KALRAY

MPPA® OpenCL User Guide

Kalray S.A.

21 pages

MPPA® OpenCL User Guide

Kalray S.A.[1]

[1] info@kalray.eu, Kalray S.A.

Abstract: This document details the Kalray OpenCL support.

The support is based on the OpenCL 1.2 specification [3] which allows to execute OpenCL kernels on MPPA® devices. This preliminary support is a subset of the OpenCL 1.2 specification [3] which allows to execute OpenCL kernels on MPPA® devices. The targeted audience should have a basic knowledge of the OpenCL 1.2 specification and of the MPPA® architecture.

Keywords: MPPA® Processor, OpenCL, Task, NDRange, Kernel, DSM.

Contents

1 Overview

This documentation presents the OpenCL support available on MPPA® devices. This support is based on the OpenCL 1.2 specifications ([3]) and targets both host runtime and device support.

On host side, buffers, events, NDRange and task kernels are amongst the supported features. In OpenCL jargon the MPPA® device is referred to as an accelerator device.

We are using our own version of LLVM to compile the kernels for MPPA®. Thus, kernels has to be written in OpenCL-C.

Regarding the execution platform, Kalray OpenCL programs can be executed on both simulator and hardware without recompiling them.

In a first part, the mapping of the OpenCL platform and memory model will be explained. A second section will introduce host API support and the different types of kernel supported. Finally, we will explain how to compile, run and optimize an OpenCL application on MPPA®.

2 OpenCL Model

2.1 Platform Model

The OpenCL platform model can be mapped to the MPPA® one as seen on figure 1. For more informations about OpenCL platform model, please refer to the OpenCL 1.2 specification ([3]).

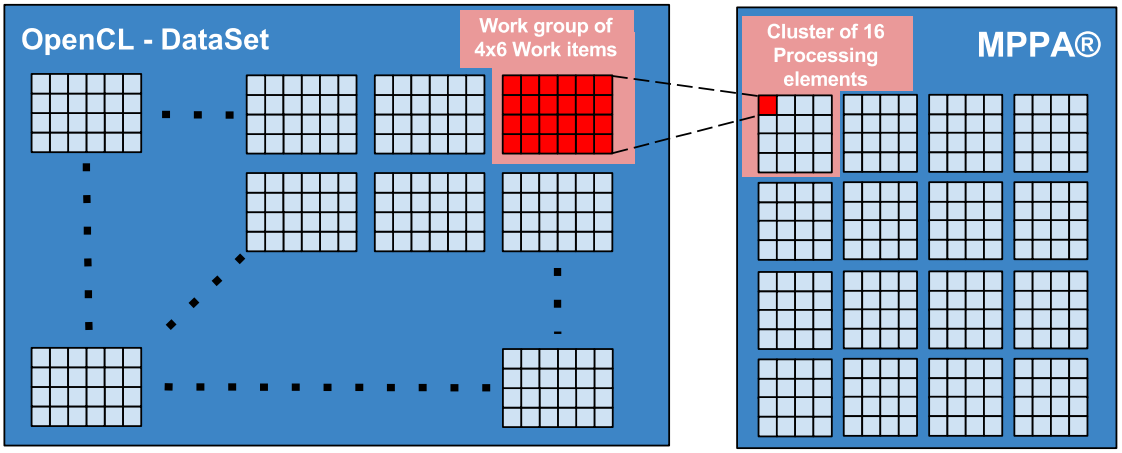


Figure 1: OpenCL platform model with MPPA® mapping

Kalray OpenCL implementation will offer 256 compute units, divided into 16 compute clusters. According to the OpenCL specification, work-groups will be executed on compute units. Work-items will be linearized onto one processing elements (PE) of one compute cluster of the MPPA®. Every PEs of one compute cluster will execute work-groups of the same kernel, as so we can't have more than 16different kernel running at the same time on the MPPA®. The number of compute devices available will be equal to the number of boards present.

2.2 Memory Model

In order to understand how the OpenCL memory model is mapped on the MPPA® memory architecture, the figure 2 represents the actual mapping. As seen on this figure, the shared memory (SMEM) is used to store both local and private data. Private data (`__private`) are located on the processing elements stacks (ie, like a thread). Local data (`__local`) are located in the SMEM and are limited to `8KB` per processing element at the moment. Finally, global buffers (`__global`) will access the DDR memory of the board using a Distributed Shared Memory (DSM) system (See section 3.4).

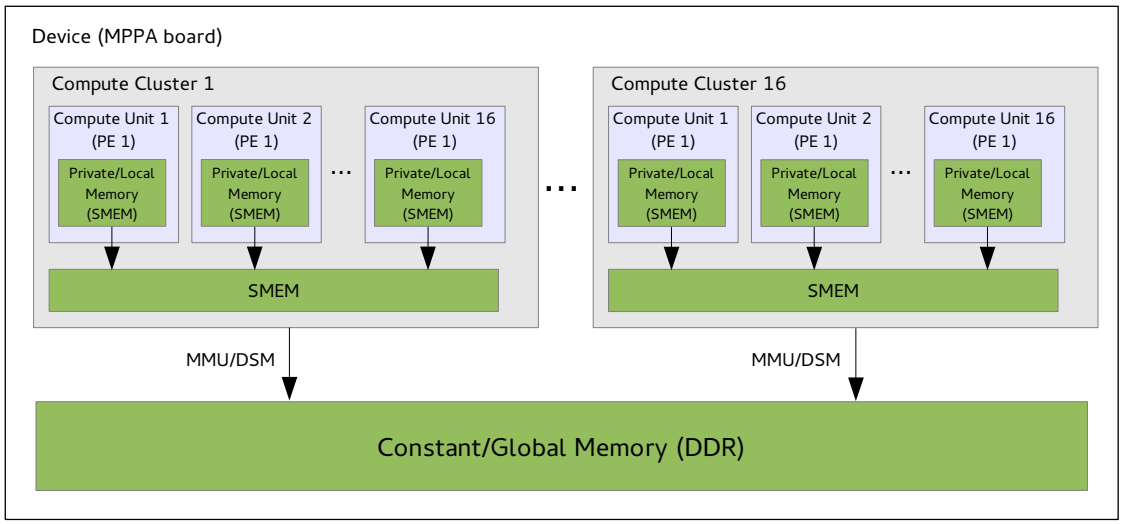


Figure 2: OpenCL memory model to MPPA® architecture

3 Kalray Implementation

3.1 OpenCL Host Support

The OpenCL host runtime support is implemented on top of the POCL Open Source project, hence the supported OpenCL API subset is mainly the one supported by POCL except for some Kalray unsupported features (section B). The following functionnalities are supported by the MPPA® OpenCL API:

- Buffers and Sub-buffers
- Out-Of-Order execution (OoO) for command queues
- Events, Wait lists
- NDRange and Task kernels
- C++ OpenCL API Wrapper

NDRange kernels use the data-parallel model and task kernel use the task-parallel model. As we are using our own version of LLVM, kernels have to be written in standard OpenCL-C 1.2.

NDRange and task kernels use the standard OpenCL kernels and programs objects (`cl_program`, `cl_kernel`) and are built using `clCreateKernel` and `clBuildProgram`. Kernels themselves are then enqueued using `clEnqueueNDRangeKernel` and `clEnqueueTask`.

3.2 Memory Spaces Keywords

`__local` buffers will be allocated into shared memory (SMEM) with a maximum of `8KB` of local data (See section A). As specified by the OpenCL specification, the local memory is shared amongst the work-items. According to the OpenCL specification, to allocate a local buffer with `clSetKernelArg`, the `arg_value` argument should be set to `NULL` and `arg_size` should reflect the amount of local memory that will be allocated.

`__global` buffers will be allocated in DDR. At the moment this memory is limited to `1GB`. Like other types of OpenCL devices, accessing the MPPA® global memory is much slower than accessing local memory. On the MPPA®, the global memory is accessed through a system called DSM for Distributed Shared Memory (See section 3.4).

`__private` buffers will be allocated into the PE stack (which is store in the SMEM). The maximum amount of private memory (ie size of stacks) is `4KB` hence declaring large array on the stack should be avoided. When doing so, the program behaviour will be unknown and will crash. To avoid such problem, when large arrays are needed, use local memory.

3.3 Kalray OpenCL Extension

Due to the specificities of the device and the runtime, an OpenCL extension is available to query or set MPPA specific informations. This extension follows the OpenCL specification for extensions and the provided library allows direct linking with the extension functions. To use it, `<CL/cl_kalray.h>` should be included in source code.

The first function provided by this extension is `clGetMPPAInfo` is provided by this extension.. This functions allows to get informations which do not fit the OpenCL abstraction but are relevant for a MPPA device. The prototype of this function is the same as `clGetDeviceInfo` (listing 3.3).

```
cl_int clGetMPPAInfo(cl_device_id  device,
                     cl_mppa_info_kalray param_name,
                     size_t       param_value_size,
                     void *       param_value,
                     size_t *     param_value_size_ret)
```

At the moment, `param_name` argument can take the value specified in table 1.

If `param_name` is not a supported value, `CL_INVALID_ARG_VALUE` will be returned. If device is not a MPPA, `CL_INVALID_DEVICE_TYPE` will be returned.

Another function provided by Kalray extension is `clSetMPPAKernelExecInfo`. This functions allows the user to set various parameters for a specific kernel execution on the MPPA® device. The prototype of the function (listing 3.3) is the following:

```
cl_int clSetMPPAKernelExecInfo(cl_kernel kernel,
```

Table 1: clGetMPPAInfo valid parameters

cl_mppa_info_kalray	Description	Return Type
CL_MPPA_PAGE_SIZE_KALRAY	MPPA global memory page size	cl_uint
CL_MPPA_BOARD_TYPE_KALRAY	Board type for the specified device	char *
CL_MPPA_PAGE_PER_CU_KALRAY	Page count per compute unit	cl_uint
CL_MPPA_PAGE_PER_PE_KALRAY	Page count per processing element	cl_uint
CL_MPPA_ARCH_TYPE_KALRAY	Integer that identify the board	cl_uint
CL_MPPA_ARCH_TYPE_STR_KALRAY	Board name	char *

```
                        cl_mppa_kernel_param_kalray param_name,
                        size_t param_value_size,
                        const void *param_value)
```

This prototype is the same as for the OpenCL 2.0 `clSetKernelExecInfo` function. The `kernel` argument is the kernel on which the execution informations will be set. The `param_name` argument cant take the value specified in table 2 and represent the property to set. `param_value_size` is the size of the value pointed by `param_value` which is the value to set for the kernel.

Table 2: clSetMPPAKernelExecInfo valid parameters

cl_mppa_kernel_param_kalray	Description
CL_MPPA_DEV_SCHED_MODE	Set the device scheduling mode
CL_MPPA_CU_SCHED_MODE	Set the compute unit workgroup scheduling mode

When setting `CL_MPPA_DEV_SCHED_MODE`, the possible `param_value` are the one visible in table 3. The default device scheduling mode can be overriden by setting the `MPPACL_DEV_SCHED_MODE` environment variable to value `cluster` or `pe`.

Table 3: `CL_MPPA_DEV_SCHED_MODE` valid values

Value	Description
CL_MPPA_DEV_SCHED_CLUSTER_KALRAY	Try to use as many cluster as possible, even with only one work-group per cluster (default)
CL_MPPA_DEV_SCHED_PE_KALRAY	Use every core of a cluster before starting of using the next

Regarding `CL_MPPA_CU_SCHED_MODE`, the possible `param_value` are the one visible in table 4 and are explained on figure 3. The default compute unit scheduling mode can be overriden by setting the `MPPACL_CU_SCHED_MODE` environment variable to value `interleave` or `sequential`.

Table 4: `CL_MPPA_CU_SCHED_MODE` valid values

Value	Description
`CL_MPPA_CU_SCHED_INTERLEAVE_KALRAY`	Execute workgroups on compute unit by interleaving (default)
`CL_MPPA_CU_SCHED_SEQUENTIAL_KALRAY`	Execute workgroups sequentially on each compute unit

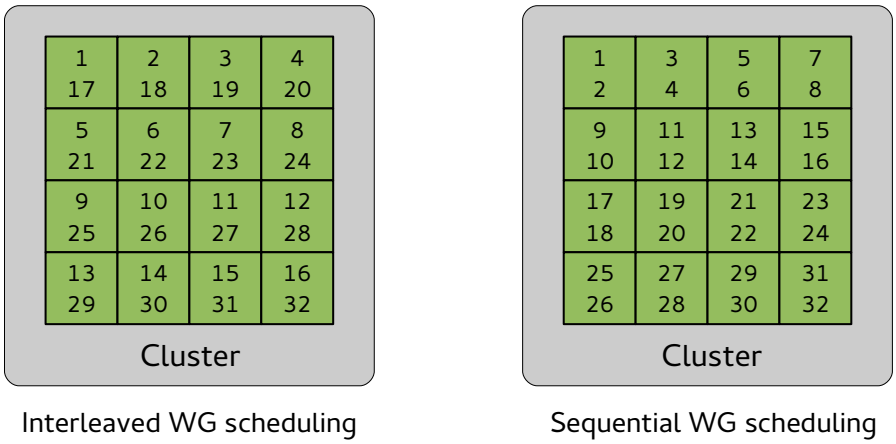


Figure 3: Available workgroups scheduling for compute units

3.4 Memory Accesses

In order to access the global memory, a Distributed Shared Memory (DSM) mechanism is used. This mechanism allows a transparent access to the DDR memory of the MPPA® IO subsystems. When accessing a buffer in global memory, the system will fetch a page (a fixed amount of memory) of data from the DDR. This page will be stored in a special global memory cache (located in shared memory) allowing for faster access when used. Each processing element can store roughly 10 pages in the global memory cache. This means that when the cache is full, a page will be evicted in order to fetch a new one.

Currently the page size is `8KB`, because it represents a good compromise between speed & flexibility. In order to access this value inside applications, host application should use the Kalray OpenCL extension (section 3.3). On device side, the `CL_MPPA_PAGE_SIZE_KALRAY` define on device side will reflect the same value.

This page size can be overriden using the environnement variable `MPPACL_PAGE_SIZE`. Authorized values for this environnement variable are `4K`, `8K` and `16K`. This value can have a significant impact on performances depending on how the kernels accesses the global buffers. In order to keep the possibility of testing different page sizes, using the definitions provided by `<CL/cl_kalray.h>` is strongly recommended. As explained in section 3.3 `clGetMPPAInfo` can be used to query the page size (with flag `CL_MPPA_PAGE_SIZE_KALRAY`). This page size will be returned accordingly to the environnement variable `MPPACL_PAGE_SIZE` Regarding

the device define `CL_MPPA_PAGE_SIZE_KALRAY`, it will also reflects the current page size and it allows compile-time optimizations.

In order to optimize the memory accesses, compute units should try work on exclusive pages (figure 4). This also allows higher speed when accessing memory.

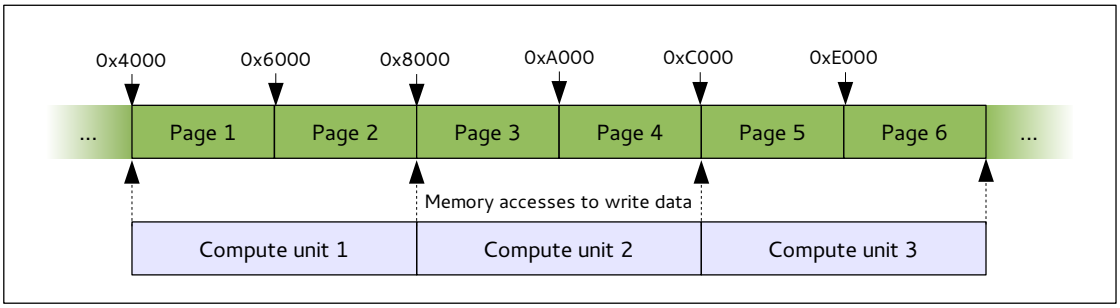


Figure 4: Optimized memory write with multiple work-groups

3.5 Out-Of-Order Execution

The current implementation allows the use of `CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE` flag for command queues. This flag enables out-of-order execution, which means that multiple command can be executed simulteanously in order to maximize the utilization of MPPA® and host. When creating a command queue with this flag, the user will have to explicit the dependencies between commands using OpenCL events (`cl_event`). This allows to pipeline commands efficiently such overlapping memory transfer with kernel execution.

For instance, suppose that a simple graphical application enqueues a write buffer (`clEnqueueWriteBuffer`) for an image processing kernel. The kernel is then enqueued (`clEnqueueNDRangeKernel`) to work on the data transfered previously. Once this kernel computation is done, the application reads the result (`clEnqueueReadBuffer`) and display the data. Then, the applications start again this cycle in order to process a video stream. This application command graph can be simplified as seen on figure 5.

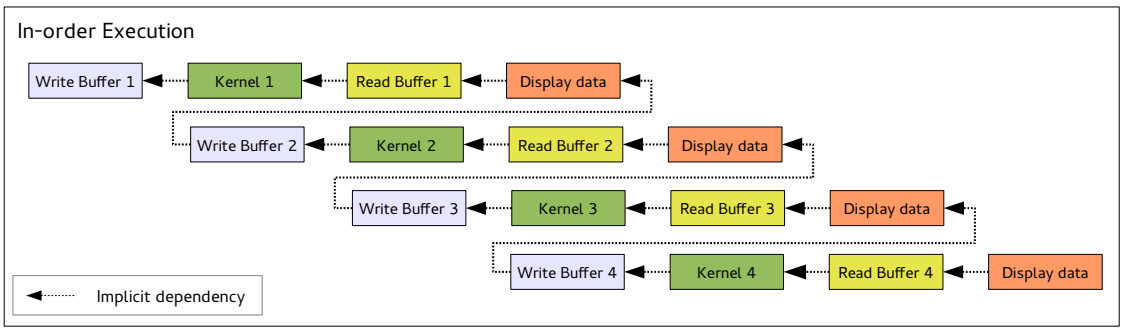


Figure 5: In-order execution

When using default execution mode for command queues, all the commands will be executed sequentially. This is clearly not optimal since we could use double buffering in order to pipeline kernel execution and write buffer. Moreover, once the first kernel is executed, it is possible to read the data while the second kernel is executing. Finally, the data display can also be done in parallel. The final command dependency graph can be represented by figure 6.

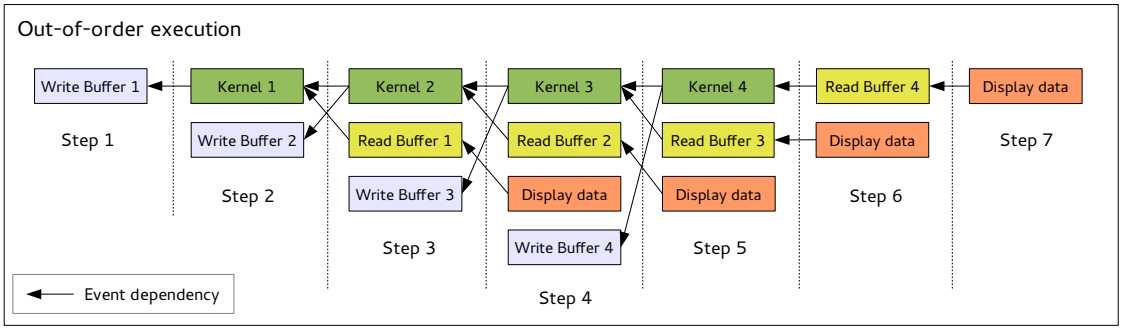


Figure 6: Out-of-order execution with command pipelining

In order to achieve pipelining, OpenCL events (`cl_event`) can be used when the command queue mode is set to `CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE`. By describing proper dependencies between enqueued command (functions beginning with `clEnqueue`), pipelining can allow noticeable gain in execution time.

The MPPA® can also execute multiple kernel at the same time as long as there is enough compute units available. This allows to execute independent kernel simulteanously (See section 3.6).

3.6 NDRange Kernel Support

Kalray OpenCL implementation supports up to three dimensions for NDRange kernels. The maximum work-group size is 256. Number of work-group size is however practically unlimited (Maximum values for dimensions are $((2^{32}-1),(2^{32}-1),(2^{32}-1))$, For more informations about OpenCL device specific values, see section A.

When using less than 256 work-groups, there is 2 differents scheduling mode available.

Either you want to schedule one work-group per compute cluster before scheduling severals work-group on the same cluster. Or you want the 16 processing elements of one compute cluster to be scheduling before scheduling other work-group on another cluster (see figure 7 and figure 8).

When using more than 256 work-groups, all processing elements of each compute cluster are schedule. Thus, there isn't different mode of scheduling (See figure 9).

Regarding the work-items, they are yet limited to 2048 on each dimensions. Work-items are linearized, so that one work-group can be executed onto one processing elements of a compute cluster. To do that, there is some compilation pass that are runned by POCL to either replicate the kernel as many time as necessary or to loop on the kernel (those pass knows how to deal with barriers amongs work-item). You can force POCL to linearized it by replication or by loop by

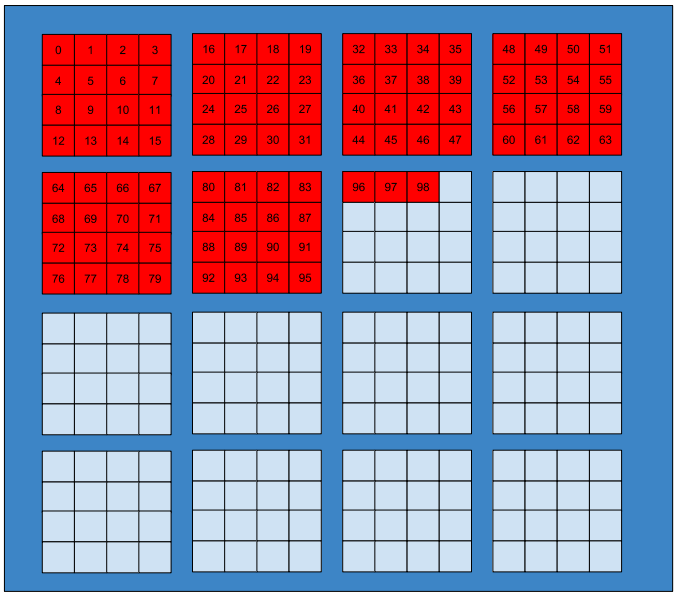


Figure 7: Scheduling for an example on 99 work-groups (numbers are the global ids of the work-group that will run on the PE)

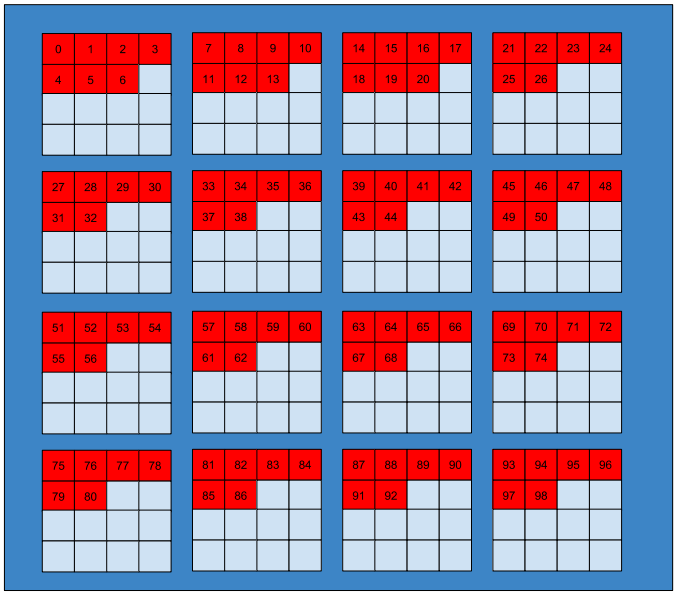


Figure 8: Scheduling for an example on 99 work-groups (numbers are the global ids of the work-group that will run on the PE)

setting this environment variable to either one or another value :

```
POCL_WORK_GROUP_METHOD = {workitemrepl, workitemloops}
```

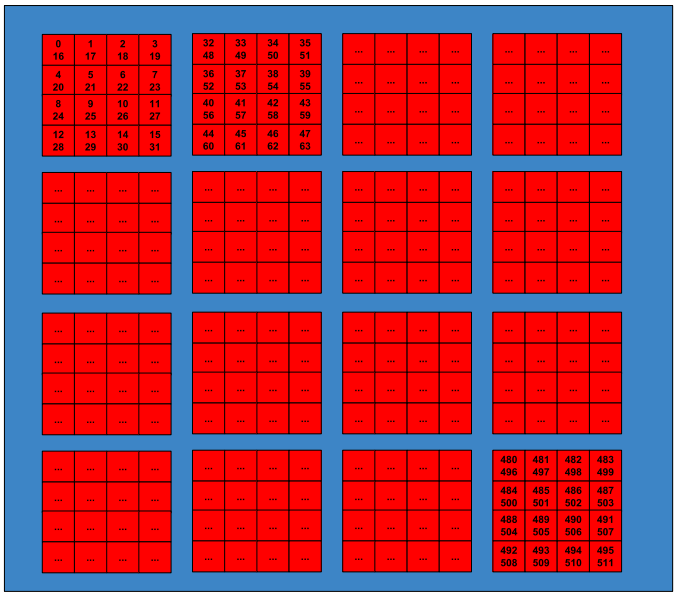


Figure 9: Scheduling for an example on 512 work-groups (numbers are the global ids of the work-groups that will run on the PE)

3.7 Tracing & Debug

Although OpenCL API offers event profiling, It might be necessary to have more precise numbers. Event profiling provides global measurements which includes runtime overhead.

To have a precise measurement of kernel execution, the MPPA® OpenCL runtime includes tracepoints activable on demand. In order to activate traces, environnement variable `MPPACL_TRACE` must be set to 1 when starting the application and traces can then be recorded using `k1-trace-util`.

In order to ease this process, `k1-opencl-trace` is available and allows to record traces (See listing 1). This in/out tracepoint will provide accurate execution time of kernels on clusters, and also the execution time of many OpenCL functions on the host. This allows the programmer to optimize his code by reducing latency and improving his scheduling.

Please refer to [1] for more information about MPPA® tracing tools.

Listing 1: k1-opencl-trace help

```
Helper script to acquire trace with OpenCL applications.

k1-opencl-trace [option] -- opencl_application

Options:
  -h: Display help and exit
  -v: Display version and exit
  -c: Nodes to trace (ie node0 for cluster 0), default: node0 (this option
      can be passed multiple times)
  -e: Trace the host
  -b: Board number to trace
```

```
  -m: MPPA number to trace (on TC3 only, starts from 0)
  -a: Do not run any post-tracing analysis
```

When the application will be over, the command line to visualize the tracepoints and display memory access will be printed.

In order to dump kernels and examine generated code, the env var `MPPACL_KERNEL_DUMP_DIR` can be set with a directory name. When set, the kernels will be dumped into the specified directory during execution.

3.8 Boards

Kalray OpenCL implementation will detect the board automatically. However, due to the differences between boards, performances can vary between two different boards.

4 Kernel Support

4.1 Overview

On MPPA® devices, kernels (NDRange and tasks) are executed on compute units. Task kernels are run on only one processing element of the MPPA whereas NDRange kernels can use all the processing elements. Consequently, when using task kernels, out-of-order execution for command queue can be used to use the 16 clusters as compute units.

The following section will present how to size the data compute per work-group and how to compile and run an OpenCL program with one example. For the sake of simplicity, this example use assertions to validate the data returned by the OpenCL API. It is recommended to check the error codes returned by those functions and compare them to the expected errors. This example is available with a Makefile in the directory `$K1_TOOLCHAIN_DIR/share/opencl/`.

4.2 NDRange Kernel Support

This example perform a vector addition on double using the data parallel paradigm. It will demonstrate the explaination of section 3.4.

4.2.1 How to size the data compute per work-group

Here is the code of the kernel of our OpenCL program. We will run this kernel with several sizes of data to be compute per work-item and several numbers of work-item per work-group. In each case the total size of data compute by the program will be the same (the number of work-group to be compute is adjust for every run).

```
const char *kernel_source=
        "__kernel void test( __constant double *a,        \n"
        "                    __constant double *b,        \n"
        "                    __global double *c,          \n"
        "                    const unsigned int n)        \n"
        "{                                                \n"
        "       unsigned int id = get_global_id(0) * n, i;\n"
```

```
        "                                                       \n"
        "        for (i = id; i < id + n; i++)                  \n"
        "                c[i] = a[i] + b[i];                    \n"
        "}                                                      \n";
```

You can see on figure 10 and figure 11 the results of this execution. The x-axe correspond to the number of data compute per work-item and the y-axe correpond to the number of work-item per work-group.

The better speed-up is obtained when the size of data compute by work-group is equal or a multiples or a divider of the page's size (`8K` in our case). You can also notice than even if when the size of data compute by work-group is a multiples of the page's size the speed-up is good, it is not better than the one for a size of data equal to the page's size. So, we recommand to have each work-group working on exactly the page's size.

We can also notice that the speed-up for 64 work-items per work-group working on 64 double is bad. It is because in this configuration each work-group will need 4 pages per gobal table (2 inputs, 1 ouput). So each work-group uses 12 pages, and we know that the distributed shared memory can store roughly 10 pages per work-group (processing element in fact), so it has to evict page from time to time which explain the bad speed-up.

4.2.2 Compiling

Compiling application can be done using the provided Makefile with `make`.

```
host_ndrange-srcs := host.c

host-bin := host_ndrange

include $(K1_TOOLCHAIN_DIR)/share/make/Makefile.opencl

run_sim: all
        $(K1_TOOLCHAIN_DIR)/bin/k1-pciesim-runner $(BIN_DIR)/host_ndrange

run_hw: all
        $(BIN_DIR)/host_ndrange
```

The Makefile is based on the Kalray Makefile framework (See Kalray Makefile documentation [2]).

However, if you want to compile the application by hand, this can be done using the following command line:

```
gcc $(k1-pkg-config --libs --cflags opencl) host.c -o host
```

Note that using the Makefile is generally easier for more complex applications.

4.2.3 Running

The OpenCL framework uses a versatile PCIe library which allows to use both simulator and device without any recompilation. Hence, to run the application on hardware (ie, a MPPA® board), simply run the following command (when compiled with the provided Makefile) :

```
./output/bin/host_ndrange
```

And to run the program using the simulator as a MPPA® board:

```
k1-pciesim-runner ./output/bin/host_ndrange
```

4.2.4 Examples

You can find some OpenCL examples to start your application here:

```
/usr/local/k1tools/share/opencl/examples
```

A OpenCL Device Informations

Table 5: Kalray OpenCL device informations

Device information	Value
CL_DEVICE_ADDRESS_BITS	32
CL_DEVICE_ENDIAN_LITTLE	CL_TRUE
CL_DEVICE_EXECUTION_CAPABILITIES	CL_EXEC_KERNEL
CL_DEVICE_GLOBAL_MEM_CACHE_SIZE	1,5MB
CL_DEVICE_GLOBAL_MEM_CACHE_TYPE	CL_READ_WRITE_CACHE
CL_DEVICE_GLOBAL_MEM_SIZE	1GB
CL_DEVICE_LOCAL_MEM_SIZE	8KB
CL_DEVICE_IMAGE_SUPPORT	CL_FALSE
CL_DEVICE_LOCAL_MEM_TYPE	CL_LOCAL
CL_DEVICE_MAX_PARAMETER_SIZE	256
CL_DEVICE_MAX_WORK_GROUP_SIZE	1024
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS	3
CL_DEVICE_MAX_WORK_ITEM_SIZES	{2048, 2048, 2048}
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS	3
CL_DEVICE_MEM_BASE_ADDR_ALIGN	8KB
CL_DEVICE_NAME	mppa
CL_DEVICE_TYPE	CL_DEVICE_TYPE_ACCELERATOR
CL_DEVICE_VENDOR	Kalray

B OpenCL Runtime Level of Support

The following section exposes the OpenCL 1.2 Kalray supported API. This list is based on the OpenCL 1.2 specification [3] and the numbers between brackets match the specification sections.
A reference card with those section is available at http://www.khronos.org/files/opencl-1-2-quick-reference-card.pdf.

- The OpenCL platform layer:
  - [4.1, 4.2]: Completely supported
  - [4.3]: clCreateSubDevices is not supported yet
- The OpenCL runtime:
  - [5.1]: Completely supported
- Buffer Objects:
  - [5.2.1, 5.2.2]: clCreateBuffer not supported with flag `CL_MEM_USE_HOST_PTR`.
  - [5.2.3]: Buffer mapping not supported
  - [5.4.1, 5.4.2]: Callbacks and buffer unmapping not supported
  - [5.4.4]: clEnqueueMigrateMemObjects not supported
  - [5.4.5]: Completely supported
- Program Objects:
  - [5.6.1]: clCreateProgramWithBuiltInKernels not supported. clCreateProgramWithBinary and clCreateProgramWithSource supported for OpenCL-C, SPIR, Kalray's machine code kernel
  - [5.6.2]: clBuildProgram supported
  - [5.6.3, 5.6.6, 5.6.7]: clGetProgramBuildInfo not supported
  - [5.6.4, 5.6.4]: All gcc options are supported except -g (Note that -cl options are not gcc options and hence are unsupported)
- Kernel and Event Objects
  - [5.7.1]: Supported
  - [5.7.2, 5.7.3]: Completely supported
  - [5.8]: partially supported. clEnqueueNativeKernel not supported
  - [5.9]: Partially supported. ClSetEventCallback not supported
  - [5.10]: Not supported
  - [5.12, 5.13]: Completely supported

- Supported datatypes
  - [6.1.1, 6.1.2]: Scalar types and vectors only
  - [6.1.3, 6.1.4]: Images and Reserved data types not supported
- Vector Component Addressing
  - [6.1.7]: Supported
- Operators and Qualifiers
  - [6.3]: Completely supported
  - [6.5]: Completely supported
  - [6.7]: Completely supported
- Specify Type Attributes
  - [6.11.1]: Endianness attribute not supported
- Integer Built-in functions
  - [6.12.3]: Completely supported
  - [6.10]: Completely supported
- Math Library:
  - [6.12.2]: Completely supported
- Geometric Built-in Function
  - [6.12.5]: Not supported
- Vector Data Load/Store
  - [6.12.7]: Completely supported
- Async Copies and Prefetch Functions
  - [6.12.10]: Not supported
- Work-Item Built-in Functions
  - [6.12.1]: Completely supported
- Common Built-in Functions
  - [6.12.4]: Not supported
- Relational Built-in Functions

  - [6.12.6]: Not supported

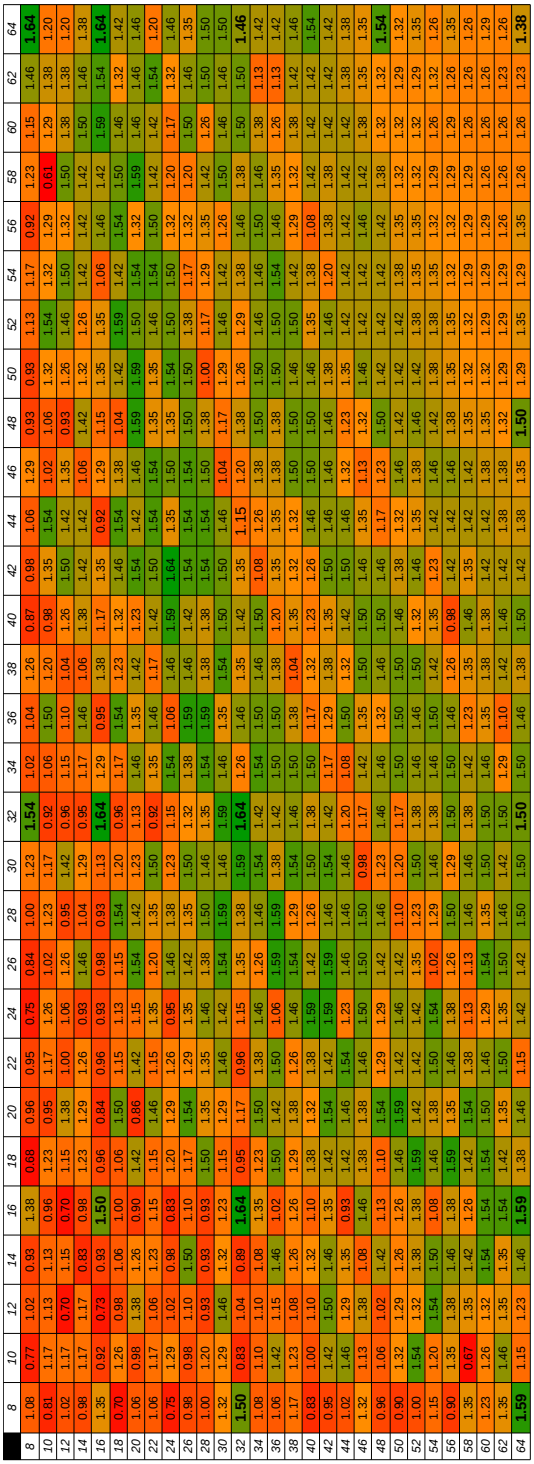
- Atomic Functions:
  - [6.12.11]: Not supported
- Conversions and Type Casting
  - [6.2]: Completely supported
- Synchronization and Explicit Memory Fence Functions
  - [6.12.8, 6.12.9]: barrier supported
- Miscellaneous Vector Functions
  - [6.12.12]: Not supported
- printf function
  - [6.12.13]: Completely supported
- OpenCL Image Processing:
  - [5.3.*] Not supported (Image, OpenGL, DX9/10)

Bibliography

[1] Kalray. MPPA™ Development using Runtime Traces.

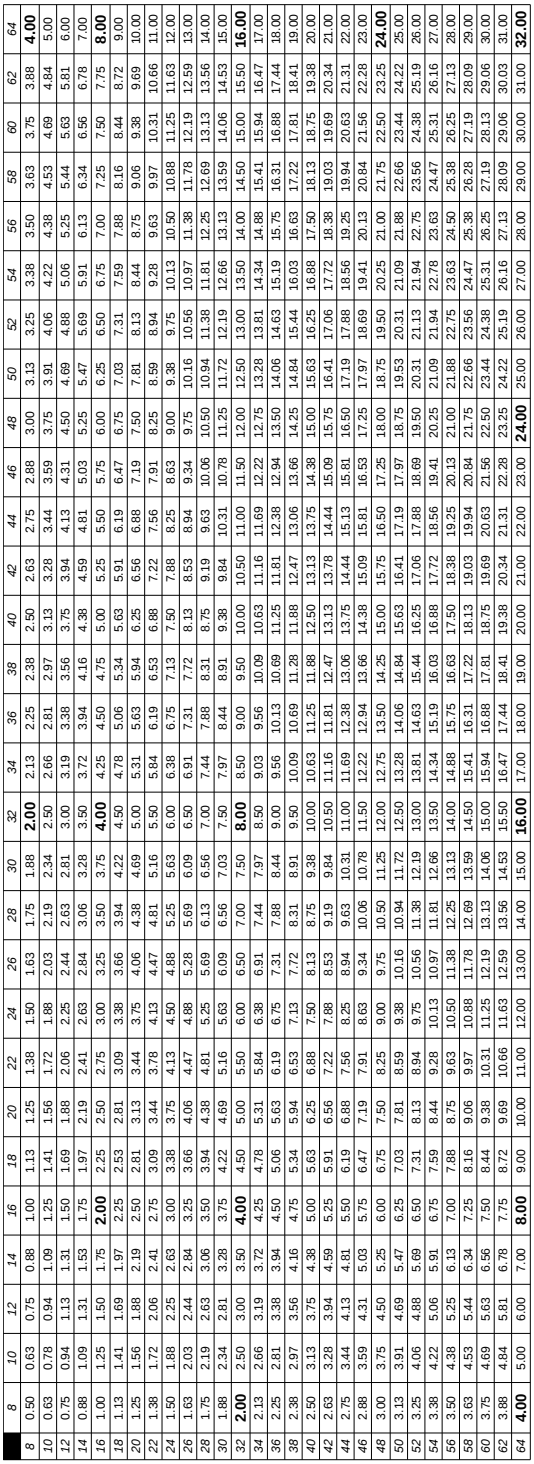
[2] Kalray. MPPA™ Development with Kalray Makefiles.

[3] Khronos OpenCL Working Group. *The OpenCL Specification, version 1.2*, 14 November 2012.



	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48	50	52	54	56	58	60	62	64
8	1.08	0.77	1.02	0.93	1.38	0.68	0.96	0.95	0.75	0.84	1.00	1.23	**1.54**	1.02	1.04	1.26	0.87	0.98	1.06	1.29	0.93	0.93	1.13	1.17	0.92	1.23	1.15	1.46	**1.64**
10	0.81	1.17	1.13	1.13	0.96	1.23	0.95	1.17	1.26	1.02	1.23	1.17	0.92	1.06	1.50	1.20	0.98	1.35	1.54	1.02	1.06	1.32	1.54	1.32	1.29	0.61	1.29	1.38	1.20
12	1.02	1.17	1.17	1.15	0.70	1.15	1.38	1.00	1.06	1.26	0.95	1.42	0.96	1.15	1.10	1.04	1.26	1.50	1.42	1.35	0.93	1.26	1.46	1.50	1.32	1.50	1.38	1.38	1.20
14	0.98	1.17	1.17	0.83	0.98	1.23	1.29	1.26	0.93	1.46	1.04	1.29	0.95	1.17	1.46	1.06	1.38	1.42	1.42	1.06	1.42	1.32	1.26	1.42	1.42	1.42	1.50	1.46	1.38
16	1.35	0.92	0.73	0.93	**1.50**	0.96	0.84	0.96	0.93	0.98	0.93	1.13	**1.64**	1.29	0.95	1.38	1.17	1.35	0.92	1.29	1.15	1.35	1.35	1.06	1.46	1.42	1.59	1.54	**1.64**
18	0.70	1.26	0.98	1.06	1.00	1.06	1.50	1.15	1.13	1.15	1.54	1.20	0.96	1.17	1.54	1.23	1.32	1.46	1.54	1.38	1.04	1.42	1.59	1.42	1.54	1.50	1.46	1.32	1.42
20	1.06	0.98	1.38	1.26	0.90	1.42	0.86	1.42	1.15	1.54	1.42	1.23	1.13	1.46	1.35	1.42	1.23	1.54	1.42	1.46	1.59	1.59	1.50	1.54	1.32	1.59	1.46	1.46	1.46
22	1.06	1.17	1.06	1.23	1.15	1.15	1.46	1.15	1.35	1.20	1.35	1.50	0.92	1.35	1.46	1.17	1.42	1.50	1.54	1.54	1.35	1.35	1.46	1.54	1.50	1.42	1.42	1.54	1.20
24	0.75	1.29	1.02	0.98	0.83	1.20	1.29	1.26	0.96	1.46	1.38	1.23	1.15	1.54	1.06	1.46	1.59	1.64	1.35	1.50	1.35	1.54	1.50	1.50	1.32	1.20	1.17	1.32	1.46
26	0.98	0.98	1.10	1.50	1.10	1.17	1.54	1.29	1.35	1.42	1.35	1.50	1.32	1.38	1.59	1.46	1.42	1.54	1.54	1.54	1.50	1.50	1.38	1.17	1.32	1.20	1.50	1.46	1.35
28	1.00	1.20	0.93	0.93	0.93	1.50	1.35	1.35	1.46	1.38	1.50	1.46	1.35	1.54	1.59	1.38	1.54	1.54	1.54	1.50	1.38	1.00	1.17	1.29	1.35	1.42	1.26	1.50	1.50
30	1.32	1.29	1.46	1.32	1.23	1.15	1.29	1.46	1.42	1.54	1.59	1.46	1.59	1.46	1.35	1.54	1.50	1.50	1.46	1.04	1.17	1.29	1.46	1.42	1.26	1.50	1.46	1.46	1.50
32	**1.50**	0.83	1.04	0.89	**1.64**	0.95	1.17	0.96	1.15	1.35	1.38	1.59	**1.64**	1.26	1.46	1.35	1.42	1.35	**1.15**	1.20	1.38	1.26	1.29	1.38	1.46	1.38	1.50	1.50	**1.46**
34	1.08	1.10	1.10	1.08	1.35	1.23	1.50	1.38	1.46	1.26	1.46	1.54	1.42	1.54	1.50	1.46	1.50	1.08	1.26	1.38	1.50	1.50	1.46	1.46	1.50	1.46	1.38	1.13	1.42
36	1.06	1.42	1.15	1.46	1.02	1.50	1.42	1.50	1.06	1.59	1.59	1.38	1.42	1.50	1.50	1.38	1.20	1.35	1.35	1.38	1.38	1.50	1.50	1.54	1.46	1.35	1.26	1.13	1.42
38	1.17	1.23	1.08	1.26	1.26	1.29	1.38	1.26	1.46	1.54	1.29	1.54	1.46	1.50	1.38	1.04	1.35	1.32	1.32	1.50	1.50	1.46	1.50	1.42	1.29	1.32	1.38	1.42	1.46
40	0.83	1.00	1.10	1.32	1.10	1.38	1.32	1.38	1.59	1.42	1.26	1.50	1.38	1.50	1.17	1.32	1.23	1.26	1.46	1.50	1.50	1.50	1.35	1.38	1.08	1.42	1.42	1.42	1.54
42	0.95	1.42	1.50	1.46	1.35	1.42	1.54	1.42	1.59	1.59	1.46	1.54	1.42	1.17	1.29	1.38	1.35	1.50	1.46	1.46	1.46	1.46	1.38	1.20	1.38	1.38	1.42	1.42	1.42
44	1.02	1.46	1.29	1.35	0.93	1.42	1.46	1.54	1.23	1.46	1.46	1.46	1.20	1.08	1.50	1.32	1.42	1.50	1.46	1.32	1.23	1.35	1.42	1.42	1.42	1.42	1.42	1.38	1.38
46	1.32	1.13	1.38	1.08	1.46	1.38	1.38	1.46	1.50	1.50	1.50	0.98	1.17	1.42	1.35	1.50	1.50	1.46	1.35	1.13	1.32	1.46	1.42	1.42	1.46	1.42	1.38	1.35	1.35
48	0.96	1.06	1.02	1.42	1.13	1.10	1.54	1.29	1.29	1.42	1.46	1.23	1.46	1.46	1.32	1.46	1.50	1.46	1.17	1.23	1.50	1.42	1.42	1.42	1.42	1.38	1.32	1.32	**1.54**
50	0.90	1.32	1.29	1.26	1.26	1.46	1.59	1.42	1.46	1.42	1.10	1.20	1.17	1.50	1.50	1.50	1.46	1.38	1.32	1.46	1.42	1.42	1.42	1.38	1.35	1.32	1.32	1.29	1.32
52	1.00	1.54	1.32	1.38	1.38	1.59	1.42	1.42	1.42	1.35	1.23	1.50	1.38	1.46	1.46	1.50	1.32	1.46	1.35	1.38	1.46	1.42	1.38	1.35	1.35	1.35	1.32	1.29	1.35
54	1.15	1.20	1.54	1.50	1.08	1.46	1.38	1.50	1.54	1.02	1.29	1.46	1.38	1.50	1.50	1.42	1.35	1.23	1.42	1.46	1.42	1.38	1.38	1.35	1.32	1.29	1.26	1.26	1.26
56	0.90	1.35	1.38	1.46	1.38	1.59	1.35	1.46	1.38	1.26	1.50	1.29	1.50	1.50	1.46	1.26	0.98	1.42	1.42	1.46	1.38	1.35	1.35	1.32	1.32	1.29	1.29	1.26	1.35
58	1.35	0.67	1.35	1.42	1.26	1.42	1.54	1.38	1.13	1.13	1.46	1.46	1.38	1.42	1.23	1.35	1.46	1.35	1.42	1.42	1.35	1.32	1.32	1.29	1.29	1.29	1.29	1.26	1.26
60	1.23	1.26	1.32	1.54	1.54	1.54	1.50	1.46	1.29	1.54	1.35	1.50	1.50	1.46	1.35	1.38	1.38	1.42	1.42	1.38	1.35	1.32	1.29	1.29	1.29	1.26	1.26	1.26	1.29
62	1.35	1.46	1.35	1.35	1.54	1.42	1.35	1.50	1.35	1.50	1.46	1.42	1.50	1.29	1.10	1.42	1.46	1.42	1.38	1.38	1.32	1.29	1.29	1.29	1.26	1.26	1.26	1.23	1.26
64	**1.59**	1.15	1.23	1.46	**1.59**	1.38	1.46	1.15	1.42	1.42	1.50	1.50	1.50	1.50	1.46	1.38	1.50	1.42	1.38	1.35	**1.50**	1.29	1.35	1.29	1.35	1.26	1.26	1.23	**1.38**

Figure 10: Speed-up compared to the same example with each work-items running on 1 double (x-axe: number of data per work-item, y-axe: number of work-items per work-group)



	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48	50	52	54	56	58	60	62	64
8	0.50	0.63	0.75	0.88	1.00	1.13	1.25	1.38	1.50	1.63	1.75	1.88	**2.00**	2.13	2.25	2.38	2.50	2.63	2.75	2.88	3.00	3.13	3.25	3.38	3.50	3.63	3.75	3.88	**4.00**
10	0.63	0.78	0.94	1.09	1.25	1.41	1.56	1.72	1.88	2.03	2.19	2.34	2.50	2.66	2.81	2.97	3.13	3.28	3.44	3.59	3.75	3.91	4.06	4.22	4.38	4.53	4.69	4.84	5.00
12	0.75	0.94	1.13	1.31	1.50	1.69	1.88	2.06	2.25	2.44	2.63	2.81	3.00	3.19	3.38	3.56	3.75	3.94	4.13	4.31	4.50	4.69	4.88	5.06	5.25	5.44	5.63	5.81	6.00
14	0.88	1.09	1.31	1.53	1.75	1.97	2.19	2.41	2.63	2.84	3.06	3.28	3.50	3.72	3.94	4.16	4.38	4.59	4.81	5.03	5.25	5.47	5.69	5.91	6.13	6.34	6.56	6.78	7.00
16	1.00	1.25	1.50	1.75	**2.00**	2.25	2.50	2.75	3.00	3.25	3.50	3.75	**4.00**	4.25	4.50	4.75	5.00	5.25	5.50	5.75	6.00	6.25	6.50	6.75	7.00	7.25	7.50	7.75	**8.00**
18	1.13	1.41	1.69	1.97	2.25	2.53	2.81	3.09	3.38	3.66	3.94	4.22	4.50	4.78	5.06	5.34	5.63	5.91	6.19	6.47	6.75	7.03	7.31	7.59	7.88	8.16	8.44	8.72	9.00
20	1.25	1.56	1.88	2.19	2.50	2.81	3.13	3.44	3.75	4.06	4.38	4.69	5.00	5.31	5.63	5.94	6.25	6.56	6.88	7.19	7.50	7.81	8.13	8.44	8.75	9.06	9.38	9.69	10.00
22	1.38	1.72	2.06	2.41	2.75	3.09	3.44	3.78	4.13	4.47	4.81	5.16	5.50	5.84	6.19	6.53	6.88	7.22	7.56	7.91	8.25	8.59	8.94	9.28	9.63	9.97	10.31	10.66	11.00
24	1.50	1.88	2.25	2.63	3.00	3.38	3.75	4.13	4.50	4.88	5.25	5.63	6.00	6.38	6.75	7.13	7.50	7.88	8.25	8.63	9.00	9.38	9.75	10.13	10.50	10.88	11.25	11.63	12.00
26	1.63	2.03	2.44	2.84	3.25	3.66	4.06	4.47	4.88	5.28	5.69	6.09	6.50	6.91	7.31	7.72	8.13	8.53	8.94	9.34	9.75	10.16	10.56	10.97	11.38	11.78	12.19	12.59	13.00
28	1.75	2.19	2.63	3.06	3.50	3.94	4.38	4.81	5.25	5.69	6.13	6.56	7.00	7.44	7.88	8.31	8.75	9.19	9.63	10.06	10.50	10.94	11.38	11.81	12.25	12.69	13.13	13.56	14.00
30	1.88	2.34	2.81	3.28	3.75	4.22	4.69	5.16	5.63	6.09	6.56	7.03	7.50	7.97	8.44	8.91	9.38	9.84	10.31	10.78	11.25	11.72	12.19	12.66	13.13	13.59	14.06	14.53	15.00
32	**2.00**	2.50	3.00	3.50	**4.00**	4.50	5.00	5.50	6.00	6.50	7.00	7.50	**8.00**	8.50	9.00	9.50	10.00	10.50	11.00	11.50	12.00	12.50	13.00	13.50	14.00	14.50	15.00	15.50	**16.00**
34	2.13	2.66	3.19	3.72	4.25	4.78	5.31	5.84	6.38	6.91	7.44	7.97	8.50	9.03	9.56	10.09	10.63	11.16	11.69	12.22	12.75	13.28	13.81	14.34	14.88	15.41	15.94	16.47	17.00
36	2.25	2.81	3.38	3.94	4.50	5.06	5.63	6.19	6.75	7.31	7.88	8.44	9.00	9.56	10.13	10.69	11.25	11.81	12.38	12.94	13.50	14.06	14.63	15.19	15.75	16.31	16.88	17.44	18.00
38	2.38	2.97	3.56	4.16	4.75	5.34	5.94	6.53	7.13	7.72	8.31	8.91	9.50	10.09	10.69	11.28	11.88	12.47	13.06	13.66	14.25	14.84	15.44	16.03	16.63	17.22	17.81	18.41	19.00
40	2.50	3.13	3.75	4.38	5.00	5.63	6.25	6.88	7.50	8.13	8.75	9.38	10.00	10.63	11.25	11.88	12.50	13.13	13.75	14.38	15.00	15.63	16.25	16.88	17.50	18.13	18.75	19.38	20.00
42	2.63	3.28	3.94	4.59	5.25	5.91	6.56	7.22	7.88	8.53	9.19	9.84	10.50	11.16	11.81	12.47	13.13	13.78	14.44	15.09	15.75	16.41	17.06	17.72	18.38	19.03	19.69	20.34	21.00
44	2.75	3.44	4.13	4.81	5.50	6.19	6.88	7.56	8.25	8.94	9.63	10.31	11.00	11.69	12.38	13.06	13.75	14.44	15.13	15.81	16.50	17.19	17.88	18.56	19.25	19.94	20.63	21.31	22.00
46	2.88	3.59	4.31	5.03	5.75	6.47	7.19	7.91	8.63	9.34	10.06	10.78	11.50	12.22	12.94	13.66	14.38	15.09	15.81	16.53	17.25	17.97	18.69	19.41	20.13	20.84	21.56	22.28	23.00
48	3.00	3.75	4.50	5.25	6.00	6.75	7.50	8.25	9.00	9.75	10.50	11.25	12.00	12.75	13.50	14.25	15.00	15.75	16.50	17.25	18.00	18.75	19.50	20.25	21.00	21.75	22.50	23.25	**24.00**
50	3.13	3.91	4.69	5.47	6.25	7.03	7.81	8.59	9.38	10.16	10.94	11.72	12.50	13.28	14.06	14.84	15.63	16.41	17.19	17.97	18.75	19.53	20.31	21.09	21.88	22.66	23.44	24.22	25.00
52	3.25	4.06	4.88	5.69	6.50	7.31	8.13	8.94	9.75	10.56	11.38	12.19	13.00	13.81	14.63	15.44	16.25	17.06	17.88	18.69	19.50	20.31	21.13	21.94	22.75	23.56	24.38	25.19	26.00
54	3.38	4.22	5.06	5.91	6.75	7.59	8.44	9.28	10.13	10.97	11.81	12.66	13.50	14.34	15.19	16.03	16.88	17.72	18.56	19.41	20.25	21.09	21.94	22.78	23.63	24.47	25.31	26.16	27.00
56	3.50	4.38	5.25	6.13	7.00	7.88	8.75	9.63	10.50	11.38	12.25	13.13	14.00	14.88	15.75	16.63	17.50	18.38	19.25	20.13	21.00	21.88	22.75	23.63	24.50	25.38	26.25	27.13	28.00
58	3.63	4.53	5.44	6.34	7.25	8.16	9.06	9.97	10.88	11.78	12.69	13.59	14.50	15.41	16.31	17.22	18.13	19.03	19.94	20.84	21.75	22.66	23.56	24.47	25.38	26.28	27.19	28.09	29.00
60	3.75	4.69	5.63	6.56	7.50	8.44	9.38	10.31	11.25	12.19	13.13	14.06	15.00	15.94	16.88	17.81	18.75	19.69	20.63	21.56	22.50	23.44	24.38	25.31	26.25	27.19	28.13	29.06	30.00
62	3.88	4.84	5.81	6.78	7.75	8.72	9.69	10.66	11.63	12.59	13.56	14.53	15.50	16.47	17.44	18.41	19.38	20.34	21.31	22.28	23.25	24.22	25.19	26.16	27.13	28.09	29.06	30.03	31.00
64	**4.00**	5.00	6.00	7.00	**8.00**	9.00	10.00	11.00	12.00	13.00	14.00	15.00	**16.00**	17.00	18.00	19.00	20.00	21.00	22.00	23.00	**24.00**	25.00	26.00	27.00	28.00	29.00	30.00	31.00	**32.00**

Figure 11: Size of data compute by each work-group (x-axe: number of data per work-item, y-axe: number of work-items per work-group)