Tera Pines

兆松科技（武汉）有限公司

Terapines Technology (Wuhan) Co. , Ltd.

详解开源乘影GPGPU OpenCL编译器技术栈

伍华林

aries.wu@terapines.com

What is OpenCL

OPEN STANDARD FOR PARALLEL PROGRAMMING OF HETEROGENEOUS SYSTEMS

OpenCL Adoption

The industry's most pervasive, cross-vendor, open standard for low-level heterogeneous parallel programming

Desktop Creative Apps

Adobe, otoy, DASSAULT SYSTEMES, SONY, Modo, darktable, AUTODESK, CHAOSGROUP, Vegas Pro, CyberLink, ptc, RADEON PRORENDER, GIMP, ArcSoft, Blackmagicdesign, Capture One, REALFLOW, SILHOUETTE, LuxCoreRender, SideFX, acdsee

Parallel Languages

SYCL, OpenACC DIRECTIVES FOR ACCELERATORS, aparapi, PyOpenCL

Linear Algebra Libraries

SYCL-BLAS, viennaCL, CLBlast

Machine Learning Libraries and Frameworks

Intel OPENVINO, TF, Intel clDNN, Xiaomi MACE, SYCL-DNN, Caffe, NNAPI, Acuity VeriSilicon, Qualcomm Neural Processing SDK for AI, Synopsis MetaWare EV, TI DL Library (TIDL), Arm Compute Library

Molecular Modelling Libraries

siremol.org, FOLDING@HOME, OpenMM, CHARMM Chemistry at HARvard Macromolecular Mechanics, ForceBalance, GROMACS FAST. FLEXIBLE. FREE.

Machine Learning Compilers

XLA, tvm.ai, GLOW, plaidML

Vision, Imaging and Video Libraries

FAST, OpenVX, Halide, VisionCpp, Metashape, OpenCV, FFmpeg

Math and Physics Libraries

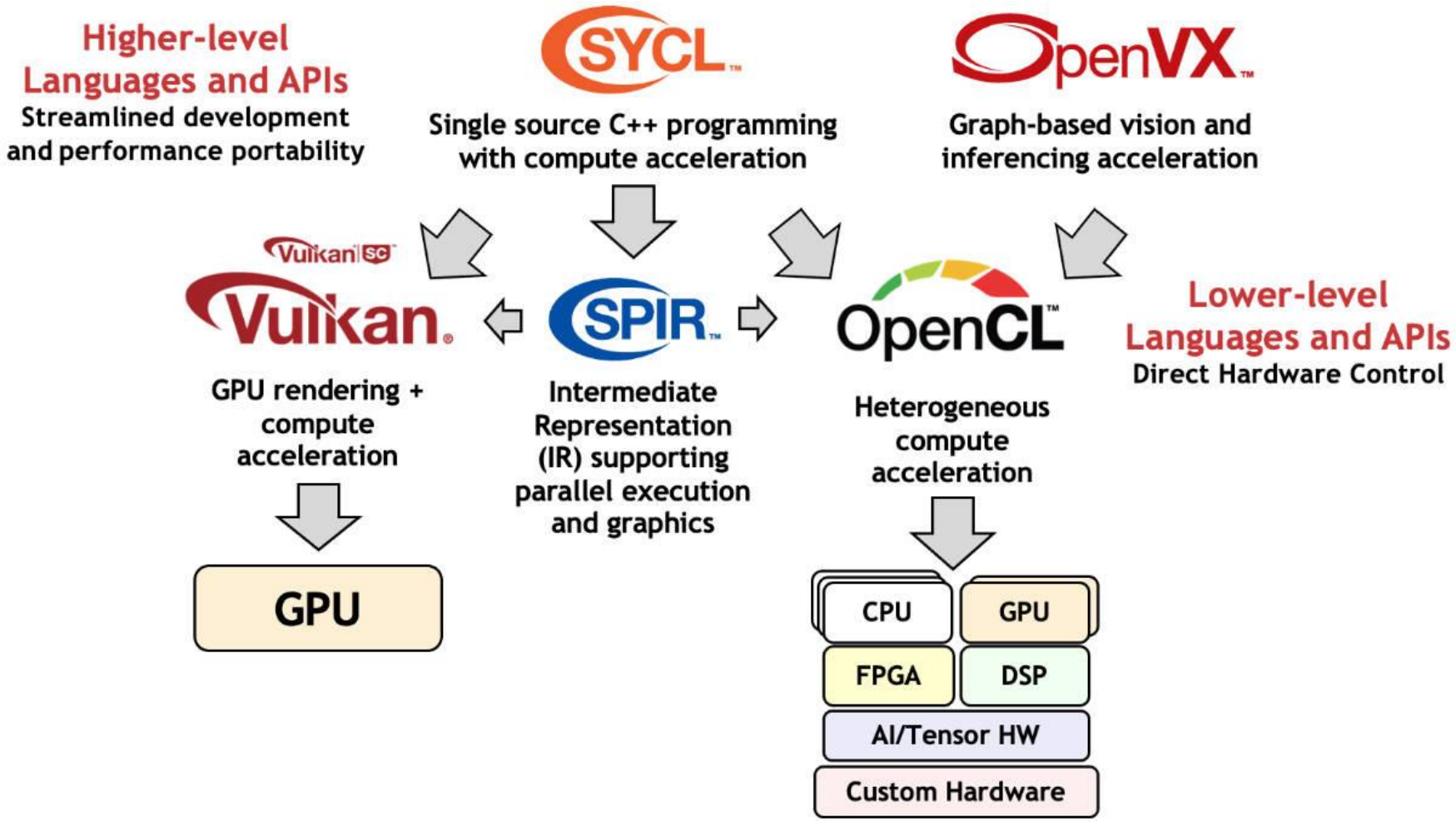
BULLET PHYSICS LIBRARY, GNU Octave, Wolfram Mathematica, ArrayFire C, C++, Fortran, Matlab

---

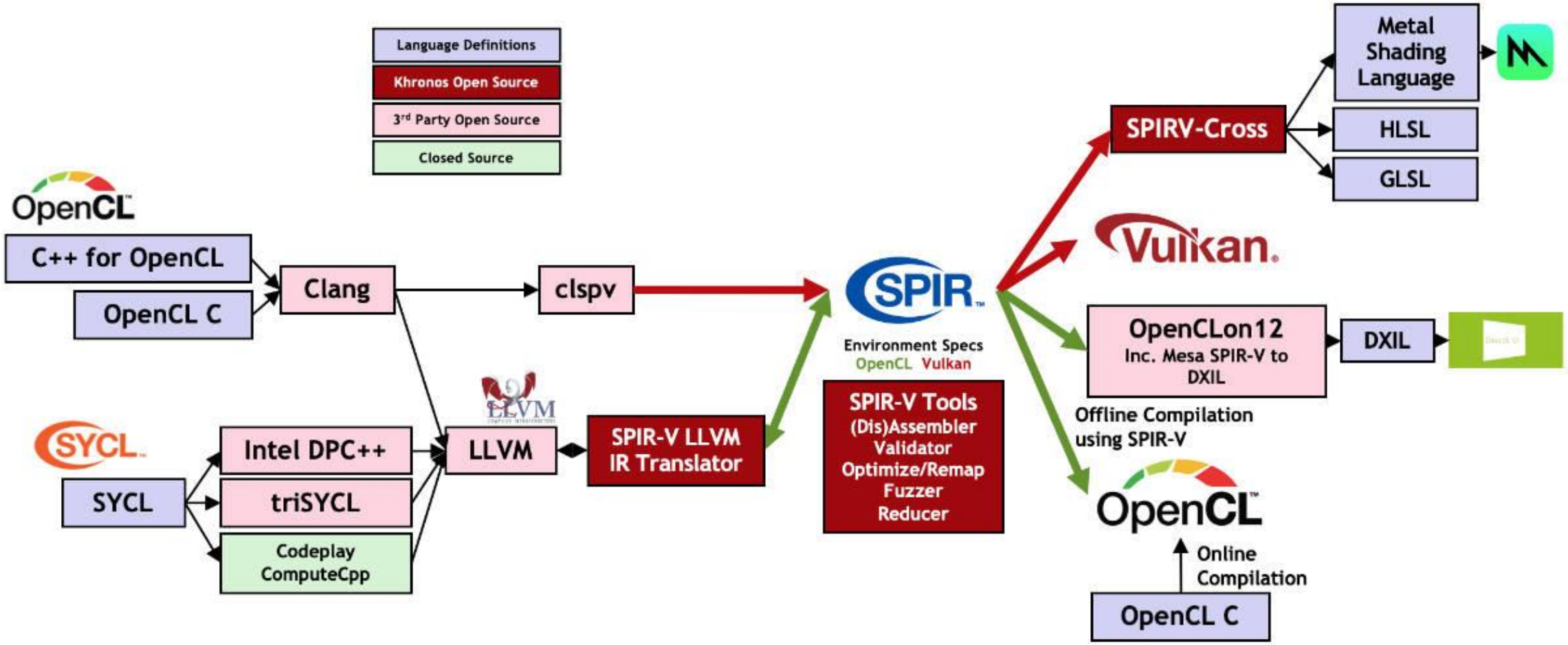
Apple, ALTERA, AMD, arm, OpenCL, Imagination, intel, KALRAY, NVIDIA, MARVELL, MEDIATEK, Qualcomm, Google, SAMSUNG, ST, Texas Instruments, VeriSilicon, XILINX

TeraPines 兆松科技

OpenCL Overview



OpenCL Overview



OpenCL Envolving

- OpenCL 1.x
- OpenCL 2.0
  - Shared Virtual Memory
  - Device Side Enqueue
  - General Address Space
  - Enhanced Image Type and Pipe
  - Enhanced Atomic Operations
- OpenCL 3.0
  - Emphasizes a return to the core principles of OpenCL 1.2 by making all features from versions 2.x optional.
  - Interoperability with other APIs like Vulkan

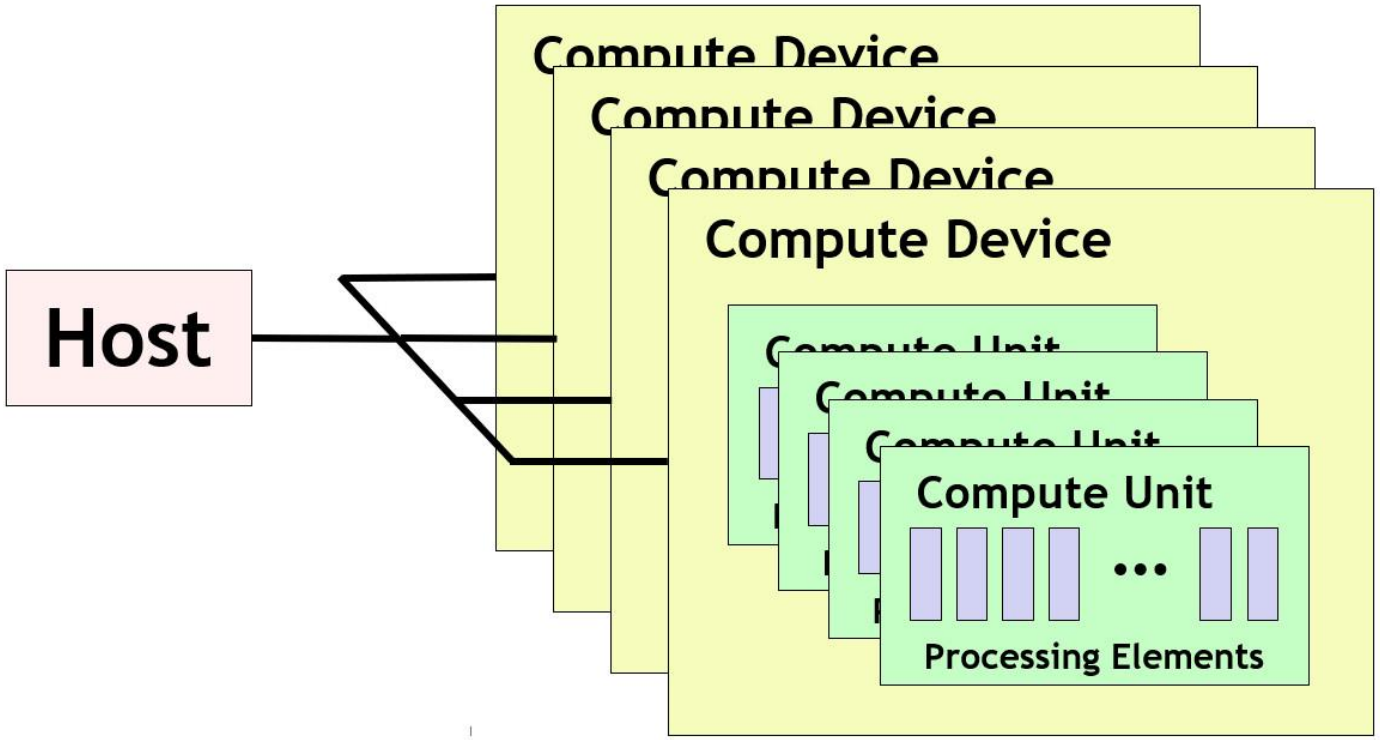
The process to claim OpenCL conformant

- Implement the OpenCL Specification
- Use the Conformance Test Suite
- Submit Results to Khronos
- Khronos Review and Approval
- Conformance Statement and Use of Logo
- Maintain Compliance

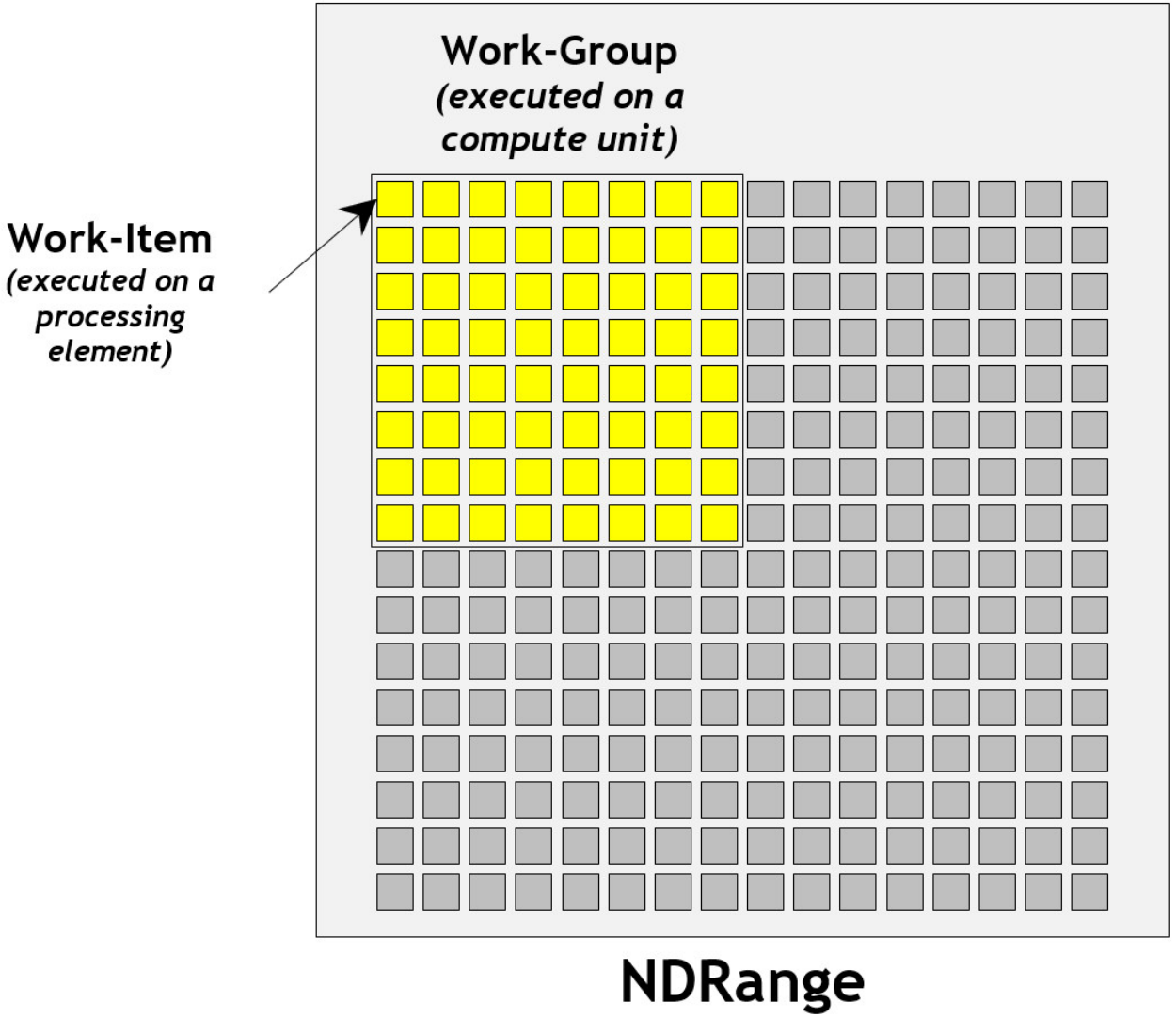
乘影GPGPU OpenCL Software Stack

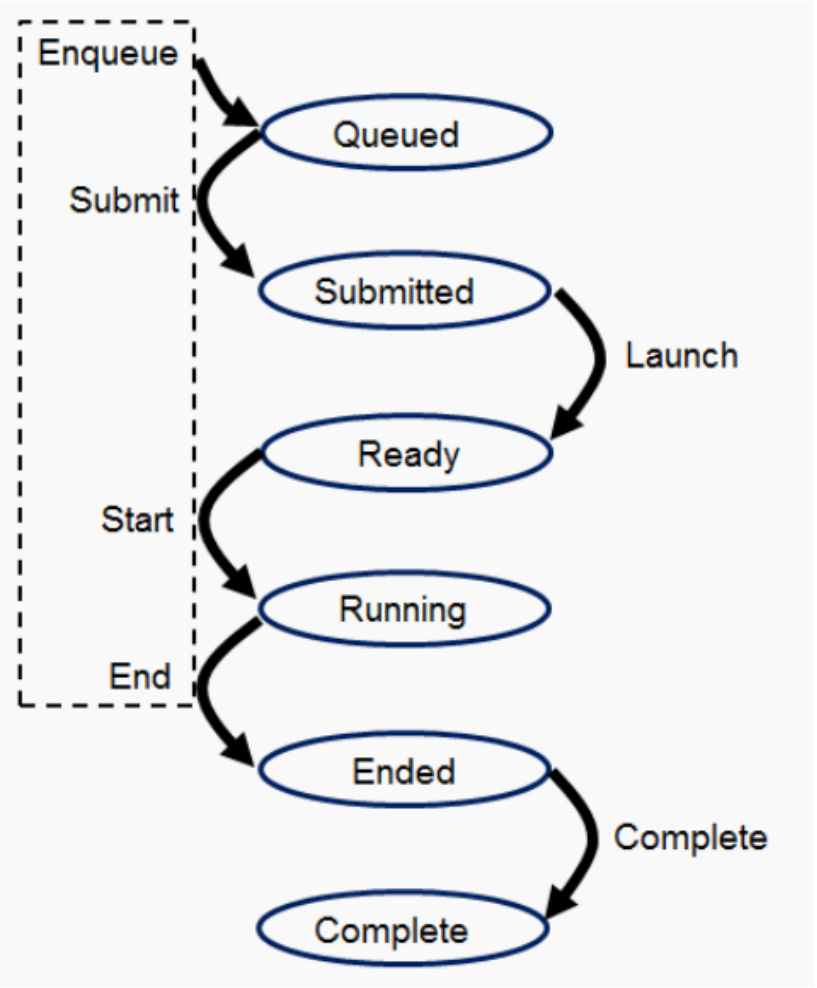
- OpenCL Driver – POCL
  - Implementation of OpenCL API
- OpenCL Compiler – LLVM
  - Implementation of OpenCL C language compiler
  - Managed by OpenCL driver, invisible to end user
- OpenCL library – libclc
  - Implementation of workitem and kernel builtin functions
- Kernel Mode Driver
  - The glue layer between POCL and Ventus GPGPU

OpenCL Platform Model

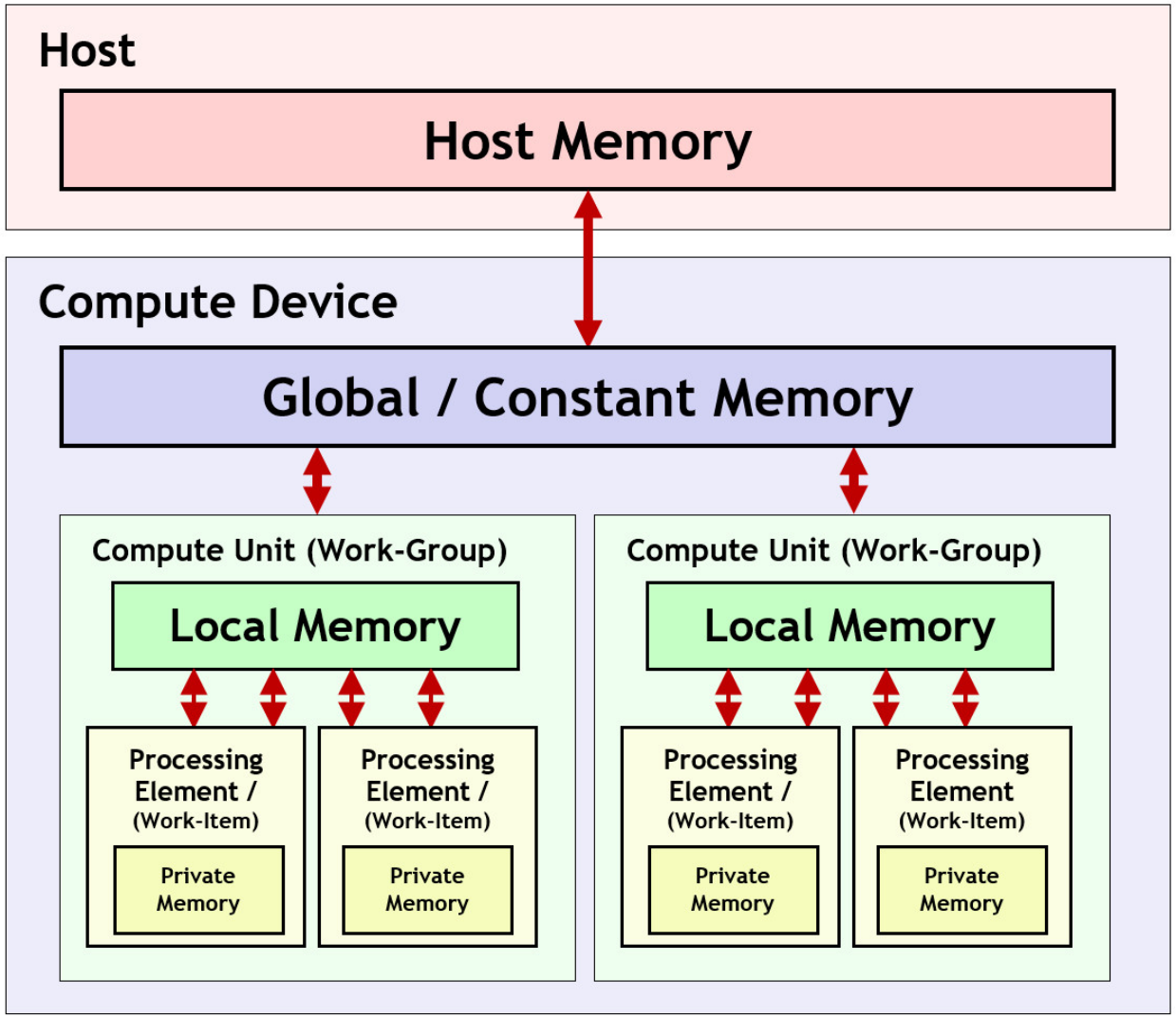


OpenCL Execution Model





OpenCL Memory Model



OpenCL Programming Model

```
void vectorAdd(int *A, int *B, int size) {
    for(int i = 0 ; i < size ; i++) {
        A[i] += B[i];
    }
}
int A[1024];
int B[1024];
vectorAdd(A, B, 1024);
```

C code

```
_kernel void vectorAdd(global int *A, global int *B) {
    int gid = get_global_id(0);
    A[gid] += B[gid];
}
```

get_global_id: work-item builtin gets total number of data at x/y/z directions

OpenCL code

OpenCL Programming Model

```
cl_int clEnqueueNDRangeKernel (
  cl_command_queue command_queue,
  cl_kernel kernel,
  cl_uint work_dim,
  const size_t *global_work_offset,
  const size_t *global_work_size,
  const size_t *local_work_size,
  cl_uint num_events_in_wait_list,
  const cl_event *event_wait_list,
  cl_event *event)
```

Work-Item Built-in Functions [C 6.13.1]

Query the number of dimensions, global, and local work size spec identifier of each work-item when this kernel is executed on a dev __opencl_c_subgroups.

uint **get_work_dim** ()	Number of dimensions in use
size_t **get_global_size** (uint *dimindx*)	Number of global work-items
size_t **get_global_id** (uint *dimindx*)	Global work-item ID value
size_t **get_local_size** (uint *dimindx*)	Number of work-items in the work-group
size_t **get_enqueued_local_size** (uint *dimindx*)	Number of work-items in a uniform work-group
size_t **get_local_id** (uint *dimindx*)	Local work-item ID
size_t **get_num_groups** (uint *dimindx*)	Number of work-groups
size_t **get_group_id** (uint *dimindx*)	Work-group ID
size_t **get_global_offset** (uint *dimindx*)	Global offset

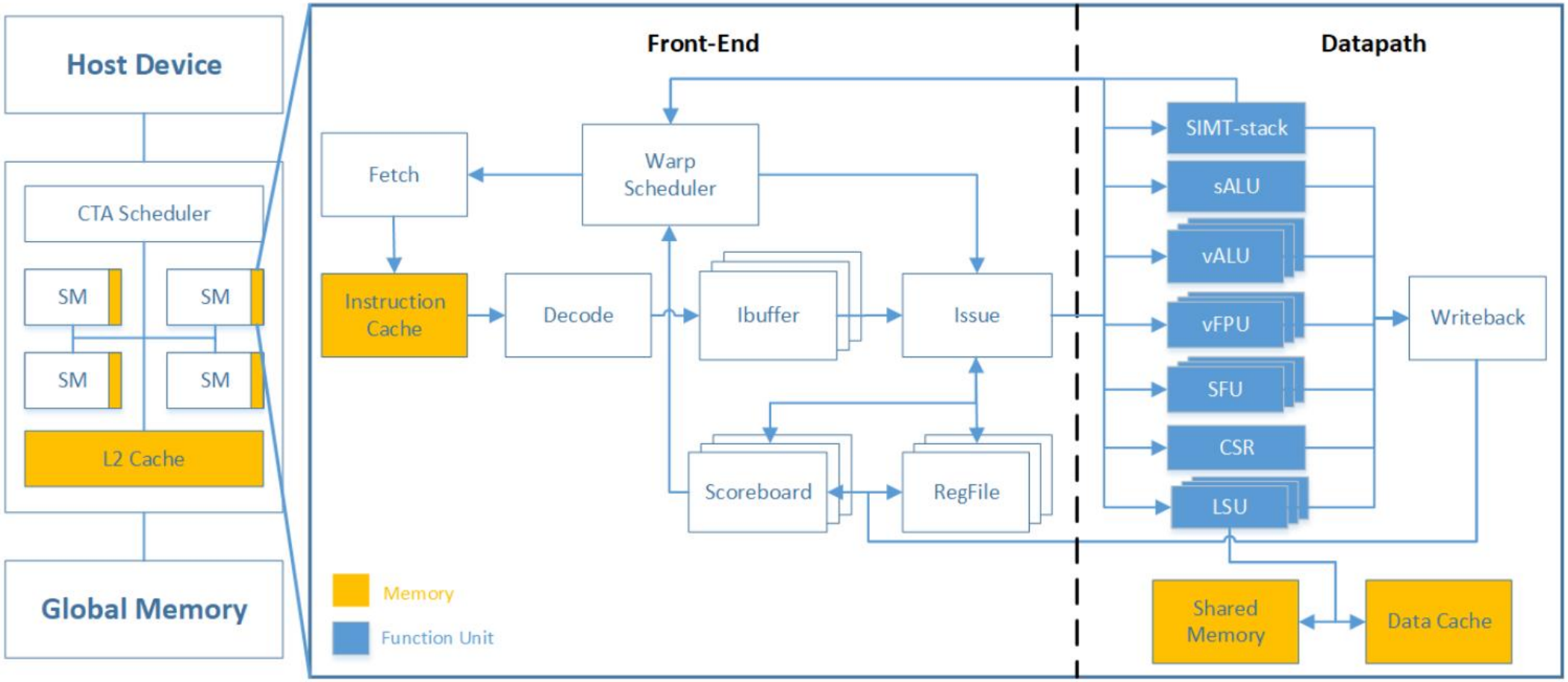
vectorAdd as example

Constant to all threads:
get_work_dim -> 1
get_global_size -> 1024
get_local_size -> 16
get_num_groups -> 1024/16
get_global_offset -> 0

Different for each threads:
get_global_id: 0~1023 for each work item
get_local_id: 0~15 for each work item in every work-group
get_group_id: 0~63 for each work group(minimal scheduling unit in compute device)

Note: Think dimension as 1, 2, 3 dimensional loops in C

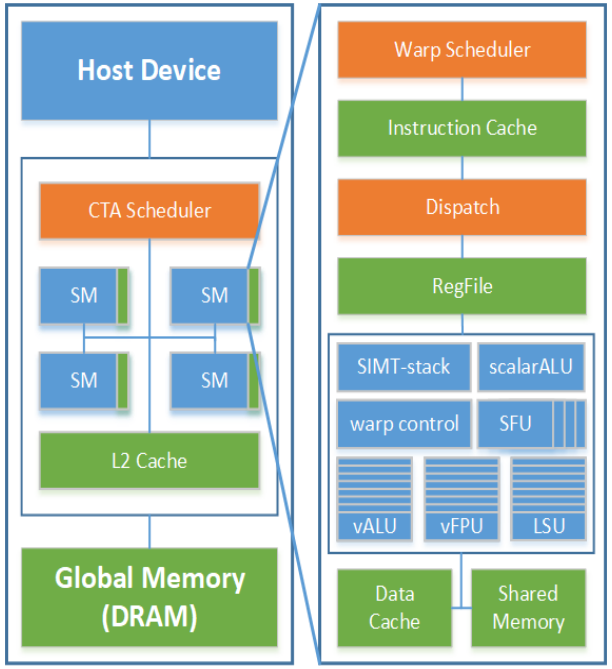
Ventus GPGPU Microarchitecture



https://github.com/THU-DSP-LAB/ventus-gpgpu

Ventus GPGPU Microarchitecture

- GPGPU programming model
  - Grid - block(CTA) - thread
  - Programmer need to declare thread numbers and describe action for every single thread to accomplish parallel programming
  - Hardware organizes the threads in the block and maps them to the hardware for execution in warp units
  - Task assignment : Host device -> CTA-scheduler -> SM(streaming processor)
- Task execution
  - Each SM can be considered a RISC-V processor with multi-warp scheduling support
  - Each warp can be considered as an RVV program, which is fetched, decoded, fired, executed and written back to a register
  - Multiple warp can be scheduled based on LocalMem/PrivateMem/VGPR/SGPR usage



Ventus GPGPU Microarchitecture - ISA

- Base ISA： RV32IMAVZfinx
- Customized Instruction extension highlights
  - barrier - Corresponding to OpenCL's barrier builtins, to achieve data synchronization between threads within the same block
  - endprg - Explicitly inserted at the end of the kernel, indicating the end of the current warp execution
  - vbeq/join： Implicit SIMT-stack for branch control
  - regext{i}： Expand the registers and imm operand encoding bits
  - vlw.v/vsw.v : Private memory access
  - vftta.vv : convolution
  - …

Ventus GPGPU Microarchitecture - Registers

- Architecture registers : 64 sGPR, 256 vGPR, 32bit width. use register pair to store 64bit width data
- Physical registers : 256 sGPR , 1024 vGPR , mapping of architectural registers to physical registers are implemented by hardware
- Each warp has 32 sGPR, 32 vGPR, the width of vGPR is 32bit*num_thread

- Vector registers (VGPR): for everything that has values that are diverging between threads in a wave. Most of your local variables will probably end up in VGPRs
- Scalar registers (SGPR): everything that is guaranteed to have the same values for all threads in a wave will be put in these. An easy example are values coming from constant buffers

Ventus GPGPU Microarchitecture – ABI & CSR

- For kernel functions， a0 is the baseaddr for arguments list, The first clSetKernelArg sets the starting address of the video memory into the a0 register, and the kernel starts loading parameters from a0 register by default
- For non-kernel functions, use v0-v31 and stack pointer to pass parameters, a0-a7 for private memory address parameters, v0-v15 to return values

name	addr	description
CSR_TID	0X800	The smallest thread id in the warp
CSR_NUMW	0X801	Total warp numbers in a workgroup
CSR_NUMT	0X802	Total thread numbers in a warp
CSR_KNL	0X803	The baseaddr of metadata buffer for workgroup
CSR_WID	0X805	The warp id in workgroup, first n-bit is the workgroup id
CSR_LDS	0X806	The baseaddr for allocated local memory in workgroup， and also the baseaddr for stack
CSR_GIDX	0X808	The x id for workgroup in NDRange
CSR_GIDY	0X809	The y id for workgroup in NDRange
CSR_GIDZ	0XC80a	The z id for workgroup in NDRange

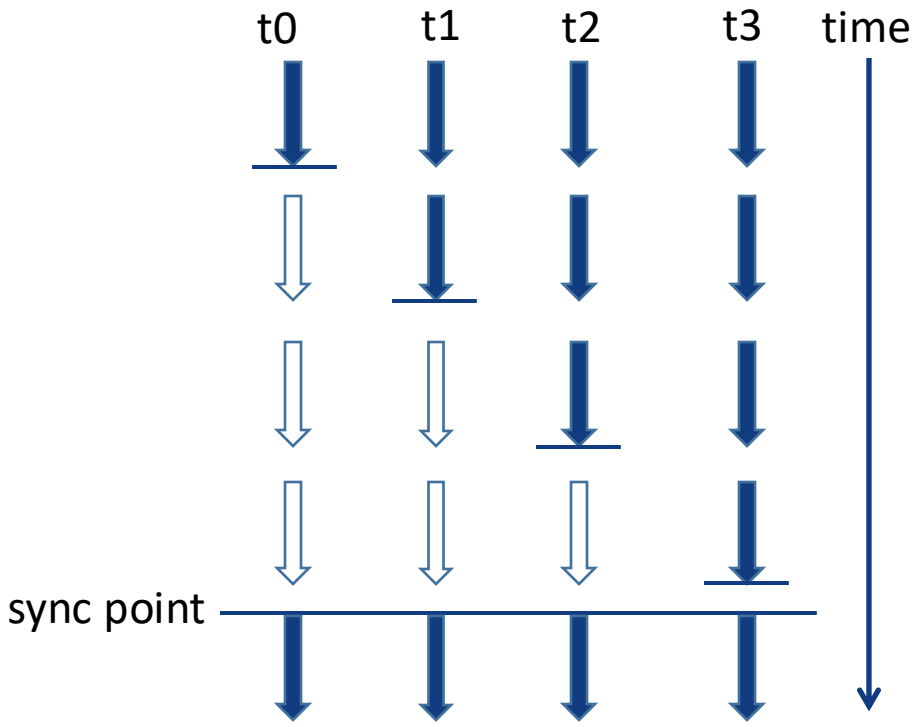
Ventus GPGPU Microarchitecture - Memory Model

- No MMU in hardware, addressing space is explicitly managed by software
- Base addresses of local/private memory is configured in CSRs
- Global memory - shared by all workgroups
- Local memory - shared by all workitems in a warp(workgroup)
- Private memory – private to a single workitem, vlw.v, vsw.v
- Memory load/store with divergence offset - vluxei.v vd, (rs1), vs2
- Memory load/store with imm offset – vlw rd, (rs1), imm11

Ventus GPGPU Microarchitecture - Data Synchronization

Threads 0 ~ 3 (in a warp) execute:

```
vsoxei32.v // Store to local memory
barrier    // Sychronization inserted explicitly by user
vloxei32.v // Load from local memory
```



Implementation in Ventus GPGPU – Kernel arguments passing

```
* kernel metadata buffer layout:
* +-------4---------+----------4----------+-----4----+-------4-------+-------4------
* | kernel func ptr | kernel arg base ptr | work_dim | global_size_x | global_size_y
* +-------4-------+------4-------+------4-------+------4-------+-------4---------
* | global_size_z | local_size_x | local_size_y | local_size_z | global_offset_x
* +--------4--------+--------4--------+----
* | global_offset_y | global_offset_z | ...
*/
```

```
*
* kernel arg buffer:
* +-------+-------+--------+-----
* | arg_0 | arg_1 |  arg_2 | ...
*/
```

Implementation in Ventus GPGPU - Divergency

- Non divergency code should be executed in sALU, all others should be executed in vALU.
- Source of divergence
  - Diverged control flows
  - get_global_id, get_local_id, load from private stack, function calls …
  - LegacyDivergenceAnalysis pass
  - RISCVTargetLowering::isSDNodeSourceOfDivergence()

Workitem and builtin functions

- Implemented in libclc under ventus-llvm compiler repo
- Builtin function categories
  - Relational
  - Geometric
  - Vector Data Load/Store
  - Memory Fence
  - Async Copies and Prefetch
  - Atomic
  - Printf
  - Workgroups
  - Pipe
  - Kernel enqueue
  - Images

Summary

- OpenCL is low level API to program accelerators
- OpenCL is widely adopted
- Ventus GPGPU supports OpenCL 2.0
- Ventus GPGPU ISA is based on RISC-V but a SIMT architecture
- Ventus GPGPU is currently going through OpenCL CTS 2.0

Thanks