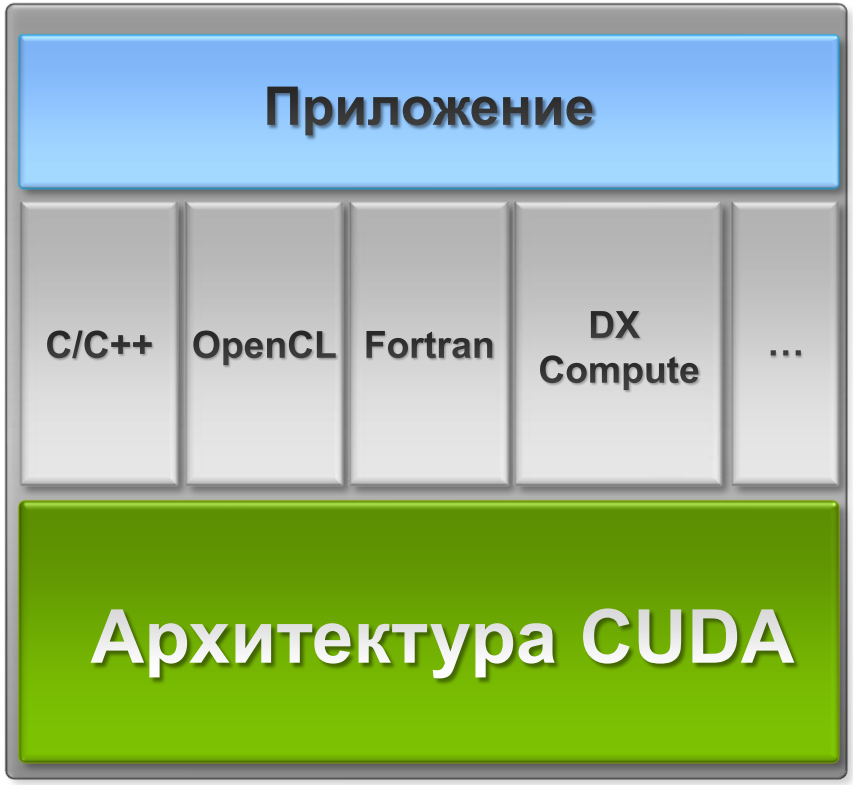
CUDA API & Multi-GPU

- Лекторы:
  - Боресков А.В. (ВМиК МГУ)
  - Харламов А. (NVidia)
  - Микушин Д. (НИВЦ)

CUDA API

- CUDA C
- CUDA Driver API
- OpenCL
- DirectX Compute



CUDA C (Runtime API)

- Расширение языка C
- CUDA API:
  - Расширения языка C
  - Runtime библиотека

CUDA C Runtime

```
float * a   = new float [N];
float * dev = NULL;

cudaMalloc( (void**)&dev, N * sizeof ( float ) );

dim3 threads = dim3( 512, 1 );
dim3 blocks  = dim3( N / threads.x, 1 );


kernel<<<blocks, threads>>> ( dev );
cudaThreadSynchronize();

cudaMemcpy(a, dev, N*sizeof(float), cudaMemcpyDeviceToHost);

cudaFree( dev );
```

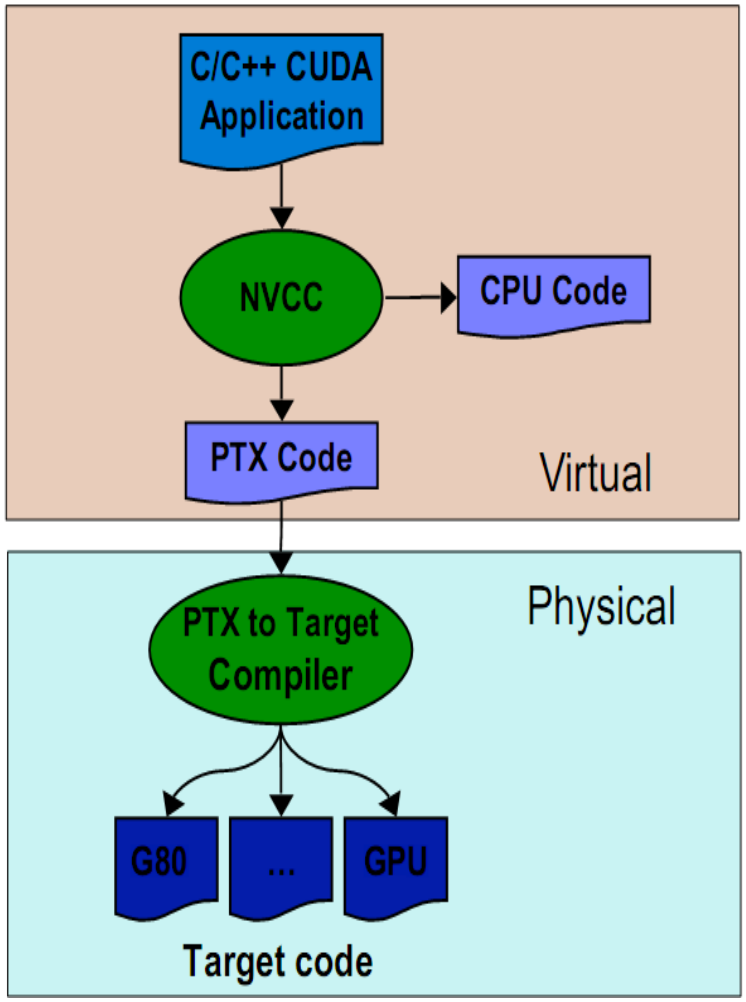
CUDA C Runtime

```
__global__ void kernel ( float * data )
{
   int idx = blockIdx.x * blockDim.x + threadIdx.x ;

   data [idx] = idx;
}
```

CUDA C Runtime

- NVCC
- .ptx
  - -keep



CUDA C Runtime

- NVCC
- .ptx
  - -keep

```
__global__ void kernel ( float * data )
{
   int idx = blockIdx.x * blockDim.x + threadIdx.x ;

   data [idx] = idx;
}
```

CUDA C Runtime

```
.entry _Z6kernelPf (.param .u32 __cudaparm__Z6kernelPf_data)
{
  .reg .u16 %rh<4>;
  .reg .u32 %r<8>;
  .reg .f32 %f<3>;
  .loc 14      6       0
  $LBB1__Z6kernelPf:
  .loc 14      10      0
  mov.u16      %rh1, %ctaid.x;  //
  mov.u16      %rh2, %ntid.x;   //
  mul.wide.u16         %r1, %rh1, %rh2;       //
  cvt.u32.u16  %r2, %tid.x;           //
  add.u32      %r3, %r2, %r1;   //
  cvt.rn.f32.s32       %f1, %r3;      //
  ld.param.u32         %r4, [__cudaparm__Z6kernelPf_data]; // id:14
  mul.lo.u32   %r5, %r3, 4;     //
  add.u32      %r6, %r4, %r5;   //
  st.global.f32        [%r6+0], %f1;          // id:15
  .loc 14      11      0
  exit;
  $LDWend__Z6kernelPf:
} // _Z6kernelPf
```

CUDA C Driver

```
CUdevice    device;
CUcontext   context;
CUmodule    module;
CUfunction  function;
CUdeviceptr pData;

float * pHostData = new float[N];

cuInit(0);

cuDeviceGetCount(&device_count);
cuDeviceGet( &device, 0 );

cuCtxCreate( &context, 0, device );

cuModuleLoad( &module, "hello.cuda_runtime.ptx" );
cuModuleGetFunction( &function, module, "_Z6kernelPf" );

cuMemAlloc( &pData, N * sizeof(float) );

// ...
```

CUDA C Driver

```
// ...
cuFuncSetBlockShape( function, N, 1, 1 );
cuParamSeti( function, 0, pData );
cuParamSetSize( function, sizeof(void *) );
cuLaunchGrid( function, 1, 1 );
cuMemcpyDtoH( pHostData, pData, N * sizeof( float) );
cuMemFree( pData );
```

OpenCL

- Кроссплатформенный стандарт
  - GPU, CPU, Cell, …
- Проблема: **функциональность, но не производительность**
  - Разный код для разных платформ
  - Разные расширения openGL-style

CUDA vs OpenCL Терминология

- CUDA C
  - Поток (thread)
  - Блок потоков (thread block)
  - Сеть (grid)
  - Ядро
- OpenCL
  - Элемент работы (work-item)
  - Группа работы (work-group)
  - N-мерное пространство индексов (ND-Range index space)
  - Ядро

CUDA vs OpenCL
Спецификаторы функций

- CUDA C
  - __global__
  - __host__
  - __device__
- OpenCL
  - __kernel
  - `n/a`
  - `n/a`

CUDA vs OpenCL Пространство памяти

- CUDA C
  - __device__
  - __shared__
  - __constant__
  - local
- OpenCL
  - __global
  - __local
  - __constant
  - __private

OpenCL

```
cl_context ctx;
cl_command_queue cmd_q;
cl_program program;
cl_kernel kernel;
cl_device_id * pDevId = NULL;
```

```
ctx = clCreateContextFromType(0,CL_DEVICE_TYPE_GPU,0,0,0);
```

```
clGetContextInfo(ctx,CL_CONTEXT_DEVICES,0,0,&dev_cnt);
clGetContextInfo(ctx,CL_CONTEXT_DEVICES,dev_cnt,pDevId,0);
```

```
cmd_q= clCreateCommandQueue(ctx,pDevId[0],0,0);
```

```
program = clCreateProgramWithSource(ctx,1,pText,0,0);
clBuildProgram(program, 0,0,0,0,0);
```

```
kernel = clCreateKernel(program, "simple", 0);
```

OpenCL

```
cl_mem mem = clCreateBuffer(ctx,CL_MEM_WRITE_ONLY,
                            N*sizeof(float),0,0);
```

```
clSetKernelArg(kernel, 0, sizeof(cl_mem),(void*) &mem);
clSetKernelArg(kernel, 1, sizeof(int),   (void*) &N);
```

```
clEnqueueNDRangeKernel(cmd_q,kernel,1,0,&N,&N,0,0,0);
```

```
clEnqueueReadBuffer(cmd_q, mem, CL_TRUE, 0,
                    N*sizeof(float), pData,0,0,0);
```

```
clReleaseMemObject(mem);
clReleaseKernel(kernel);
clReleaseProgram(program);
clReleaseCommandQueue(cmd_q);
clReleaseContext(ctx);
```

DirectX Compute

- Microsoft API
- Тесно интегрирован с Direct3D
- Доступен
  - CS 4.x: DirectX 10 HW
  - CS 5.x: DirectX 11 HW

DirectX

- ID3D11Device
  - ID3D11Resource
  - ID3D11View
- ID3D11DeviceContext

CUDA vs DirectX
Спецификаторы функций

- CUDA C
  - __global__
  - __host__
  - __device__
- DirectX
  - Compute Shader
  - `n/a`
  - `n/a`

CUDA vs DirectX Compute Пространство памяти

- CUDA C
  - __device__
  - __shared__
  - __constant__
  - local
- DirectX
  - [Structured]Buffer
  - groupshared
  - Constant Buffer
  - n/a

DirectX

- ID3D11Device
  - ID3D11Resource
  - ID3D11View
- ID3D11DeviceContext
- ID3D11Asynchronous
  - ID3D11Query
- ID3D11ComputeShader
- ID3DX11Effect

DirectX

- ID3D11Device
  - ID3D11Resource
    - Buffer
    - StructuredBuffer
    - Texture
  - ID3D11View
    - ShaderResourceView
    - UnorderedAccessView
    - RenderTargetView

DirectX

- ID3D11DeviceContext
  - Dispatch(bx, by, bz)
  - DispatchIndirect(pBuffer, offset)
  - End(pQuery)
  - GetData(g_pQuerry, NULL, 0, 0)

DirectX

ID3D11ComputeShader

- ConstantBuffer
- ShaderResourceView
- UnorderedAccessView

ID3D11Effect

- ConstantBuffer
- ShaderResourceView
- UnorderedAccessView

DirectX

ID3D11ComputeShader

```
pContext->CSSetShader(pCS, NULL, 0);
pContext->CSSetUnorderedAccessViews(0, 1, &pRWBufUAV, NULL);
```

ID3D11Effect

```
pEffect->GetVariableByName("tSimple")->AsUnorderedAccessView()-
   >SetUnorderedAccessView(pRWBufUAV);

pEffect->GetTechniqueByName("tSimple")->GetPassByName("pSimple")-
   >Apply(0, pContext);
```

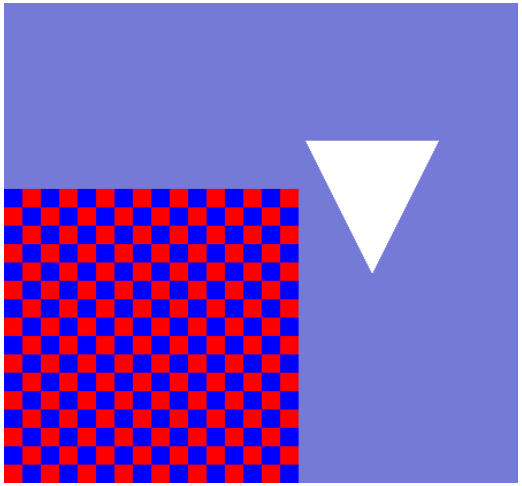
```
20  #define GROUP_SIZE 16
21
22  unsigned int    g_bufPitch = 0;
23  unsigned int    g_bufWidth = 3;
24  unsigned int    g_bufHeight = 0;
25
26  RWBuffer<float4>  g_bufResult : register ( u0 );
27
28  [numthreads( GROUP_SIZE, GROUP_SIZE, 1 )]
29  void CS_Main_RWBuffer(uint3 gid : SV_GroupID,
30                        uint3 tid : SV_GroupThreadID,
31                        uint3 id : SV_DispatchThreadID)
32  {
33    if (id.x > g_bufWidth || id.y > g_bufHeight) return;
34
35    float4 pos[3] = { {0.0, 1.0, 0.0, 1.0}, {-1.0, -1.0, 0.0, 1.0}, {1.0, -1.0, 0.0, 1.0}};
36
37    int index = id.x % 3;
38
39    g_bufResult[id.x + id.y * g_bufPitch] = pos[index];
40  }
41
```

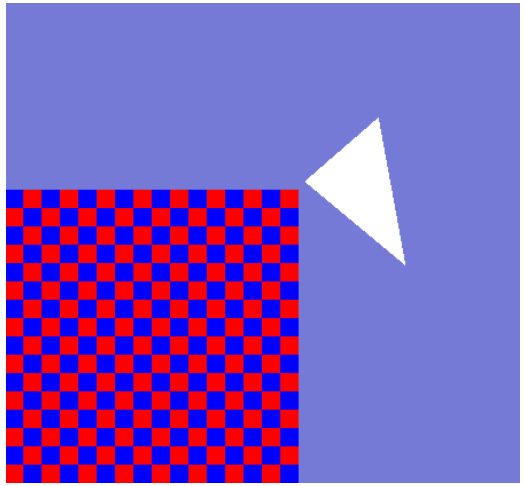
DirectX Compute

```
41  unsigned int     g_t2dWidth = 512;
42  unsigned int     g_t2dHeight = 512;
43  RWTexture2D<float4> g_t2dResult : register ( u1 );
44
45  [numthreads( GROUP_SIZE, GROUP_SIZE, 1 )]
46  void CS_Main_RWTexture2D(uint3 gid : SV_GroupID,
47                           uint3 tid : SV_GroupThreadID,
48                           uint3 id : SV_DispatchThreadID)
49  {
50    if (id.x > g_t2dWidth || id.y > g_t2dHeight) return;
51
52    if ((gid.x % 2 == 0 && gid.y % 2 == 0) || (gid.x % 2 == 1 && gid.y % 2 == 1))
53      g_t2dResult[id.xy] = float4(1, 1, 0, 1);
54    else
55      g_t2dResult[id.xy] = float4(1, 0, 1, 1);
56  }
57
```

DirectX Compute

```
59  technique11 tFillUAV
60  {
61      pass pFillRWTexture2D
62      {
63          SetComputeShader(CompileShader(cs_5_0, CS_Main_RWTexture2D()));
64      }
65
66      pass pFillRWBuffer
67      {
68          SetComputeShader(CompileShader(cs_5_0, CS_Main_RWBuffer()));
69      }
70  };
```

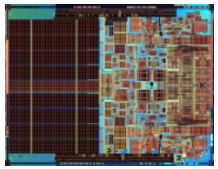




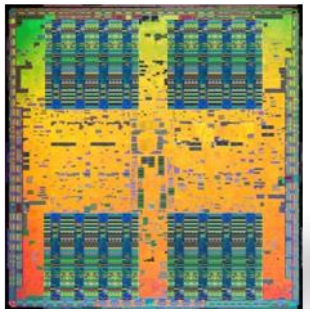
Multi-GPU

- CUDA
- OpenMP
- MPI
- OS Threads

Multi-GPU



CPU

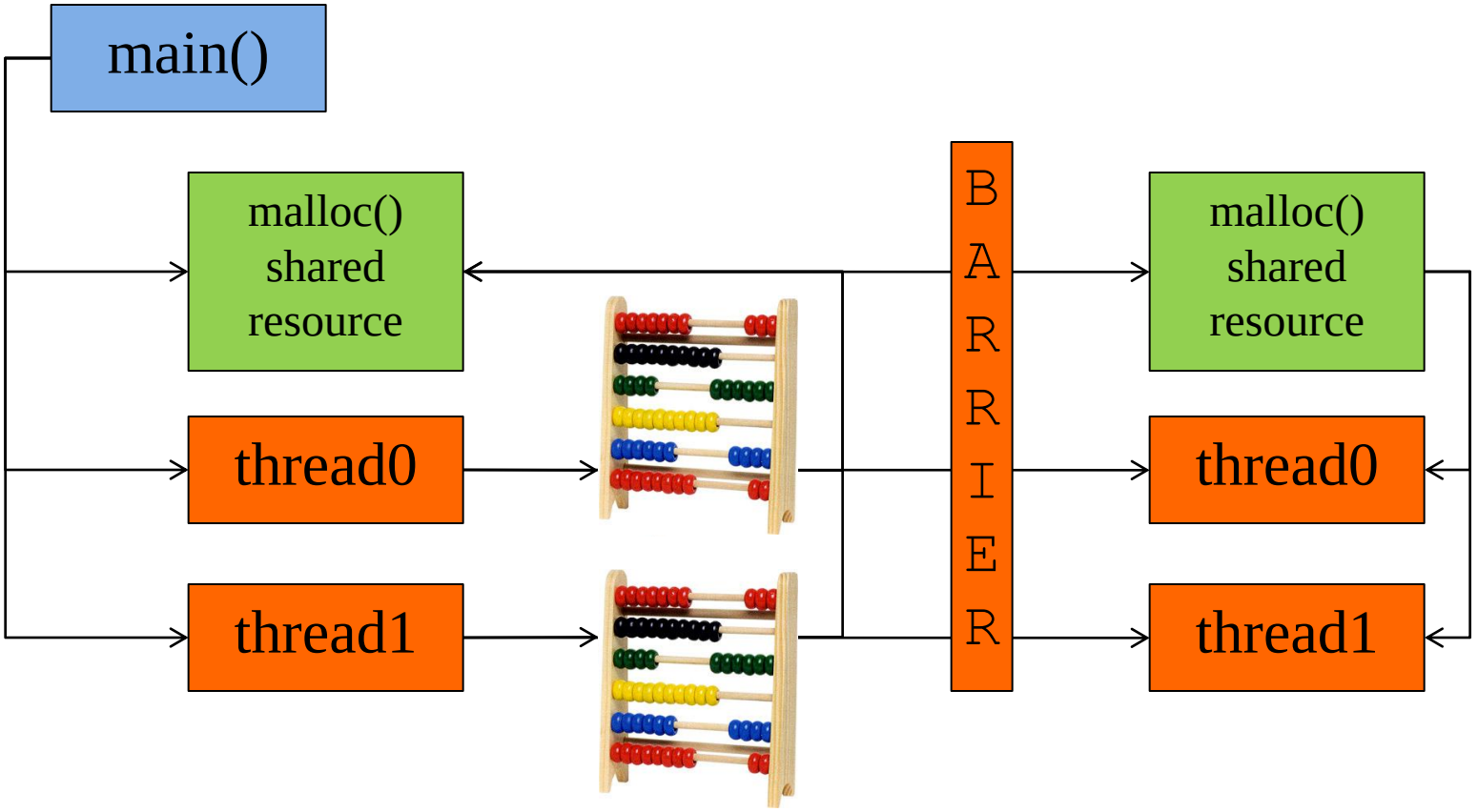


GPU

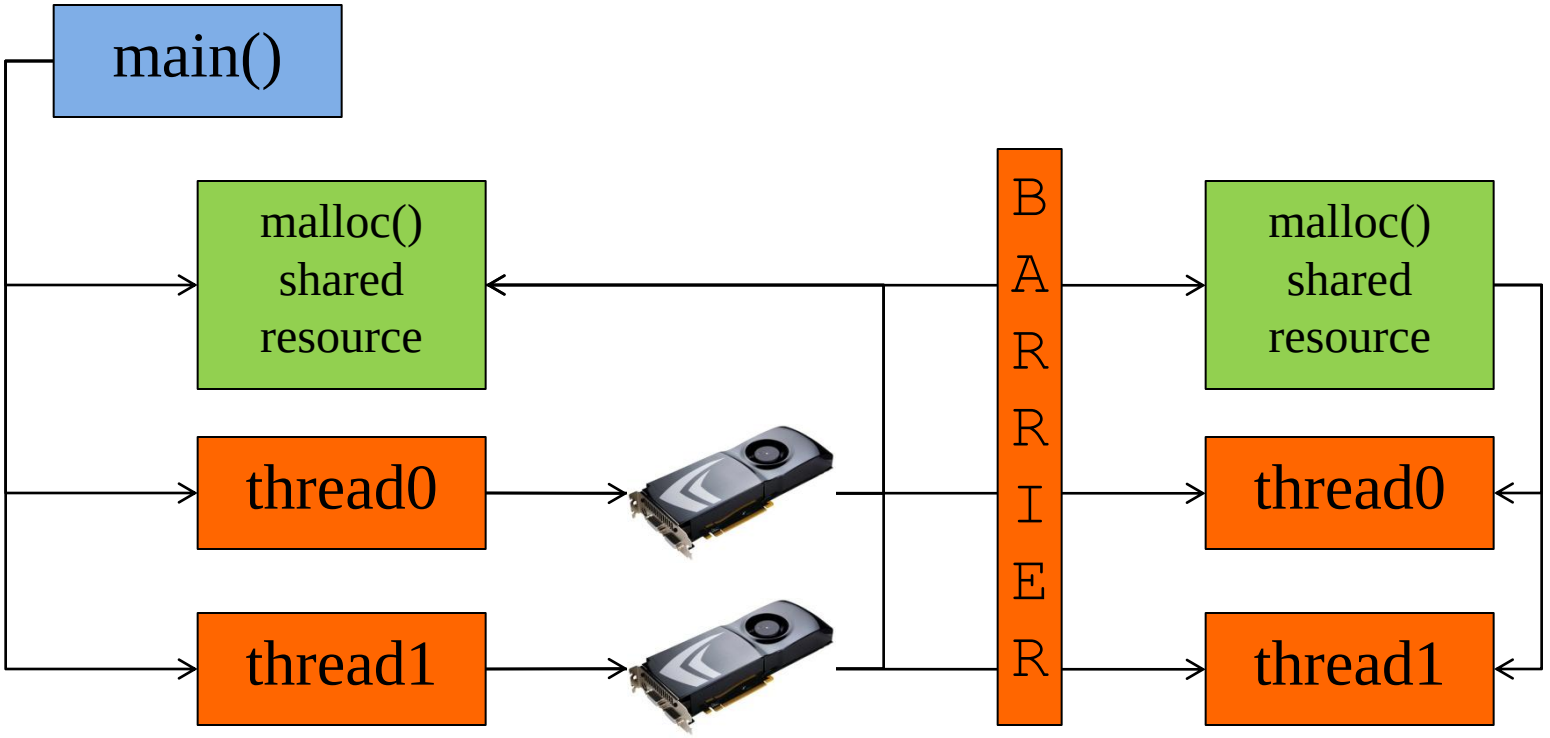
CUDA

Кластер

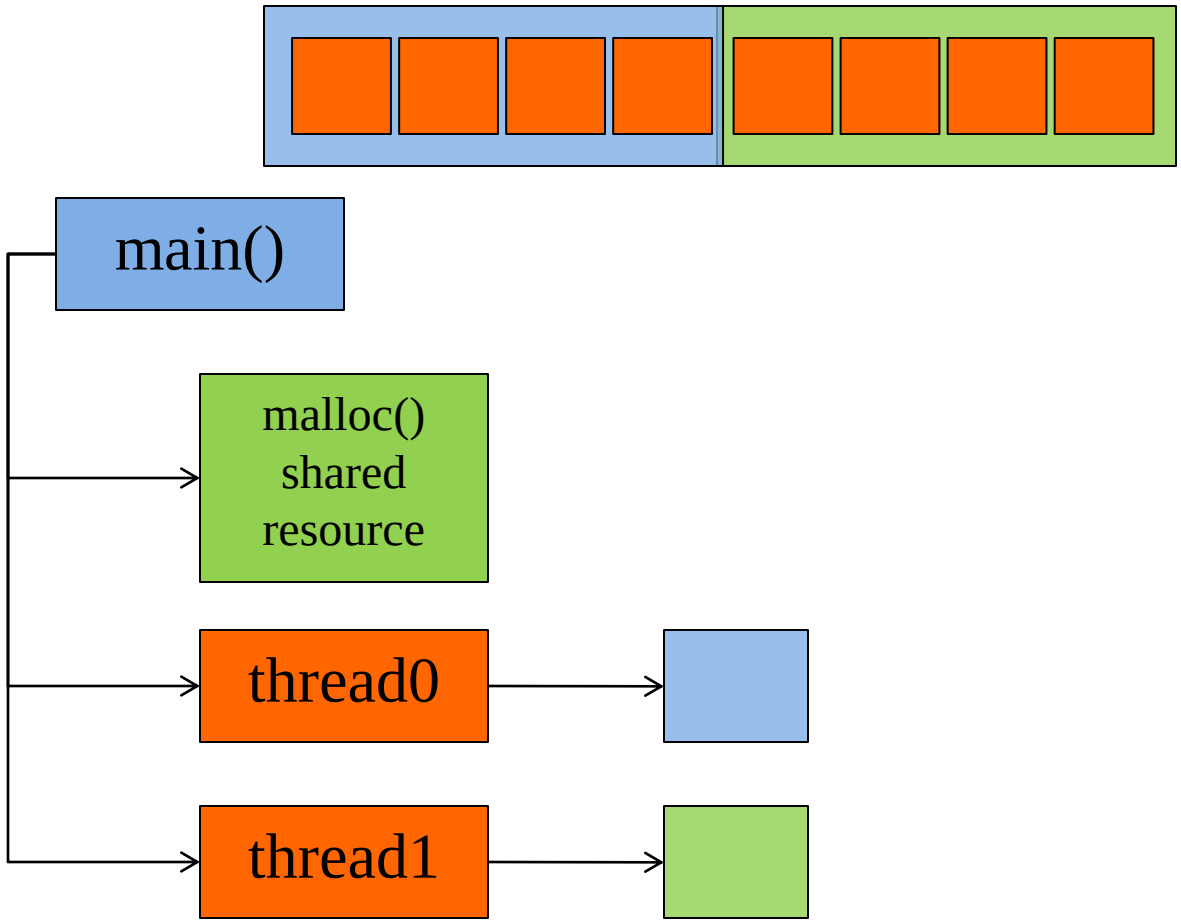
Пример


main()
malloc()
shared
resource
thread0
thread1
BARRIER
malloc()
shared
resource
thread0
thread1

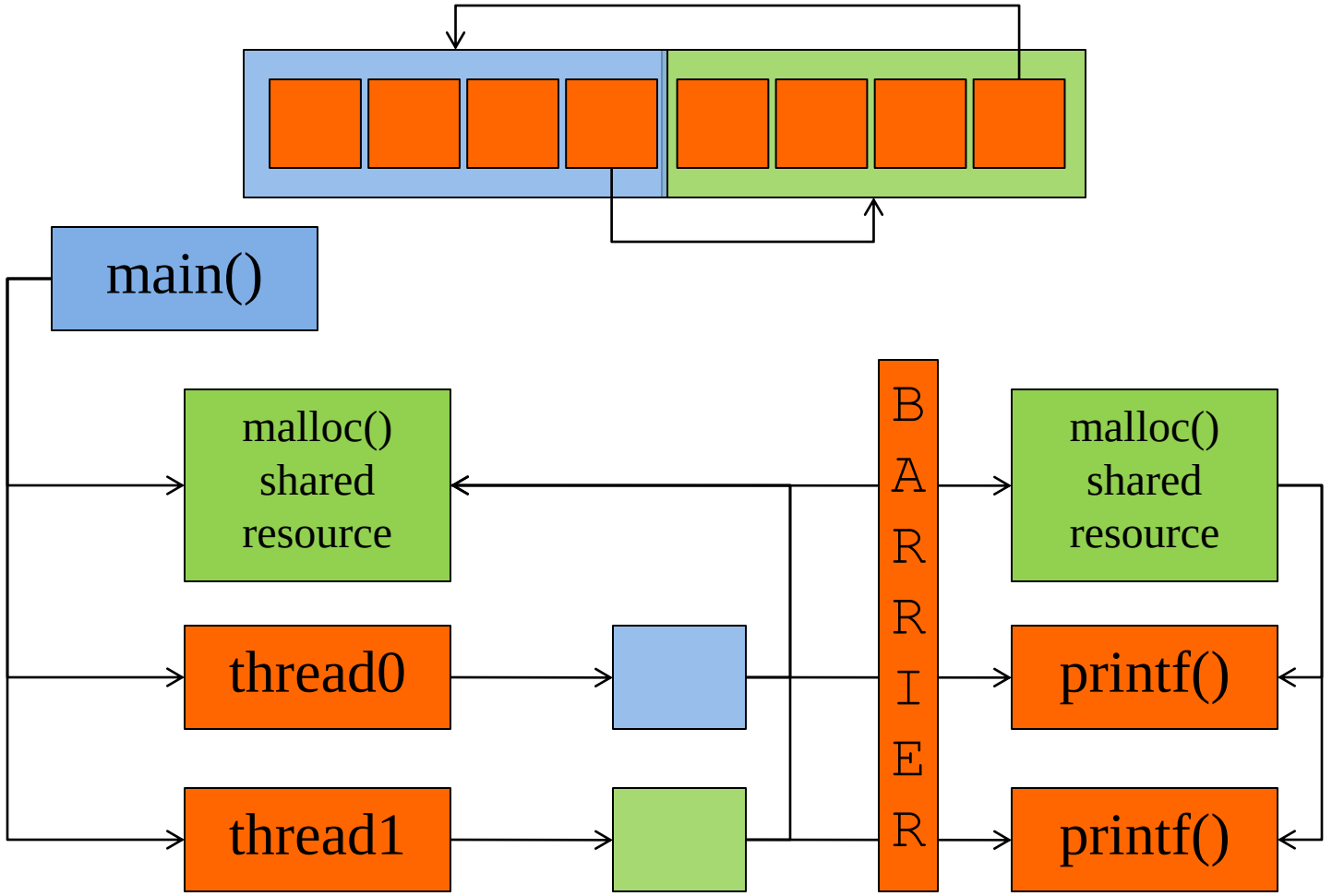
Пример


main()
malloc()
shared
resource
thread0
thread1
BARRIER
malloc()
shared
resource
thread0
thread1

Модельная задача



Модельная задача

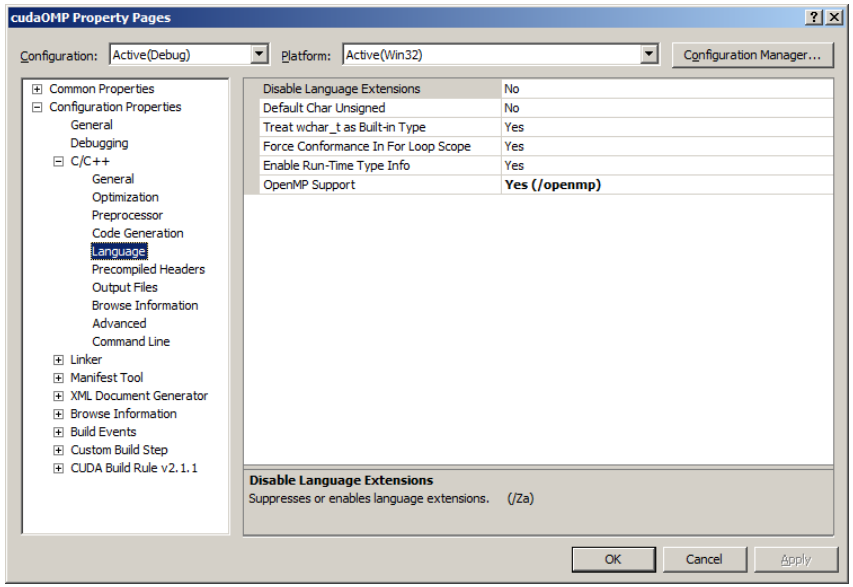


CUDA

- cudaGetDeviceCount()
- cudaSetDevice()

OpenMP

- omp_set_num_threads()
- omp_get_thread_num()
- omp_get_num_threads()
- #pragma omp parallel
- #pragma omp barrier



Модельная задача

```
9   void Kernel_Wrapper(float *, int, int);
10
11  int main(int argc, char * argv[])
12  {
13    int device_count = 0;
14    int N = 32;
15
16    cudaGetDeviceCount(&device_count);
17    omp_set_num_threads(device_count);
18
19    float *shared_mem = new float[device_count];
20
```

Модельная задача

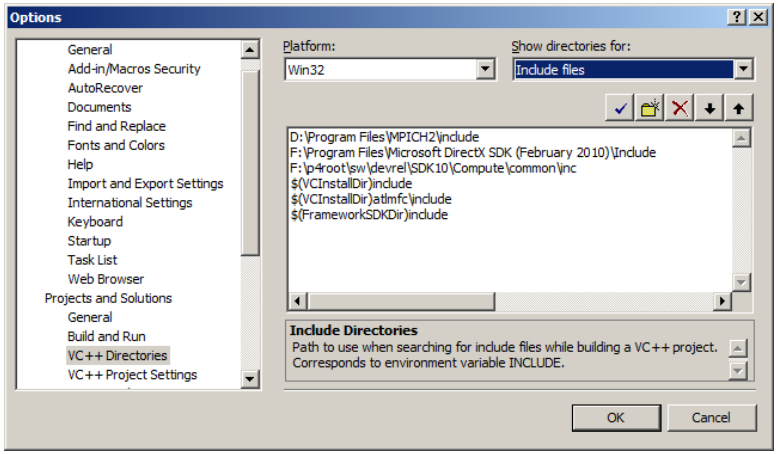
```
21  #pragma omp parallel
22    {
23      unsigned int cpu_thread_id = omp_get_thread_num();
24      unsigned int num_cpu_threads = omp_get_num_threads();
25
26      cudaSetDevice(cpu_thread_id);
27
28      fprintf(stdout, "Rank %d size %d device %d / %d\n",
29        cpu_thread_id, num_cpu_threads, device_count);
30
31      float * dData = NULL;
32      float * hData = NULL;
33
34      cudaMallocHost((void **) &hData, N * sizeof(float));
35      cudaMalloc((void **) &dData, N * sizeof(float));
36
37      Kernel_Wrapper(dData, cpu_thread_id, N);
38
39      cudaMemcpy(hData, dData, N * sizeof(float), cudaMemcpyDeviceToHost);
40
41      fprintf(stdout, "0 : %f | 1 : %f | 2 : %f | 3 : %f\n",
42        hData[0], hData[1], hData[2], hData[3]);
43
44      shared_mem[(cpu_thread_id + 1) % num_cpu_threads] = hData[3];
```

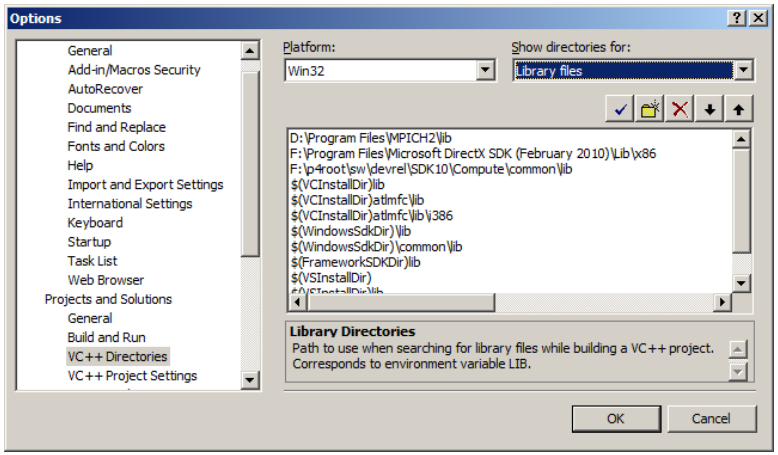
Модельная задача

```
46  #pragma omp barrier
47
48      fprintf(stdout, "Thread %d recv %f\n", cpu_thread_id, shared_mem[cpu_thread_id]);
49
50      cudaFree(dData);
51      cudaFreeHost(hData);
52    }
53
54    delete [] shared_mem;
55
56    return 0;
```

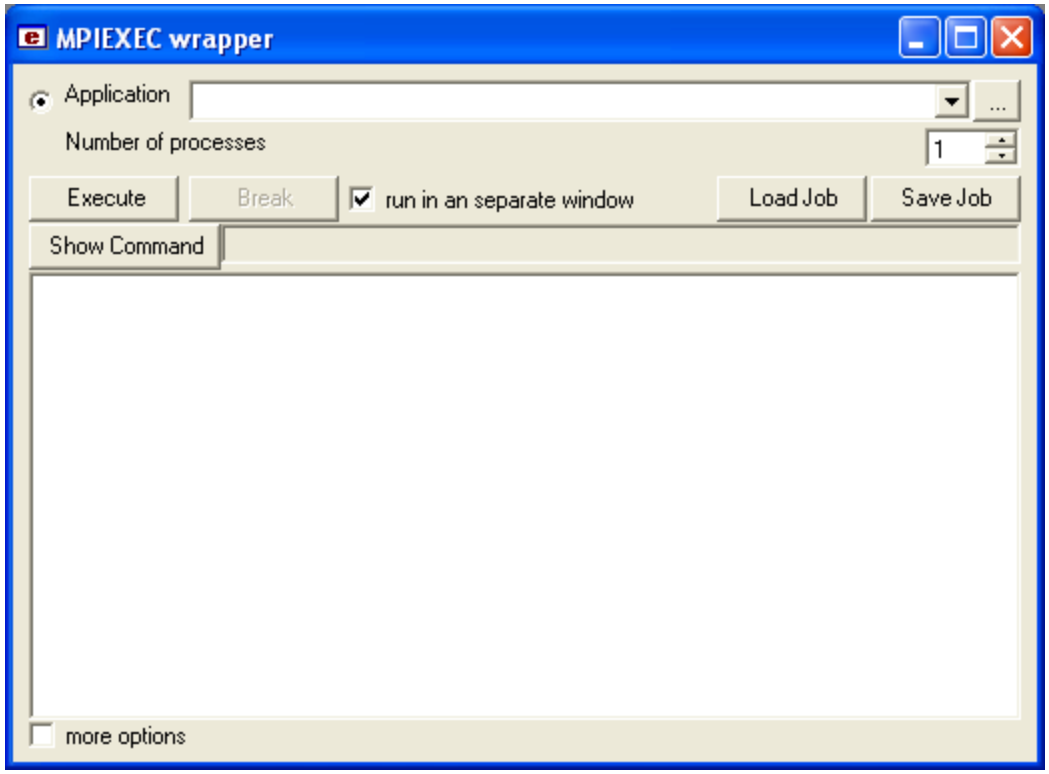
MPI

- MPICH2
- Инструкция по установке
- Programming Guide





Запуск MPI


MPIEXEC wrapper
Application
...
Number of processes
1
Execute
Break
run in an separate window
Load Job
Save Job
Show Command
more options

Модельная задача

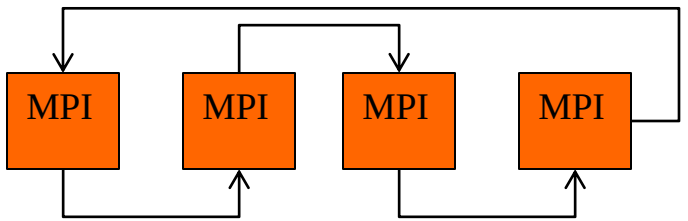
```
9  void Kernel_Wrapper(float *, int, int);
10
11 int main(int argc, char * argv[])
12 {
13   MPI_Init(&argc, &argv);
14   int N = 32;
15   int rank, size;
16
17   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
18
19   MPI_Comm_size(MPI_COMM_WORLD, &size);
20
21   int device_count = 0;
22
23   cudaGetDeviceCount(&device_count);
24
25   int set_device = min(device_count-1, rank % size);
26
27   cudaSetDevice(set_device);
28
29   fprintf(stdout, "Rank %d size %d device %d / %d\n",
30     rank, size, set_device, device_count);
```

Модельная задача

```
32    float * dData = NULL;
33    float * hData = NULL;
34
35    cudaMallocHost((void **) &hData, N * sizeof(float));
36    cudaMalloc((void **) &dData, N * sizeof(float));
37
38    Kernel_Wrapper(dData, rank, N);
39
40    cudaMemcpy(hData, dData, N * sizeof(float),
41      cudaMemcpyDeviceToHost);
42
43    fprintf(stdout, "0 : %f | 1 : %f | 2 : %f | 3 : %f\n",
44      hData[0], hData[1], hData[2], hData[3]);
45
```

Модельная задача

```
46    float recvbuf = 0.0f;
47    MPI_Status st;
48
49    MPI_Send(&hData[3], 1, MPI_FLOAT, (rank+1) % size,
50      0, MPI_COMM_WORLD);
51    MPI_Recv(&recvbuf, 1, MPI_FLOAT, (size + rank-1) % size,
52      0, MPI_COMM_WORLD, &st);
53
54    fprintf(stdout, "Rank %d rec %f\n", rank, recvbuf);
55
56    cudaFree(dData);
57    cudaFreeHost(hData);
58
59    MPI_Finalize();
60
61    return 0;
62  }
```



Ссылки

- http://iproc.ru/programming/openmp-visual-studio/
- http://iproc.ru/programming/mpich-windows/
- http://www.mcs.anl.gov/research/projects/mpich2/

Ресурсы нашего курса

- CUDA.CS.MSU.SU
  - Место для вопросов и дискуссий
  - Место для материалов нашего курса
  - Место для ваших статей!
    - Если вы нашли какой-то интересный подход!
    - Или исследовали производительность разных подходов и знаете, какой из них самый быстрый!
    - Или знаете способы сделать работу с CUDA проще!
- Steps3d.narod.ru
- www.nvidia.ru