Padrões de Projeto em Desenvolvimento de Aplicações Paralelas em Hardware Heterogêneo

Rafael Monteiro e Pereira
Faculdade de Computação e informática
Universidade Presbiteriana Mackenzie

II Escola Regional de Alto Desempenho
São José dos Campos - 28/07/2011

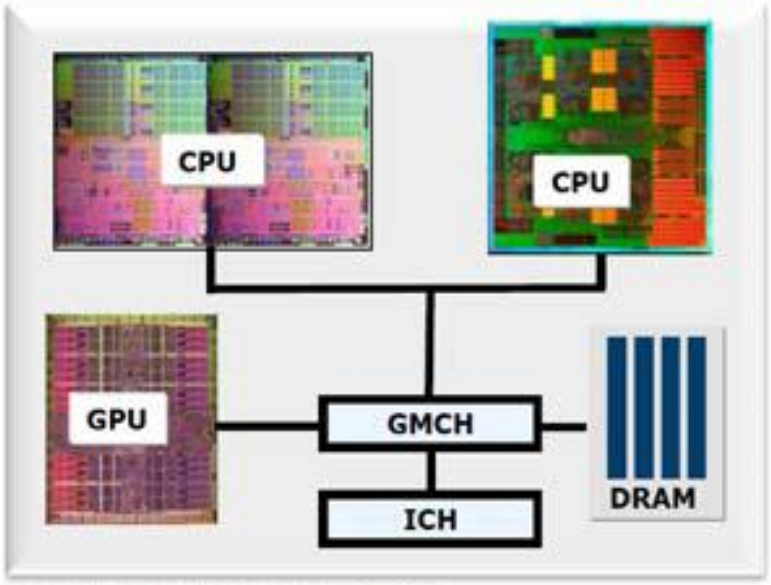
Agenda

- Motivação
- Sistemas Heterogêneos
- OpenCL
- Padrões de Projeto
- Estrutura do Projeto
- Exemplo de Aplicação

Motivação

- Arquiteturas heterogêneas
- Dificuldade de integração ➔ APIs
- Programação homogênea

Sistemas Heterogêneos



- Um sistema pode ser composto por diversos dispositivos diferentes
- Detecção do hardware ➔ escolha do melhor dispositivo

OpenCL

- Vantagens
  - Padrão aberto
  - Execução em diversos dispositivos diferentes
- Modelo de programação
  - Contextos
  - Filas
  - Kernels

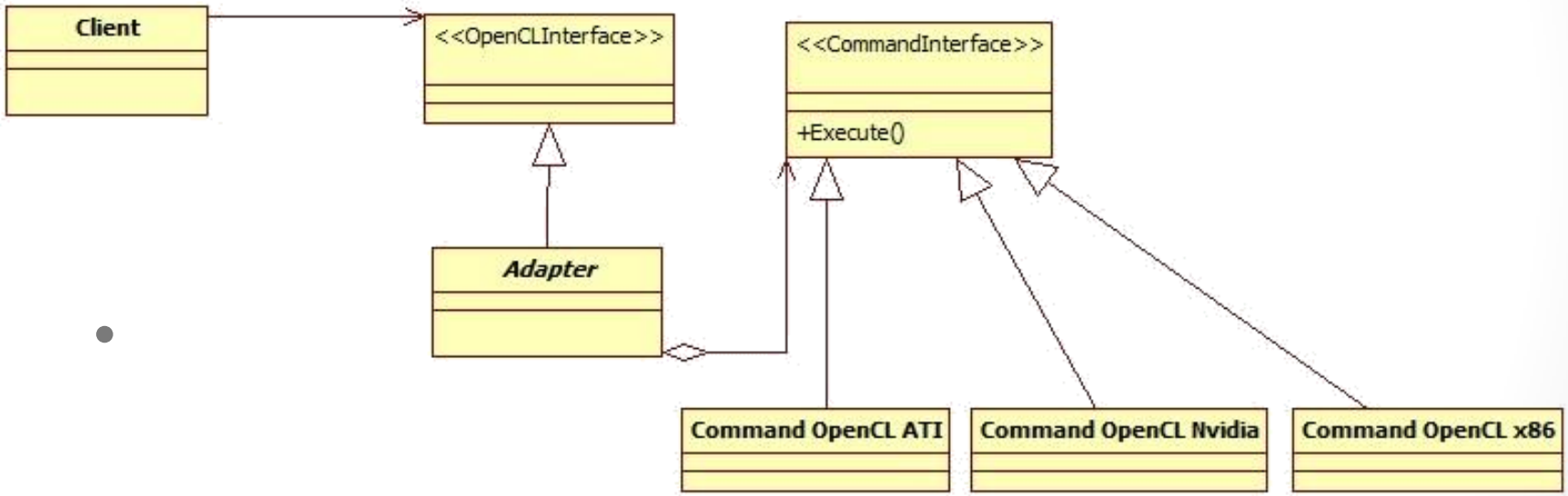
Padrões de Projeto

- O que são Padrões de Projeto?

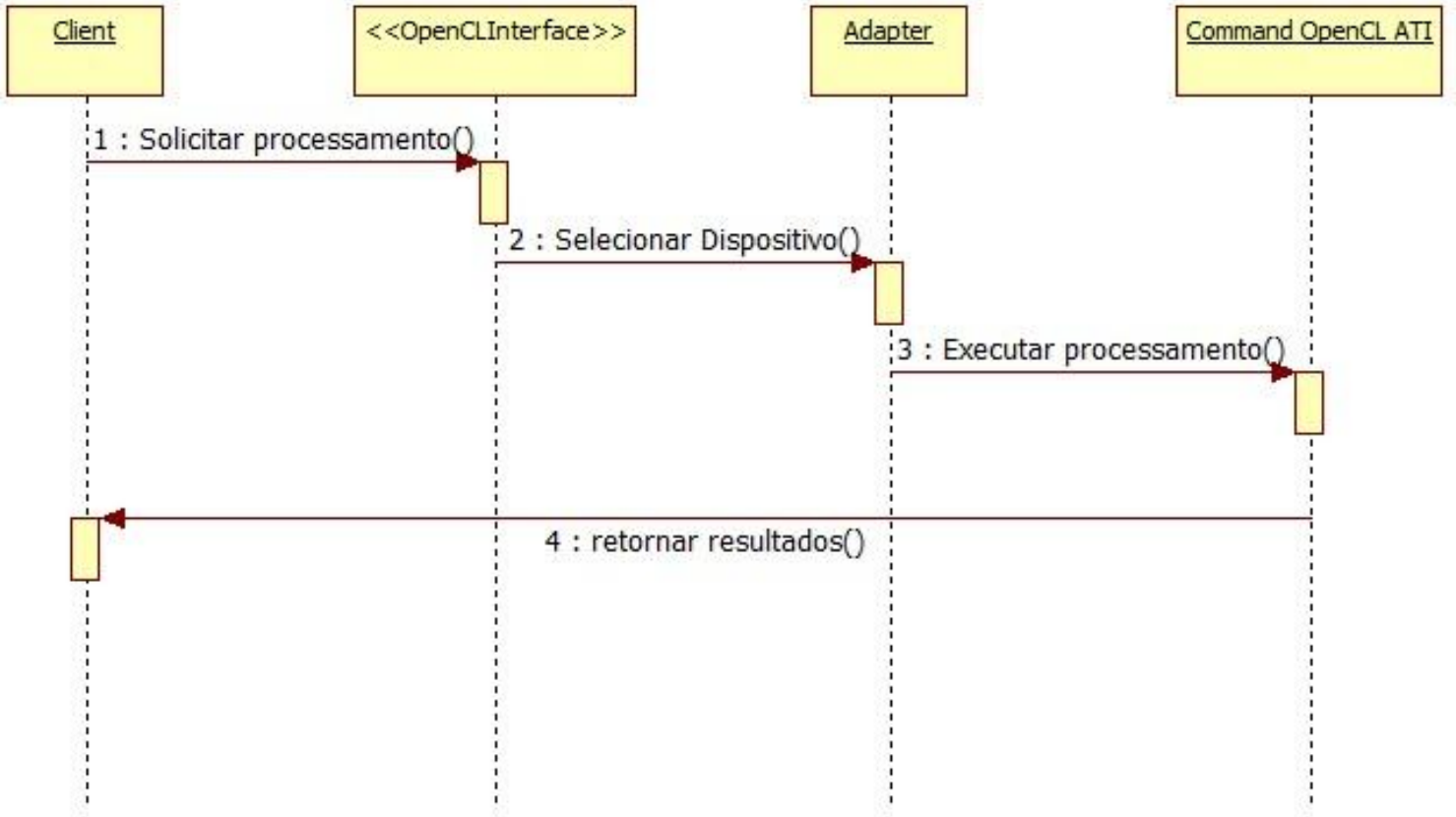
- Padrões utilizados:
  - Padrão Adapter - estrutural
  - Padrão Command- comportamental

Estrutura do Projeto

- Aplicação dos padrões



Estrutura do Projeto



Exemplo de Aplicação

```
class CommandInterface {
        public:
        virtual ~CommandInterface {}
        virtual void execute() = 0 ;
        }
```

Trecho de código 1: Implementação da interface *Command*.

```
class CommandOpenCLATI :
public CommandInterface {
        public void execute() {
        /* código da API*/
        }
}
```

Trecho de código 2: Implementação do *command* concreto.

```
class Adapter : OpenCLInterface {
  private:
  Device selectDevice(); //implementa a heurística de seleção de dispositivo
public:
  void runCommand(Arguments args);
};
Adapter::runCommand(Arguments args){
  selectDevice().getCommand().execute(args);
}
```

Trecho de código 3: Implementação do *Adapter*.

Exemplo de Aplicação

- const char* kernelSource = "__kernel void convolveKernel(global uchar *in,uint inWidth, uint inHeight,global uint *out, uint outWidth, uint outHeight,global float *convKernel,uint convKernelWidth,uint convKernelHeight)”

Exemplo de Aplicação

- // Dispositivos

```
std::vector<cl::Device>devices;
platforms[0].getDevices(CL_DEVICE_TYPE_GPU,
&devices);
assert(devices.size() > 0);
assert(devices[0].getInfo<CL_DEVICE_TYPE>()
==CL_DEVICE_TYPE_GPU);
```

Exemplo de Aplicação

- // Contextos

```
cl::Context context(devices);

// Criação de buffers GPU

cl::Buffer inGPU( context,CL_MEM_READ_ONLY,
in.width *in.height *sizeof(uchar));

cl::Buffer convKernelGPU(context,
CL_MEM_READ_ONLY,convKernel.width
*convKernel.height *sizeof(float));

cl::Buffer outGPU( context,CL_MEM_WRITE_ONLY,
out.width *out.height *sizeof(uint));
```

Exemplo de Aplicação

- // Commandqueue

```
cl::CommandQueue queue(context, devices[0], 0);
queue.enqueueWriteBuffer(inGPU,false, // FIFO
0,in.width * in.height *sizeof(uchar),in.data);
queue.enqueueWriteBuffer(convKernelGPU,true,0,con
vKernel.width *convKernel.height
*sizeof(float),convKernel.data);
cl ::Program::Sources source(1,
std::make_pair(kernelSource,strlen(kernelSource)));
cl::Program program(context, source);
program.build(devices);
```

Próximas etapas

- Candidatos a Implementação
  - Aplicação para render de imagens
  - Cálculo de hash criptográfico

Obrigado!

rafamonteiro.pereira@gmail.com