

Bitonic Sorting

Sample User's Guide

Intel® SDK for OpenCL * Applications - Samples

Document Number: 325262-003US

Contents

Contents	2
Legal Information	3
About Bitonic Sorting Sample	4
Motivation	4
Algorithm	4
OpenCL* Implementation	4
Understanding OpenCL Performance Characteristics	5
Reference (Native) Implementation	5
Controlling the Sample	5
References	6

Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to:

<http://www.intel.com/design/literature.htm>.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to:

http://www.intel.com/products/processor_number/.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Intel, Intel logo, Intel Core, VTune, Xeon are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission from Khronos.

Microsoft product screen shot(s) reprinted with permission from Microsoft Corporation.

Copyright © 2010-2013 Intel Corporation. All rights reserved.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

About Bitonic Sorting Sample

Bitonic Sorting sample illustrates implementing calculation kernels using OpenCL* C99 and parallelizing kernels by running several work-groups in parallel.

This sample demonstrates how to sort arbitrary input array of integer values with OpenCL using Single Instruction Multiple Data (SIMD) bitonic sorting networks. This implementation is very general, so it permits you to add *<key/value>* sorting with relatively low effort.

Motivation

Sorting algorithms are among most widely used building blocks. Bitonic sorting algorithm implemented in this sample is based on properties of bitonic sequence and principles of so-called sorting networks. It enables efficient SIMD-style parallelism through OpenCL vector data types.

Algorithm

For an array of length $2^N \cdot 4$, this algorithm completes N stages of sorting. The first stage has one pass. As a result, the kernel forms bitonic sequences of size four using SIMD sorting network inside each item of the input array.

For each successive stage, the number of passes is incremented by one and the sequence size is doubled by merging two neighboring items.

For general reference on bitonic sorting networks, see "Bitonic Sort" at <http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/bitonicen.htm>. For reference on sorting networks using SIMD data types, see "Efficient implementation of sorting on multi-core SIMD CPU architecture" at <http://portal.acm.org/citation.cfm?id=1454159.1454171&coll=GUIDE&dl=GUIDE&CFID=105910684&CFTOKEN=82233064>.

OpenCL* Implementation

Code Highlights

Bitonic sort OpenCL* kernel of `BitonicSort.cl` file performs the specified stage of each pass. Every input array item or item pair (depending on the pass number) corresponds to a unique global ID that the kernel uses for their identification. The full sorting sequence consists of repetitive kernel calls performed in `ExecuteSortKernel()` function of `BitonicSort.cpp` file.

Limitations

For the sake of simplicity, the current version of the sample requires input array of size of $4 \cdot 2^N$ 32-bit integer items, where N is a positive integer.

Understanding OpenCL Performance Characteristics

Benefits of Using Vector Data Types

This sample implements the bitonic sort algorithm using vector data types. Explicit usage of these types, such as `int4` or `float4`, enables the following optimizations:

- You can work with quads instead of single integers. This removes unnecessary branches, saves memory bandwidth, and optimizes CPU cache usage.
- You can use sorting network inside a single vector item during the last pass on every stage. This permits merging two last passes together to save an extra kernel invocation per stage, thus decreasing execution overhead.

Beside the maximum possible 4x speedup brought by SIMD register usage, these optimizations bring additional 25% speedup to the explicitly vectorized version. As a result, you get approximately 5x speedup in total.

Work-Group Size Considerations

Valid work-group sizes on Intel platforms range from 1 to 8192 elements. The sample uses `NULL` argument for local work group size to allow runtime to choose sub-optimal workgroup size by itself depending on device.

Reference (Native) Implementation

Reference implementation is done in `ExecuteSortReference()` routine of `BitonicSort.cpp` file. This is single-threaded code that performs exactly the same bitonic sort sequence as OpenCL code, but uses pure scalar C nested loop.

Controlling the Sample

The sample executable is a console application. To set the sorting direction and input array size, use command-line arguments. If the command line is empty, the sample uses default values.

The sample supports the following command-line parameters:

Option	Description
<code>-h, --help</code>	Show this help text and exit.
<code>-p, --platform number-or-string</code>	Select platform, devices of which are used.
<code>-t, --type all cpu gpu acc default <OpenCL constant for device type></code>	Select the device by type on which the OpenCL kernel is executed.
<code>-d, --device number-or-string</code>	Select the device on which all stuff is executed.
<code>-r, --reverse-sort</code>	Perform descending sort (default is ascending).
<code>-s, --size <integer></code>	Set input/output array size.

References

- H. W. Lang. Bitonic Sort at <http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/bitonicen.htm>
- Jatin Chhugani, Anthony D. Nguyen, Victor W. Lee, William Macy, Mostafa Hagog, Yen-Kuang Chen, Akram Baransi, Sanjeev Kumar, Pradeep Dubey: Efficient implementation of sorting on multi-core SIMD CPU architecture. PVLDB 1(2): 1313-1324 (2008) at <http://portal.acm.org/citation.cfm?id=1454159.1454171&coll=GUIDE&dl=GUIDE&CFID=105910684&CFTOKEN=82233064>