III КОНФЕРЕНЦИЯ FPGA РАЗРАБОТЧИКОВ

FPGA-Systems 2021.2

Доступно в записи на Youtube

Конференция в Москве

Конференция в Санкт-Петербурге

Приходи на следующую конференцию

fpga-systems.ru/meet

Поддержи мероприятие

Способ 1

Способ 2

Высокоуровневый синтез на Intel FPGA: от HLS к OpenCL и OneAPI

Александр Корнев
«Акселтех»

О компании «Акселтех»

Ускорительные решения на реконфигурируемых сопроцессорах

— Разработка и производство аппаратуры реконфигурируемых сопроцессоров (РФ)

— Разработка программно-аппаратных комплексов и интеграция IP блоков

EULER FMC, SFP+, QSFP+

SONET/SDH, OT
ETHERNET, IP SE

EULER PRIME HPC 2XDDR4

EULER EMBEDDED SOM

OPENCL, ONEA
NEURAL NETWO
OPENVINO

EULER LINE NET 20G/80G/10

4G LTE CPRI
5G L1 ACCELERA

План

- Основная проблематика HLS на Intel FPGA
  - конвейеризация
  - синтез памяти
- Стандарт OpenCL
  - модель применения
  - 3 вида параллелизма
- От OpenCL к OneAPI
  - модель “one code”
  - инструменты разработчика
- Обзор применения технологий Intel FPGA в продуктах компании «Акселтех»

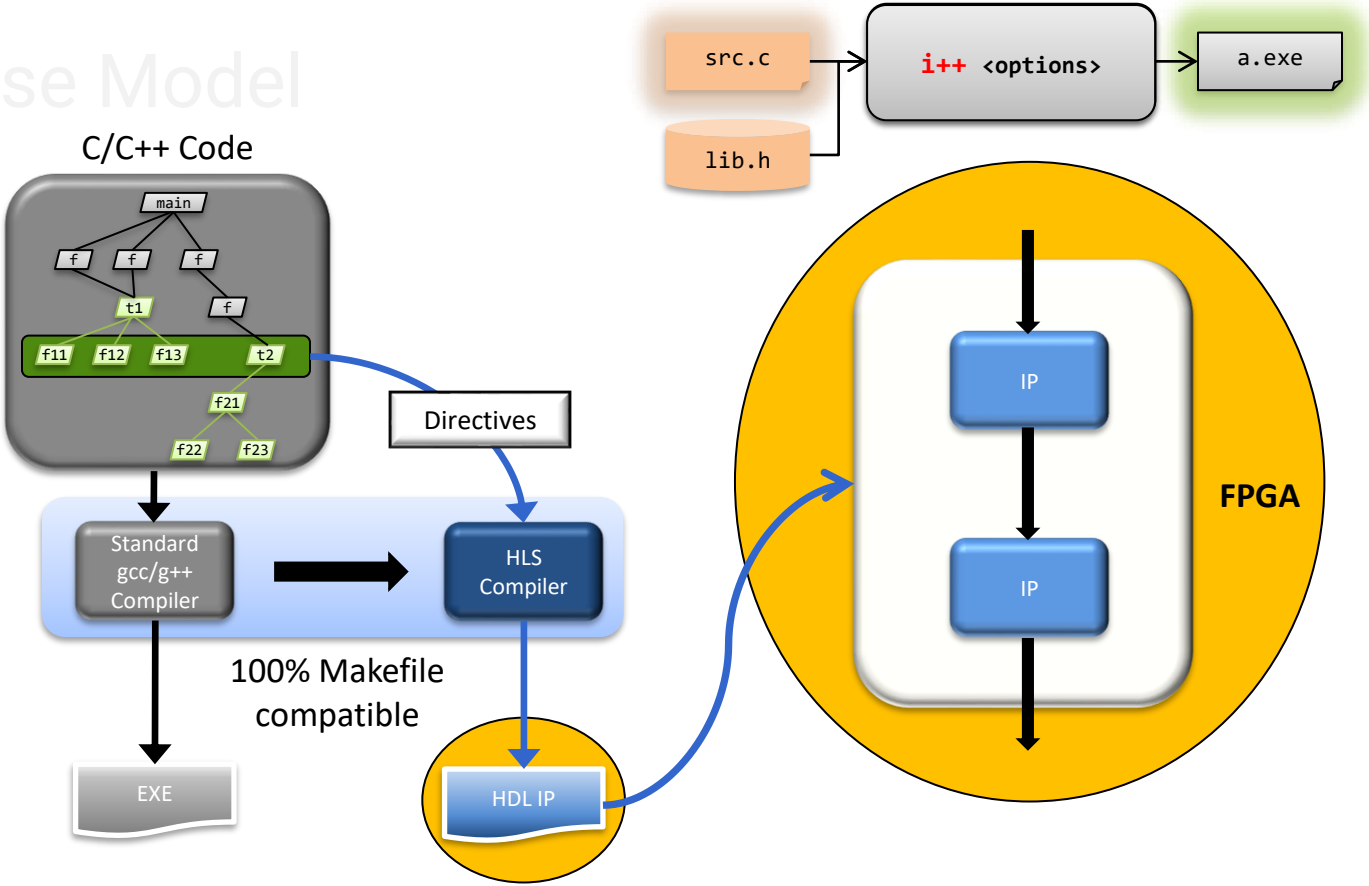
Задачи высокоуровневого синтеза

- Уменьшение Time-to-Market
- Снижение «порога вхождения» при использовании FPGA
- Применение FPGA в гетерогенных системах

Этапы развития высокоуровневого синтеза Intel

- HLS (High-level Synthesys)
- OpenCL BSP для Intel FPGA
- Intel OneAPI

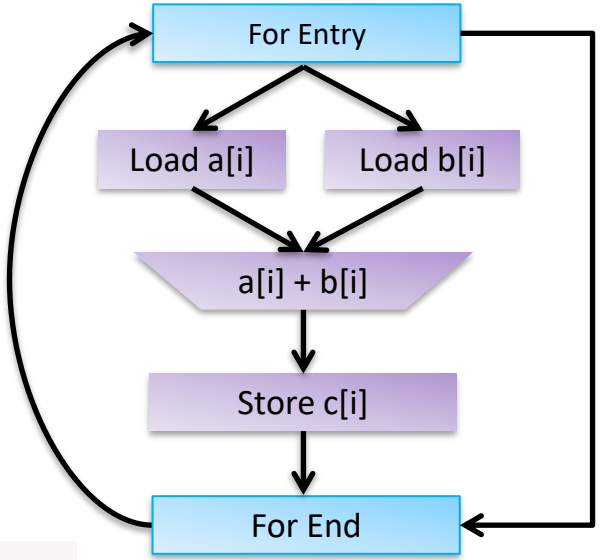
HLS: модель применения



HLS: пример

```
void my_component(          int *a,
                            int *b,
                            int *c,
                            int N)
{
        int i;
        for (i = 0; i < N; i++)
                c[i] = a[i] + b[i];
}
```

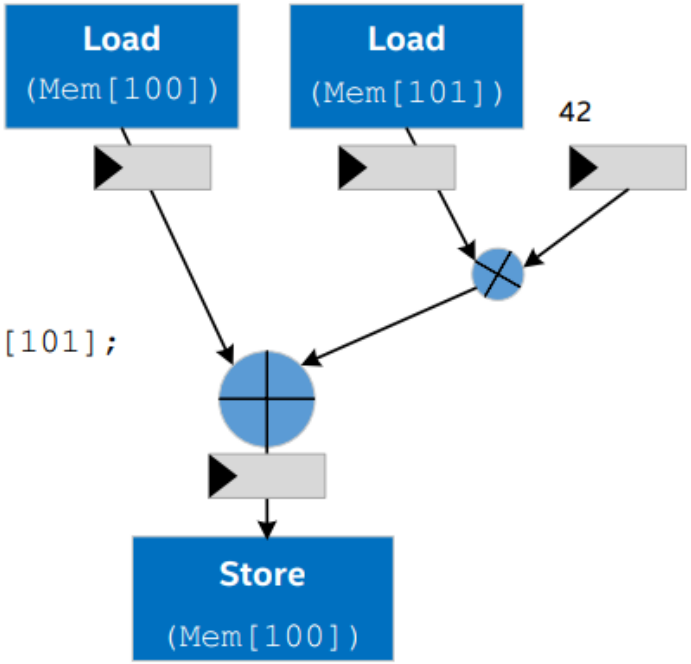
i++

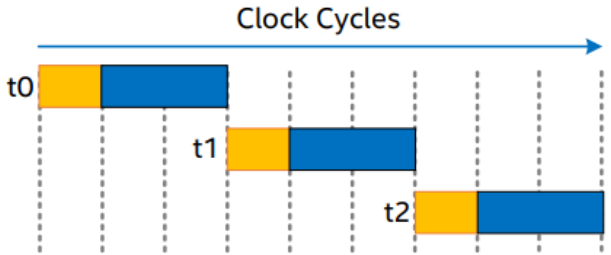



```
i++ -c m_component.cpp -march=Arria10
i++ --fpga-only m_component.o
```

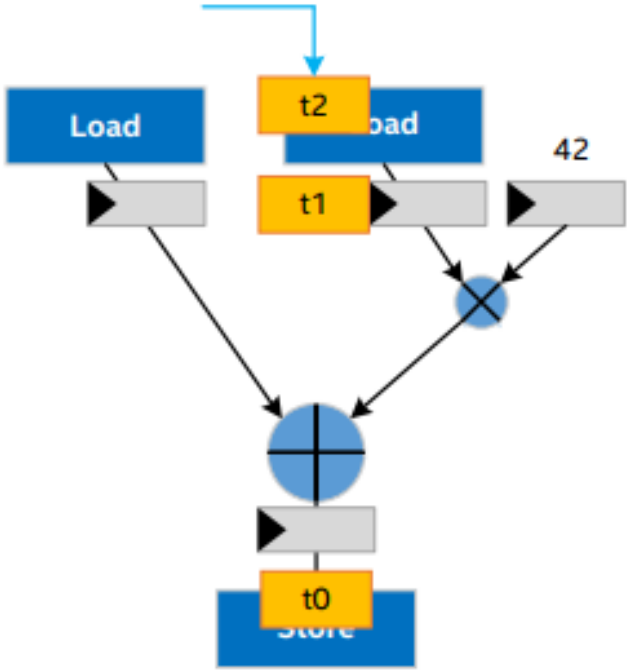
HLS: последовательное выполнение

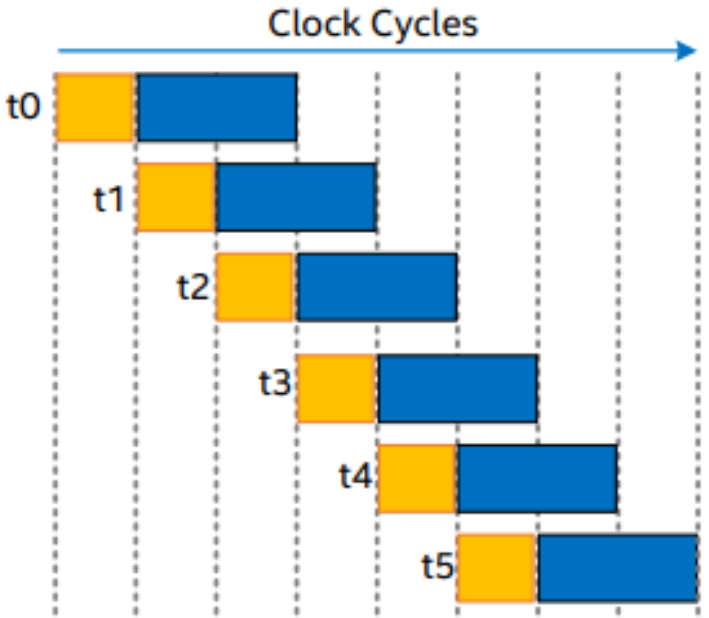
```
Mem[100] += 42 * Mem[101];
```





HLS: синтез конвейера – оптимальный вариант

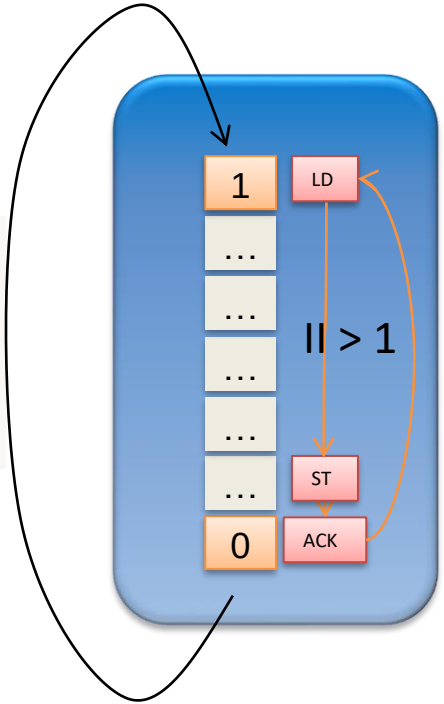




HLS: зависимости как проблемы конвейеризации

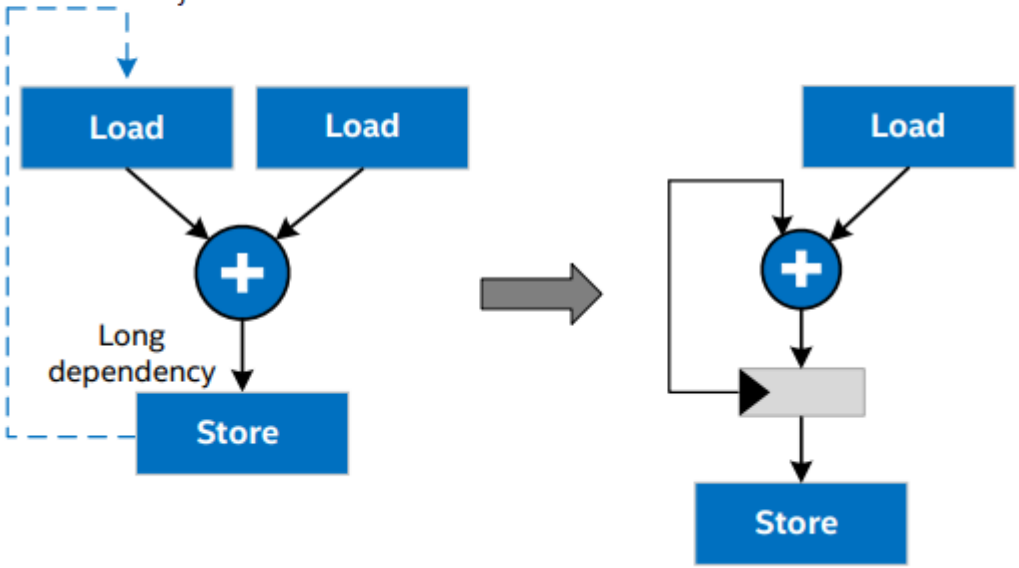
- Зависимость по данным
- Зависимость по памяти

```
for ( … ) {
    A[x] = A[y];
    …
}
```



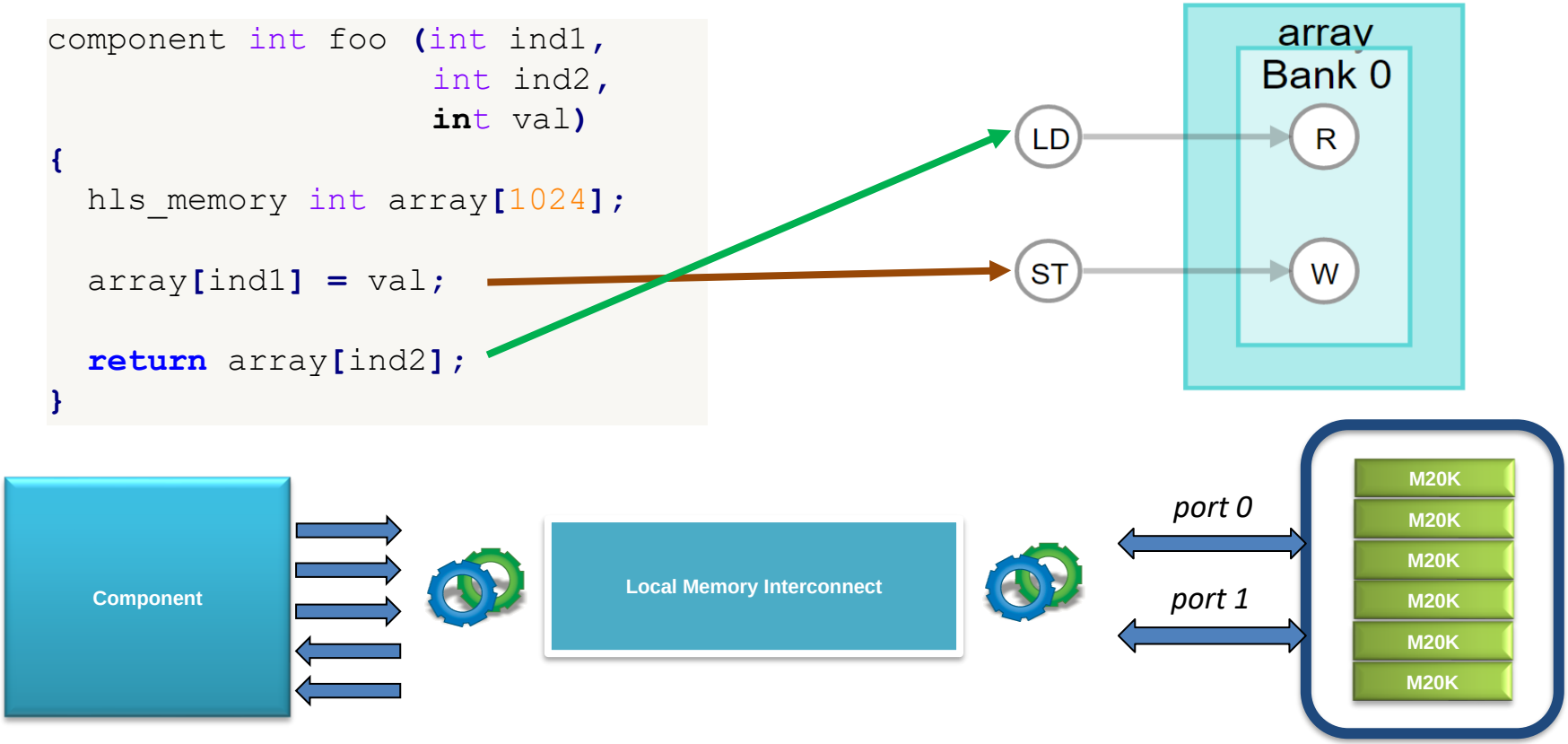
HLS: устранение зависимости по данным в цикле

```
for ( int i = 0; i < n; i++ )
{
    c[i] = c[i-1] + b[i];
}
```

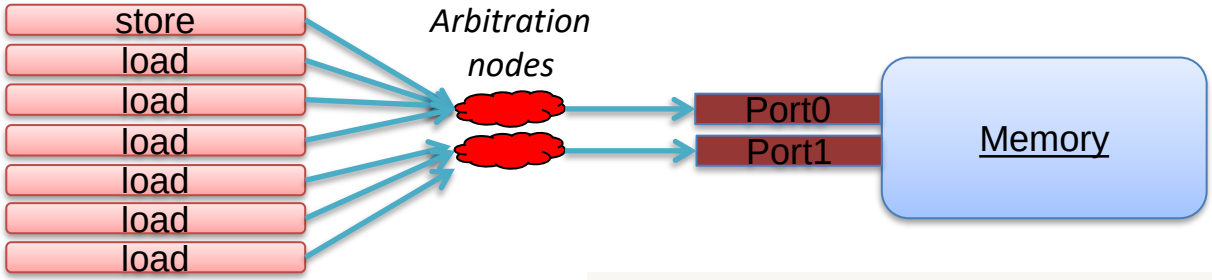


HLS: автоматический синтез локальной памяти

```
component int foo (int ind1,
                   int ind2,
                   int val)
{
  hls_memory int array[1024];

  array[ind1] = val;

  return array[ind2];
}
```

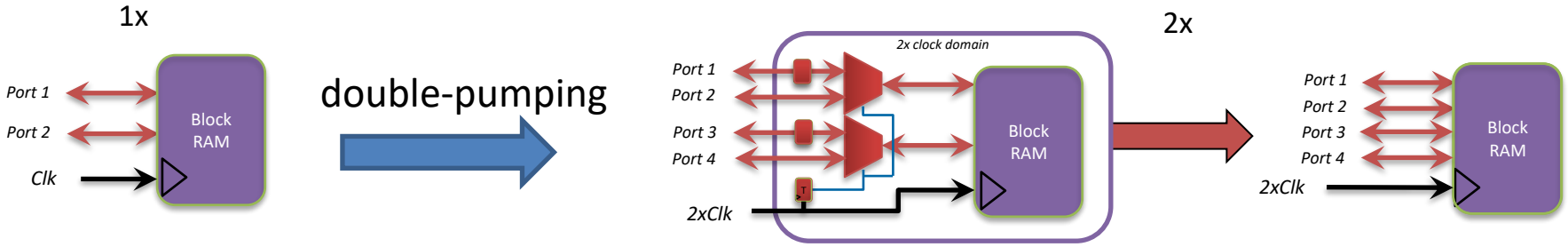


HLS: проблема арбитража

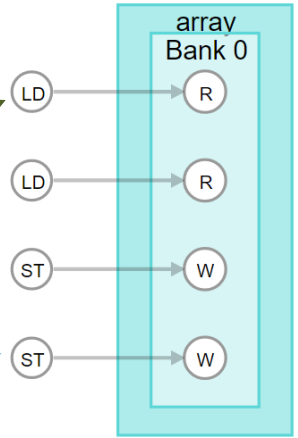



```
component
int foo_arb(int ind1, int ind2,
            int ind3, int val1, int val2)
{
  hls_memory int array[1024];
  array[ind1] = val1;
  array[ind1+1] = val1;
  array[ind2+1] = val2;
  array[ind2] = val2;
  return array[ind3];
}
```

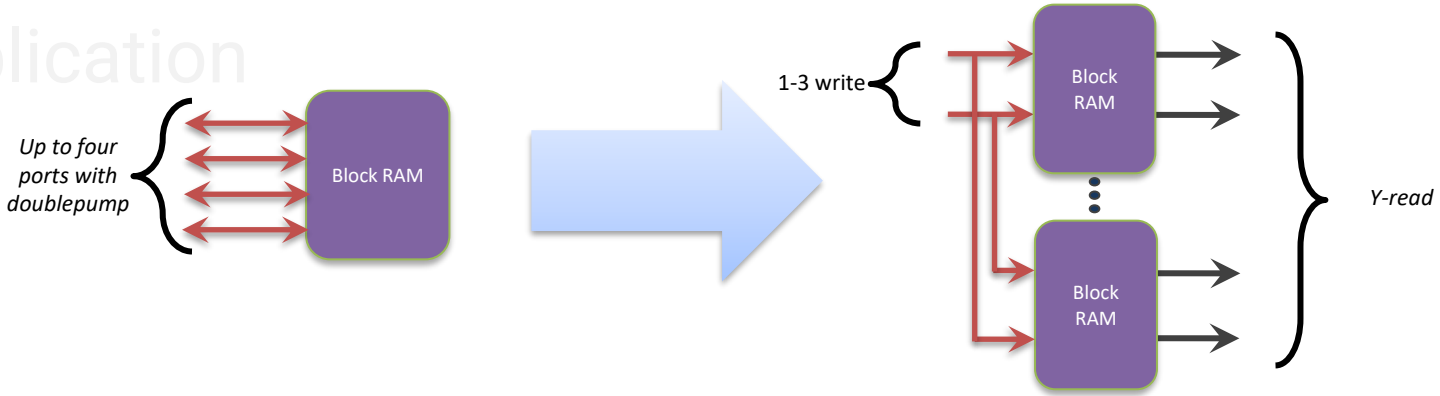
HLS: оптимизация памяти – double pumping

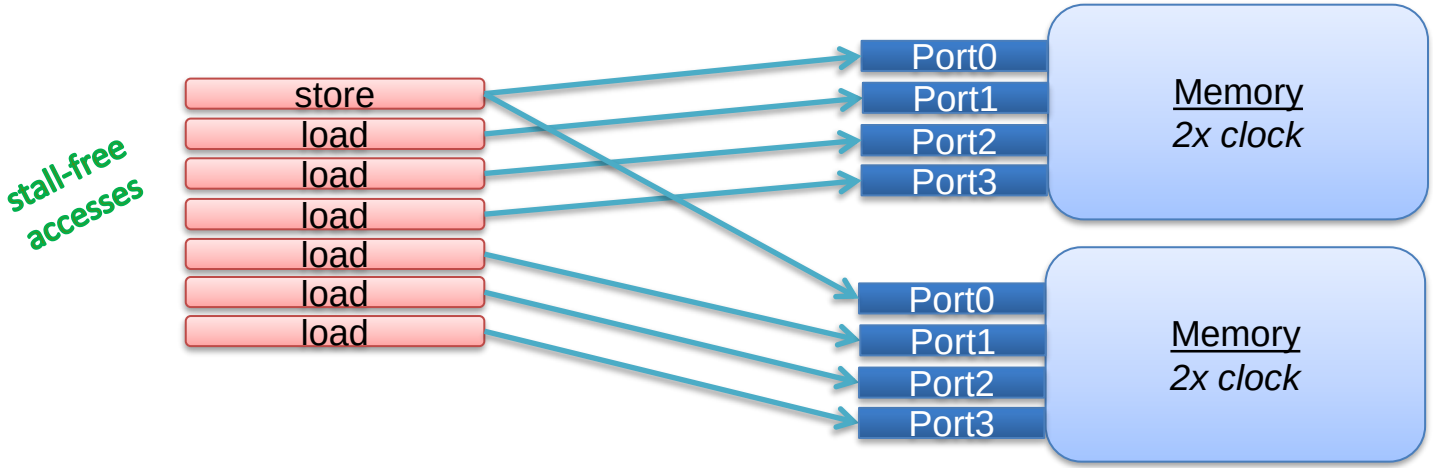



```
component
int bar (int ind1, int ind2,
         int val)
{
  hls_memory int array[1024];

  array[ind1] = val;

  array[ind1+1] = val;

  return array[ind2] + array[ind2+1];
}
```



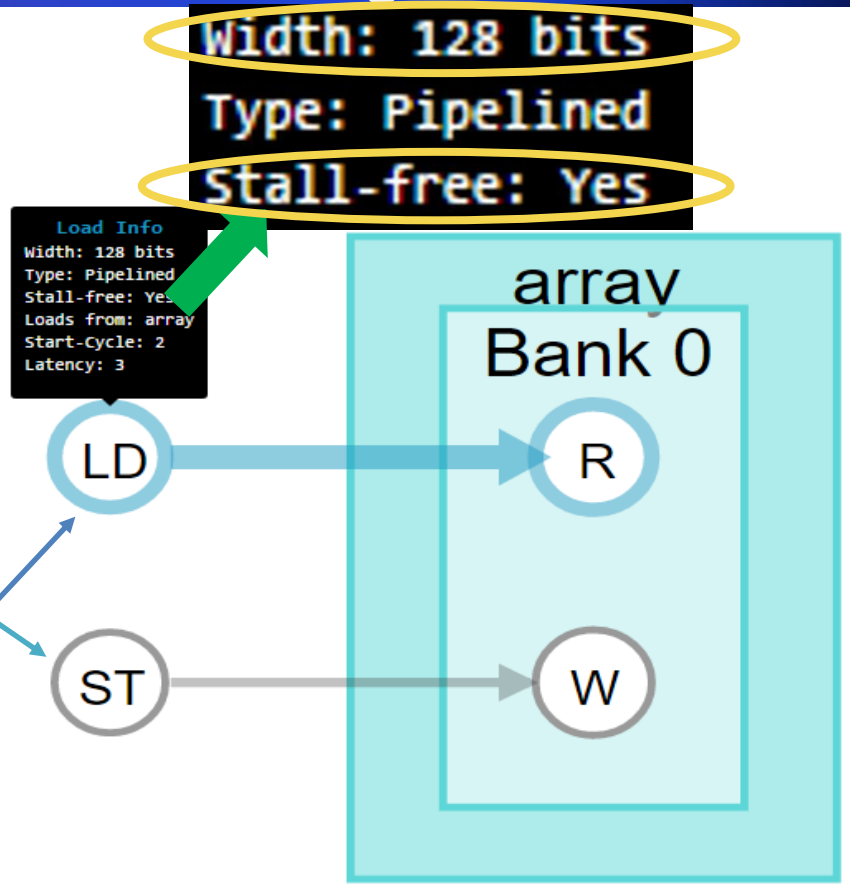
HLS: оптимизация памяти – replication





HLS: оптимизация памяти – coalescing

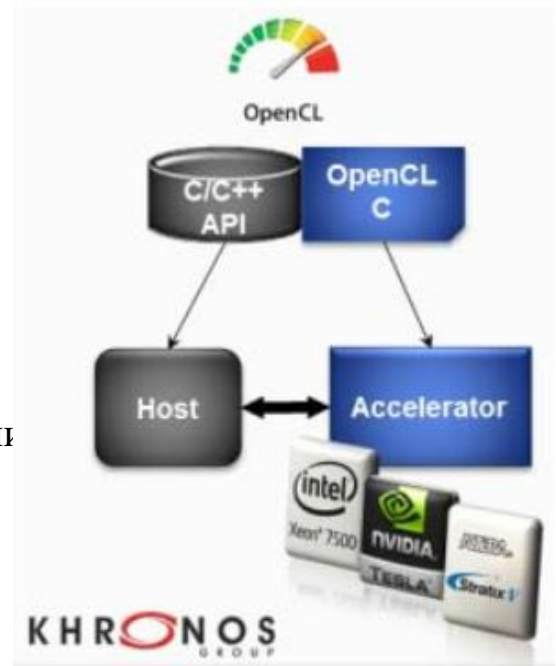
```
component
int foo_coal (int ind1,int ind2,int val)
{
        hls_memory int array[1024];
        int res = 0;

        #pragma unroll
        for (int i = 0; i < 4; i++)
            array[ind1*4 + i] = val;

        #pragma unroll
        for (int i = 0; i < 4; i++)
            res += array[ind2*4 + i];

  return res;
}
```



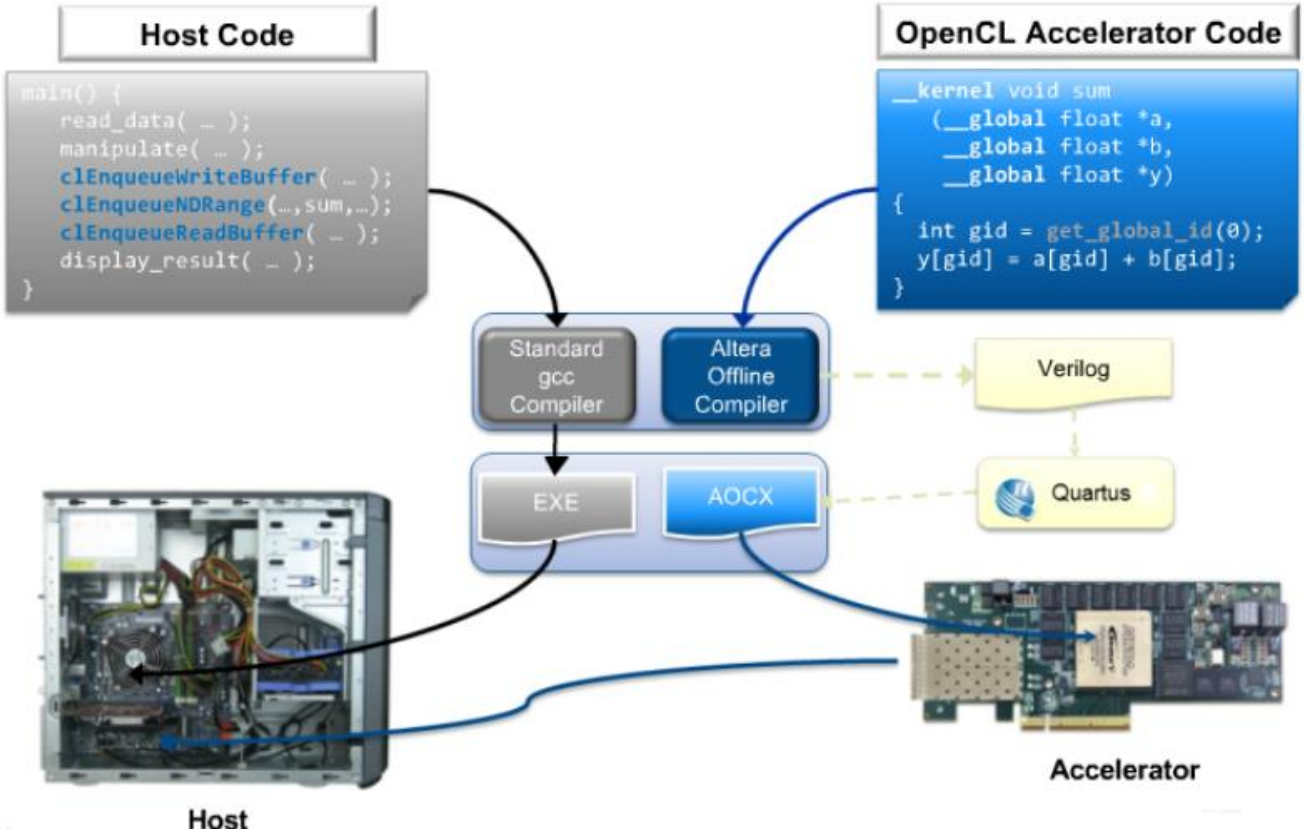
Часть 2. Стандарт OpenCL

От HLS к OpenCL

1. Уровень абстракции относительно САПР Quartus;
2. Обеспечивает параллелизм по данным и по задачам;
3. Поддерживает язык программирования *C99;*
4. Поддерживает стандарт *IEEE* 754 (float point);
5. Поддерживает одновременную работу с несколькими устройствами



OpenCL: модель применения



От программы на C к программе на OpenCL C

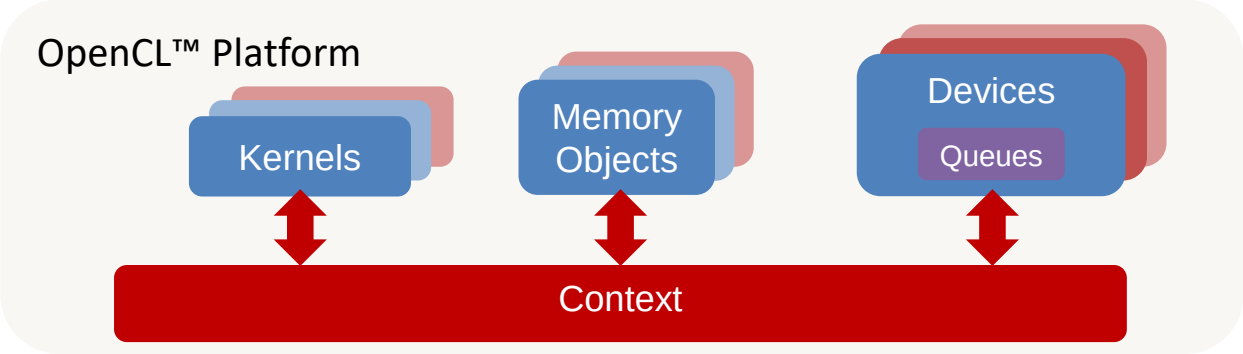
C

```
void inc(float *a, float c, int N)
{
    for(int i = 0; i<N; i++)
        a[i] = a[i] + c;
}

void main() {
    …
    inc(a,c,N);
    …
}
```

OpenCL

```
__kernel
void inc(__global float *a, float c)
{
    int i = get_global_id(0);
    a[i] = a[i] + c;
}

void main() {
    …
    clEnqueueNDRangeKernel(…,&N,…)
    …
}
```

OpenCL Хост: управления kernel




```
//Get the Platforms
std::vector<cl::Platform> plist;
err=cl::Platform::get(&plist);
// Get the FPGA devices in the first platform
std::vector<cl::Device> mydevlist;
err=plist[0].getDevices(CL_DEVICE_TYPE_ACCELERATOR, &mydevlist);
//Create an OpenCL™ context for the FPGA devices
cl::Context mycontext (&mydevlist);
```

OpenCL Хост: планирование с помощью Queue

Constructor

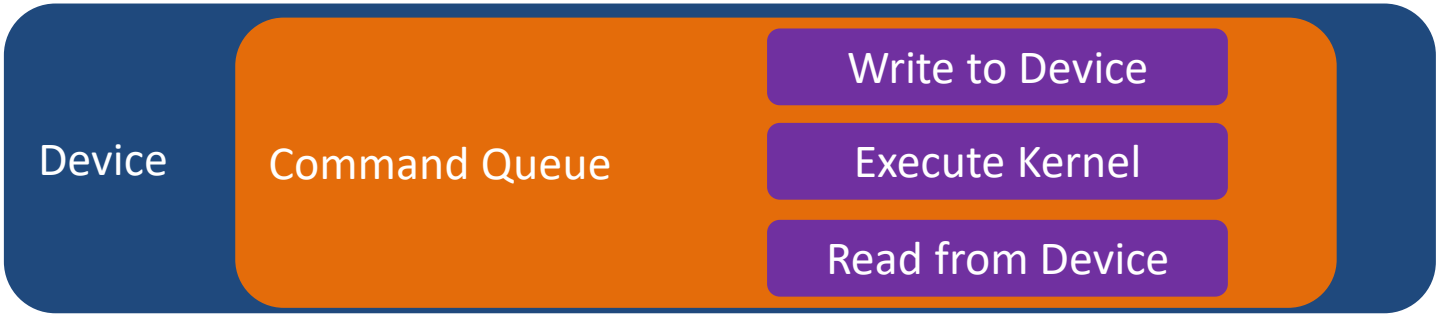
```
cl::CommandQueue::CommandQueue(
                        const Context& context,
                        const Device& device,
                        cl_command_queue_properties properties=0,
                        cl_int *errcode_ret=NULL)
```

Valid context

A device associated with *context*

Error code

Queue properties
e.g. Turn on profiling



OpenCL Хост: применение Queue

```
const int N = 5;
int nBytes = N*sizeof(int);
int hostarr [N] = {3,1,4,1,5};

//Create an OpenCL™ command queue
cl::CommandQueue myq(mycontext, mydevlist[0]);

// Allocate memory on device
cl::Buffer buf_a(mycontext, CL_MEM_READ_WRITE, nBytes);

// Transfer Memory
cl_int err;
err=myq.enqueueWriteBuffer(buf_a, CL_FALSE, 0, nBytes, hostarr);
```

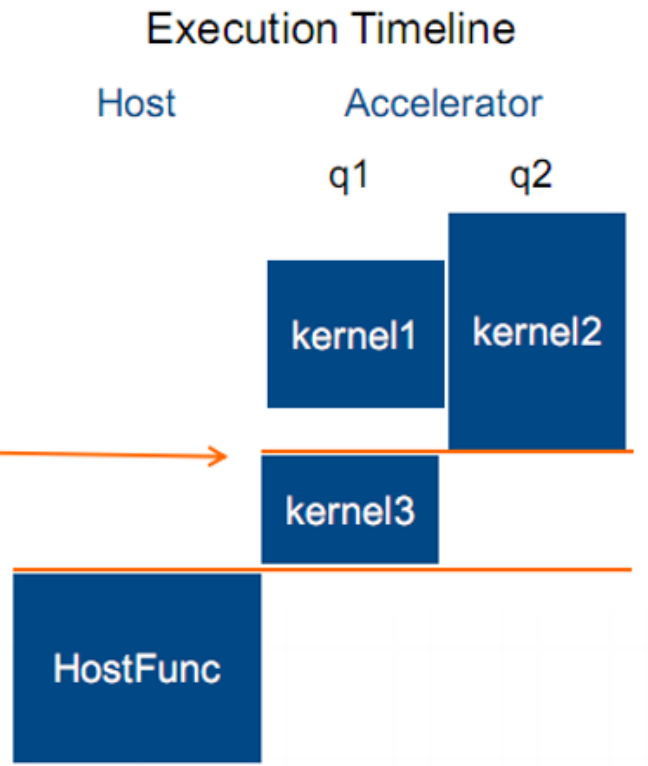
OpenCL Хост: Модель выполнения

Host Code

```
setup_memory_buffers();
transfer_data_to_fpga();

clEnqueueTask(myqueue, mykernel, …);

read_data_from_fpga();
```

OpenCL Хост: синхронизация на основе Queue

```
cl_command_queue q1, q2;
cl_event e1, e2, e3;

clEnqueueNDRangeKernel(q1,k1,…, &e1);

clEnqueueNDRangeKernel(q2,k2,…, &e2);

cl_event elist[2];
elist[0]=e1;
elist[1]=e2;

clEnqueueNDRangeKernel(q1,k3,…,2,elist,&e3);

//Sync Pt, could also use clFinish
clWaitForEvents(1, &e3);

HostFunc();
```



OpenCL: виды параллельных вычислений

- Параллелизм по задачам (task)
- Конвейерное выполнение (pipeline)
- Параллелизм по данным (SIMD)

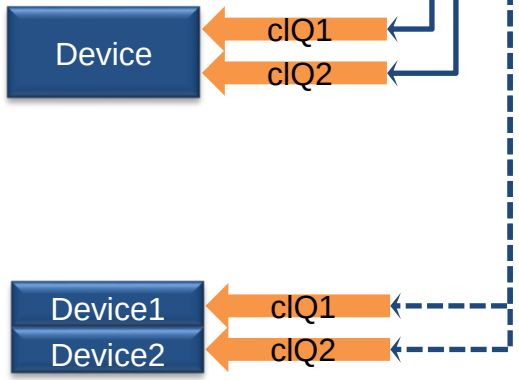
OpenCL: параллелизм по задачам

Implicit Parallelism

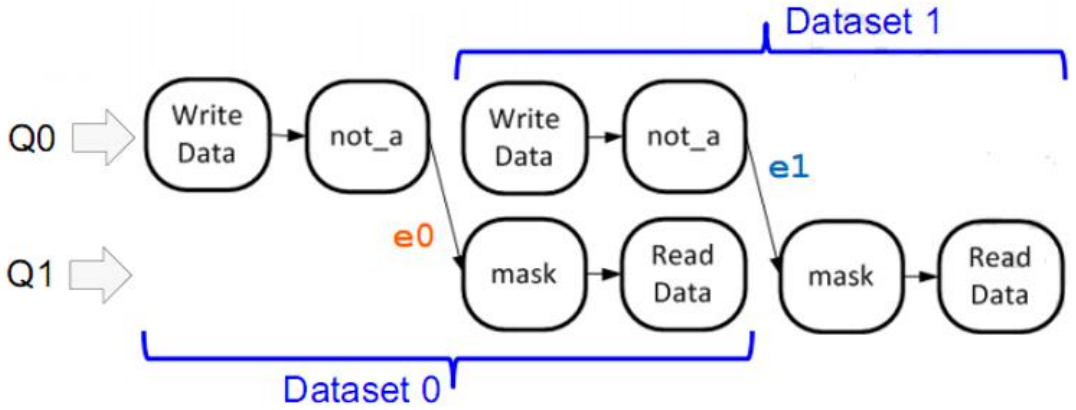
```
u = foo(x);
y = bar(x);
```

Task Parallelism (SMT)

```
clQ1.enqueueNDRangeKernel(cl_foo,…)
clQ2.enqueueNDRangeKernel(cl_bar,…)
```



OpenCL: конвейерное выполнение




```
cl_command_queue q0, q1;
cl_event e0, e1;

// Queue 0, Potentially could make this a loop if there are more data sets
clEnqueueWriteBuffer(q0, …); // Write Dataset 0
clEnqueueNDRangeKernel(q0,not_a,…, &e0);  //Execute Kernel 0 dataset 0
clEnqueueWriteBuffer(q0, …); // Write Dataset 1
clEnqueueNDRangeKernel(q0,not_a,…, &e1);  //Execute Kernel 0 dataset 1

// Queue 1, Potentially could make this a loop if there are more data sets
clEnqueueNDRangeKernel(q1,mask,…, 1, &e0, NULL); //Execute Kernel 1 dataset 0
clEnqueueReadBuffer(q1, …);                      //Read back dataset 0
clEnqueueNDRangeKernel(q1,mask,…, 1, &e1, NULL); //Execute Kernel 1 dataset 1
clEnqueueReadBuffer(q1, …);                   //Read back dataset 1
```

OpenCL: SIMD-параллелизм

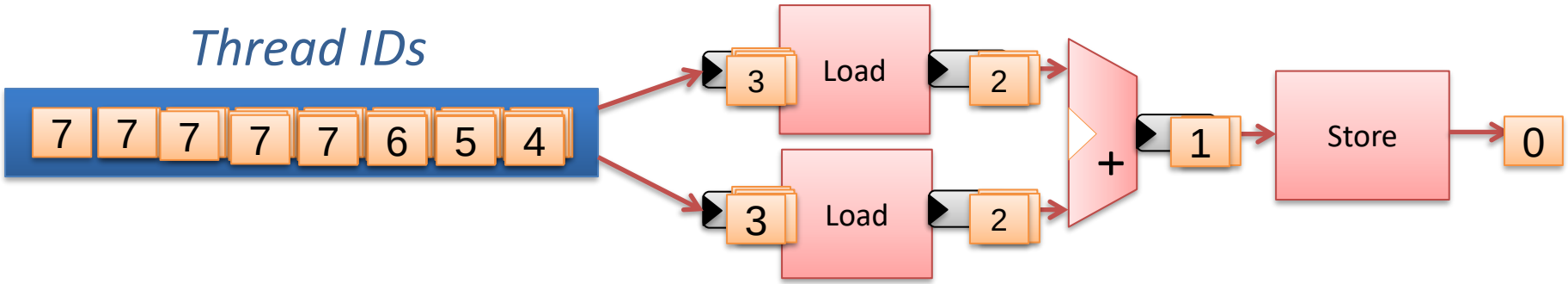
Vectored addition of A and B example

C

```
for (int i=0; i<N; i++)
{
    C[i] = A[i] + B[i];
}
```

OpenCL™ Kernel

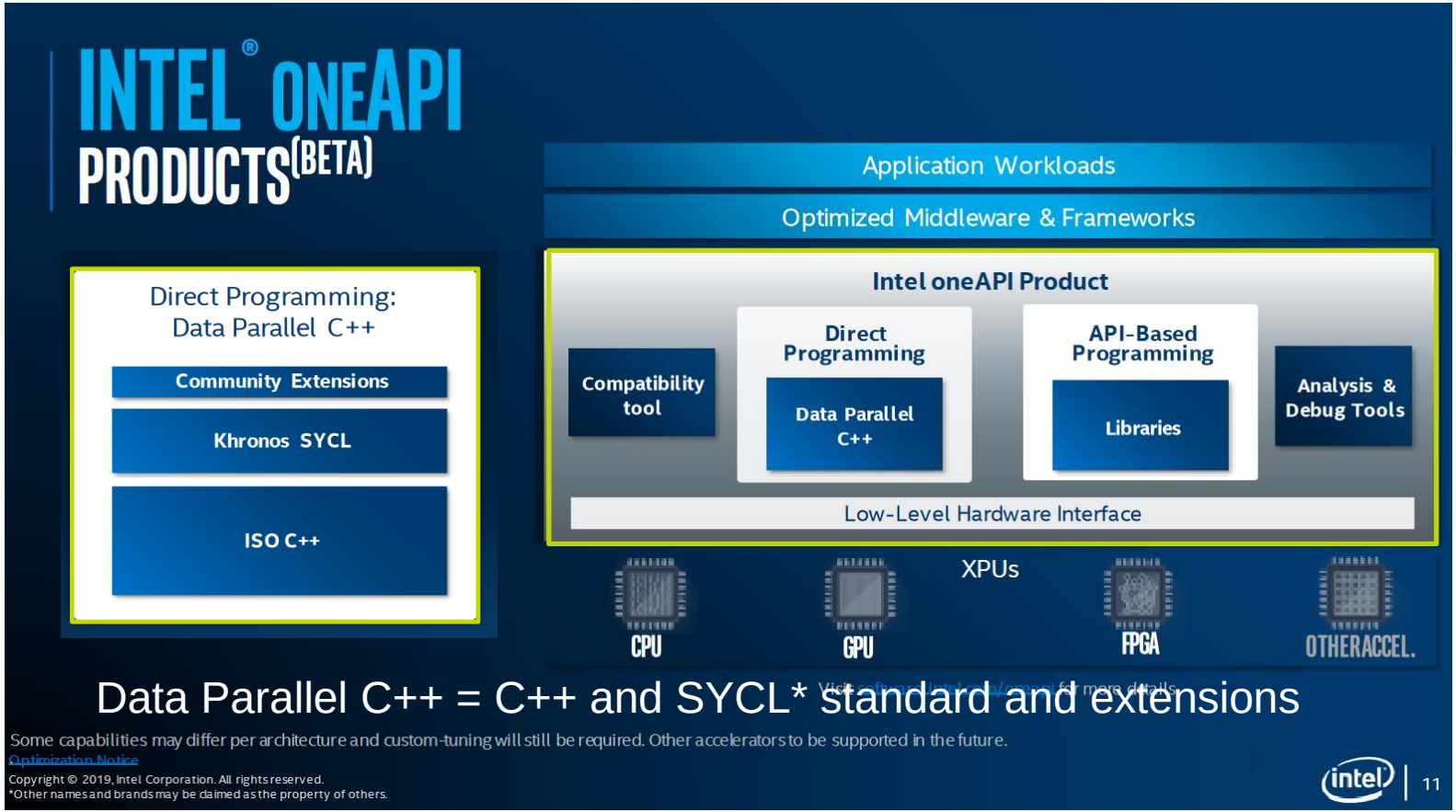
```
// N work-items to be created
__kernel void vecadd(__global int *C,
                     __global int *A,
                     __global int *B)
{
        int tid = get_global_id(0);
        C[tid] = A[tid] + B[tid];
}
```



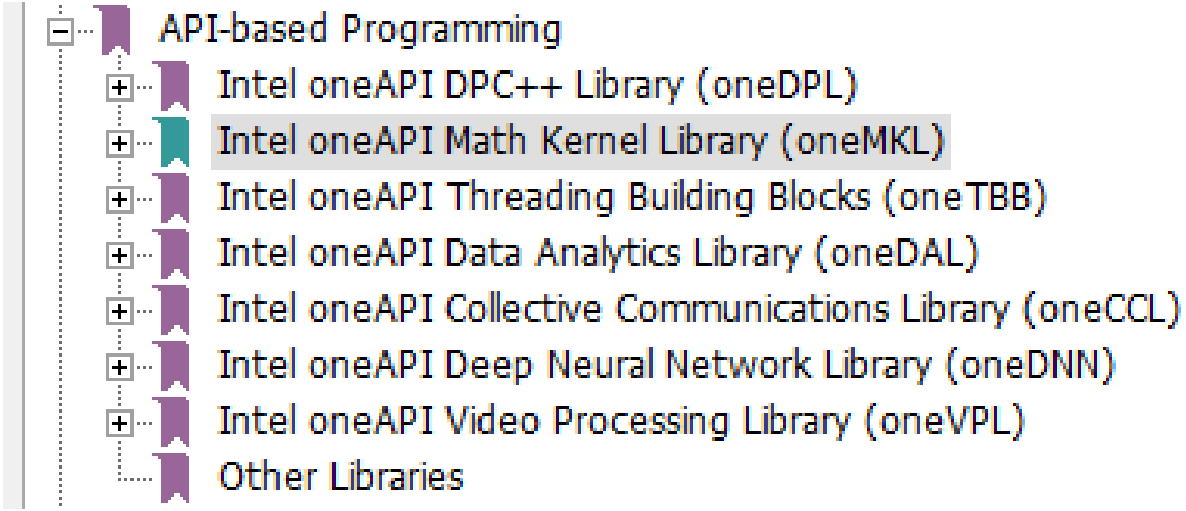
От OpenCL к OneAPI

Часть 3. OneAPI и Data Parallel C++

OneAPI: Унифицированная модель разработки



OneAPI: Библиотеки



DPC++: стратегия “one code”

```
#include <CL/sycl.hpp>
#include <sycl/ext/intel/fpga_extensions.hpp>

using namespace sycl;
static const int N = 16;
int main()
{
  ext::intel::fpga_emulator_selector device_selector;
  queue q { device_selector };
  std::cout << "Device: " << q.get_device().get_info<info::device::name>() << std::endl;

  int *data = malloc_shared<int>(N, q);

  //# Initialization
  for(int i=0; i<N; i++) data[i] = i;

  //# Offload parallel computation to device
  q.parallel_for(range<1>(N), [=] (id<1> i){
    data[i] *= 2;
  }).wait();

  //# Print Output
  for(int i=0; i<N; i++) std::cout << data[i] << std::endl;

  free(data, q);
  return 0;
}
```

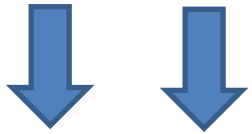
DPC++: Devices and Queues

```
  // Select either:
  //  - the FPGA emulator device (CPU emulation of the FPGA)
  //  - the FPGA device (a real FPGA, can be used for simulation too)
#if defined(FPGA_EMULATOR)
  ext::intel::fpga_emulator_selector device_selector;
#else
  ext::intel::fpga_selector device_selector;
#endif

  queue q(device_selector);
```

```
q.submit([&](handler& h) {
    // COMMAND GROUP CODE
});
```

DPC++: Single Task Kernel vs Parallel Kernel

```
for(int i=0; i < 1024; i++){
    a[i] = b[i] + c[i];
});
```




```
h.single_task([=](){
    for (int i=0; i < 1024; i++) {
        A[i] =  B[i] + C[i];
    }
});
```

```
h.parallel_for(range<1>(1024), [=](id<1> i){
    A[i] =  B[i] + C[i];
});
```

DPC++: асинхронное выполнение

ASYNCHRONOUS EXECUTION (CONT'D)

Host

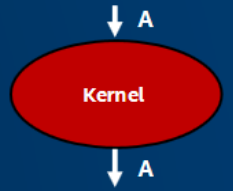
Host code execution

Enqueues kernel to graph, and keeps going

```
#include <CL/sycl.hpp>
#include <iostream>
constexpr int num=16;
using namespace cl::sycl;

int main() {
  auto R = range<1>{ num };
  buffer<int> A{ R };

  queue{}.submit([&](handler& h) {
    auto out = A.get_access<access::mode::write>(h);
    h.parallel_for(R, [=](id<1> idx) {
      out[idx] = idx[0]; }); });

  auto result = A.get_access<access::mode::read>();
  for (int i=0; i<num; ++i)
    std::cout << result[i] << "\n";

  return 0;
}
```

Graph

Graph executes asynchronously to host program



DPC++: модель памяти

MEMORY MODEL

Buffers and Accessors coordinate memory between host and devices. Ensure correctness and performance.

Buffer encapsulates a 1, 2, 3-dimensional array to share between host and devices.

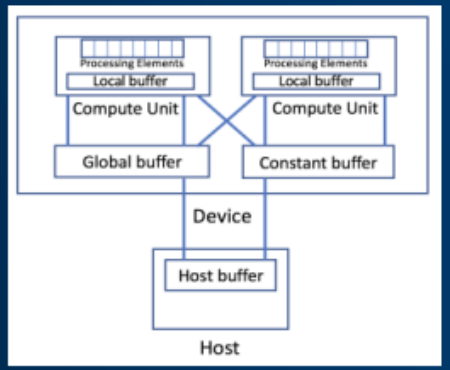
Member functions to obtain size, range, number of elements.

Access target specifies memory location requirement.

Private memory is determined by compiler or by employing private_memory class.

```
sycl::buffer<int>   a_device(a.data(), a_size);
sycl::buffer<int>   c_device(c.data(), a_size);

d_queue.submit([&](sycl::handler  &cgh) {
  sycl::accessor<int,  1, sycl::access::mode::discard_write,
    sycl::access::target::global_buffer> c_res(c_device, cgh);
  sycl::accessor<int,  1, sycl::access::mode::read,
    sycl::access::target::constant_buffer> a_res(a_device, cgh);
```

Access Targets



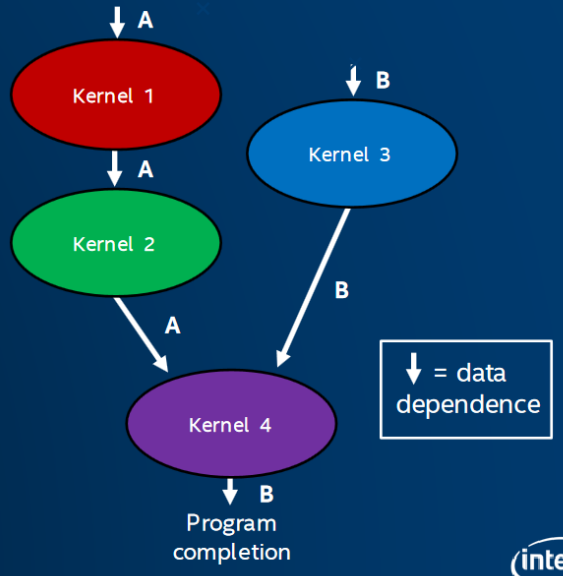
Memory Hierarchy

DPC++: автоматическое планирование

GRAPH OF KERNEL EXECUTIONS

```
int main() {
  auto R = range<1>{ num };
  buffer<int> A{ R }, B{ R };
  queue Q;

  Q.submit([&](handler& h) {
    auto out = A.get_access<access::mode::read_write>(h);
    h.parallel_for(R, [=](id<1> idx) {
      out[idx] = idx[0]; }); });                    } Kernel 1

  Q.submit([&](handler& h) {
    auto out = A.get_access<access::mode::read_write>(h);
    h.parallel_for(R, [=](id<1> idx) {
      out[idx] = idx[0]; }); });                    } Kernel 2

  Q.submit([&](handler& h) {
    auto out = B.get_access<access::mode::read_write>(h);
    h.parallel_for(R, [=](id<1> idx) {
      out[idx] = idx[0]; }); });                    } Kernel 3

  Q.submit([&](handler& h) {
  auto in = A.get_access<access::mode::read>(h);
  auto inout =
    B.get_access<access::mode::read_write>(h);
  h.parallel_for(R, [=](id<1> idx) {
    inout[idx] *= in[idx]; }); });                  } Kernel 4
}
```

Automatic data and control dependence resolution!



DPC++: Unified Shared Pointers

Аннотированные указатели

```
T* ptr = malloc_device<T>(1024, Queue);
...
cgh.single_task<class DeviceAnnotation>([=]() {
  Ptr[0] = 42;  // load-store unit connected to both device and host memories
  device_ptr<T> DevicePtr(Ptr);
  DevicePtr[1] = 43;  // load-store unit connected only to the device memory
});
```

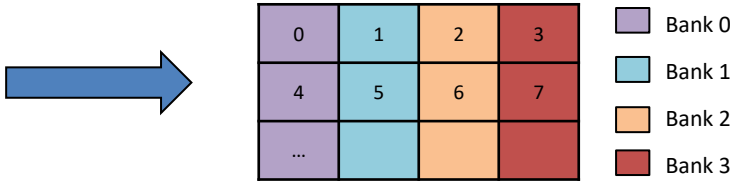
```
T* ptr = malloc_host<T>(1024, Queue);
...
cgh.single_task<class HostAnnotation>([=]() {
  Ptr[0] = 42;  // load-store unit connected to both device and host memories
  host_ptr<T> HostPtr(Ptr);
  HostPtr[1] = 43;  // load-store unit connected only to the host memory
});
```

Доступ к памяти «Zero-copy»

DPC++ и HLS: управление синтезом памяти

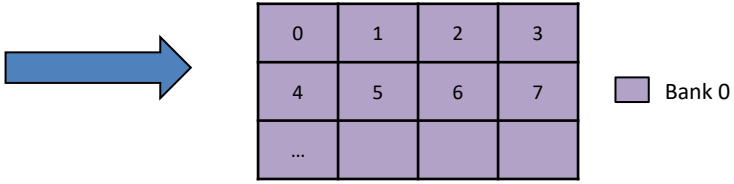
- Compiler creates memory geometry based on how an array is accessed, not how it's declared
- Array could be banked:

```
int lmem[N];
```



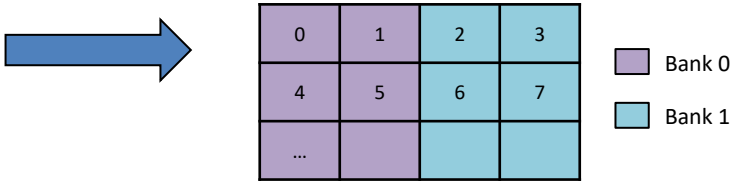
- Coalesced

```
int lmem[N];
```



- Or coalesced and banked:

```
int lmem[N];
```




```
[[intel::numbanks(8), intel::bankwidth(16)]] int lmem[8][4];
#pragma unroll
for (int i = 0; i < 4; i+=2) {
  lmem[i][x & 0x3] = ...;
}
```

HLS, OpenCL, DPC++: Режим эмуляции

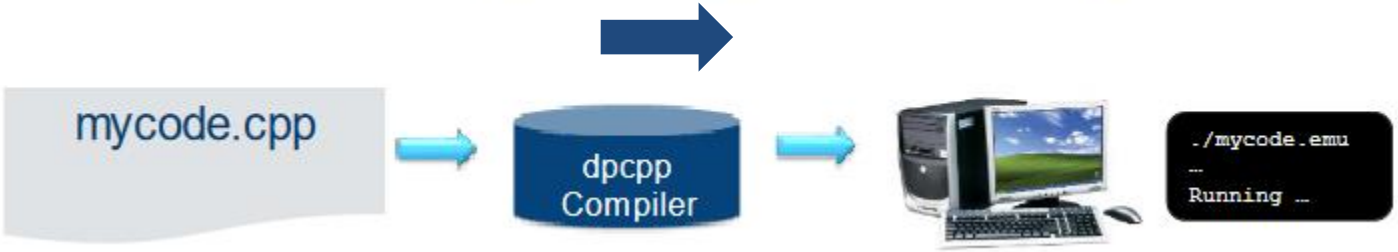
```
component int accelerate(int a, int b) {
            return a+b;
}
```

`accelerate()` becomes an FPGA component

- Use `--component` i++ argument or `component` attribute in source

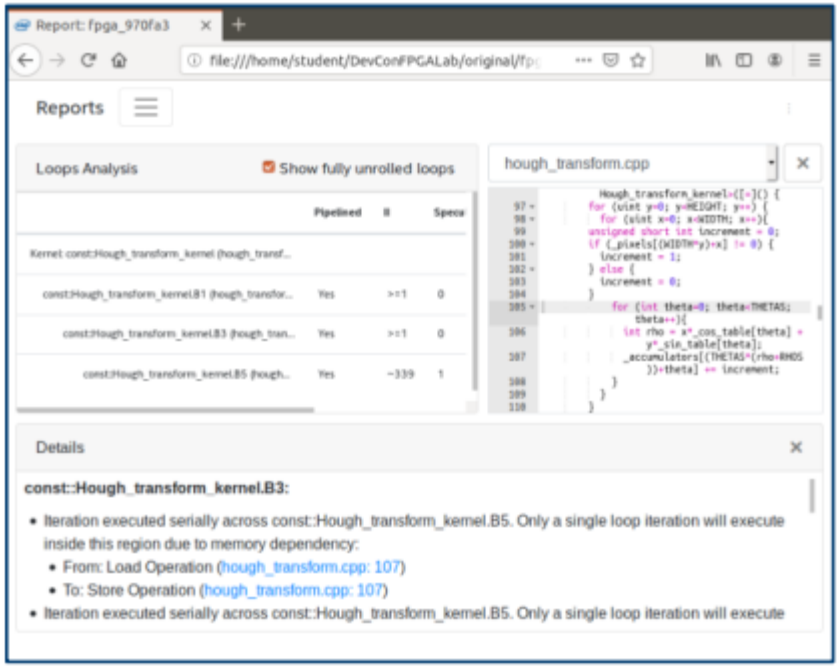
```
i++ -march=<fpga family> --component accelerate mysource.cpp
```

```
dpcpp -fintelfpga <source_file>.cpp -DFPGA_EMULATOR
```



Optimization Report – Throughput Analysis

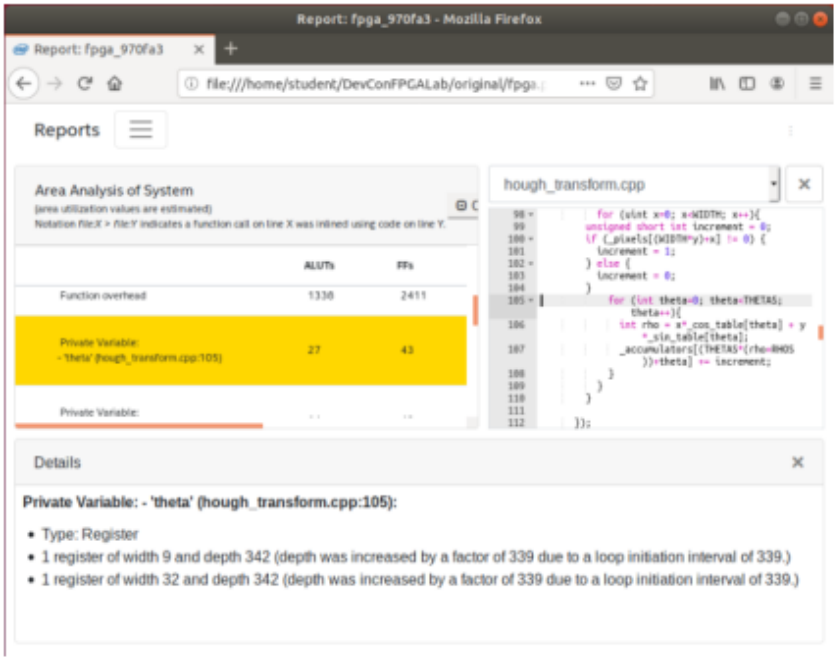
- Loops Analysis and Fmax II sections
- Actionable feedback on pipeline status of loops
- Show estimated Fmax of each loop



Optimization Report – Area Analysis

Generate detailed estimated area utilization report of kernel scope code

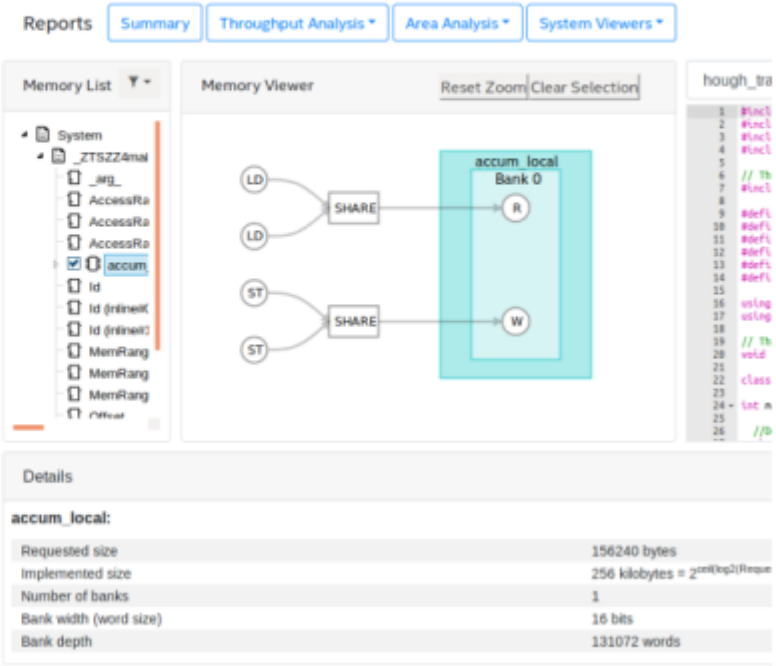
- Detailed breakdown of resources by system blocks
- Provides architectural details of HW
  - Suggestions to resolve inefficiencies



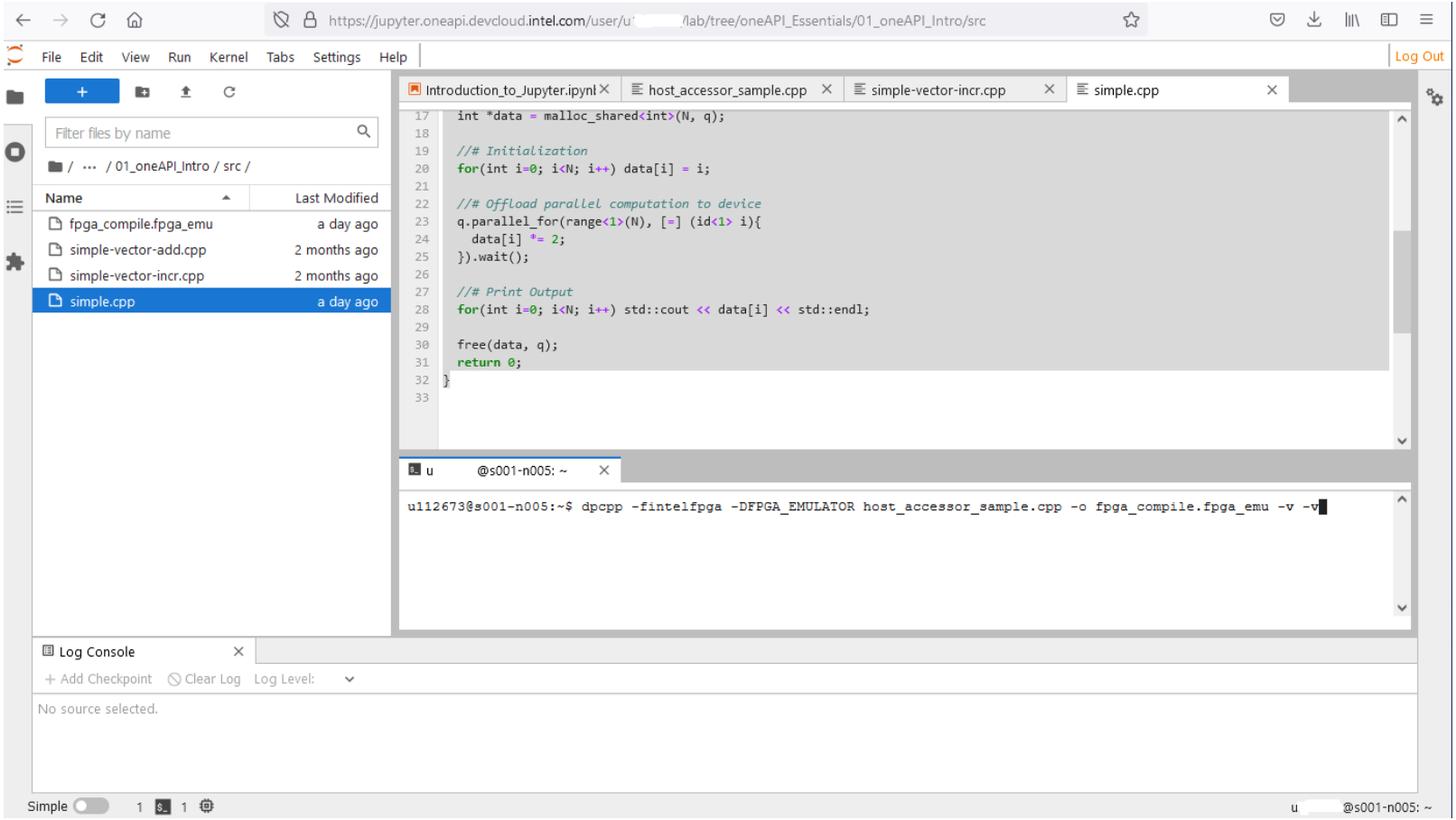
HTML Kernel Memory Viewer

Helps you identify data movement bottlenecks in your kernel design. Illustrates:

- Memory replication
- Banking
- Implemented arbitration
- Read/write capabilities of each memory port



Intel Devcloud: Jupiter Notebooks

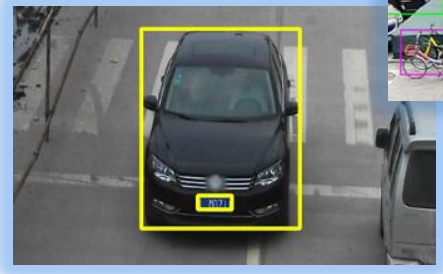


Практические кейсы компании «Акселтех»

- Видеоаналитика на Intel OpenVINO
- DPI и анализ трафика
- eCPRI и OpenRAN
- Кодирование данных: архивирование, перекодирование медиа данных
- Криптографические средства

Intel OpenVINO: видеоаналитика на OpenCL

- Generic Age & Gender Recognition
- Camera Tampering Detection
- Generic Face Detection
- Face Detection for Retail
- Person Detection for Retail
- Face Detection for Automotive
- Person, Vehicle & Bike Detection
- Vehicle License Plate Detection



- License Plate Recognition
- Age & Gender Recognition for Retail
- Vehicle Attributes Recognition
- Head Pose Estimation for Automotive
- Semantic Segmentation
- Road Segmentation
- Person Attributes
- Person Re-identification
- Pedestrian Detection
- Pedestrian & Vehicle Detection
- Emotions Recognition

- Зеленым выделены приоритетно запущенные модели на нашей аппаратуре и готовые к тестированию,
- Оранжевым выделены модели , которые будут скоро доступны для теста

Euler Line – ускоритель для видеоаналитики

Серийно выпускаемый ускоритель EulerProject
Разработка – г. Москва
Производство – г. Москва

- Форм-фактор ½ PCI Express
- 168 mm x 69 mm x 28 mm
- Хост интерфейс
- 8-lane PCI-Express Gen 3.0
- Сетевые интерфейсы
- 2x10GE SFP+ и 1GE (версия NET)
- Два DDR4 контроллера (версия HPC)
- Каждый банк по 8-16ГБ
- ПЛИС Intel Arria-10 FPGA 20nm
- По-умолчанию емкость: GX 1150

Высокопроизводительная FPGA Arria-10

Промышленный диапазон

- До 1.5 TFLOPS
- Срок жизни более 10 лет
- Не требует лицензирования

Поддержка SDK Intel

- Intel FPGA OpenCL SDK
- Quartus Prime HDL
- OpenVINO SDK* возможна поддержка
- DDR4 SDRAM x 72 bits в SODIMM до 16GB
- QDR4 Cypress до 144Мбит (разъем HILO)

EULER PROJECT

intel ARRIA 10 inside

DISCOVER.
DESIGN.
DEVELOP.

yadro.com

Генеральный партнер конференции FPGA-Systems 2021.2

tech@exponenta.ru
exponenta.ru

ЭКСПОНЕНТА

ЦЕНТР ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ
И МОДЕЛИРОВАНИЯ

- Технические **консультации**
- Подбор **инструментов**
- **Обучение** специалистов
- Работа **на заказ**

EREMEX

Генеральный партнёр конференции FPGA-Systems 2021.2

DeltaDesign

Первая современная отечественная САПР, реализующая сквозной цикл проектирования печатных плат

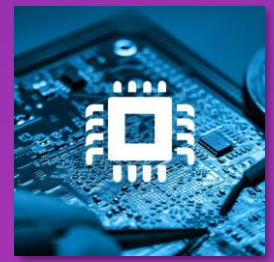
www.eremex.ru

Информационные партнеры

КОНВЕРУМ Сообщество приборостроителей

Русская электроника

ДАЙДЖЕСТ ИНФОРМАЦИОННЫХ РЕСУРСОВ ПО ТЕМАТИКЕ РАДИОЭЛЕКТРОННОЙ ПРОМЫШЛЕННОСТИ



НОВОСТИ МИРА ЭК

iMAXAi

thirdpin

PCBSOFT PCB&IC SOFTWARE

Где найти FPGA комьюнити?

fpga-systems.ru

t.me/fpgasystems <=> @fpgasystems

youtube.com/c/fpgasystems

admin@fpga-systems.ru