

USING OPENCL

Advances in Parallel Computing

This book series publishes research and development results on all aspects of parallel computing. Topics may include one or more of the following: high-speed computing architectures (Grids, clusters, Service Oriented Architectures, etc.), network technology, performance measurement, system software, middleware, algorithm design, development tools, software engineering, services and applications.

Series Editor:

Professor Dr. Gerhard R. Joubert

Volume 21

Recently published in this series

- Vol. 20. I. Foster, W. Gentsch, L. Grandinetti and G.R. Joubert (Eds.), High Performance Computing: From Grids and Clouds to Exascale
- Vol. 19. B. Chapman, F. Desprez, G.R. Joubert, A. Lichnewsky, F. Peters and T. Priol (Eds.), Parallel Computing: From Multicores and GPU's to Petascale
- Vol. 18. W. Gentsch, L. Grandinetti and G. Joubert (Eds.), High Speed and Large Scale Scientific Computing
- Vol. 17. F. Xhafa (Ed.), Parallel Programming, Models and Applications in Grid and P2P Systems
- Vol. 16. L. Grandinetti (Ed.), High Performance Computing and Grids in Action
- Vol. 15. C. Bischof, M. Bückner, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr and F. Peters (Eds.), Parallel Computing: Architectures, Algorithms and Applications

Volumes 1–14 published by Elsevier Science.

ISSN 0927-5452 (print)

ISSN 1879-808X (online)

Using OpenCL

Programming Massively Parallel Computers

Janusz Kowalik

16477-107th PL NE, Bothell, WA 98011, USA

and

Tadeusz Puźniakowski

UG, MFI, Wit Stwosz Street 57, 80-952 Gdańsk, Poland

IOS
Press

Amsterdam • Berlin • Tokyo • Washington, DC

© 2012 The authors and IOS Press.

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, without prior written permission from the publisher.

ISBN 978-1-61499-029-1 (print)

ISBN 978-1-61499-030-7 (online)

Library of Congress Control Number: 2012932792

doi:10.3233/978-1-61499-030-7-i

Publisher

IOS Press BV

Nieuwe Hemweg 6B

1013 BG Amsterdam

Netherlands

fax: +31 20 687 0019

e-mail: order@iospress.nl

Distributor in the USA and Canada

IOS Press, Inc.

4502 Rachael Manor Drive

Fairfax, VA 22032

USA

fax: +1 703 323 3668

e-mail: iosbooks@iospress.com

LEGAL NOTICE

The publisher is not responsible for the use which might be made of the following information.

PRINTED IN THE NETHERLANDS

This book is dedicated to Alex, Bogdan and Gabriela
with love and consideration.

Preface

This book contains the most important and essential information required for designing correct and efficient OpenCL programs. Some details have been omitted but can be found in the provided references. The authors assume that readers are familiar with basic concepts of parallel computation, have some programming experience with C or C++ and have a fundamental understanding of computer architecture. In the book, all terms, definitions and function signatures have been copied from official API documents available on the page of the OpenCL standards creators.

The book was written in 2011, when OpenCL was in transition from its infancy to maturity as a practical programming tool for solving real-life problems in science and engineering. Earlier, the Khronos Group successfully defined OpenCL specifications, and several companies developed stable OpenCL implementations ready for learning and testing. A significant contribution to programming heterogeneous computers was made by NVIDIA which created one of the first working systems for programming massively parallel computers – CUDA. OpenCL has borrowed from CUDA several key concepts. At this time (fall 2011), one can install OpenCL on a heterogeneous computer and perform meaningful computing experiments. Since OpenCL is relatively new, there are not many experienced users or sources of practical information. One can find on the Web some helpful publications about OpenCL, but there is still a shortage of complete descriptions of the system suitable for students and potential users from the scientific and engineering application communities.

Chapter 1 provides short but realistic examples of codes using MPI and OpenMP in order for readers to compare these two mature and very successful systems with the fledgling OpenCL. MPI used for programming clusters and OpenMP for shared memory computers, have achieved remarkable worldwide success for several reasons. Both have been designed by groups of parallel computing specialists that perfectly understood scientific and engineering applications and software development tools. Both MPI and OpenMP are very compact and easy to learn. Our experience indicates that it is possible to teach scientists or students whose disciplines are other than computer science how to use MPI and OpenMP in a several hours time. We hope that OpenCL will benefit from this experience and achieve, in the near future, a similar success.

Paraphrasing the wisdom of Albert Einstein, we need to simplify OpenCL as much as possible but not more. The reader should keep in mind that OpenCL will be evolving and that pioneer users always have to pay an additional price in terms of initially longer program development time and suboptimal performance before they gain experience. The goal of achieving simplicity for OpenCL programming requires an additional comment. OpenCL supporting heterogeneous computing offers us opportunities to select diverse parallel processing devices manufactured by different vendors in order to achieve near-optimal or optimal performance. We can select multi-core CPUs, GPUs, FPGAs and other parallel processing devices to fit the problem we want to solve. This flexibility is welcomed by many users of HPC technology, but it has a price.

Programming heterogeneous computers is somewhat more complicated than writing programs in conventional MPI and OpenMP. We hope this gap will disappear as OpenCL matures and is universally used for solving large scientific and engineering problems.

Acknowledgements

It is our pleasure to acknowledge assistance and contributions made by several persons who helped us in writing and publishing the book.

First of all, we express our deep gratitude to Prof. Gerhard Joubert who has accepted the book as a volume in the book series he is editing, *Advances in Parallel Computing*. We are proud to have our book in his very prestigious book series.

Two members of the Khronos organization, Elizabeth Riegel and Neil Trevett, helped us with evaluating the initial draft of Chapter 2 *Fundamentals* and provided valuable feedback. We thank them for the feedback and for their offer of promoting the book among the Khronos Group member companies.

Our thanks are due to NVIDIA for two hardware grants that enabled our computing work related to the book.

Our thanks are due to Piotr Arłukowicz, who contributed two sections to the book and helped us with editorial issues related to using \LaTeX and the Blender^{3D} modeling open-source program.

We thank two persons who helped us improve the book structure and the language. They are Dominic Eschweiler from FIAS, Germany and Roberta Scholz from Redmond, USA.

We also thank several friends and family members who helped us indirectly by supporting in various ways our book writing effort.

Janusz Kowalik
Tadeusz Puźniakowski

How to read this book

The text and the source code presented in this book are written using different text fonts. Here are some examples of different typography styles collected.

variable – for example:

... the variable *platform* represents an object of class ...

`type` or `class name` – for example:

... the value is always of type `cl_ulong`...

... is an object of class `cl::Platform`...

`constant` or `macro` – for example:

... the value `CL_PLATFORM_EXTENSIONS` means that...

function, **method** or **constructor** – for example:

... the host program has to execute `clGetPlatformIDs`...

... can be retrieved using `cl::Platform::getInfo` method...

... the context is created by `cl::Context` constructor...

`file name` – for example:

... the `cl.h` header file contains...

keyword – for example:

... identified by the `__kernel` qualifier...

Contents

1	Introduction	1
1.1	Existing Standard Parallel Programming Systems	1
1.1.1	MPI	1
1.1.2	OpenMP	4
1.2	Two Parallelization Strategies: Data Parallelism and Task Parallelism	9
1.2.1	Data Parallelism	9
1.2.2	Task Parallelism	9
1.2.3	Example	10
1.3	History and Goals of OpenCL	12
1.3.1	Origins of Using GPU in General Purpose Computing	12
1.3.2	Short History of OpenCL	13
1.4	Heterogeneous Computer Memories and Data Transfer	14
1.4.1	Heterogeneous Computer Memories	14
1.4.2	Data Transfer	15
1.4.3	The Fourth Generation CUDA	15
1.5	Host Code	16
1.5.1	Phase a. Initialization and Creating Context	17
1.5.2	Phase b. Kernel Creation, Compilation and Preparations	17
1.5.3	Phase c. Creating Command Queues and Kernel Execution	17
1.5.4	Finalization and Releasing Resource	18
1.6	Applications of Heterogeneous Computing	18
1.6.1	Accelerating Scientific/Engineering Applications	19
1.6.2	Conjugate Gradient Method	19
1.6.3	Jacobi Method	21
1.6.4	Power Method	22
1.6.5	Monte Carlo Methods	22
1.6.6	Conclusions	23
1.7	Benchmarking CGM	24
1.7.1	Introduction	24
1.7.2	Additional CGM Description	24
1.7.3	Heterogeneous Machine	24
1.7.4	Algorithm Implementation and Timing Results	24
1.7.5	Conclusions	25

2	OpenCL Fundamentals	27
2.1	OpenCL Overview	27
2.1.1	What is OpenCL	27
2.1.2	CPU + Accelerators	27
2.1.3	Massive Parallelism Idea	27
2.1.4	Work Items and Workgroups	29
2.1.5	OpenCL Execution Model	29
2.1.6	OpenCL Memory Structure	30
2.1.7	OpenCL C Language for Programming Kernels	30
2.1.8	Queues, Events and Context	30
2.1.9	Host Program and Kernel	31
2.1.10	Data Parallelism in OpenCL	31
2.1.11	Task Parallelism in OpenCL	32
2.2	How to Start Using OpenCL	32
2.2.1	Header Files	33
2.2.2	Libraries	33
2.2.3	Compilation	34
2.3	Platforms and Devices	34
2.3.1	OpenCL Platform Properties	36
2.3.2	Devices Provided by Platform	37
2.4	OpenCL Platforms – C++	40
2.5	OpenCL Context to Manage Devices	41
2.5.1	Different Types of Devices	43
2.5.2	CPU Device Type	43
2.5.3	GPU Device Type	44
2.5.4	Accelerator	44
2.5.5	Different Device Types – Summary	44
2.5.6	Context Initialization – by Device Type	45
2.5.7	Context Initialization – Selecting Particular Device	46
2.5.8	Getting Information about Context	47
2.6	OpenCL Context to Manage Devices – C++	48
2.7	Error Handling	50
2.7.1	Checking Error Codes	50
2.7.2	Using Exceptions – Available in C++	53
2.7.3	Using Custom Error Messages	54
2.8	Command Queues	55
2.8.1	In-order Command Queue	55
2.8.2	Out-of-order Command Queue	57
2.8.3	Command Queue Control	60
2.8.4	Profiling Basics	61
2.8.5	Profiling Using Events – C example	61
2.8.6	Profiling Using Events – C++ example	63
2.9	Work-Items and Work-Groups	65
2.9.1	Information About Index Space from a Kernel	66
2.9.2	NDRange Kernel Execution	67
2.9.3	Task Execution	70
2.9.4	Using Work Offset	70

2.10	OpenCL Memory	71
2.10.1	Different Memory Regions – the Kernel Perspective	71
2.10.2	Relaxed Memory Consistency	73
2.10.3	Global and Constant Memory Allocation – Host Code	75
2.10.4	Memory Transfers – the Host Code	78
2.11	Programming and Calling Kernel	79
2.11.1	Loading and Compilation of an OpenCL Program	81
2.11.2	Kernel Invocation and Arguments	88
2.11.3	Kernel Declaration	90
2.11.4	Supported Scalar Data Types	90
2.11.5	Vector Data Types and Common Functions	92
2.11.6	Synchronization Functions	94
2.11.7	Counting Parallel Sum	96
2.11.8	Parallel Sum – Kernel	97
2.11.9	Parallel Sum – Host Program	100
2.12	Structure of the OpenCL Host Program	103
2.12.1	Initialization	104
2.12.2	Preparation of OpenCL Programs	106
2.12.3	Using Binary OpenCL Programs	107
2.12.4	Computation	109
2.12.5	Release of Resources	113
2.13	Structure of OpenCL host Programs in C++	114
2.13.1	Initialization	115
2.13.2	Preparation of OpenCL Programs	115
2.13.3	Using Binary OpenCL Programs	116
2.13.4	Computation	120
2.13.5	Release of Resources	121
2.14	The SAXPY Example	122
2.14.1	Kernel	123
2.14.2	The Example SAXPY Application – C Language	123
2.14.3	The example SAXPY application – C++ language	128
2.15	Step by Step Conversion of an Ordinary C Program to OpenCL	131
2.15.1	Sequential Version	132
2.15.2	OpenCL Initialization	132
2.15.3	Data Allocation on the Device	134
2.15.4	Sequential Function to OpenCL Kernel	135
2.15.5	Loading and Executing a Kernel	136
2.15.6	Gathering Results	139
2.16	Matrix by Vector Multiplication Example	139
2.16.1	The Program Calculating $matrix \times vector$	140
2.16.2	Performance	142
2.16.3	Experiment	142
2.16.4	Conclusions	144
3	Advanced OpenCL	147
3.1	OpenCL Extensions	147
3.1.1	Different Classes of Extensions	147

3.1.2	Detecting Available Extensions from API	148
3.1.3	Using Runtime Extension Functions	149
3.1.4	Using Extensions from OpenCL Program	153
3.2	Debugging OpenCL codes	155
3.2.1	Printf	155
3.2.2	Using GDB	157
3.3	Performance and Double Precision	162
3.3.1	Floating Point Arithmetics	162
3.3.2	Arithmetics Precision – Practical Approach	165
3.3.3	Profiling OpenCL Application	172
3.3.4	Using the Internal Profiler	173
3.3.5	Using External Profiler	180
3.3.6	Effective Use of Memories – Memory Access Patterns	183
3.3.7	Matrix Multiplication – Optimization Issues	189
3.4	OpenCL and OpenGL	194
3.4.1	Extensions Used	195
3.4.2	Libraries	196
3.4.3	Header Files	196
3.4.4	Common Actions	197
3.4.5	OpenGL Initialization	198
3.4.6	OpenCL Initialization	201
3.4.7	Creating Buffer for OpenGL and OpenCL	203
3.4.8	Kernel	209
3.4.9	Generating Effect	213
3.4.10	Running Kernel that Operates on Shared Buffer	215
3.4.11	Results Display	216
3.4.12	Message Handling	218
3.4.13	Cleanup	219
3.4.14	Notes and Further Reading	221
3.5	Case Study – Genetic Algorithm	221
3.5.1	Historical Notes	221
3.5.2	Terminology	221
3.5.3	Genetic Algorithm	222
3.5.4	Example Problem Definition	225
3.5.5	Genetic Algorithm Implementation Overview	225
3.5.6	OpenCL Program	226
3.5.7	Most Important Elements of Host Code	234
3.5.8	Summary	241
3.5.9	Experiment Results	241
A	Comparing CUDA with OpenCL	245
A.1	Introduction to CUDA	245
A.1.1	Short CUDA Overview	245
A.1.2	CUDA 4.0 Release and Compatibility	245
A.1.3	CUDA Versions and Device Capability	247
A.2	CUDA Runtime API Example	249
A.2.1	CUDA Program Explained	251

A.2.2	Blocks and Threads Indexing Formulas	257
A.2.3	Runtime Error Handling	260
A.2.4	CUDA Driver API Example	262
B	Theoretical Foundations of Heterogeneous Computing	269
B.1	Parallel Computer Architectures	269
B.1.1	Clusters and SMP	269
B.1.2	DSM and ccNUMA	270
B.1.3	Parallel Chip Computer	270
B.1.4	Performance of OpenCL Programs	270
B.2	Combining MPI with OpenCL	277
C	Matrix Multiplication – Algorithm and Implementation	279
C.1	Matrix Multiplication	279
C.2	Implementation	279
C.2.1	OpenCL Kernel	279
C.2.2	Initialization and Setup	280
C.2.3	Kernel Arguments	282
C.2.4	Executing Kernel	282
D	Using Examples Attached to the Book	285
D.1	Compilation and Setup	286
D.1.1	Linux	286
D.1.2	Windows	286
	Bibliography and References	289

