Programação Paralela: das *threads* aos FPGAs

Open Computing Language (OpenCL™)

Prof. Ricardo Menotti
menotti@ufscar.br

Prof. Maurício Acconcia Dias
macccdias@gmail.com

Prof. Helio Crestana Guardia
helio.guardia@ufscar.br

Departamento de Computação
Universidade Federal de São Carlos

Atualizado em: 2 de junho de 2020

Conteúdo

Introdução

Linguagem
- Código do *host* (em C)
- Código do *kernel* em OpenCL

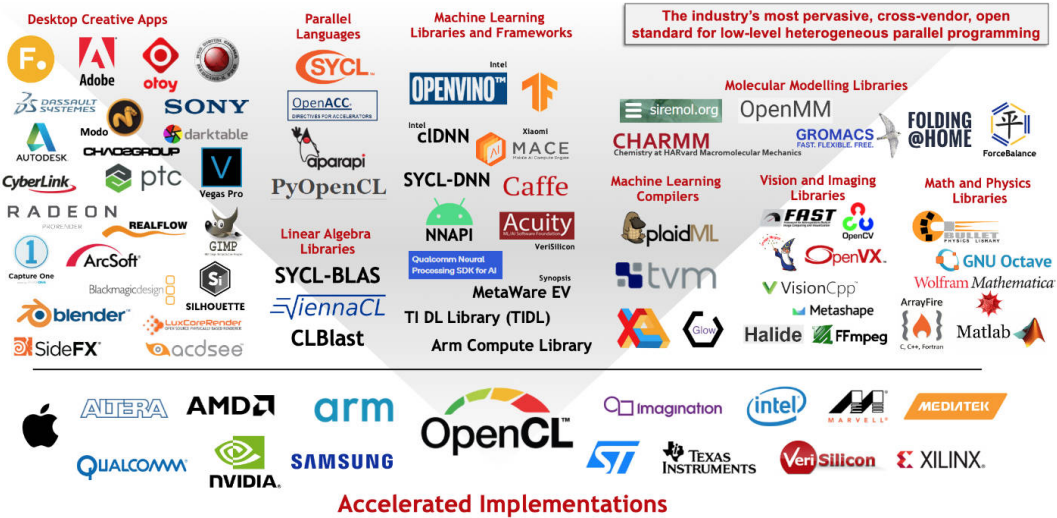
Bibliografia

OpenCL™ (Open Computing Language)

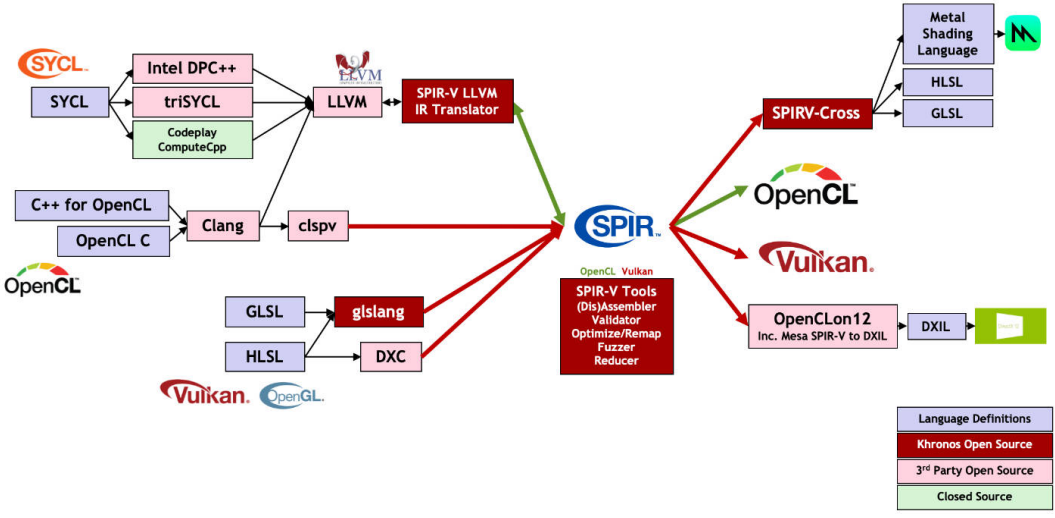
OPEN STANDARD FOR PARALLEL PROGRAMMING OF HETEROGENEOUS SYSTEMS

OpenCL™ (Open Computing Language) is an open, royalty-free standard for cross-platform, parallel programming of diverse accelerators found in supercomputers, cloud servers, personal computers, mobile devices and embedded platforms. OpenCL greatly improves the speed and responsiveness of a wide spectrum of applications in numerous market categories including professional creative tools, scientific and medical software, vision processing, and neural network training and inferencing.

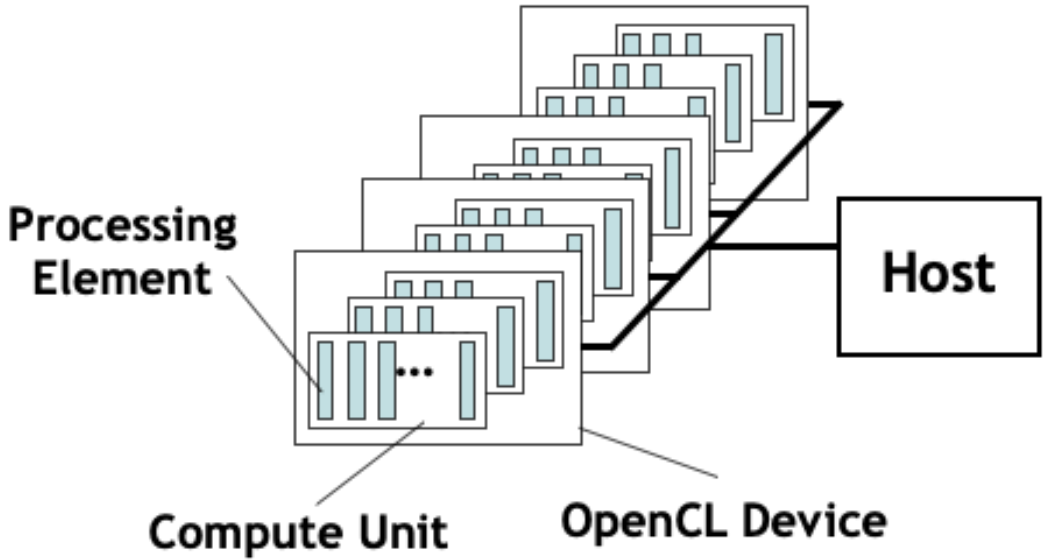
OpenCL is Widely Deployed and Used



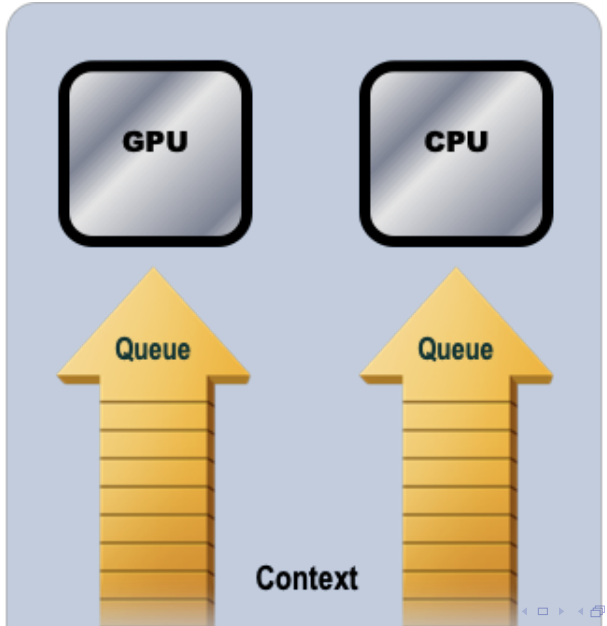
Ecosistema


SYCL
SYCL
Intel DPC++
triSYCL
Codeplay ComputeCpp
LLVM
SPIR-V LLVM IR Translator
C++ for OpenCL
OpenCL C
Clang
clspv
OpenCL
GLSL
HLSL
glslang
DXC
Vulkan
OpenGL
SPIR
OpenCL Vulkan
SPIR-V Tools
(Dis)Assembler
Validator
Optimize/Remap
Fuzzer
Reducer
SPIRV-Cross
Metal Shading Language
HLSL
GLSL
OpenCL
Vulkan
OpenCLon12
Inc. Mesa SPIR-V to DXIL
DXIL
Language Definitions
Khronos Open Source
3rd Party Open Source
Closed Source

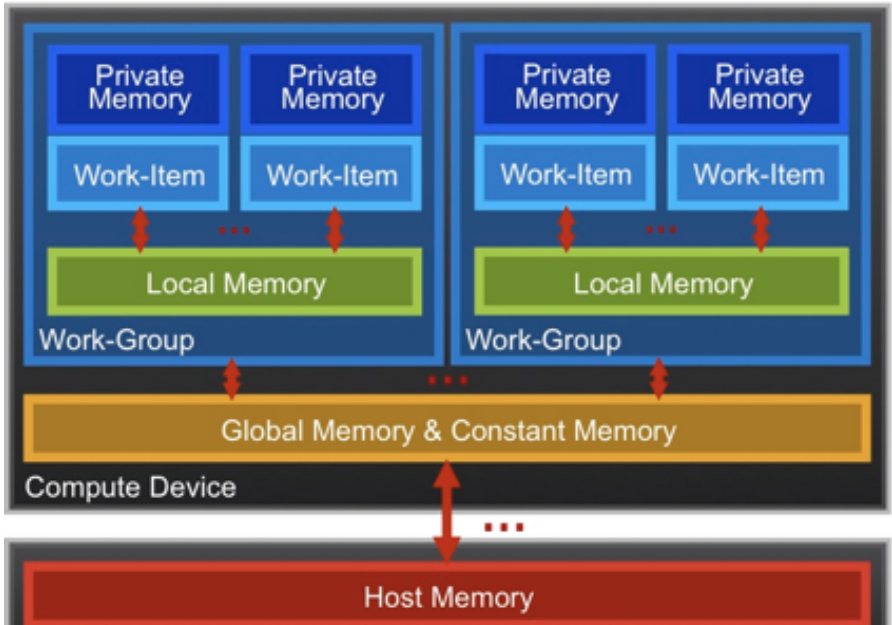
Modelo



Contexto



Memória


Private Memory
Private Memory
Work-Item
Work-Item
Local Memory
Work-Group
Private Memory
Private Memory
Work-Item
Work-Item
Local Memory
Work-Group
Global Memory & Constant Memory
Compute Device
Host Memory

Código do *host* (em C)

1. Definir a plataforma

```
1  // Pegue a primeira plataforma disponível:
2  err = clGetPlatformIDs(1, &firstPlatformId, &numPlatforms);
3
4  // Use o primeiro dispositivo de CPU fornecido pela plataforma:
5  err = clGetDeviceIDs(firstPlatformId, CL_DEVICE_TYPE_CPU, 1, &device_id, NULL);
6
7  // Crie um contexto simples com um único dispositivo:
8  context = clCreateContext(firstPlatformId, 1, &device_id, NULL, NULL, &err);
9
10 // Crie uma fila de comandos simples para alimentar seu dispositivo:
11 queue = clCreateCommandQueue(context, device_id, 0, &err);
```

Código do *host* (em C)

2. Criar e compilar o programa (kernel)

```
1  // Crie o objeto do programa:
2  program = clCreateProgramWithSource(context, 1,
3            (const char**) &KernelSource, NULL, &err);
4
5  // Compile o programa para criar uma ``biblioteca dinâmica''
6  //   a partir da qual kernels específicos podem ser extraídos:
7  err = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
8
9  // Teste e imprima mensagens de erro:
10 if (err != CL_SUCCESS) {
11   size_t len;
12   char buffer[2048];
13   clGetProgramBuildInfo(program, device_id,
14     CL_PROGRAM_BUILD_LOG, sizeof(buffer), buffer, &len);
15   printf("%s\n", buffer);
16 }
```

Código do *host* (em C)

3. Configure objetos de memória

```
1  // Crie vetores de entrada e atribua valores ao host:
2  float h_a[LENGTH], h_b[LENGTH], h_c[LENGTH];
3  for (i = 0; i < length; i++) {
4      h_a[i] = rand() / (float)RAND_MAX;
5      h_b[i] = rand() / (float)RAND_MAX;
6  }
7  // Defina objetos de memória OpenCL:
8  d_a = clCreateBuffer(context, CL_MEM_READ_ONLY,
9                       sizeof(float)*count, NULL, NULL);
10 d_b = clCreateBuffer(context, CL_MEM_READ_ONLY,
11                      sizeof(float)*count, NULL, NULL);
12 d_c = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
13                      sizeof(float)*count, NULL, NULL);
```

Código do *host* (em C)

4. Defina o kernel

```
1  // Crie um objeto do kernel a partir da função do kernel "mmul":
2  kernel = clCreateKernel(program, "mmul", &err);
3
4  // Anexe argumentos da função do kernel aos objetos de memória:
5  err  = clSetKernelArg(kernel, 0, sizeof(cl_mem), &d_a);
6  err |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &d_b);
7  err |= clSetKernelArg(kernel, 2, sizeof(cl_mem), &d_c);
8  err |= clSetKernelArg(kernel, 3, sizeof(unsigned int), &count);
```

Código do *host* (em C)

5. Enfileire comandos

```
1  // Escreva os buffers do host na memória global (como operações não-blocantes):
2  err = clEnqueueWriteBuffer(commands, d_a, CL_FALSE,
3           0, sizeof(float)*count, h_a, 0, NULL, NULL);
4  err = clEnqueueWriteBuffer(commands, d_b, CL_FALSE,
5           0, sizeof(float)*count, h_b, 0, NULL, NULL);
6  // Enfileire o kernel para execução
7  const size_t global[2] = {N, N};
8  err = clEnqueueNDRangeKernel(commands, kernel, 2,
9                 NULL, &global, &local, 0, NULL, NULL);
10 // Leia o resultado
11 err = clEnqueueReadBuffer(commands, d_c, CL_TRUE,
12             sizeof(float)*count, h_c, 0, NULL, NULL);
```

Código original em C

```
1  void mmul(
2      int N,
3      float* A,
4      float* B,
5      float* C)
6  {
7      int i, j, k;
8      for (i = 0; i < N; i++) {
9          for (j = 0; j < N; j++) {
10             C[i*N+j] = 0.0f;
11             for (k = 0; k < N; k++) {
12               C[i*N+j] += A[i*N+k] * B[k*N+j];
13             }
14         }
15     }
16 }
```

Código do *kernel* em OpenCL

```
1  __kernel void mmul(
2      const int N,
3      __global float* A,
4      __global float* B,
5      __global float* C)
6  {
7      int k;
8      int i = get_global_id(0);
9      int j = get_global_id(1);
10     float tmp;
11     if ((i < N) && (j < N))
12     {
13         tmp = 0.0;
14         for (k = 0; k < N; k++)
15             tmp += A[i*N+k] * B[k*N+j];
16         C[i*N+j] = tmp;
17     }
18 }
```

Bibliografia

- Open Computing Language (OpenCL™)
- Hands On OpenCL
- Matrix Multiplication, Wolfram Demonstrations Project
- Prof. Jukka Suomela, Aalto University

Programação Paralela: das *threads* aos FPGAs

Open Computing Language (OpenCL™)

Prof. Ricardo Menotti
menotti@ufscar.br

Prof. Maurício Acconcia Dias
macccdias@gmail.com

Prof. Helio Crestana Guardia
helio.guardia@ufscar.br

Departamento de Computação
Universidade Federal de São Carlos

Atualizado em: 2 de junho de 2020