Using hStreams Programming Library for Accelerating a Real-Life Application on Intel MIC

Lukasz Szustak[1], Kamil Halbiniak[1(✉)], Adam Kulawik[1], Roman Wyrzykowski[1], Piotr Uminski[2], and Marcin Sasinowski[2]

[1] Czestochowa University of Technology, Częstochowa, Poland
{lszustak,khalbiniak,adam.kulawik,roman}@icis.pcz.pl
[2] Intel Corporation, Santa Clara, USA
{piotr.uminski,marcin.sasinowski}@intel.com

Abstract. The main goal of this paper is the suitability assessment of the hStreams programming library for porting a real-life scientific application to heterogeneous platforms with Intel Xeon Phi coprocessors. This emerging library offers a higher level of abstraction to provide effective concurrency among tasks, and control over the overall performance. In our study, we focus on applying the FIFO streaming model for a parallel application which implements the numerical model of alloy solidification. In the paper, we show how scientific applications can benefit from multiple streams. To take full advantages of hStreams, we propose a decomposition of the studied application that allows us to distribute tasks belonging to the computational core of the application among two logical streams within two logical/physical domains. Effective overlapping computations with data transfers is another goal achieved in this way. The proposed approach allows us to execute the whole application 3.5 times faster than the original parallel version running on two CPUs.

Keywords: Intel MIC · Hybrid architecture · Numerical modeling of solidification · Heterogeneous programming · Hstreams library · Task and data parallelism

1 Introduction

Efficient concurrency on the task level is difficult to achieve, especially on heterogeneous platforms. An emerging effort on the way to meet this challenge is the hStreams programming framework [1–3], a new heterogeneous streaming library. It is based on a simple FIFO streaming model, and supports concurrency across nodes, among tasks within a node, and between data transfers and computation.

This research was conducted with the financial support of National Science Centre grant no. UMO-2011/03/B/ST6/03500. The authors are grateful to the Czestochowa University of Technology for granting access to Intel Xeon Phi coprocessors provided by the MICLAB project no. POIG.02.03.00.24-093/13.

© Springer International Publishing AG 2016
J. Carretero et al. (Eds.): ICA3PP 2016 Workshops, LNCS 10049, pp. 373–382, 2016.
DOI: 10.1007/978-3-319-49956-7_30

This framework is aimed at making it easier to port and tune task-parallel codes by offering such features as [1]: (i) separation of concerns, (ii) sequential semantics, (iii) task concurrency, (iv) pipeline parallelism, and (v) unified interface to heterogeneous platforms. In particular, the first feature addresses key programming productivity issues by allowing a separation of concerns between (1) the expression of functional semantics and disclosure of task parallelism, and (2) the performance tuning and control over mapping tasks onto a platform. As a result, while creators of scientific algorithms receive something simple and intuitive, code tuners may work long after them, having the freedom to control over the code execution without the need for application domain expertise. A detailed comparison of hStreams with other heterogeneous programming environments such as OpenMP, OmpSs, Offload Streams and CUDA Streams is presented in paper [1].

Heterogeneous platforms become increasingly popular in many application domains [2–4]. The combination of using a general-purpose CPUs combined with specialized computing devices (e.g., GPU, Intel Xeon Phi or FPGA) enabled in many cases for accelerating an application by significant amounts [4–6]. However, realizing these performance potentials remains a challenging issue.

The main goal of this paper is the suitability assessment of the hStreams framework for porting a real-life scientific application to heterogeneous platforms with Intel Xeon Phi coprocessors. We focus on utilizing the FIFO streaming model in a parallel application which implements a numerical model of alloy solidification. This application has been already studied in our previous work [7], where we developed an approach for porting and optimizing the application on computing platforms with a single Intel Xeon Phi accelerator [4]. The proposed scheme of parallelization and workload distribution was implemented using the offload interface [7], dedicated directly for the Intel MIC architecture.

The contribution of this paper to the area of co-design technologies are as follows: (1) demonstration of applicability of the hStreams programming framework for porting a complex application to a heterogeneous platform in a relative quick and easy way, which justifies the conclusion that using the hStreams framework increases the level of abstraction for the code development in hybrid hardware environments; (2) hardware-aware performance tuning of the resulting code with its experimental evaluation showing practically the same performance as in the case of the low-level offload interface.

The material of the paper is organized as follow. Section 2 provides an overview of the hStreams library, while Sect. 3 introduces the numerical model of solidification, and the idea of its parallelization on platforms containing Intel Xeon Phi coprocessors. The next section outlines the most important details of mapping the solidification application onto heterogeneous streams. Section 5 presents performance results achieved for the proposed approach, while Sect. 6 concludes the paper and addresses future works.

2 Introduction to Hetero Streams Library

2.1 Overview of hStreams

The hetero Streams library (hStreams) [1,2] allows stream programming in heterogeneous platforms consisting of Intel Xeon CPUs and Intel Xeon Phi coprocessors. Stream programming model assumes existence of one or more FIFOs abstractions, where computation jobs are submitted on the computing entities.

Before proceeding further, the introduction is needed for two key definitions that hStreams uses: **source** is a place where work is enqueued to be performed, and **sink** is a place where work is executed. Source and sink can either share resources of the same processor or reside on separate ones. In a typical scenario, source resides on an Intel Xeon CPU, while sinks are present on the same CPU, as well as on Intel coprocessors connected to the main processor over PCIe.

Memory resources shared between source and sinks are called **logical buffers**. Logical buffers are registered by the application on the source. Once the hStreams run-time is aware of the buffer, a pointer to a memory location anywhere inside that buffer is recognized as a handle, and can be used for performing data transfers or compute actions involving that buffer. A logical buffer created by the user may have instantiations in many **logical domains** beside the source. Those instantiations of the buffer are called **physical buffers**. A logical buffer must have a corresponding physical buffer on the logical domain where it is intended to be used (either as an operand of a data transfer or a compute action).

Actions are enqueued in a FIFO queue called **stream**. From the operating systems point of view, the stream is a subset of processor cores with access to the local memory. There are three categories of actions: **task computations**, **memory transfers** and **synchronization**. The task computation is performed entirely on the sink side. Memory transfers are performed between buffers on the source and their sink instantiations. Transfers are defined by the direction (source to sink or sink to source) and source-side buffer addresses. Synchronization actions involve the sink endpoint of a stream waiting on a collection of events, triggered by the completion of actions enqueued in any stream.

Streams are organized into **logical domains**. Memory buffers are shared by all streams inside a single logical domain, while being disjointed from other logical domains. One or more logical domains belong to a **physical domain**. The physical domain can be treated as physical device: an Intel Xeon Phi coprocessor or an Intel Xeon server. This approach allows us to have the same API for a coprocessor and server, while also sharing the same memory on server.

Internally, hStreams has implicit dependency management. By default a task enqueued in a stream depends on the previous task in this stream and on all buffers used by this task, but does not depend on memory transfers of buffers not related to the previous task. Dependencies can also be controlled explicitly by the application - for example, the application can wait for completion of one or more

previously defined events. Such dependency management allows programmers to hide communication behind computation.

Two levels of API are exposed by hStreams - the higher level App API and lower level Core API. The former offers a subset of the hStreams functionality and is designed to allow a novice user to quickly start writing programs. Its productivity is boosted by helper functions and common building blocks. The Core API - on the other hand - exposes the full functionality of hStreams, and is targeted at a more advanced user. Currently, the hStreams library supports Intel Xeon CPUs and the first generation of Intel Xeon Phi coprocessors. Support for other configurations may be added in the future. The hStreams framework was created by Intel and is maintained on the public repository. Its latest version and source code is available at https://github.com/01org/hetero-streams.

2.2 Comparison of hStreams with OpenMP

In this section, we briefly compare hStreams with OpenMP as the most popular parallel programming standard, which offers support for heterogeneous computing [8]. The most obvious differences between them is that hStreams represents a library-based API, while OpenMP is a compiler-based language extension [1]. An important advantage of hStreams is independence from the compilers. In this case, the utilization of new features requires only updating the library, unlike OpenMP where new mechanisms are available only after updating the compiler to the latest version. For many programmers who prefer to change compilers rarely, the use of the library-based extensions seems to be most attractive.

Unlike OpenMP, the Hetero Streams library provides an uniform interface for heterogeneous platforms [1,2]. Both environments are based on the host-centric model, where one of the host threads transfers data and computation to the platform components. However, in hStreams all the resources of a platform are handled in uniform way, whie OpenMP separates constructs used to assign the application workload to host and remote devices. The current version of the hStreams library gives also the possibility for offloading computation to remote nodes over fabric. The great advantage of hStreams over OpenMP is the ability to subdividing a device, that allows executing multiple offload regions concurrently.

The differences between hStreams and OpenMP are noticeable also in data management. Both environments give the possibility to transfer data from memory of one device to another. In hStreams, buffers used for data movements have to be allocated before starting the transfer, while in OpenMP data allocations can be performed explicitly or implicitly. Opposite to OpenMP, the hStreams framework provides also an efficient support for memory allocations in different memory types [1].

3 Application: Numerical Model of Solidification and Parallelization on Platforms with Intel MIC

3.1 Numerical Model

The phase-field method is a powerful tool for solving interfacial problems in materials science [9]. It has mainly been applied to solidification dynamics [10], but it has also been used for other phenomena such as viscous fingering [11], fracture dynamics, [12], and vesicle dynamics [9]. The number of scientific papers related to the phase-field method grows since the 90 years of XX century, reaching for the last 7 years more than 400 positions (according to the SCOPUS database) [13].

In the numerical examples studied in this paper, a binary alloy of Ni-Cu is considered as a system of the ideal metal mixture in the liquid and solid phases. The numerical model [14] refers to the dendritic solidification process in the isothermal conditions with constant diffusivity coefficients for both phases. It allows us to use the field-phase model defined by Warren and Boettinger [14]. In this model, the growth of microstructure during the solidification process is determined by solving a system of two PDEs [14,15], which define the phase content ϕ and concentration c of the alloy dopant (one of the alloy components).

The resulting numerical scheme belongs to the group of forward-in-time iterative algorithms [7]. The application code consists of two main blocks of computation, which are responsible for determining either the phase content (Fig. 1) or the dopant concentration. In the model studied in the paper, values of ϕ and c are calculated for nodes uniformly distributed across a square domain. However, the presented approach, which is based on the generalized finite difference method, allows for solving PDEs not only for regular, but also irregular grids.

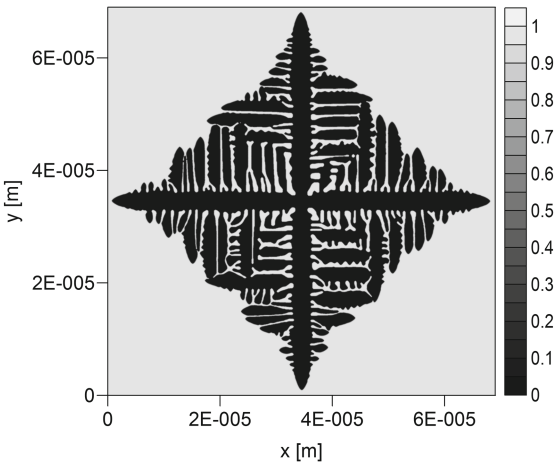

Fig. 1. Phase content for the simulated time $t = 2.75 \times 10^{-3} s$ (original code)

3.2 Idea of Parallelization for Platforms with MIC

In the studied application, computation are interleaved with writing partial results to a file. In the original version (Fig. 2a), parallel computations are executed for subsequent time steps, while writing results to the file is performed after the first time step, and then after every package of 2000 time steps. Figure 2b shows the idea of adapting the application to platforms with a single Intel Xeon Phi. In this approach, the coprocessor is employed to perform major parallel workloads, while the rest of application is assigned to CPU, as not requiring massively parallel resources. In consequence, writing data to the file is the responsibility of CPU, while the coprocessor provides execution of parallel regions of the code.

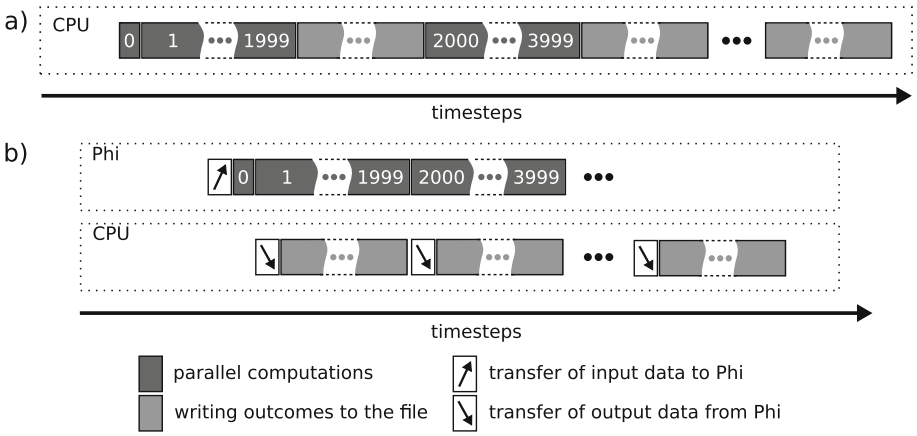

Fig. 2. Idea of adapting solidification application to platforms with Intel MIC [7]

At the beginning, all the input data are transferred from CPU to the coprocessor, which then starts computation for the first time step. After finishing it, all the results are transferred back to CPU. During this transfer, coprocessor starts computations for the next package of 2000 time steps. At the same time, CPU begins writing results to the file, immediately after receiving outcomes from the coprocessor. Such a scheme is repeated for every package of 2000 time steps. A critical performance challenge here is to overlap workload performed by the coprocessor with data movements. To meet this challenge, data transfers between CPU and Xeon Phi, writing data to the file, as well as computation have to be performed simultaneously.

4 Porting with hStreams

4.1 Mapping Application Workload onto Heterogeneous Streams

The hStreams library supports the task parallelism by creating multiple streams. This advantage can be efficiently applied for the proposed idea of adapting the

solidification application to platforms with a single Intel MIC. Our approach distinguishes the two main tasks: (i) writing outcomes to the file, and (ii) running parallel computation. These tasks are mapped onto two logical streams created within two logical domains. This solution allows for executing streams on different computing resources, such as processor and coprocessor.

The idea of mapping the application on heterogeneous streams is illustrated in Fig. 3. While the first stream is responsible for parallel computation performed for subsequent packages of 2000 time steps, the second one has to provide transfers of outcomes from the first stream, in order to write them further to the file. These streams are assigned respectively to the coprocessor and CPU.

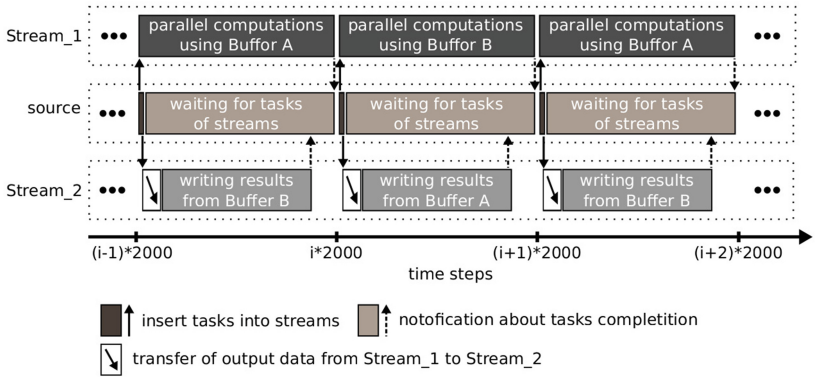

Fig. 3. Mapping solidification application onto heterogeneous streams

Because of mutual dependencies between the execution of streams, their synchronizations becomes a crucial issue. The hStreams library offers two scenarios for solving this issue that correspond to various ways of filling queues of streams. The first way requires to fill the FIFO queue before the stream execution, while the second one refill the FIFO queue during the execution of stream. In both cases, the source process is responsible for the management of queues. For the first way, the stream synchronization is based on the completion of events that have to be inserted into the streams before execution. The second way, called the active synchronization, employs the source process to provide the synchronization of streams during execution. In this scenario, the source process waits for the completion of tasks of a stream, in order to insert subsequent tasks to queues, and then run streams. In our approach, the second scenario is chosen (see Fig. 3) as more suitable for the proposed idea of parallelization. As a results, the synchronization points occur after every package of 2000 time steps.

Selecting an appropriate method for providing efficient data transfers is important for the overall performance. In the proposed approach, data are transferred excluding the source process. It allows us to reduce the communication path from the default scheme Stream_1 → source → Stream_2 to the shorter

one, where data are transferred directly between streams: Stream_1 → Stream_2. The task of data movement that downloads outcomes from the logical domain of Stream_1 is inserted into queue of Stream_2.

To overlap computation with data transfers, the double buffering techniques is applied: the first buffer is used to provide computation while the second one is responsible for data movement of outcomes of the previous time steps (Fig. 3). However, a right policy of hStreams data dependencies has to be applied for enabling this optimization. The default police `HSTR_DEP_POLICY_CONSERVATIVE` prevents the simultaneous execution of tasks for data transfers and computation. To solve this problem, the policy `HSTR_DEP_POLICY_BUFFERS` has to be set in order to ensure the asynchronous execution of these tasks using different buffers.

By defauult, streams are executed on coprocessors. Since Stream_2 should run on the CPU site, the hStreams Core API has to be used to provide such a mapping. This API allows programmers to perform a more advanced management of the hStream library.

4.2 Data Parallelization Within Streams

The original CPU version of the application uses the OpenMP standard to utilize cores/threads, based on the OpenMP construction `#pragma omp parallel for`. Since the Intel Xeon Phi coprocessors supports OpenMP, the application code can be rather easily ported to this platform. To ensure the best overall performance without significant modifications in the source code, we use several compiler-friendly optimizations, and empirically determine the best OpenMP setup for the loop scheduling.

The utilization of vector processing is crucial for ensuring the best performance on Intel Xeon Phi. The quickest way to achieve this goal is the compiler-based automatic vectorization. However, in the studied case the innermost loop cannot be vectorized safely, mainly because of data dependencies. To solve this problem, we propose to change slightly the code by adding temporary vectors responsible for loading the necessary data from the irregular memory region, and than providing SIMD computations (see our previous work [5,7]).

5 Performance Results

In this study, we use the platform [16] equipped with two Intel Xeon E5-2699 v3 CPUs (Haswell-EP), and Intel Xeon Phi 7120P coprocessor (Knight Corner). The benchmarks are compiled using the Intel icpc compiler (v.15.0.2) with the same optimization flags. All tests are performed for modeling solidification application using the double precision floating-point format, 110000 time steps, and grid with 4000000 nodes (2000 nodes along each dimensions x and y).

Table 1 presents the comparison of the execution times obtained for: (i) original CPU parallel version of the solidification application, (ii) optimized parallel version based on the offload interface (see our previous work [7]), and (iii) new parallel code programmed with hStreams. Both the offload- and hStreams-based

versions correspond to the proposed adaptation of the studied application to platforms with a single Intel Xeon Phi.

The total execution time of the original version (see Fig. 2a) includes the sum of execution times necessary for performing parallel computation and writing outcomes to the file. The proposed approach (see Fig. 2b) allows us to hide more than 99 % of computations behind data movements, for both the offload- and hStreams-based versions, and finally accelerate the whole application about 3.50x. Comparing the execution times of the hStreams- and offload-based codes, we can see that the difference is negligible, since it is equal to 0.28 %.

Table 1. Performance results for different versions of the solidification application

	Tasks			
Code version	data movements	parallel computation	Time	Speedup
original	CPU	CPU	641 min 32 s	-
offload-based	CPU	MIC	183 min 08 s	3.50x
hStreams-based	CPU	MIC	183 min 39 s	3.49x

6 Conclusions and Future Works

The hStreams programming library is a promising solution for the exploration of emerging multi- and manycore architectures that become increasingly complex, hierarchical and heterogeneous. It is expected that the potential of using hStreams on current and future platforms will be manifested for a wide range of real-life applications. Our research allow us to conclude that the hStream library enables for porting such applications on modern architectures, including Intel MIC, in a relatively quick and easy way.

The streaming abstraction is one of advantages of this library which enables for mapping concurrent tasks onto computing resources. A rich functionality of hStreams, including synchronization scenarios and overlapping tasks, makes this library programmer-friendly, and increases the level of abstraction for the code development. The performed benchmark confirms that the hStreams library allows for achieving the performance results at the same level as the offload model, dedicated directly for the Intel MIC architecture. At the same time, it is worth to mention that the proposed adaptation of the solidification application to platforms with Intel MIC plays the main role in accelerating computations, while the hStreams library and offload interface are “only” tools that allows us to reach this goal.

The performance results achieved in this study provide the basis for further research on the development and optimization of code. The primary direction of our future work is to utilize hStreams for porting the studied application on heterogeneous platforms with more than one Intel Xeon Phi coprocessor, and taking advantage of all the computing resources to process together the application workload. Also, we plan to use our application as a valuable benchmark

for comparing hStreams with other programming models and languages interfaces, and in particular, with the OpenMP 4.x support [17] for heterogeneous computing and task-parallelism.

References

1. Newburn, C.J., et al.: Heterogeneous streaming. In: IPDPSW, AsHES (2016)
2. Jeffers, J., Reinders, J.: Fast matrix computations on heterogeneous streams. In: Jeffers, J., Reinders, J. (eds.), High Performance Parallelism Pearls: Multicore and Many-core Programming Approaches, vol. 2, pp. 49–52. Morgan Kaufmann (2015)
3. Li, Z., et al.: Evaluating the Performance Impact of Multiple Streams on the MIC-based Heterogeneous Platform (2016). arXiv preprint arXiv:1603.08619
4. Szustak, L., Rojek, K., Olas, T., Kuczynski, L., Halbiniak, K., Gepner, P.: Adaptation of MPDATA heterogeneous stencil computation to Intel Xeon Phi coprocessor. Sci. Program. (2015). http://dx.doi.org/10.1155/2015/642705
5. Szustak, L., Halbiniak, K., Kuczynski, L., Wrobel, J., Kulawik, A.: Porting, optimization of solidification application for CPU-MIC hybrid platforms. Accepted to print: Int. J. High Perform. Comput. Appl., 13 (2016)
6. Rojek, K., et al.: Adaptation of fluid model EULAG to graphics processing unit architecture. Concurrency Computations Pract. Experience **27**(4), 937–957 (2015)
7. Szustak, L., Halbiniak, K., Kulawik, A., Wrobel, J., Gepner, P.: Toward parallel modeling of solidification based on the generalized finite difference method using intel xeon phi. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Kitowski, J., Wiatr, K. (eds.) PPAM 2015. LNCS, vol. 9573, pp. 411–422. Springer, Heidelberg (2016). doi:10.1007/978-3-319-32149-3_39
8. OpenMP Application Programming Interface (2015)
9. Steinbach, I.: Phase-field models in materials science. Model. Simul. Mater. Sci. Eng. **17**(7), 73001 (2009)
10. Provatas, N., Elder, K.: Phase-Field Methods in Materials Science and Engineering. Wiley, New York (2010)
11. Folch, R., Casademunt, J., Hernandez-Machado, A., Ramirez-Piscina, L.: Phase-field model for Hele-Shaw flows with arbitrary viscosity contrast. II. Numer. Study. Phys. Rev. E **60**(2), 1734–1740 (1999)
12. Karma, A., Kessler, D., Levine, H.: Phase-field model of mode III dynamic fracture. Phys. Rev. Lett. **87**(4), 40401 (2001)
13. Takaki, T.: Phase-field modeling and simulations of dendrite growth. ISIJ Int. **54**(2), 437–444 (2014)
14. Warren, J.A., Boettinger, W.J.: Prediction of dendritic growth and microsegregation patterns in a binary alloy using the phase-field method. Acta Metall. et Mater. **43**(2), 689–703 (1995)
15. Longinova, T., Amberg, G., Ågren, J.: Phase-field simulations of non-isothermal binary alloy solidification. Acta Mater. **49**(4), 573–581 (2001)
16. Pilot Laboratory of Massively Parallel Systems (MICLab). http://miclab.pl
17. Michael Klemm. Heterogeneous Programming with OpenMP 4.5. https://www.scc.kit.edu/downloads/sca/Heterogeneous%20Programming%20with%20OpenMP%204.5.pdf