Семинар 6а. Разработка гибридного приложения.

1. Идея и структура приложения.

Цель – создание гибридного приложения, способного работать как на CPU, так и на GPU. Используемые гибридные вычислительные кластеры изображены на Рис. 1, 2.

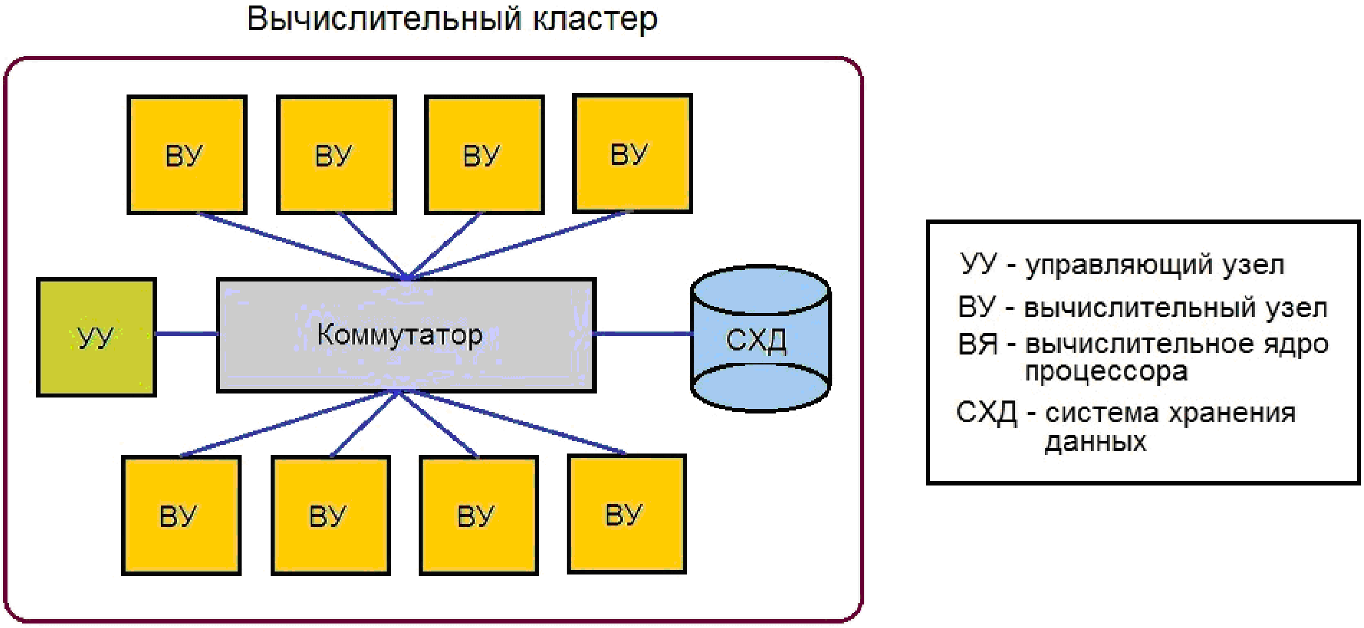

Рис. 1. Вычислительный кластер.

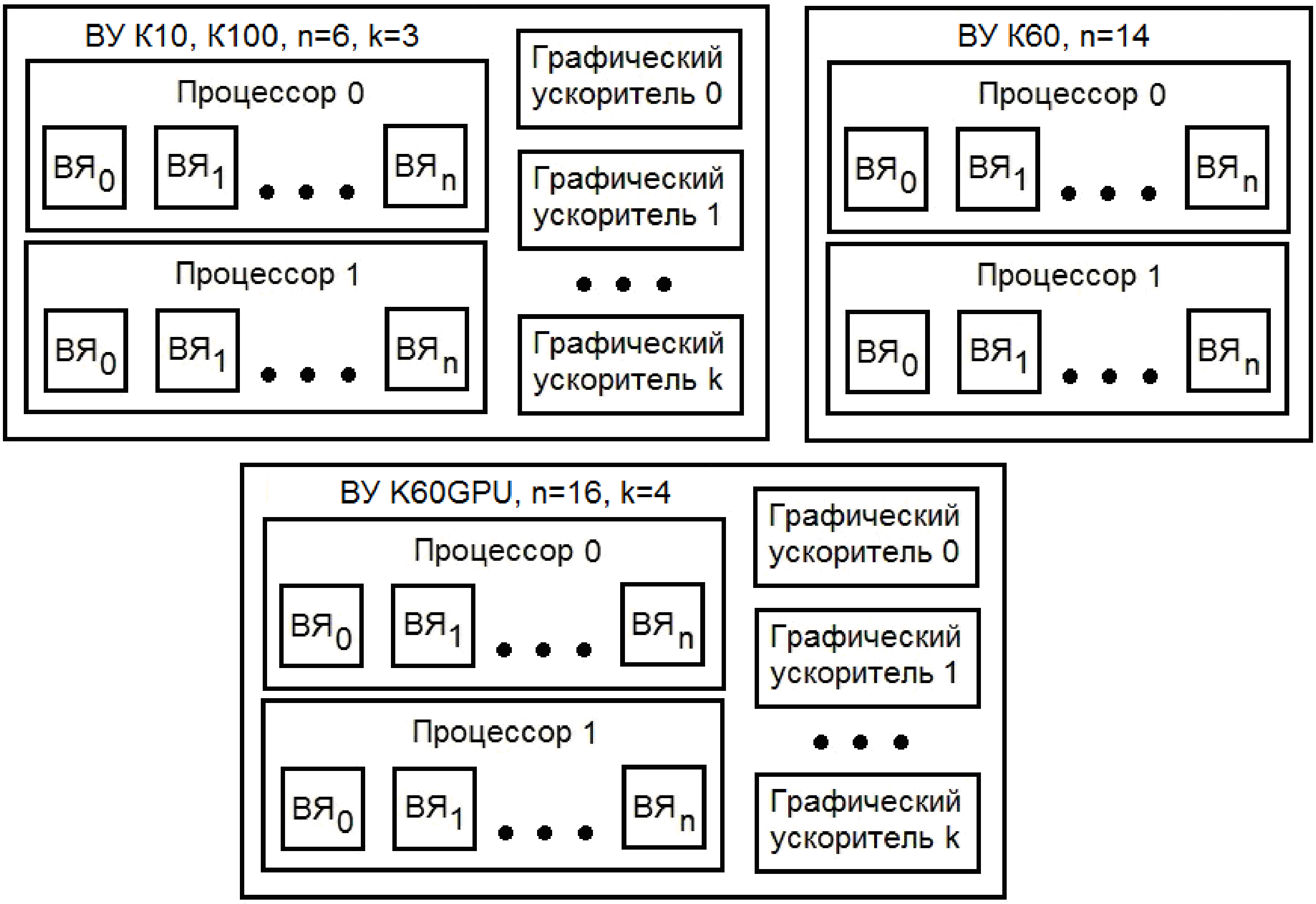

Рис. 2. Архитектуры узлов ГСК К100, К10, К60, К60GPU

Технологии программирования – MPI (поддержка сетевой модели), OpenMP (вычисления на CPU и поддержка индивидуальной работы с GPU), CUDA (использование GPU).

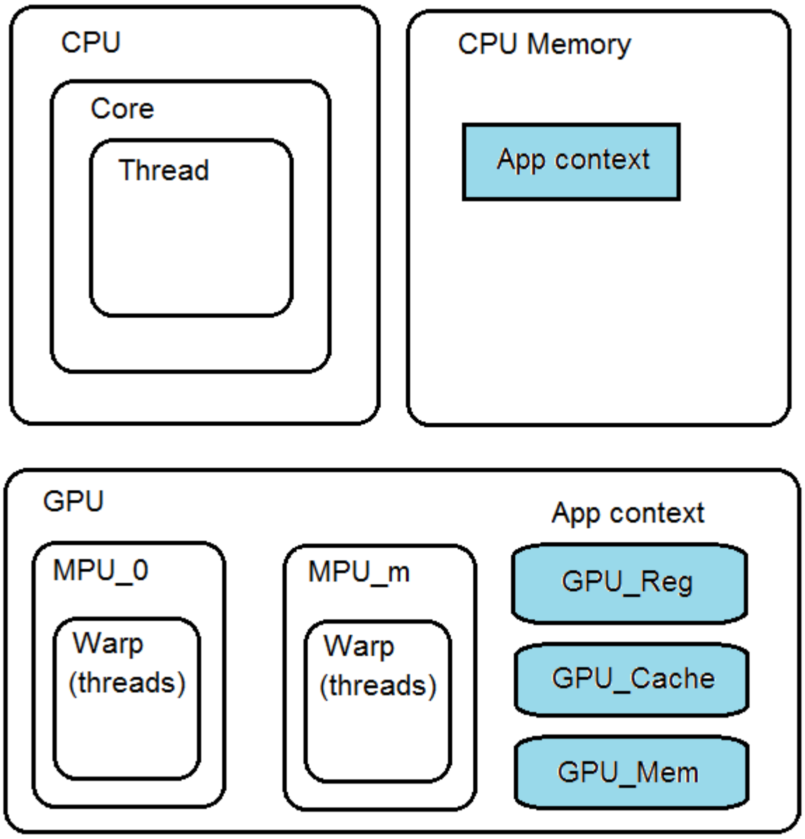

Рис. 3. Архитектура гибридного приложения

2. Пример – Задача численного интегрирования для сравнения производительности CPU и GPU.

```
Заголовочные файлы:
Заголовки для CPU программы (cudaf.h):
CUDA_C_PREF int MyCudaDevCount();
CUDA_C_PREF int MyCudaSetDev(int device);
CUDA_C_PREF int MyCudaProcess(int mp, int mt,
                              int ngb, int ngt,
                              int i1, int i2,
                              double a, double h,
                              double *s);

Заголовки для GPU программы (cudafs.h):
CUDA_C_PREF int MyCudaDevCount();
CUDA_C_PREF int MyCudaSetDev(int device);
CUDA_C_PREF int MyCudaProcess(int mp, int mt,
                              int ngb, int ngt,
                              int i1, int i2,
                              double a, double h,
                              double *s);

__global__ void MyCudaProcessReal(int i1, int i2,
                                  double a, double h,
                                  double *sum);
__device__ double MyFun(double x);
__device__ void MyRange(int np, int mp, int ia, int ib,
                        int *i1, int *i2, int *nc);

Код программы для CPU (main.c):
#define MAIN_FILE 1
#define CUDA_C_PREF
#define _GNU_SOURCE

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#include <errno.h>

#include <unistd.h>
#include <pthread.h>
#include <sched.h>

#include <mpi.h>
#include "cudaf.h"

// MPI error messages:

void mpierr(char *msg, int mp, const int n);
void mpierr(char *msg, int mp, const int n)
{
  fprintf(stderr,"Process %d message: %s\n",mp,msg);
  MPI_Abort(MPI_COMM_WORLD,n);
}

// Network:

static int np, mp, lname;
static char pname[MPI_MAX_PROCESSOR_NAME];

void MyRange(int np, int mp, int ia, int ib, int *i1, int *i2, int *nc);
void MyRange(int np, int mp, int ia, int ib, int *i1, int *i2, int *nc)
{
  if (np<2) { *i1=ia; *i2=ib; *nc=ib-ia+1; }
  else {
    int ni, mi, nn;
    nn = ib - ia + 1; ni = nn / np;  mi = nn - ni * np;
    if (mp+1<=mi)
      { *i1 = ia + mp * (ni+1); *i2 = *i1 + ni; }
    else
      { *i1 = ia + mi * (ni+1) + (mp-mi) * ni; *i2 = *i1 + ni - 1; }
    *nc = *i2 - *i1 + 1;
  }
  return;
}

// Job parameters:

static int mode = 0;
static int ni = 1024 * 1024 * 1024;
static double a = 0;
static double b = 1;
static double h;
static double sum = 0;

double MyFun(double x);
double MyFun(double x)
{
  double c = cos(x);
  c = c * tan(x-0.25);
  c = c * tan(x-0.5);
  c = c * exp(-x);
  c = c * log(1.25+x);
  return 4.0 / (1.0 + x * x * (2.0 - c) / (2.0 + c));
}

// CPU & GPU threads:

typedef struct tag_data_t {
  int nt, mt;
  double *sum;
} data_t;

static int nt, ng, ngb, ngt, nt_max, nd_max;
static data_t    *ThreadDtArray;
static pthread_t *ThreadHnArray;
static pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;

void* myjobt(void* d);
void* myjobt(void* d)
{
  int err, i, i1, i2, nc, nn, mm;
  int k, kk = 20;
  double s = 0;
  data_t* dd = (data_t *)d;
  int nt = dd->nt;
  int mt = dd->mt;

// Subdomain:

  nn = np * nt;
  mm = mp * nt + mt;
  MyRange(nn,mm,1,ni,&i1,&i2,&nc);

  fprintf(stderr,"[%04d,%04d,%04d]: np=%d nt=%d nn=%d\n",mp,mt,mm,np,nt,nn);
  fprintf(stderr,"[%04d,%04d,%04d]: i1=%d i2=%d nc=%d\n",mp,mt,mm,i1,i2,nc);

  if (mode==0) {
    for (k=0; k<kk; k++)
    for (i=i1; i<=i2; i++) {
      double x = a + h * (1.0 * i - 0.5);
      s += MyFun(x) * h;
    }
    s /= (1.0*kk);
  }
  else {
    err = MyCudaProcess(mp,mt,ngb,ngt,i1,i2,a,h,&s);
    fprintf(stderr,"[%04d,%04d,%04d]: cuda compute retcode is %d\n",mp,mt,mm,err);
  }

  pthread_mutex_lock(&mut); // lock

  *dd->sum += s;

  pthread_mutex_unlock(&mut); // unlock

  return 0;
}

void ThreadWork();
void ThreadWork()
{
  int i;

  if (!(ThreadHnArray = (pthread_t*) malloc(nt*sizeof(pthread_t))))
    mpierr("Not enough memory",mp,1);
  if (!(ThreadDtArray = (data_t*) malloc(nt*sizeof(data_t))))
    mpierr("Not enough memory",mp,2);

  for (i=0; i<nt; i++){
    (ThreadDtArray+i)->nt=nt;
    (ThreadDtArray+i)->mt=i;
    (ThreadDtArray+i)->sum = &sum;

    if (pthread_create(ThreadHnArray+i,0,myjobt,(void*)(ThreadDtArray+i)))
      mpierr("Can not create thread",mp,3);
  }

  for (i=0; i<nt; i++)
    if (pthread_join(ThreadHnArray[i],0))
      mpierr("Can not close thread",mp,4);
  free(ThreadHnArray);
  free(ThreadDtArray);

  return;
}

int MyGetCPUCount();
int MyGetCPUCount()
{
  int i = sysconf(_SC_NPROCESSORS_ONLN);
  return i;
}

int main(int argc, char *argv[])
{
  int i;
  double t0,t1,t2;

  MPI_Init(&argc, &argv);
  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Comm_size(MPI_COMM_WORLD, &np);
  MPI_Comm_rank(MPI_COMM_WORLD, &mp);
  MPI_Get_processor_name(pname, &lname);

  MPI_Barrier(MPI_COMM_WORLD);

  nt_max = MyGetCPUCount();
  nd_max = MyCudaDevCount();

  fprintf(stderr,
    "Netsize: %d, process: %d, system: %s, cpu_count: %d, gpu_count: %d\n",
    np,mp,pname,nt_max,nd_max);

  MPI_Barrier(MPI_COMM_WORLD);

//  if (nd_max > 0)
//    MyCudaInfo(mp);

  if (nd_max<1 || nt_max<1)
    mpierr("Bad count of devices",mp,10);

  mode = 0; // Computation mode
  nt   = 1; // Thread or GPU number
  ngb  = 1; // GPU block number
  ngt  = 1; // GPU total threads number

  if (mp==0)
    fprintf(stderr,
      "Usage: %s <mode> <cpu_threads> <gpu_blocks> <gpu_threads>\n",argv[0]);

  if (argc>1) {
    i = sscanf(argv[1],"%d",&mode);
    if (mode<1) mode = 0;        // CPU calculations
    else        mode = 1;        // GPU calculations
  }

  if (argc>2) {
    i = sscanf(argv[2],"%d",&nt);
    if (nt<1) nt = 1;
  }

  if (argc>3) {
    i = sscanf(argv[3],"%d",&ngb);
    if (ngb<1) ngb = 1;
  }

  if (argc>4) {
    i = sscanf(argv[4],"%d",&ngt);
    if (ngt<1) ngt = 1;
  }

  if (mode==0) {
    if (nt>nt_max) nt = nt_max;
  }
  else {
    if (nt>nd_max) nt = nd_max;
  }

  ng = ngb * ngt;

  h = (b-a) / ni;

  MPI_Barrier(MPI_COMM_WORLD);

  t0 = MPI_Wtime();

  ThreadWork();

  t1 = MPI_Wtime();

  if (np>1) {
    double sum0;
    MPI_Barrier(MPI_COMM_WORLD);
    sum0 = sum;
    MPI_Reduce(&sum0, &sum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    t2 = MPI_Wtime();
  }
  else {
    t2 = t1;
  }

  fprintf(stderr,
    "mode=%d np=%d nt=%d ng=%d ngb=%d ngt=%d sum=%le "
    "t1=%le t2=%le t3=%le mp=%d node=%s\n",
    mode, np, nt, ng, ngb, ngt, sum, t1-t0, t2-t1, t2-t0, mp, pname);

  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Finalize();
  return 0;
}
```

Код программы для GPU (cudaf.cu):

```
#define CUDA_C_PREF extern "C"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>
#include "cudafs.h"

CUDA_C_PREF double MyTime()
{
  struct timeval tv;
  struct timezone tz;
  gettimeofday(&tv, &tz);
  return (double)(tv.tv_sec) + (double)(tv.tv_usec)*1e-6;
}

CUDA_C_PREF int MyCudaDevCount()
{
  int deviceCount;
  cudaGetDeviceCount ( &deviceCount );
  cudaGetDeviceCount ( &deviceCount );
  return deviceCount;
}

CUDA_C_PREF int MyCudaSetDev(int device)
```

```
{
  cudaError_t get;
  get = cudaSetDevice(device);
  if (get != cudaSuccess)
    return -1;
  else
    return 0;
}

CUDA_C_PREF int MyCudaProcess(int mp, int mt,
                              int ngb, int ngt,
                              int i1, int i2,
                              double a, double h,
                              double *s)
{
  cudaError_t get;
  double *S_CPU;
  double *S_GPU;
  double t;
  int ndev, mdev, ldev;

  cudaGetDeviceCount (&ndev);
  cudaGetDeviceCount (&ndev);
  if (ndev < 1) return -1;
  if (ndev > 4) return -2;

  fprintf(stderr,"[%02d,%02d]: ndev=%d\n",mp,mt,ndev);

  mdev = mp % ndev;

  get = cudaSetDevice(mdev);
  if (get != cudaSuccess) return 1;

  get = cudaGetDevice(&ldev);
  if (get != cudaSuccess) return 2;

  fprintf(stderr,"[%02d,%02d]: mdev=%d %d\n",mp,mt,mdev,ldev);

  t = MyTime();

  get = cudaMallocHost ((void **)&S_CPU, ngb * sizeof(double));
  if (get != cudaSuccess) return 3;

  get = cudaMalloc ((void **)&S_GPU, ngb * sizeof(double));
  if (get != cudaSuccess) return 4;

  {
    dim3 threads (ngt, 1);
    dim3 blocks  (ngb, 1);
    MyCudaProcessReal<<<blocks, threads>>>(i1,i2,a,h,S_GPU);
  }

  get = cudaMemcpy (S_CPU, S_GPU, ngb * sizeof(double), cudaMemcpyDeviceToHost);
  if (get != cudaSuccess) return 5;

  {
    int i;
    double p = 0;
    for (i=0; i<ngb; i++) p += S_CPU[i];
    *s = p;
  }

  get = cudaFree(S_GPU);
  if (get != cudaSuccess) return 6;

  get = cudaFreeHost(S_CPU);
  if (get != cudaSuccess) return 7;
  t = MyTime() - t;

  fprintf(stderr,"[%02d,%02d]: time=%.6lf\n",mp,mt,t);

  return 0;
}

__device__ void MyRange(int np, int mp, int ia, int ib,
                        int *i1, int *i2, int *nc)
{
  if (np<2) { *i1=ia; *i2=ib; *nc=ib-ia+1; }
  else {
    int ni, mi, nn;
    nn = ib - ia + 1; ni = nn / np;  mi = nn - ni * np;
    if (mp+1<=mi)
      { *i1 = ia + mp * (ni+1); *i2 = *i1 + ni; }
    else
      { *i1 = ia + mi * (ni+1) + (mp-mi) * ni; *i2 = *i1 + ni - 1; }
    *nc = *i2 - *i1 + 1;
  }
  return;
}

__device__ double MyFun(double x)
{
  double c = cos(x);
  c = c * tan(x-0.25);
  c = c * tan(x-0.5);
  c = c * exp(-x);
  c = c * log(1.25+x);
  return 4.0 / (1.0 + x * x * (2.0 - c) / (2.0 + c));
}

__global__ void MyCudaProcessReal(int i1, int i2,
                                  double a, double h,
                                  double *DATAOUT)
{
  __shared__ double cache[1024]; // 1 <= thread count <= 512
  int cacheIndex = threadIdx.x;
  int ng = blockDim.x * gridDim.x;
  int mg = blockDim.x * blockIdx.x + threadIdx.x;
  int j, j1, j2, jc;
  int k, kk = 20;
  double s=0;

  MyRange(ng,mg,i1,i2,&j1,&j2,&jc);

  for (k=0; k<kk; k++)
  for (j=j1; j<=j2; j++) {
    double x = a + h * (1.0 * j - 0.5);
    s += MyFun(x) * h;
  }
  s /= (1.0*kk);

// Save to cache & synchronize:
  cache[cacheIndex] = s;
  __syncthreads();

// Reduction:

  j = blockDim.x/2;

  while (j != 0) {
    if (cacheIndex < j)
      cache[cacheIndex] += cache[cacheIndex + j];
    __syncthreads();
    j /= 2;
  }
  if (cacheIndex == 0)
    DATAOUT[blockIdx.x] = cache[0];

  __syncthreads();
}
```

Трансляция:

```
nvcc --compiler-options -O2 -arch sm_20 --ptxas-options=-v -c cudaf.cu
mpicc -O2 -c main.c
mpicc -o main.px cudaf.o main.o -L/common/cuda-8.0/lib64 -lm -lstdc++ -lcudart
```

Файл запусков (commands):

```
#
# CPU tasks:
#
# MPI:
mpirun -np  1 -resource gpu=3 -maxtime 90 -stderr ./main_cpu_001.err ./main.px 0 1
mpirun -np 12 -resource gpu=3 -maxtime 10 -stderr ./main_cpu_012.err ./main.px 0 1
mpirun -np 24 -resource gpu=3 -maxtime 10 -stderr ./main_cpu_024.err ./main.px 0 1
#
# MPI + OpenMP:
mpirun -np 1 -ppn 12 -resource gpu=3 -maxtime 10 -stderr ./main_cpu_112.err ./main.px 0 12
mpirun -np 2 -ppn 12 -resource gpu=3 -maxtime 10 -stderr ./main_cpu_124.err ./main.px 0 12
#
# GPU tasks:
#
# MPI + CUDA:
mpirun -np 1 -ppn 1 -resource gpu=3 -stderr ./main_gpu_001.err ./main.px 1 1 160 512
mpirun -np 2 -ppn 2 -resource gpu=3 -stderr ./main_gpu_002.err ./main.px 1 1 160 512
mpirun -np 3 -ppn 3 -resource gpu=3 -stderr ./main_gpu_003.err ./main.px 1 1 160 512
mpirun -np 4 -ppn 3 -resource gpu=3 -stderr ./main_gpu_004.err ./main.px 1 1 160 512
mpirun -np 5 -ppn 3 -resource gpu=3 -stderr ./main_gpu_005.err ./main.px 1 1 160 512
mpirun -np 6 -ppn 3 -resource gpu=3 -stderr ./main_gpu_006.err ./main.px 1 1 160 512
#
# MPI + OpenMP + CUDA:
mpirun -np 1 -ppn 1 -resource gpu=3 -stderr ./main_gpu_101.err ./main.px 1 1 160 512
mpirun -np 1 -ppn 1 -resource gpu=3 -stderr ./main_gpu_102.err ./main.px 1 2 160 512
mpirun -np 1 -ppn 1 -resource gpu=3 -stderr ./main_gpu_103.err ./main.px 1 3 160 512
mpirun -np 2 -ppn 1 -resource gpu=3 -stderr ./main_gpu_104.err ./main.px 1 2 160 512
mpirun -np 2 -ppn 1 -resource gpu=3 -stderr ./main_gpu_106.err ./main.px 1 3 160 512
#
```

Результаты запусков:

```
>grep "sum=3.158" *.err
main_cpu_001.err:mode=0 np= 1 nt= 1 ng=1 ngb=1 ngt=1         sum=3.158489e+00 t1=2.351120e+03 t2=0.000000e+00 t3=2.351120e+03 mp=0 node=node25
main_cpu_012.err:mode=0 np=12 nt= 1 ng=1 ngb=1 ngt=1         sum=3.158489e+00 t1=1.942838e+02 t2=4.271979e+01 t3=2.370036e+02 mp=0 node=node39
main_cpu_024.err:mode=0 np=24 nt= 1 ng=1 ngb=1 ngt=1         sum=3.158489e+00 t1=9.636746e+01 t2=2.376444e+01 t3=1.201319e+02 mp=0 node=node52
main_cpu_112.err:mode=0 np= 1 nt=12 ng=1 ngb=1 ngt=1         sum=3.158489e+00 t1=2.358813e+02 t2=0.000000e+00 t3=2.358813e+02 mp=0 node=node39
main_cpu_112.err:mode=0 np= 1 nt=12 ng=1 ngb=1 ngt=1         sum=3.158489e+00 t1=2.355502e+02 t2=0.000000e+00 t3=2.355502e+02 mp=0 node=node10
main_gpu_001.err:mode=1 np= 1 nt= 1 ng=81920 ngb=160 ngt=512 sum=3.158489e+00 t1=2.124470e+01 t2=0.000000e+00 t3=2.124470e+01 mp=0 node=node25
main_gpu_001.err:mode=1 np= 1 nt= 1 ng=81920 ngb=160 ngt=512 sum=3.158489e+00 t1=2.124757e+01 t2=0.000000e+00 t3=2.124757e+01 mp=0 node=node39
main_gpu_002.err:mode=1 np= 2 nt= 1 ng=81920 ngb=160 ngt=512 sum=3.158489e+00 t1=1.069607e+01 t2=3.504753e-05 t3=1.069611e+01 mp=0 node=node39
main_gpu_003.err:mode=1 np= 3 nt= 1 ng=81920 ngb=160 ngt=512 sum=3.158489e+00 t1=7.216949e+00 t2=1.165485e-02 t3=7.228604e+00 mp=0 node=node52
main_gpu_004.err:mode=1 np= 4 nt= 1 ng=81920 ngb=160 ngt=512 sum=3.158489e+00 t1=5.447716e+00 t2=1.421309e-02 t3=5.461929e+00 mp=0 node=node27
main_gpu_005.err:mode=1 np= 5 nt= 1 ng=81920 ngb=160 ngt=512 sum=3.158489e+00 t1=4.393088e+00 t2=1.068211e-02 t3=4.403770e+00 mp=0 node=node49
main_gpu_006.err:mode=1 np= 6 nt= 1 ng=81920 ngb=160 ngt=512 sum=3.158489e+00 t1=3.686455e+00 t2=3.510213e-02 t3=3.721557e+00 mp=0 node=node54
main_gpu_102.err:mode=1 np= 1 nt= 2 ng=81920 ngb=160 ngt=512 sum=3.158489e+00 t1=2.124617e+01 t2=0.000000e+00 t3=2.124617e+01 mp=0 node=node49
main_gpu_103.err:mode=1 np= 1 nt= 3 ng=81920 ngb=160 ngt=512 sum=3.158489e+00 t1=2.124790e+01 t2=0.000000e+00 t3=2.124790e+01 mp=0 node=node53
main_gpu_104.err:mode=1 np= 2 nt= 2 ng=81920 ngb=160 ngt=512 sum=3.158489e+00 t1=1.065907e+01 t2=4.911423e-05 t3=1.065911e+01 mp=0 node=node54
main_gpu_106.err:mode=1 np= 2 nt= 3 ng=81920 ngb=160 ngt=512 sum=3.158489e+00 t1=1.065624e+01 t2=7.388496e-02 t3=1.073012e+01 mp=0 node=node56
```