ implementation uses an atomic section implemented at low level with a *futex* lock.

On the contrary, for larger workloads such as those shown in Fig. 7, the DPC++ versions outperformed the OpenMP counterparts in most cases. The DPC++-based benchmarks were up to 4 times faster than the OpenMP equivalents.

4.2.5. Performance of the kernel execution on Intel GPUs

The following subsection addresses the performance results on two Intel GPUs: an integrated GPU (UHD P630) and a discrete GPU (Intel Iris Xe MAX DG1).

Once again, we split the execution times into those that take less than a second (Fig. 8) and those that lasted more than a second (Fig. 9). Most benchmarks (20/21) run successfully in the P630 GPU but the DG1 does not support double-precision arithmetic and cannot run 10 benchmarks.

As expected, the Intel DG1 performed better than the integrated one. The average speedup achieved by the DG1 over the UHD P630 is around ×2.4. Nevertheless, higher performance can be achieved in both GPUs adjusting parameters such the *ndrange* or the *simd length* [16].

5. Conclusions

OneAPI aims to provide a unified and straightforward programming method across different heterogeneous systems, independently of the accelerator used or the vendor, allowing high and low level approaches, thus facilitating the programmer's work in this highly heterogeneous world we are entering.

This work has evaluated the Intel DPCT tool using the well-known Rodinia benchmarks as a workload. The main conclusions from our research are the following:

- DPCT greatly streamlines the migration process from CUDA to oneAPI. Twenty out of the twenty three benchmarks were successfully migrated without major developer interventions.
- Memory operations (device memory management operations and data transfers between host and device memories) take roughly the same time in the migrated and native codes.
- While some migrated applications achieved similar performance to the original CUDA versions, others experience significant overheads (from 25% to 190%). This performance degradation is caused by diverse factors that strongly depend on the actual benchmarks. The most important issues we found where a higher number of instructions (*particlefilter*), higher memory contention (*cfd* and *euler3d*) and lower hardware occupancy (*nw*).

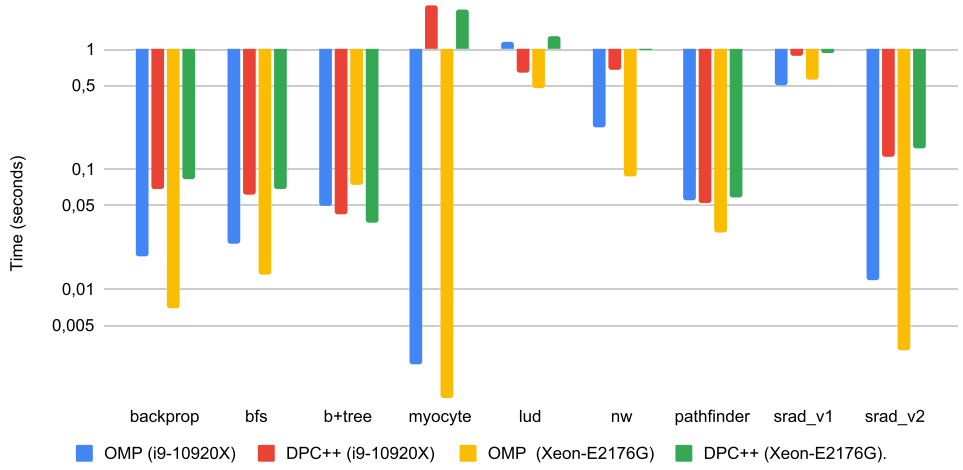

Fig. 6. Performance comparison of OpenMP and DPC++. Rodinia kernel times less than one second.

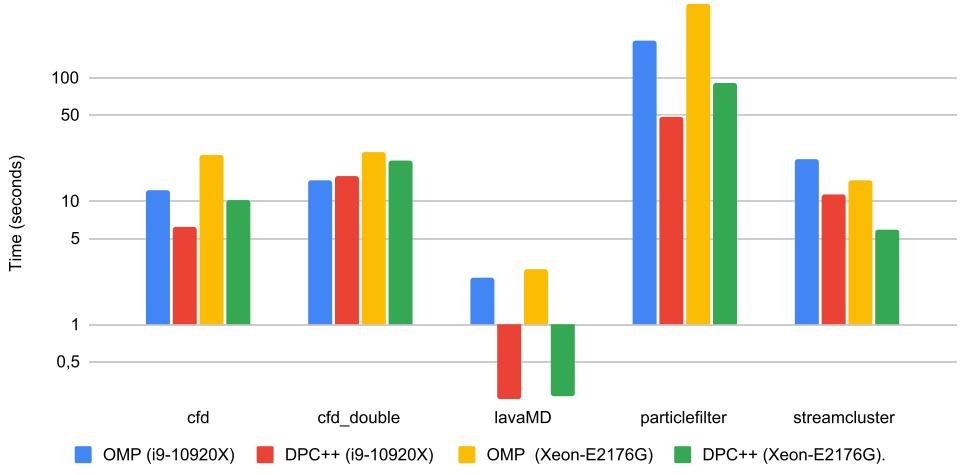

Fig. 7. Performance comparison of OpenMP and DPC++. Rodinia kernel times more than one second.

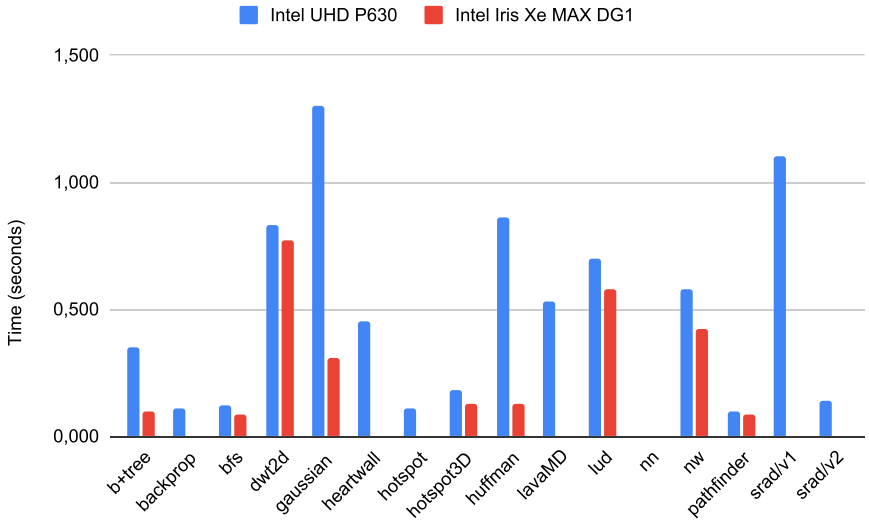

Fig. 8. Time comparison on the Intel GPUs (tests taking less than one second).

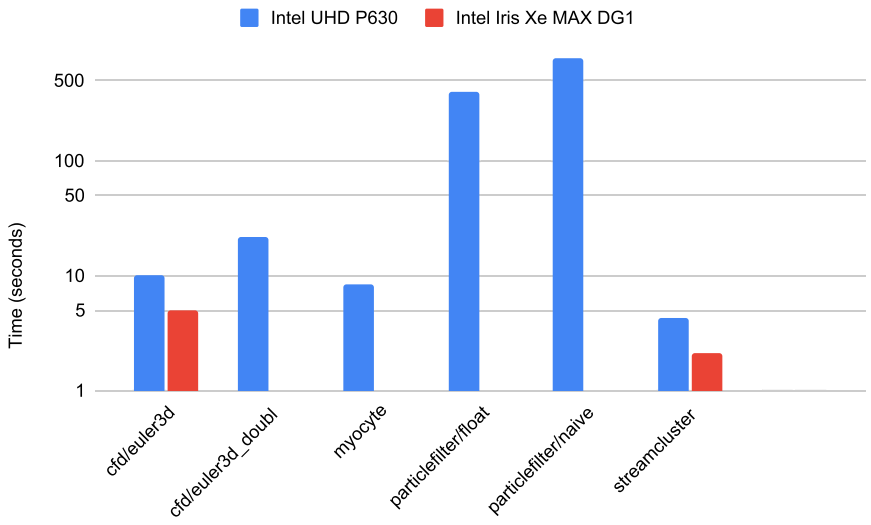

Fig. 9. Time comparison on the Intel GPUs (tests taking more than one second).

Focusing on other relevant aspects such as performance portability on other types of devices such as CPUs or GPUs, we can conclude that:

- Lightweight benchmarks translated with DPCT perform poorly on CPUs, where OpenMP counterparts clearly outperform the migrated code. Synchronization issues in parallel for regions are the leading cause of this behavior. However, the DPCT versions are faster than the OpenMP codes for computationally-demanding benchmarks.
- Not all benchmarks run successfully on Intel's DG1 GPU, due to its lack of double-precision support. As expected, performance results are better on the DG1 than on the integrated UHD P630 GPU. Nevertheless, as Rodinia benchmarks are not fully optimized for GPU devices, higher performance rates can be achieved by tuning parameters related to data-parallelism exploitation.

Finally, in order to facilitate the reproducibility of our experiments and to encourage further comparisons, the source codes and log files generated in this work have been publicly released at https://github.com/artecs-group/rodinia-dpct-dpcpp.

CRediT authorship contribution statement

Germán Castaño: conception and design of study, acquisition of data, analysis and/or interpretation of data, drafting the manuscript, approval of the version of the manuscript to be published. **Youssef El Faqir El Rhazoui:** acquisition of data, analysis and/or interpretation of data, approval of the version of the manuscript to be published. **Carlos García:** conception and design of study, acquisition of data, analysis and/or interpretation of data, revising the manuscript critically for important intellectual content, drafting the manuscript, approval of the version of the manuscript to be published. **Manuel Prieto-Matias:** analysis and/or interpretation of data, revising the manuscript critically for important intellectual content, drafting the manuscript, approval of the version of the manuscript to be published.

Declaration of competing interest

The authors certify that they have NO affiliations with or involvement in any organization or entity with any financial interest (such as honoraria; educational grants; participation in speakers' bureaus; membership, employment, consultancies, stock ownership, or other equity interest; and expert testimony or patent-licensing arrangements), or non-financial interest (such as personal or professional relationships, affiliations, knowledge or beliefs) in the subject matter or materials discussed in this manuscript.

Acknowledgment

This paper has been supported by the Community of Madrid; UE FEDER and the Spanish MINECO under grants S2018/TCS-4423, RTI2018-093684-B-I00 and PID2021-126576NB-I00.

References

[1] B. Aktemur, M. Metzger, N. Saiapova, M. Strasuns, Debugging SYCL programs on heterogeneous Intel® architectures, in: Proceedings of the International Workshop on OpenCL, IWOCL '20, 2020, pp. 1–10.
[2] A. Alekseenko, S. Páll, E. Lindahl, Experiences with adding sycl support to gromacs, in: International Workshop on OpenCL, IWOCL'21, Association for Computing Machinery, New York, NY, USA, 2021, https://doi.org/10.1145/3456669.3456690.
[3] Aksel Alpay, hipSYCL implementation, https://github.com/illuhad/hipSYCL, 2019.
[4] Apple unleashes M1, https://www.apple.com/newsroom/2020/11/apple-unleashes-m1, 2020.
[5] B. Ashbaugh, A. Bader, J. Brodman, J. Hammond, M. Kinsner, J. Pennycook, R. Schulz, J. Sewall, Data parallel C++: enhancing SYCL through extensions for productivity and performance, in: Proceedings of the International Workshop on OpenCL, IWOCL '20, Association for Computing Machinery, New York, NY, USA, 2020, https://doi.org/10.1145/3388333.3388653.
[6] S. Che, M. Boyer, J. Meng, D. Tarjan, J.W. Sheaffer, S.-H. Lee, K. Skadron, Rodinia: a benchmark suite for heterogeneous computing, in: 2009 IEEE International Symposium on Workload Characterization (IISWC), IEEE, 2009, pp. 44–54.
[7] S. Christgau, T. Steinke, Porting a legacy CUDA stencil code to oneAPI, in: 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2020, pp. 359–367.
[8] Codeplay Software, Computecpp, https://codeplay.com/products/computesuite/computecpp, 2019.
[9] M. Costanzo, E. Rucci, C. Garcia Sanchez, M. Naiouf, Early experiences migrating CUDA codes to oneAPI, short papers of the 9th conference on cloud computing conference, big data & emerging topics, https://doi.org/10.35537/10915/121564, 2021.
[10] DPC++ Intel® compatibility tool developer guide and reference, https://software.intel.com/content/www/us/en/develop/documentation/intel-dpcpp-compatibility-tool-user-guide/top.html, 2021.
[11] DPCPP CUDA examples docker image, https://github.com/Ruyk/dpcpp-cuda-examples-docker, 2020.
[12] R. Farber, Parallel Programming with OpenACC, 1st edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2016.

[13] D.B. Kirk, W.-m.W. Hwu, Programming Massively Parallel Processors: A Hands-on Approach, 1st edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.
[14] R. Nozal, J.L. Bosque, Exploiting co-execution with oneAPI: heterogeneity from a modern perspective, in: Euro-Par 2021: Parallel Processing. Euro-Par 2021, 2021.
[15] NVIDIA visual profiler, https://developer.nvidia.com/nvidia-visual-profiler, 2021.
[16] oneAPI GPU optimization guide, https://www.intel.com/content/dam/develop/external/us/en/documents/oneapi-gpu-optimization-guide.pdf, 2021.
[17] J. Reinders, B. Ashbaugh, J. Brodman, M. Kinsner, J. Pennycook, X. Tian, Data Parallel C++: Mastering DPC++ for Programming of Heterogeneous Systems Using C++ and SYCL, Springer Nature, 2021.
[18] R. Reyes, V. Lomüller, SYCL: single-source C++ accelerator programming, in: G.R. Joubert, H. Leather, M. Parsons, F.J. Peters, M. Sawyer (Eds.), Parallel Computing: On the Road to Exascale, Proceedings of the International Conference on Parallel Computing, ParCo 2015, 1-4 September 2015, Edinburgh, Scotland, UK, in: Advances in Parallel Computing, vol. 27, IOS Press, 2015, pp. 673–682.
[19] Khronos SYCL working group, SYCL 1.2.1 specification, https://www.khronos.org/registry/SYCL/specs/sycl-1.2.1.pdf, 2020.
[20] The oneAPI specification, https://spec.oneapi.com/.
[21] The OpenACC specification, https://www.openacc.org/.
[22] Khronos OpenCL Working Group, The OpenCL specification, version 1.1, https://www.khronos.org/registry/cl/specs/opencl-1.1.pdf, 2011.
[23] The OpenMP specification, https://www.openmp.org/.
[24] The triSYCL project, https://github.com/triSYCL/triSYCL, 2019.
[25] W. Yong, Z. Yongfa, W. Scott, Y. Wang, X. Qing, W. Chen, Developing medical ultrasound imaging application across gpu, fpga, and cpu using oneapi, in: International Workshop on OpenCL, IWOCL'21, Association for Computing Machinery, New York, NY, USA, 2021, https://doi.org/10.1145/3456669.3456680.

Germán Castaño received the degree in computer science at Complutense University of Madrid last 2021 June. His final degree project dealt with the use of the Intel'oneAPI toolkit for DPCT code migration.

Youssef El Faqir El Rhazoui is a PhD candidate at Complutense University of Madrid studying computer science. His research includes parallel computation, and GPGPU algorithm design for massively parallel simulations. Actually he is focuses on comparing SYCL and DPC ++ with other paradigms such as CUDA or OpenMP of heterogeneous programming under the prism of performance and programmability.

Carlos García received the B.S. degree in physics and the Ph.D. degree in computer science from the Universidad Complutense de Madrid (UCM), Madrid, Spain, in 1999 and 2007, respectively. He is currently an Associate Professor with UCM. He has authored more than 50 research papers, including more than 25 peer-reviewed articles in international journals. He has frequently served as a referee for international journals on image processing and high-performance computing. His research interests include high- performance computing for heterogeneous parallel architecture, including efficient parallel exploitation on modern devices such as multicore, many- core, GPUs, and FPGAs.

Manuel Prieto-Matias received the Ph.D. degree in computer science from the Complutense University of Madrid (UCM), Madrid, Spain, in 2000. He is currently a Full Professor with the Department of Computer Architecture, UCM. He has co-written numerous articles in journals and for international conferences in the field of parallel computing and computer architecture. His research interests include parallel computing and computer architecture. Most of his activities have focused on leveraging parallel computing platforms and on complexity-effective microarchitecture design. His research addresses emerging issues related to heterogeneous systems, memory hierarchy performance, and energy-aware computing, with a special emphasis on the interaction between the system software and the underlying architecture.