Portable Parallelization Performance

Where stands SYCL in this landscape ?

Gray-Scott Battle
David Chamont, july 2023

OpenCL (2008)

vectorization + acceleration portable
across CPU, GPU, FPGA ... damned !?!

Gray-Scott Battle
David Chamont, july 2023

OpenCL History

- 2008: OpenCL announced by Apple
- 2009: OpenCL 1.0 released by the Khronos Group
- 2010: OpenCL 1.1, new features
- 2011: OpenCL 1.2, expanded language support
- 2013: OpenCL 2.0, image types and pipes
- 2015: OpenCL 2.1, better performance and productivity
- 2017: OpenCL 2.2, additional features and bug fixes
- 2020: OpenCL 3.0, unified model for CPU and GPU

OpenCL example / kernel

```
#include <CL/opencl.h>

.....

const char * KernelSource = "                    \n"
"  __kernel void square(                         \n"
"    __global float* input,                      \n"
"    __global float* output,                     \n"
"    const unsigned int count)                   \n"
"  {                                             \n"
"    int i = get_global_id(0);                   \n"
"    if(i < count)                               \n"
"      output[i] = input[i]*input[i];             \n"
"  }                                             \n"
"                                                \n";

.....
```

OpenCL example / main

```
.....

int main(int argc, char** argv)
{
  ..... create and initialize host input/output arrays .....

  // Prepare kernel
  cl_int err;
  cl_platform_id platform_id;
  err = clGetPlatformIDs(1, &platform_id, NULL);
  cl_device_id device_id ;
  err = clGetDeviceIDs(platform_id,CL_DEVICE_TYPE_GPU,1,&device_id,NULL);
  cl_context context = clCreateContext(0,1,&device_id,NULL,NULL,&err);
  cl_command_queue queue = clCreateCommandQueue(context,device_id,0,&err);
  cl_program prog = clCreateProgramWithSource(context,1,(const char **)&KernelSource,...
  err = clBuildProgram(prog,0,NULL,NULL,NULL,NULL);
  cl_kernel kernel = clCreateKernel(prog,"square",&err);

  // Device io arrays
  cl_mem input_buffer = clCreateBuffer(context,CL_MEM_READ_ONLY,sizeof(float)*count,...
  cl_mem output_buffer = clCreateBuffer(context,CL_MEM_WRITE_ONLY,sizeof(float)*count,...

  .....
```

OpenCL example / main

```
.....

// Write our data set into the input array in device memory
err = clEnqueueWriteBuffer(queue,input_buffer,CL_TRUE,0,sizeof(float)*count,input,0,...

// Set arguments to kernel
err  = clSetKernelArg(kernel,0,sizeof(cl_mem),&input_buffer);
err |= clSetKernelArg(kernel,1,sizeof(cl_mem),&output_buffer);
err |= clSetKernelArg(kernel,2,sizeof(unsigned int),&count);

// Queue kernel and wait for its execution end
size_t global = count;
err = clEnqueueNDRangeKernel(queue,kernel,1,NULL,&global,NULL,0,NULL,NULL);
clFinish(commands);

// Read back the results from the device to verify the output
err = clEnqueueReadBuffer(queue,output_buffer,CL_TRUE,0,sizeof(float)*count,output,0,...

.....
```

OpenCL example / main

```
.....

// Process results
.....

// Shutdown and cleanup
clReleaseMemObject(input);
clReleaseMemObject(output);
clReleaseProgram(prog);
clReleaseKernel(kernel);
clReleaseCommandQueue(commands);
clReleaseContext(context);
return 0 ;
}
```

SYCL (2013)

rise to C++

Gray-Scott Battle
David Chamont, july 2023

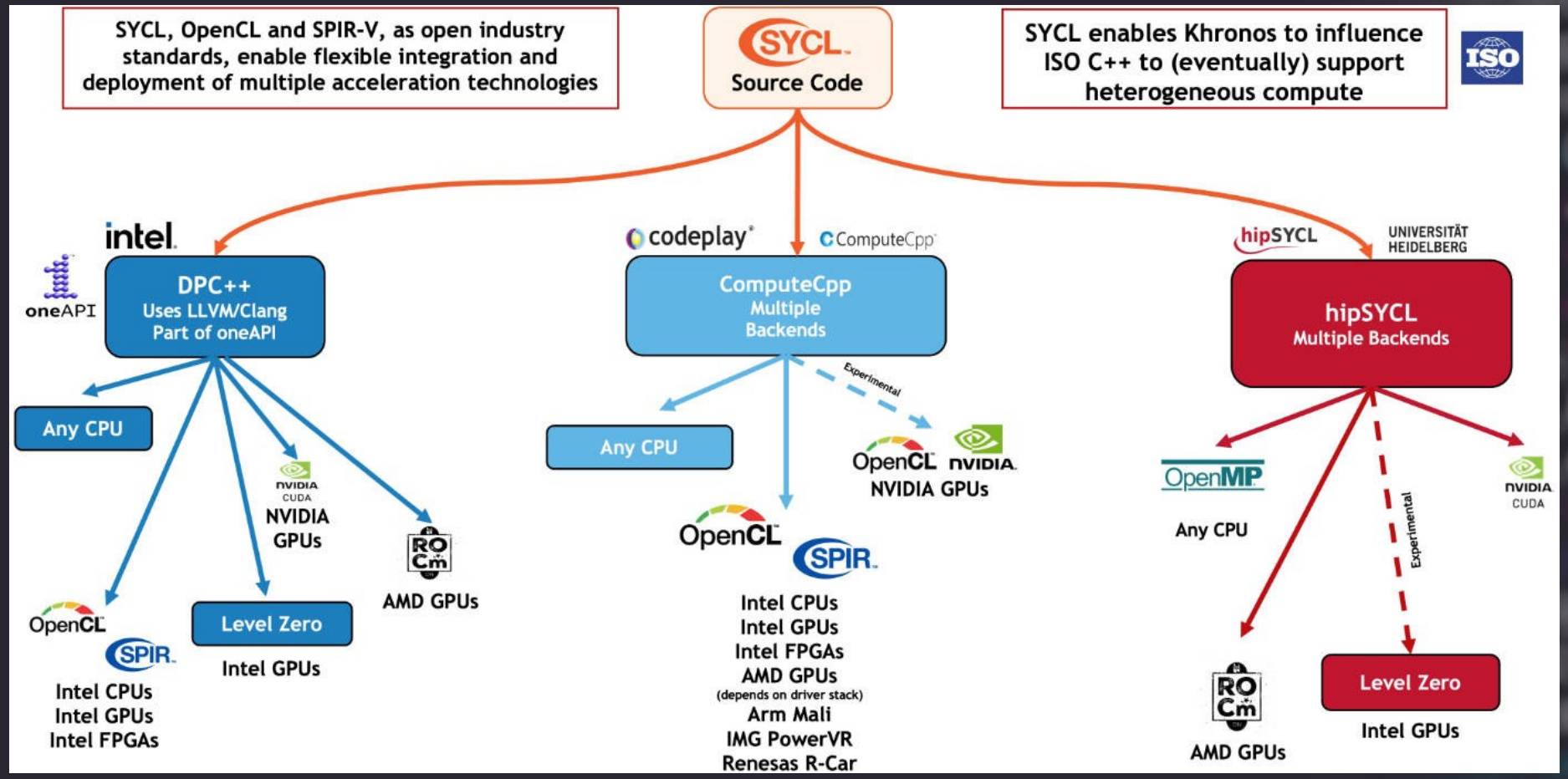
SYCL History

- 2013: initial proposal from Codeplay Software
- 2015: SYCL 1.2, released by the Khronos Group
- 2016: integrated into Intel FPGA SDK for OpenCL
- 2016: SYCL 2.2, target C++14 and OpenCL 2.2
- 2018: Xilinx announced support for SyCL
- 2020: SYCL 2020, open the door for CUDA backend
- 2022: SYCL 2020 rev 6, target C++17, OpenCL 3.0 and IA
- Next: target C++20, *and converge with C++29 ?*

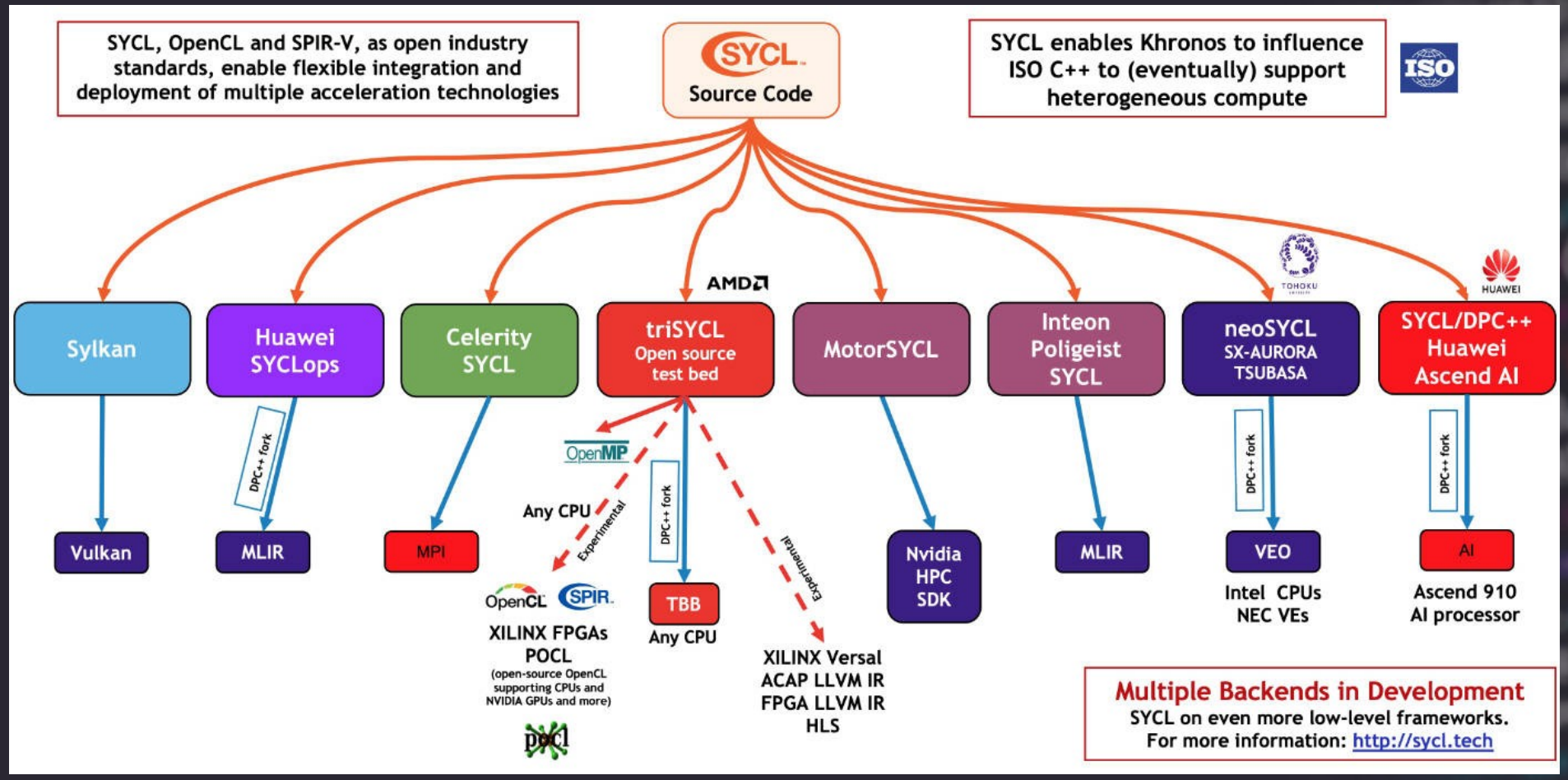
SYCL example

```
#include <sycl.hpp>
using namespace cl::sycl;

.....

    queue q ;


    buffer inputBuff(input) ;
    buffer outputBuff(output);

    q.submit( [&](handler& cgh){

        accessor inputAcc(inputBuff, cgh, read_only);
        accessor outputAcc(outputBuff, cgh, write_only);

        cgh.parallel_for<class square_kernel>( range<1>(SIZE), [=](id<1> idx) {

            outputAcc[idx] = inputAcc[idx] * inputAcc[idx];

        } );

    } );

    q.wait();

.....
```

SYCL implems & backends



SYCL implems & backends



OneAPI (2018)

the Intel touch

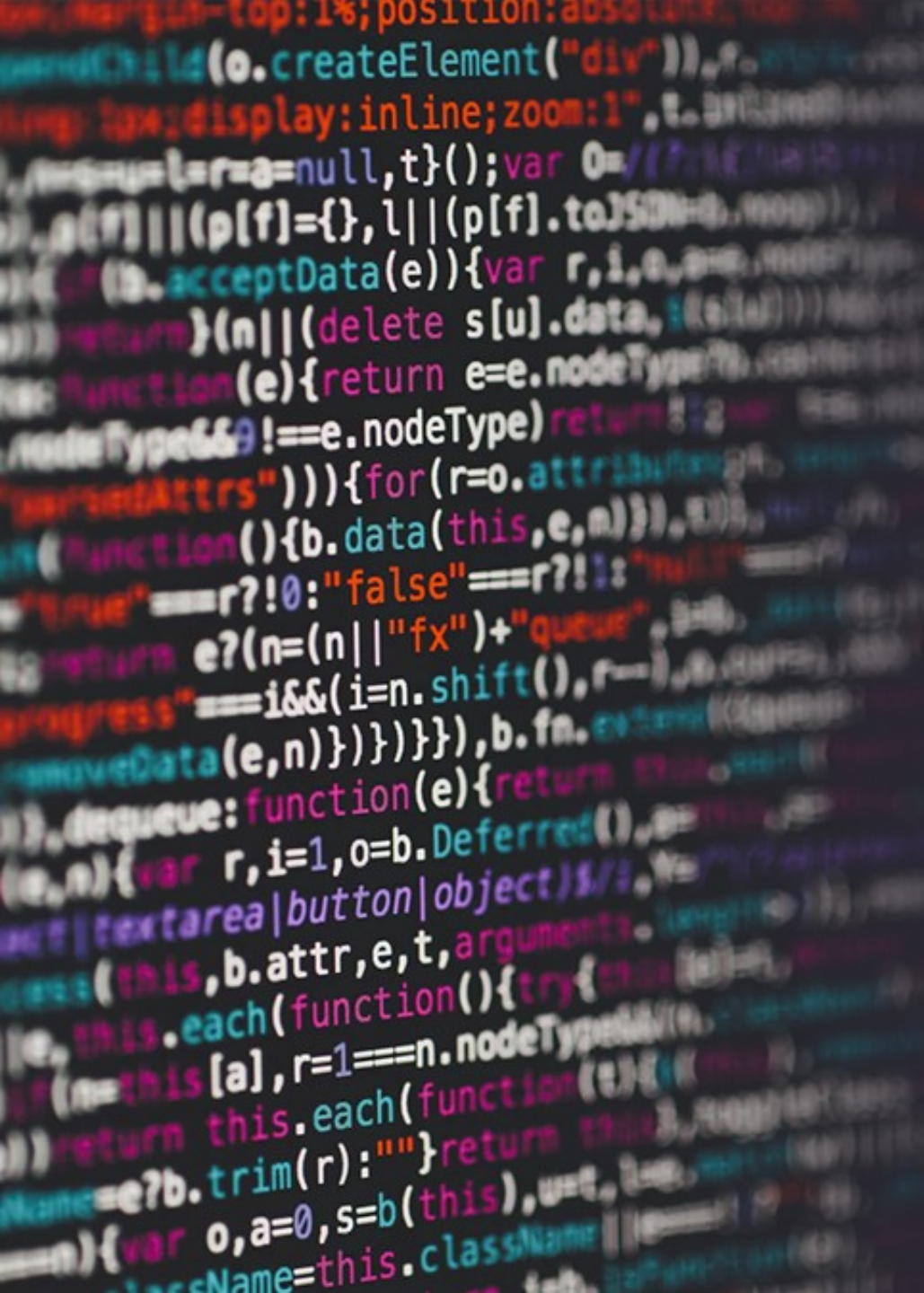
Gray-Scott Battle
David Chamont, july 2023

oneAPI History

- 2018: oneAPI announced at the Intel Developer Forum
- 2019: Base Toolkit released
- 2020: Intel
  - launches oneAPI devcloud
  - introduces Level Zero API
  - opens the oneAPI specification

oneAPI ecosystem

- Level Zero: the low-level hardware interface
- DPC++: an implementation of SYCL with extensions (powered by LLVM)
- SYCLomatic: CUDA-to-SYCL code migration tool
- oneDPL (algorithms), oneMKL (math), oneDAL (machine learning), oneDNN (deep learning), oneCCL (communication), oneTBB (threads), oneVPL (video), ...



And yet… (2023)

… in high energy physics, no consensus on some common portable parallelization solution

*(following information courtesy of the **HEP-CCE** project)*

Gray-Scott Battle
David Chamont, july 2023

Hardware vs Solutions (2019)

	CUDA	Kokkos	SYCL	HIP	OpenMP	alpaka	std::par
NVIDIA GPU			*codeplay*	*hipcc*			*nvc++*
AMD GPU			*hipSYCL*	*hipcc*			
Intel GPU			*oneAPI*				*oneAPI:dpl*
x86 CPU			*oneAPI*				*gcc*
FPGA							

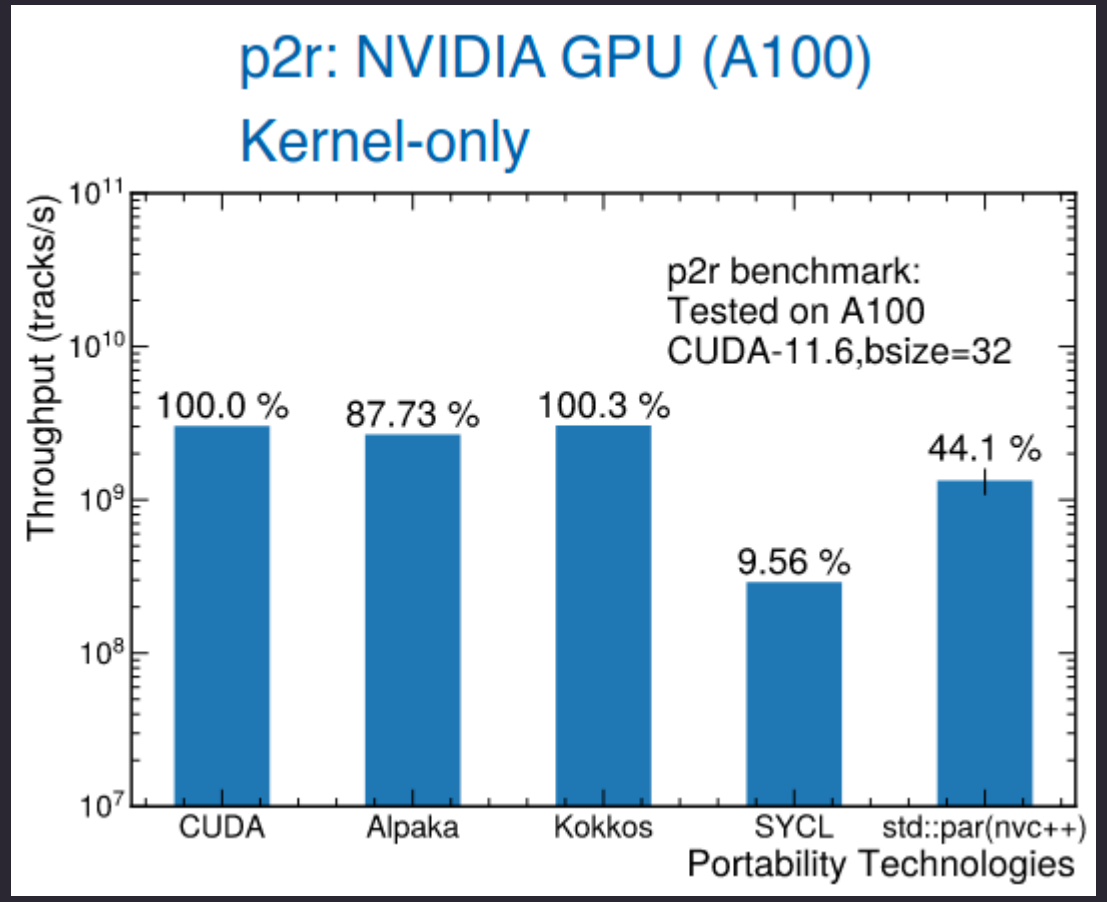
Hardware vs Solutions (2023)

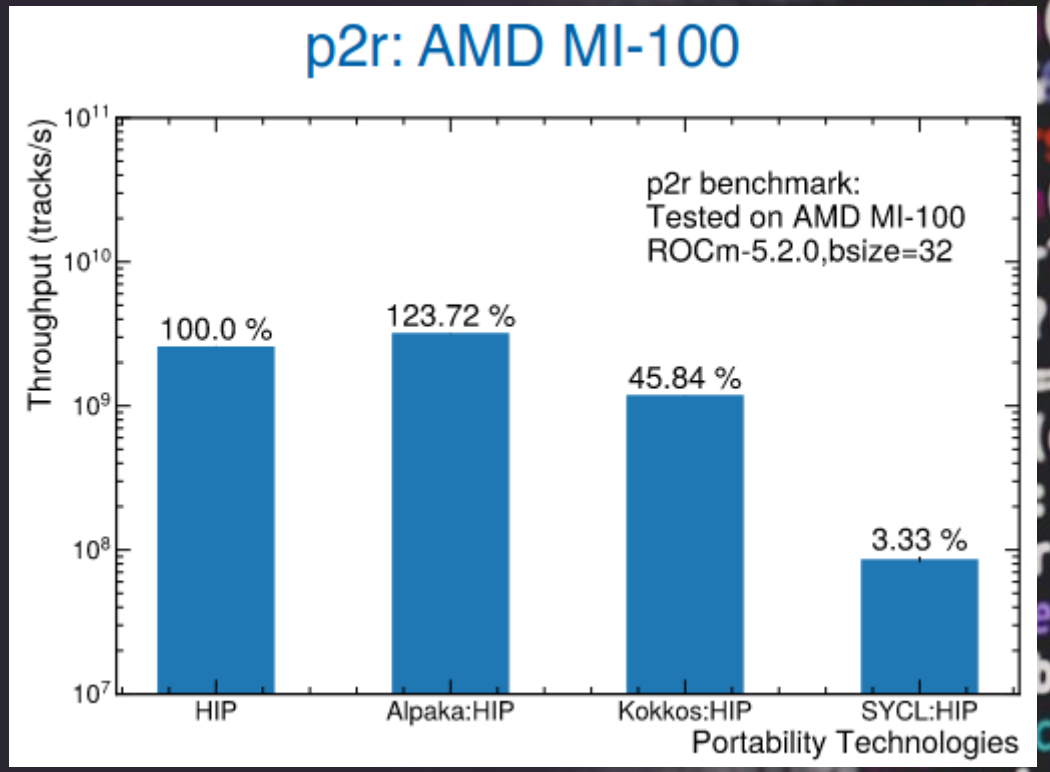
	CUDA	Kokkos	SYCL	HIP	OpenMP	alpaka	std::par
NVIDIA GPU				*hipcc*	*nvc++ LLVM, Cray GCC, XL*		*nvc++*
AMD GPU			*openSYCL intel/llvm*	*hipcc*	*AOMP LLVM Cray*		
Intel GPU			*oneAPI intel/llvm*	CHIP-SPV: *early prototype*	*Intel OneAPI compiler*	*prototype*	*oneapi::dpl*
x86 CPU			*oneAPI intel/llvm openSYCL*	*via HIP-CPU Runtime*	*nvc++ LLVM, CCE, GCC, XL*		
FPGA				*via Xilinx Runtime*	*prototype compilers (OpenArc, Intel, etc.)*	*protytype via SYCL*	

Applications vs Solutions

	Kokkos	SYCL	OpenMP	Alpaka	std::par
Patatrack	Done	Done*	WIP	Done*	Done compiler bugs
Wirecell	Done	Done	Done	no	Done
FastCaloSim	Done	Done	Done	Done	Done
P2R	done	Done	OpenACC	Done	Done

SyCL p2r poor performance





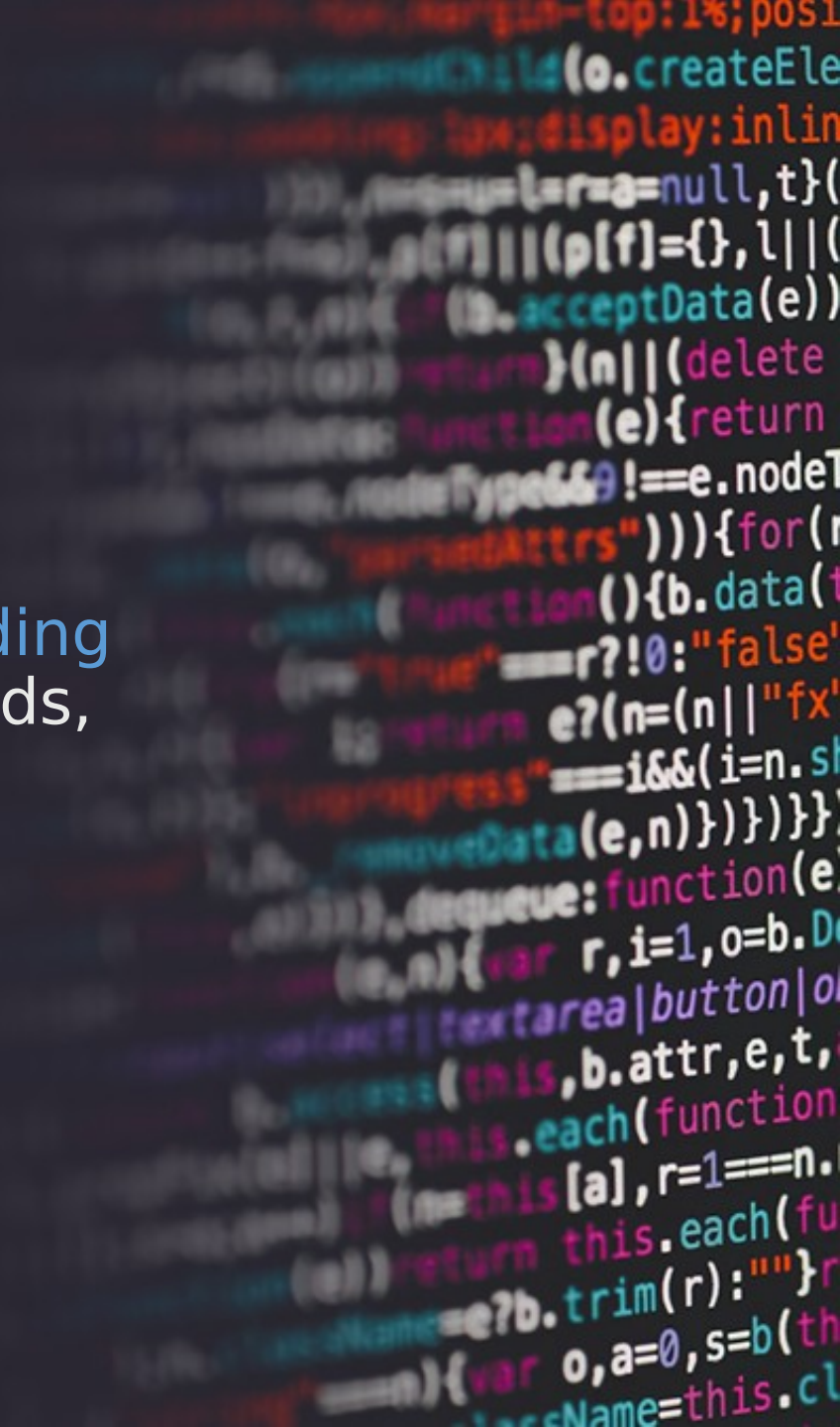
Conclusions

Gray-Scott Battle

David Chamont, july 2023

Bad News

- SYCL brings functional portability, but no implementation has yet proved performance portability.
- We can only dream of generic installation & building portability of the many implementations, backends, plugins…
- The same for generic tooling for debugging and profiling, which is far beyond CUDA ecosystem.



Good News

- More and more hardware is adressed by one SYCL implementation or the other.
- Intel is fully involved.
- Choosing the right portable parallelization solution, and the right implementation, is not such an issue.
- The real challenge is to “think parallel” and review your data structures and algorithms accordingly.
- It is worth the effort : wether the performance is better or not, this will improve your code quality, and ease the transition to any other solution.

That’s all folks

Gray-Scott Battle

David Chamont, july 2023