

GPU-Quicksort in OpenCL 2.0: Nested Parallelism and Work-Group Scan Functions

By [Robert I. \(Intel\)](https://software.intel.com/en-us/user/414915) (<https://software.intel.com/en-us/user/414915>), Updated March 4, 2015

Translate ▶

- [Introduction](#)
- [A Brief History of Quicksort](#)
- [A Brief Introduction to GPU-Quicksort](#)
- [GPU-Quicksort in OpenCL 1.2](#)
- [Converting GPU-Quicksort to OpenCL 2.0](#)
- [Tutorial Requirements](#)
- [Running the Tutorial](#)
- [Conclusion](#)
- [References](#)
- [About the Author](#)
- [Download Code](#)

Introduction

This tutorial shows how to use two powerful features of OpenCL™ 2.0: `enqueue_kernel` functions that allow you to enqueue kernels from the device and `work_group_scan_exclusive_add` and `work_group_scan_inclusive_add`, two of a new set of work-group functions that were added to OpenCL 2.0 to facilitate scan and reduce operations across work-items of a work-group. This tutorial demonstrates these features on our very own GPU-Quicksort implementation in OpenCL, which, as far as we know, is the first known implementation of that algorithm in OpenCL. The tutorial shows an important design pattern of enqueueing kernels of `NDRange` of size 1 to perform housekeeping and scheduling operations previously reserved for the CPU.

In this tutorial you'll see how to use Intel® tools for OpenCL. The Intel® SDK for OpenCL™ Applications is the development platform for OpenCL applications on Intel® Architecture. In addition to the SDK, the Intel® VTune™ Amplifier XE is the tool to analyze and optimize applications on Intel platforms on both CPU and GPU, and it includes support for OpenCL. Learn more about Intel tools for OpenCL at <http://intel.com/software/opengl> (<http://intel.com/software/opengl>).

I want to thank Deepti Joshi, Allen Hux, Aaron Kunze, Dillon Sharlet, Adam Lake, Michal Mrozek, Ben Ashbaugh, Ayal Zaks, Yuri Kulakov, Joseph Wells, Lynn Putnam, Jerry Baugh, Jerry Makare, and Chris Davis for their help in developing, reviewing and publishing this article, accompanying code and editing the video. I also want to thank my wife Ellen and my kids Michael and Celine for their unwavering support and understanding.

GPU-Quicksort in OpenCL 1.2

- Three parts:
 - CPU Part: **GPUQSort** function, which splits the sequence into blocks, iteratively launches **gqsort** kernel, when sequences are short enough, launches **lqsort** kernel
 - First Phase GPU Kernel: **gqsort** kernel, which sorts sequences until they are small enough to be sorted by one work group
 - Uses two-pass partitioning technique
 - On the first pass, calculates items less than and greater than the pivot
 - Uses cumulative sums to figure out where to write
 - On the second pass, writes the data to predetermined less than and greater than locations
 - Second Phase GPU Kernel: **lqsort** kernel, which sorts small subsequences
 - Uses same two-pass partitioning technique as **gqsort** kernel
 - Uses simulated stack
 - Switches to Bitonic sort for sequences ≤ 512 elements

13

Intel

A Brief History of Quicksort

Quicksort is an algorithm invented by C.A.R. (“Tony”) Hoare in 1960 while at Moscow State University. Tony Hoare was a graduate student studying Probability Theory under Professor A.N. Kolmogorov. Hoare learned Russian while in the Royal Navy where his uncle was a captain. Hoare was trying to sort words of each incoming Russian sentence into ascending order while working on problems of machine translation of languages. The basic idea of the Quicksort algorithm is to partition the sequence to be sorted around the pivot element, which could be selected from the sequence in a number of different ways, where all the elements less than the pivot are placed on the left hand side of the array, all the elements greater than the pivot are placed on the right hand side of the array, and all the elements that are equal to the pivot are placed in the middle of the array. After the array has been partitioned, the quicksort algorithm is reapplied to both the left hand side and the right hand side of the sequence.

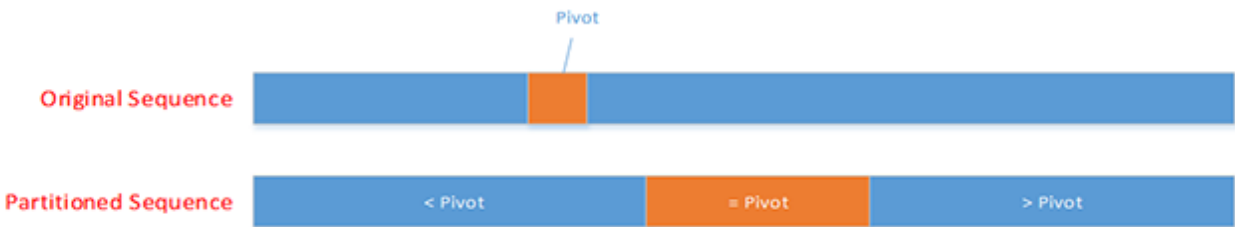


Figure 1. Partitioning the sequence in Quicksort

A Brief Introduction to GPU-Quicksort

The first known quicksort algorithm for the GPUs was developed and described by Shubhabrata Sengupta, Mark Harris, Yao Zhang, and John D. Owens in their paper "Scan Primitives for GPU Computing". The algorithm that we are presenting here, GPU-Quicksort, was first described by Daniel Cederman and Philippas Tsigas in their paper “GPU-Quicksort: A practical Quicksort algorithm for graphics processors” and was an improvement on the first quicksort on the GPU. While originally written in CUDA* to run on Nvidia* discrete graphics cards, the algorithm is easily implementable in OpenCL to run on any hardware that supports the OpenCL API. The algorithm was specifically designed to take advantage of the high GPU bandwidth and runs well on both OpenCL 1.2 (Intel® HD Graphics 4600) and OpenCL 2.0 drivers (Intel® HD Graphics 4600 and Intel® HD Graphics 5300 or above). For an excellent overview of the GPU-Quicksort, see Section 3.1 of the Cederman and Tsigas paper.

As the original Quicksort, GPU-Quicksort recursively partitions the sequence of elements around the pivot until the entire sequence is sorted. Since the algorithm is written for the GPU, it consists of two phases, where in the first phase several work-groups work on different parts of the same sequence until subsequences are small enough to be fully sorted by each work-group in the second phase.

The main idea of GPU-Quicksort is to divide the input sequence into blocks and assign them to different work-groups, where each work-group participates in partitioning the input sequence around the pivot in a two-pass process: first each work-item of a work-group calculates the number of elements less than and greater than the pivot that it sees in the block. The work-group scan add built-ins are used to find the cumulative totals of elements less than and greater than the pivot. An auxiliary array is used to allocate space for each work-group. The second pass thru the data writes the elements less than and greater than the pivot into the allocated space. Finally, the last work-group fills the gap with the pivot value.

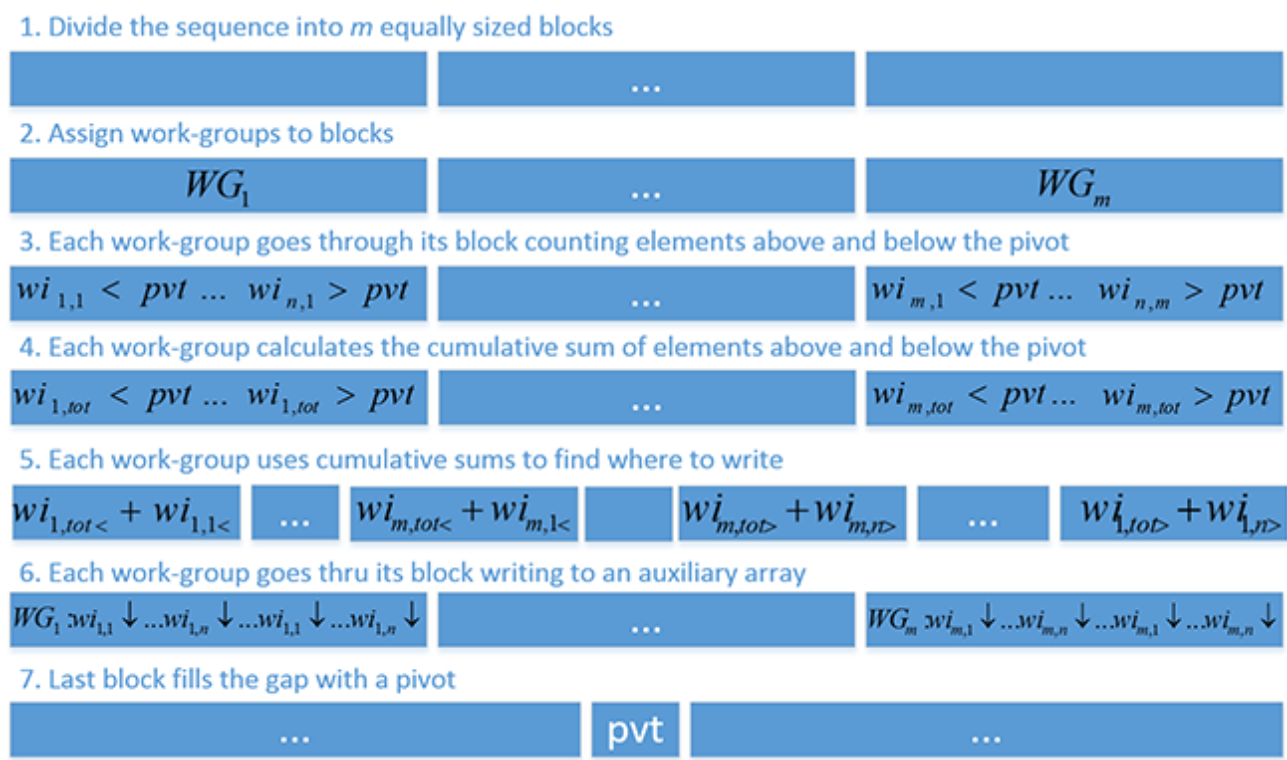


Figure 2. Partitioning the sequence in GPU-Quicksort

GPU-Quicksort in OpenCL 1.2

The GPU-Quicksort in OpenCL 1.2 is a straightforward implementation following the algorithms described in the Cederman/Tsigas paper. The three parts are the CPU portion implemented by the GPUQSort function, the GPU First Phase Kernel implemented by the gqsort_kernel OpenCL kernel function, and the GPU Second Phase Kernel implemented by the lqsort_kernel OpenCL kernel function. The GPUQSort function iteratively launches gqsort_kernel until the initial sequence is partitioned into small enough chunks such that each chunk can be sorted by one work-group. At this point GPUQSort launches lqsort_kernel to complete the sorting job.

The GPU-Quicksort implementation requires barriers and atomic functions that are available in OpenCL 1.2 and are supported by practically all modern hardware capable of running OpenCL 1.2. Specifically, atomic_add, atomic_sub, and atomic_dec are used in implementing gqsort_kernel. Barriers are used extensively in both gqsort_kernel and lqsort_kernel.

The disadvantages of using OpenCL 1.2 are:

1.

lack of work-group scan primitives, which required using the algorithm described in a famous paper by Guy Blelloch to implement prefix sums
2.

frequent round-trips between the CPU and the GPU for launching gqsort kernel and the final launch of the lqsort kernel
3.

necessity to perform housekeeping operations prior to kernel launches on the CPU, while the data for that is generated and readily available on the GPU.

All three of these disadvantages are addressed in our OpenCL 2.0 conversion. You will see that by addressing these shortcomings, the performance of the algorithm improved significantly.

Converting GPU-Quicksort to OpenCL 2.0

In this section we will review the changes made to transform the OpenCL 1.2 implementation to an OpenCL 2.0 implementation that takes advantage of the new device-side enqueue and work-group scan functions.

Work-group functions can improve performance and readability of code in OpenCL C

The first and easiest step of converting GPU-Quicksort to OpenCL 2.0 is to take advantage of the readily

available work-group scan functions: specifically, `usework_group_scan_inclusive_add` and `work_group_scan_exclusive_add` functions in both `gqsort_kernel` and `lqsort_kernel`. We not only get some measurable performance benefits (~8%), but we gain in code conciseness, maintainability, and clarity reducing the code size associated with prefix sum calculations almost 3X.

Blocks allow us to take advantage of nested parallelism via device-side enqueue

Next, we translated the logic that sorts the records after each `gqsort_kernel` run into the records suitable for `lqsort_kernel` processing and the ones that need further subdivision, calculates block and parent records, and launches either `gqsort_kernel` or `lqsort_kernel` from C++ into OpenCL C. The `lqsort_kernel` will be launched only once at the very end of the GPU-Quicksort run. In our first implementation that logic resided at the end of `gqsort_kernel`, which required additional global variables and atomics, and made `gqsort_kernel` relatively slow, even though the whole sample performance improved compared to OpenCL 1.2 version. We moved this logic into a separate `relauncher_kernel`, which is now launched as a very first kernel and then at the end of every `gqsort_kernel` run:

```
01 // now let's recompute and relaunch
02 if (get_global_id(0) == 0) {
03     uint num_workgroups = get_num_groups(0);
04     queue_t q = get_default_queue();
05     enqueue_kernel(q, CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
06                   ndrange_1D(1),
07                   ^{ relauncher_kernel(d, dn,
08                   blocks, parents, result, work, done, done_size,
09                   MAXSEQ, num_workgroups);
10                   });
11 }
```

Note that `relauncher_kernel` is launched on only a single work-item (`ndrange = 1`). Isolating the relauncher logic from the `qsort_kernel` significantly simplified the `qsort_kernel` and had the added benefit of improving the performance of the algorithm.

A couple of points about the way we enqueue `relauncher_kernel`: we use a new feature introduced in OpenCL 2.0—blocks. Since the kernels do not require local memory arguments, we use blocks of type `void (^) (void)`. Blocks are similar to C++ lambdas, but have a different syntax. The block captures the parameters passed into it and therefore avoids the painful job of setting the arguments for the kernel via `clSetKernelArg` equivalent. A second point worth mentioning is the way `num_workgroups` is calculated outside the block and then passed into it. Note that in the code example above the `num_workgroups` will be captured by value, which is exactly what we want. If we used `get_num_groups(0)` directly instead of `num_workgroups`, it would have been called after the block containing `relauncher_kernel` call was enqueued and executed, so the value passed to `relauncher_kernel` would have been 1, instead of the number of work-groups of the `gqsort_kernel`. For more information on the block syntax used by OpenCL C, see the OpenCL C specification [3].

The `relauncher_kernel` launches either `gqsort_kernel` or `lqsort_kernel` depending on whether there is still work available (`work_size != 0`) for the `gqsort_kernel` to perform:

```
01 if (work_size != 0) {
02     // Calculate block size, parents and blocks
03     ...
04
05     enqueue_kernel(q, CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
06                   ndrange_1D(GQSORT_LOCAL_WORKGROUP_SIZE *
07                               blocks_size,
08                               GQSORT_LOCAL_WORKGROUP_SIZE),
09                   ^{ gqsort_kernel(d, dn,
09                   blocks, parents, result, work,
10                   done,
11                   done_size, MAXSEQ, 0); });
12 } else {
13     enqueue_kernel(q, CLK_ENQUEUE_FLAGS_WAIT_KERNEL,
14                   ndrange_1D(LQSORT_LOCAL_WORKGROUP_SIZE * done_size,
15                               LQSORT_LOCAL_WORKGROUP_SIZE),
```



```
15 |         ^{ lqsort_kernel(d, dn, done); });
16 |     }
```

Once we launch the first `relauncher_kernel` from the CPU, subsequent kernel launches are all performed from the GPU. After all the input data is fully sorted, control is returned to the CPU.

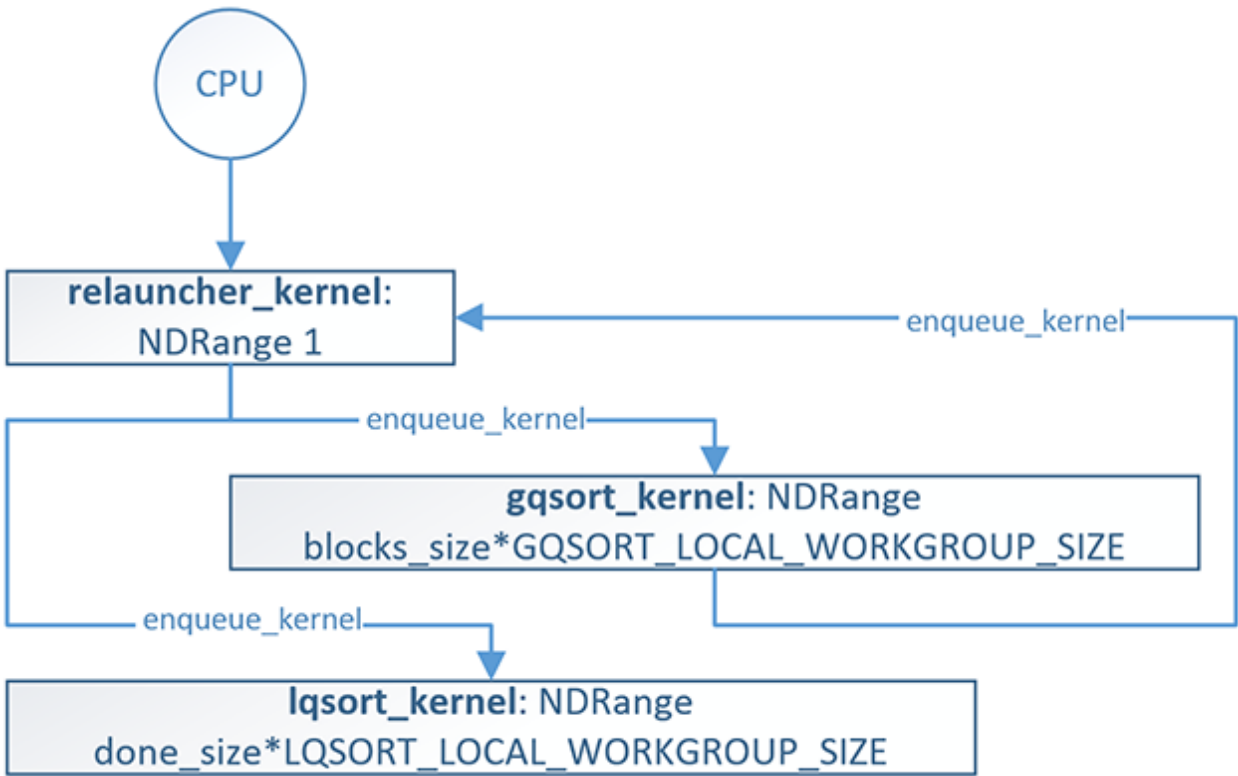


Figure 3. GPU-Quicksort kernel enqueueing sequence in OpenCL 2.0

Tutorial Requirements

Building and running this tutorial requires a PC that meets the following requirements:

- A processor from the Intel® Core™ processors series code-named Broadwell
- Microsoft Windows* 8 or 8.1
- Intel® SDK for OpenCL™ Applications 2014 R2 or above
- Microsoft Visual Studio* 2012 or above

Running the Tutorial

The tutorial is a console application that generates an array of size Width*Height of random unsigned integers. It then proceeds to sort the copies of that array using `std::sort`, regular single-threaded quicksort, followed by a number of iterations of sorting with GPU-Quicksort in OpenCL 2.0. The tutorial supports a few command-line options:

Option	Description
-h, -?	Show help text and exit
[num test iterations]	Number of times to run GPU-Quicksort on the same data
[cpu gpu]	Whether to run the tutorial using CPU or GPU OpenCL
[intel amd nvidia]	Select vendor whose OpenCL device you want to run on
[width]	“width” of the input – makes it easier to enter large numbers
[Height]	“height” of the input – makes it easier to enter large numbers, e.g., instead of 67M elements we could provide 8192 8192

Option	Description
[show_CL no_show_CL]	Whether to show detailed OpenCL device info or not

Try running the tutorial with parameters `5 gpu intel 8192 8192 show_CL`.

Conclusion

An Intel processor with Intel® HD graphics 5300 or above is a complex piece of hardware, but when coupled with the OpenCL 2.0 capable driver, can bring dramatic performance improvements to your OpenCL code. OpenCL 2.0 provides a number of powerful features to the GPGPU programmer. We covered just two of these features: `enqueue_kernel` function and `work_group_scan_exclusive_add` and `work_group_scan_inclusive_add` functions, and showed that they bring dramatic performance improvements at a relatively small cost, while simplifying and increasing readability of your code. Please consider reading other Intel tutorials on other OpenCL features at [Intel® SDK for OpenCL™ Applications Support \(https://software.intel.com/intel-opengl-support\)](https://software.intel.com/intel-opengl-support) website.

References

1. [Intel SDK for OpenCL Applications – Optimization Guide \(/sites/products/documentation/ioclSDK/2013XE/OG/index.htm\)](#)
2. [Khronos OpenCL 2.0 API Specification \(https://www.khronos.org/registry/cl/specs/opengl-2.0.pdf\)](https://www.khronos.org/registry/cl/specs/opengl-2.0.pdf)
3. [Khronos OpenCL 2.0 C Language Specification \(https://www.khronos.org/registry/cl/specs/opengl-2.0-openccl.pdf\)](https://www.khronos.org/registry/cl/specs/opengl-2.0-openccl.pdf)
4. Wikipedia article on [Quicksort \(http://en.wikipedia.org/wiki/Quicksort\)](http://en.wikipedia.org/wiki/Quicksort)
5. “My Early Days at Elliots” (<http://www.cs.man.ac.uk/CCS/res/res48.htm>) by Tony Hoare
6. [Scan Primitives for GPU Computing \(http://www.idav.ucdavis.edu/publications/print_pub?pub_id=915\)](http://www.idav.ucdavis.edu/publications/print_pub?pub_id=915) by Shubhabrata Sengupta, Mark Harris, Yao Zhang, and John D. Owens, Graphics Hardware 2007, pages 97--106, August 2007.
7. [GPU-Quicksort: A practical Quicksort algorithm for graphics processors \(http://dl.acm.org/citation.cfm?id=1564500\)](http://dl.acm.org/citation.cfm?id=1564500) by Daniel Cederman and Philippas Tsigas, Journal of Experimental Algorithmics, Volume 14, 2009, Article No. 4
8. [Prefix Sums and Their Applications \(http://www.cs.cmu.edu/~blelloch/papers/Ble93.pdf\)](http://www.cs.cmu.edu/~blelloch/papers/Ble93.pdf) by Guy Blelloch
9. [enqueue_kernel \(http://www.khronos.org/registry/cl/sdk/2.0/docs/man/xhtml/enqueue_kernel.html\)](http://www.khronos.org/registry/cl/sdk/2.0/docs/man/xhtml/enqueue_kernel.html) functions online documentation.
10. [work_group_scan_inclusive \(http://www.khronos.org/registry/cl/sdk/2.0/docs/man/xhtml/work_group_scan_inclusive.html\)](http://www.khronos.org/registry/cl/sdk/2.0/docs/man/xhtml/work_group_scan_inclusive.html) functions online documentation
11. [work_group_scan_exclusive \(https://www.khronos.org/registry/cl/sdk/2.0/docs/man/xhtml/work_group_scan_exclusive.html\)](https://www.khronos.org/registry/cl/sdk/2.0/docs/man/xhtml/work_group_scan_exclusive.html) functions online documentation
12. [Intel® SDK for OpenCL™ Applications \(/intel-opengl\)](#)
13. [Intel® SDK for OpenCL Applications 2013 R2 Optimization Guide \(http://software.intel.com/sites/products/documentation/ioclSDK/2013/OG/index.htm\)](http://software.intel.com/sites/products/documentation/ioclSDK/2013/OG/index.htm)
14. [The Full Checklist for Optimized OpenCL Application \(http://software.intel.com/en-us/articles/tips-and-tricks-for-kernel-development/\)](http://software.intel.com/en-us/articles/tips-and-tricks-for-kernel-development/)
15. [Writing Optimal OpenCL Code with Intel® OpenCL SDK \(http://software.intel.com/file/37171\)](http://software.intel.com/file/37171) - in pdf format

About the Author



Robert Ioffe is a Technical Consulting Engineer at Intel’s Software and Solutions Group. He is an expert in OpenCL programming and OpenCL workload optimization on Intel Iris and Intel Iris Pro Graphics with deep knowledge of Intel Graphics Hardware. He was heavily involved in Khronos standards work, focusing on prototyping the latest features and making sure they can run well on Intel architecture. Most recently he has been working on prototyping Nested Parallelism (`enqueue_kernel` functions) feature of OpenCL 2.0 and wrote a number of samples that demonstrate Nested Parallelism functionality, including GPU-Quicksort for OpenCL 2.0. He also recorded and released two Optimizing Simple OpenCL Kernels videos and is in the process of recording a third video on Nested Parallelism.

You might also be interested in the following:

- [Sierpiński Carpet in OpenCL 2.0 \(https://software.intel.com/en-us/articles/sierpinski-carpet-in-opengl-20\)](https://software.intel.com/en-us/articles/sierpinski-carpet-in-opengl-20)
- [Optimizing Simple OpenCL Kernels: Modulate Kernel Optimization \(https://software.intel.com/en-us/videos/optimizing-simple-opengl-kernels-modulate-kernel-optimization\)](https://software.intel.com/en-us/videos/optimizing-simple-opengl-kernels-modulate-kernel-optimization)
- [Optimizing Simple OpenCL Kernels: Sobel Kernel Optimization \(https://software.intel.com/en-us/videos/optimizing-simple-opengl-kernels-sobel-kernel-optimization\)](https://software.intel.com/en-us/videos/optimizing-simple-opengl-kernels-sobel-kernel-optimization)

Download the Code

For more complete information about compiler optimizations, see our [Optimization Notice \(/en-us/articles/optimization-notice#opt-en\)](#).

Attachment	Size
 GPU-Quicksort_OpenCL_2.0.zip (https://software.intel.com/sites/default/files/managed/e2/bd/GPU-Quicksort_OpenCL_2.0.zip)	19.76 KB
 GPU-Quicksort_OpenCL_1.2.zip (https://software.intel.com/sites/default/files/managed/84/c4/GPU-Quicksort_OpenCL_1.2.zip)	20.13 KB

- Hardware Developers

▪ [Resource and Design Center](#)
- Open Source

▪ [01.org](#)
- Manage Your Tools

▪ [Download Center](#)
- Stay Up-to-Date

▪ [Forums](#)

- [Shop Intel](#)
 - [Firmware](#)
- [Clear Linux* Project](#)
 - [Zephyr Project](#)
- [Online Service Center](#)
 - [Registration Center](#)
- [Recent Updates](#)
 - [Subscribe to our YouTube Channel](#)
 - [Newsletter Archives](#)

Rate Us





[Get the Newsletter](#)

[Terms of Use](#)

[*Trademarks](#)

[Privacy](#)

[Cookies](#)

Follow us:









