Методика создания переносимых программ математического моделирования для различных типов гибридных суперЭВМ

Е.А.Генрих, А.В.Снытников
Лаборатория Параллельных Алгоритмов Решения Больших Задач
Институт Вычислительной Математики и Математической Геофизики
СО РАН

Актуальность

- Желание иметь возможность использовать наиболее мощные гибридные суперЭВМ, а такие сейчас строятся (в том числе) на базе Intel Xeon Phi
- Доклад А.О.Лациса “Что же делать с этим многообразием суперкомпьютерных миров?” на конференции “Научный сервис в сети Интернет-2014”
- Большое количество различных суперкомпьютерных архитектур приводит к необходимости разрабатывать отдельный вариант программы под каждую из них

Принципиальные вопросы

- Необходимо решить вопрос о переносе между **наиболее распространенными типами** суперкомпьютерных архитектур
  - Кластера на основе Nvidia Kepler
  - Кластера на основе Intel Xeon Phi
  - Кластера на основе Intel Xeon или Sun UltraSPARC
- Рассматривается перенос программы с GPU на Intel Xeon Phi (**не наоборот!!!**)
- Не рассматривается (**пока!**) вопрос оптимизации под ту или иную архитектуру

Основные проблемы переноса с CUDA на MIC

- Компиляция ядер CUDA
- и в особенности вызовов ядер CUDA без компилятора Nvidia
- Пропуск операций копирования между различными видами памяти в CUDA
- Определение типов данных и ключевых слов, входящих в расширение языка C, используемое в CUDA

Технология переноса программ

- Архитектурно-зависимые участки кода
  - Сводятся к минимуму
  - Оформляются в виде процедур
  - Выносятся во внешнюю подключаемую библиотеку
- Таким образом в тексте программы присутствует некий обобщенный вызов процедуры, который приобретает конкретную форму при компиляции в зависимости
  - От компилятора
  - От архитектуры

Пример: моделирование динамики плазмы

$$\vec{p}=\gamma\vec{v},\ \gamma^{-1}=\sqrt{1-v^2}$$

$$\vec{F}=q_{i,e}\left(\vec{E}+\frac{1}{c}[\vec{v},\vec{B}]\right)$$

$$\vec{j}=\sum_{i,e}q_{i,e}\int f_{i,e}\vec{v}\,d\vec{v}$$

$$\rho=\sum_{i,e}q_{i,e}\int f_{i,e}\,d\vec{v}$$

Основные уравнения

$$\frac{\partial f_{i,e}}{\partial t}+\vec{v}\frac{\partial f_{i,e}}{\partial \vec{x}}+\vec{F}\frac{\partial f_{i,e}}{\partial \vec{v}}=0$$

$$\nabla\times\vec{B}=4\pi\vec{j}+\frac{1}{c}\frac{\partial\vec{E}}{\partial t}$$

$$\nabla\times\vec{E}=-\frac{1}{c}\frac{\partial\vec{B}}{\partial t}$$

$$\nabla\vec{E}=4\pi\rho$$

$$\nabla\vec{B}=0$$

Начальные условия

- $\rho_e = 1000$, $\rho_b = 1$

  $\rho = \rho_e + \rho_b$
- Импульсы электронов плазмы:

  p_x, p_y, p_z — максвелловское распределение, $\sigma = T_e = 1.0$

$$f=\exp\left(\frac{-p^2}{\sigma}\right)$$

- Импульсы ионов плазмы:0
- Импульс электронов пучка:

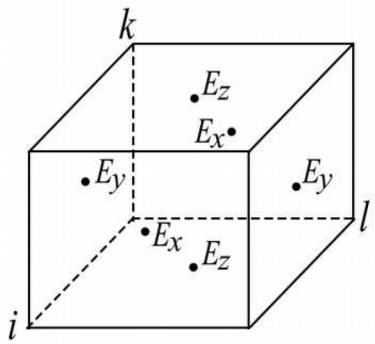
  $p_x = 50$ $p_y = p_z = 0$

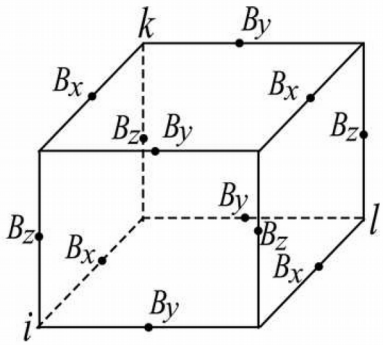
Эйлеров этап метода частиц в ячейках: **вычисление полей**

Схема Лэнгдона-Лазински

$$\frac{B^{m+1/2}-B^{m-1/2}}{\tau}=-\mathrm{rot}_h E^m$$

$$\frac{E^{m+1}-E^m}{\tau}=\mathrm{rot}_h B^{m+1/2}-j^{m+1/2}$$



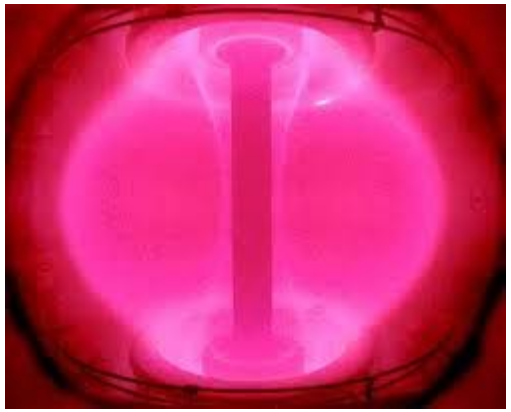


$$\rho=\sum q_p v_p^{m+1/2}\overline{R}(x_p,x_i)$$

$$\frac{\rho^{m+1}-\rho^m}{\tau}+\mathrm{div}_h j^{m+1/2}=0$$

$$\mathrm{div}_h B=\frac{B_{x,i+1/2,k,l}-B_{x,i-1/2,k,l}}{h_x}+\frac{B_{y,i,k+1/2,l}-B_{y,i,k-1/2,l}}{h_y}+\frac{B_{z,i,k,l+1/2}-B_{z,i,k,l-1/2}}{h_z}$$

$$\mathrm{rot}_h B=\begin{pmatrix}\frac{B_{z,i,k,l-1/2}-B_{z,i,k-1,l-1/2}}{h_y}-\frac{B_{y,i,k-1/2,l}-B_{y,i,k-1,l-1}}{h_z}\\ \frac{B_{x,i-1/2,k,l}-B_{x,i-1/2,k,l-1}}{h_z}-\frac{B_{z,i,k,l-1/2}-B_{z,i-1,k,l-1/2}}{h_x}\\ \frac{B_{y,i,k-1/2,l}-B_{y,i-1/2,k,l}}{h_x}-\frac{B_{x,i-1/2,k,l}-B_{x,i-1/2,k-1,l}}{h_x}\end{pmatrix}$$



Вшивков В.А. и др.,
Вычислительные технологии, Том 6, № 2, 2001.

Сведение к минимуму архитектурно-зависимых участков кода

- В коде имеется 15-20 вызовов небольших вычислительных фрагментов,
  - выполняющих обработку узлов сетки, модельных частиц, границ расчетной области
  - оформленных в виде ядер CUDA
  - таким образом, не подлежащих компиляции в помощью компилятора Intel и пр.
- Идея: сделать такой “непроходной” участок кода по крайней мере единственным

Универсальная процедура запуска

```
int Kernel_Launcher(
Cell<Particle>  **cells,KernelParams *params,
unsigned int grid_size_x,unsigned int grid_size_y,unsigned int grid_size_z,
        unsigned int block_size_x,unsigned int block_size_y,unsigned int block_size_z,
        int shmem_size,
        SingleNodeFunctionType h_snf,char *name)
{
struct timeval tv1,tv2;
#ifdef __CUDACC__
    dim3
blocks(grid_size_x,grid_size_y,grid_size_z),threads(block_size_x,block_size_y,block_size_z
);

    gettimeofday(&tv1,NULL);
    GPU_Universal_Kernel<<<blocks,threads,shmem_size>>>(cells,params,h_snf);
    DeviceSynchronize();
    gettimeofday(&tv2,NULL);
#else
    char hostname[1000];
    gethostname(hostname,1000);

#ifdef OMP_OUTPUT
    printf("function %s executed on %s \n",name,hostname);
#endif

    gettimeofday(&tv1,NULL);

    omp_set_num_threads(OMP_NUM_THREADS);

#pragma omp parallel for
    for(int i = 0;i < grid_size_x;i++)
    {
            ….
                    h_snf(cells,params,i,j,k,i1,j1,k1);
….
```

Работа с расширениями языка C (CUDA C/C++)

- Специальные типы данных: int3, double3, dim3,...
  - доопределить, при возможности скопировать файл cuda.h
- Специальные ключевые слова: __global__, __device__, …
  - замаскировать с помощью директив условной компиляции
- Функции CUDA API: копирование из одного типа памяти в другой, обработка ошибок и пр.
  - Оформить в виде универсальной процедуры, пригодной для компиляции на обоих архитектурах

Конкретный механизм реализации
технологии переноса программ

- Процедурные переменные C/C++
- Директивы условной компиляции
- Универсальный набор параметров счетных процедур

Процедурные переменные C/C++

```
typedef void (*SingleNodeFunctionType)(GPUCell<Particle>  **cells,KernelParams *params,
        unsigned int bk_nx,unsigned int bk_ny,unsigned int bk_nz,
        unsigned int nx,unsigned int ny,unsigned int nz
        );
```

Тип универсальной счетной процедуры

```
__device__ void GPU_eme_SingleNode(

                    Cell<Particle>  **cells,
                    KernelParams *params,
                    unsigned int bk_nx,unsigned int bk_ny,unsigned int bk_nz,
                    unsigned int tnx,unsigned int tny,unsigned int tnz
        )
{
    unsigned int nx = bk_nx*params->blockDim_x + tnx;
    unsigned int ny = bk_ny*params->blockDim_y + tny;
    unsigned int nz = bk_nz*params->blockDim_z + tnz;
    Cell<Particle>  *c0 = cells[0];

    emeElement(c0,params->i_s+nx,params->l_s+ny,params->k_s+nz,params->E,params->H1,params->H2,
            params->J,params->c1,params->c2,params->tau,
            params->dx1,params->dy1,params->dz1,
            params->dx2,params->dy2,params->dz2);
}
```

Syntax error
Press 'F2' for focus

Пример процедуры с унифицированной сигнатурой: вычисление электрического поля в узле сетки

```
void emeElement(Cell<Particle> *c,int i,int l,int k,double *E,double *H1, double *H2,
        double *J,double c1,double c2, double tau,
        int dx1,int dy1,int dz1,int dx2,int dy2,int dz2
        )
{
   int n  = c->getGlobalCellNumber(i,l,k);
  int n1 = c->getGlobalCellNumber(i+dx1,l+dy1,k+dz1);
  int n2 = c->getGlobalCellNumber(i+dx2,l+dy2,k+dz2);

  E[n] += c1*(H1[n] - H1[n1]) - c2*(H2[n] - H2[n2]) - tau*J[n];
```

Реальную работу выполняет эта процедура

Директивы условной компиляции

```
 int MemoryCopy(void* dst,void *src,size_t size,int dir)
{
    int err = 0;

#ifdef __CUDACC__
    cudaMemcpyKind cuda_dir;

    if(dir == HOST_TO_DEVICE) cuda_dir = cudaMemcpyHostToDevice;
    if(dir == HOST_TO_HOST) cuda_dir = cudaMemcpyHostToHost;
    if(dir == DEVICE_TO_HOST) cuda_dir = cudaMemcpyDeviceToHost;
    if(dir == DEVICE_TO_DEVICE) cuda_dir = cudaMemcpyDeviceToDevice;



    return err = (int)cudaMemcpy(dst,src,size,cuda_dir);
#else
    memcpy(dst,src,size);
#endif
    return err;
}
```

Универсальный набор параметров

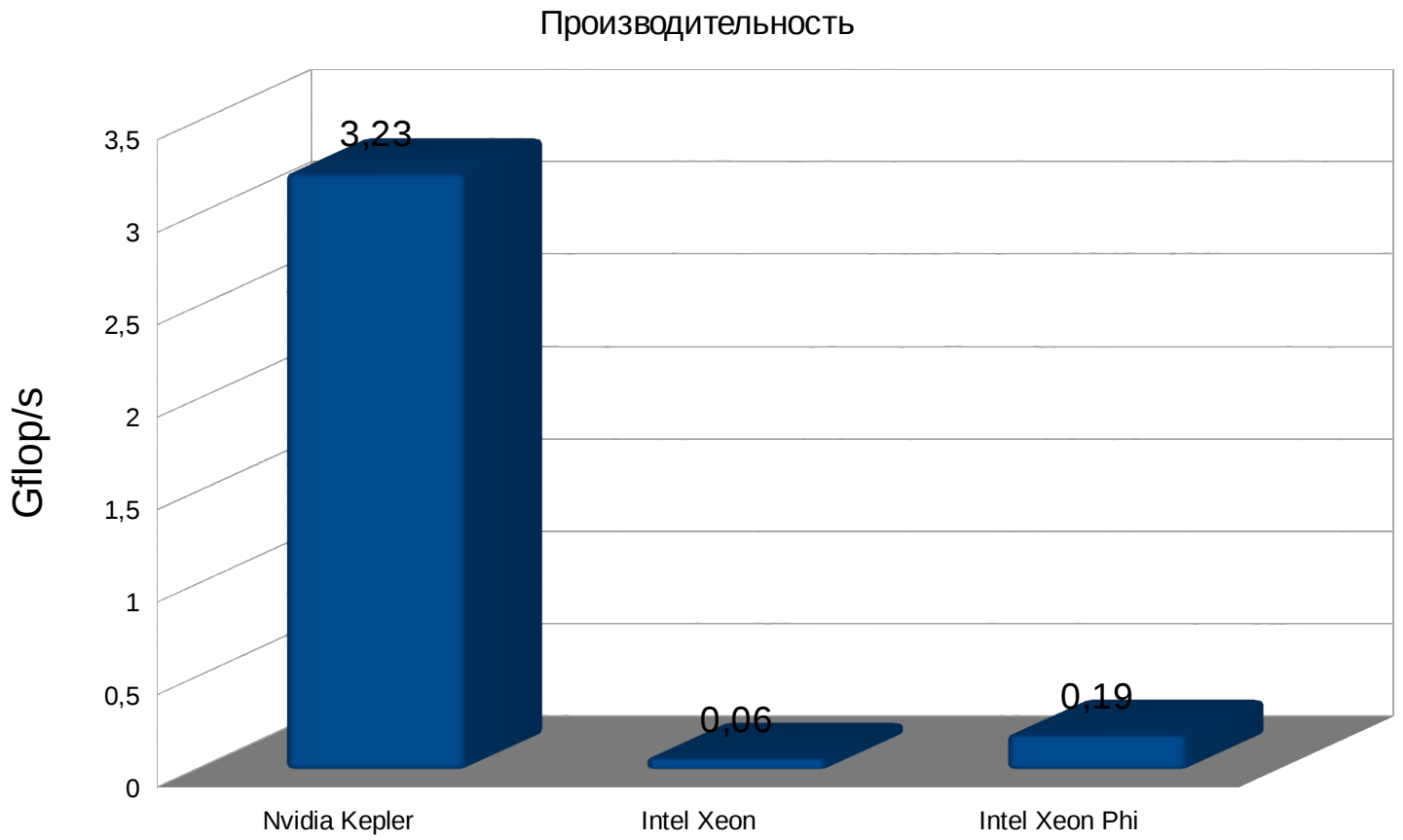
```
typedef struct {
double d_ee; //electric energy
double *d_Ex,*d_Ey,*d_Ez; // electric field
double *d_Hx,*d_Hy,*d_Hz; // magnetic field
double *d_Jx,*d_Jy,*d_Jz; // currents
double *d_Rho;
int nt;                                  // timestep
int *d_stage;                            // checking system (e.g. for flow-out
particles)
int *numbers;                             // number of particles in each cell
double mass,q_mass;
double *d_ctrlParticles;
int jmp;
//                                           for periodical FIELDS
int i_s,k_s;                          //
double *E;                               //the field
int dir;                                 // the direction being processed
int to,from;                             // the range along the direction

//                                           for periodical CURRENTS
int dirE;                             // directions
int N;                              // variables

//                                        electric field solver
int l_s;                           // variables
double *H1,*H2;                           // magnetic fields (orthogonal)
double *J;                               // current
double c1,c2,tau;                        //grid steps squared
int dx1,dy1,dz1,dx2,dy2,dz2;             //shifts

//                                           magnetic field solver
double *Q;                               // magnetic field at half-step
double *H;                               // magnetic field
double *E1,*E2;                          // electric fields (orthogonal)
int particles_processed_by_a_single_thread;
unsigned int blockDim_x,blockDim_y,blockDim_z; // block for field solver
} KernelParams;
```

Перенос vs Оптимизация

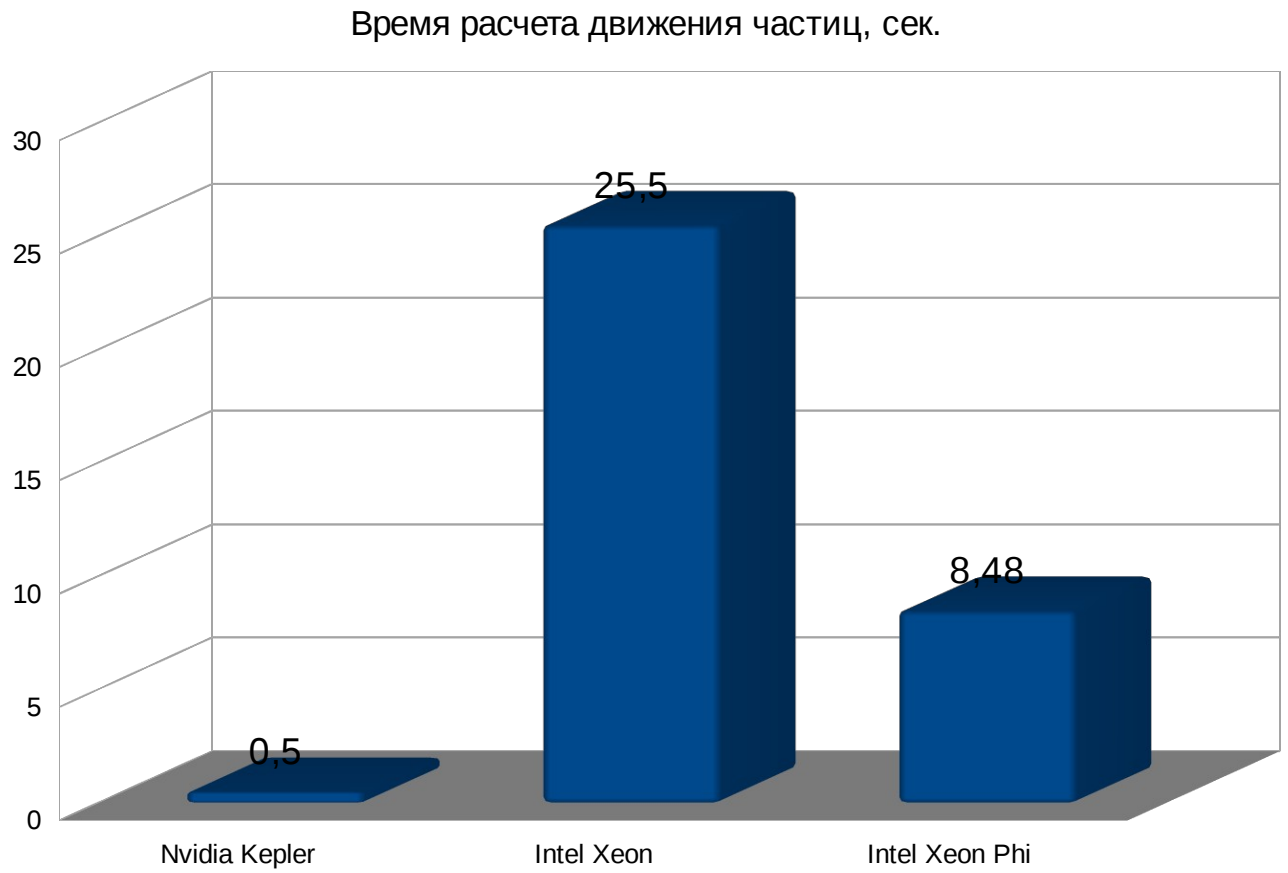
- Насколько серьезно отличается архитектура CUDA от MIC и, соответственно, насколько отличаются стратегии оптимизации?
  - Основным вопросом в обоих случаях является локальность данных
  - Векторизация циклов для MIC во всяком случае, не ухудшит производительность CUDA
- Какие элементы технологии переноса могут ухудшить производительность
  - Процедурные переменные не поддерживаются при CUDA Compute Capability < 3.5
  - Имитация вызова ядер CUDA с помощью директив OpenMP для MIC требует дополнительных индивидуальных настроек для достижения высокой производительности

Оценка производительности

~250 операций на модельную частицу



Производительность



- 6.4 млн. частиц (в среднем 3.5-4 тыс. в одной ячейке)
- Процессор Intel Xeon имеет 6 ядер (Intel Xeon Phi – 61)
- т.е. всего в 10 раз больше ядер, причем существенно менее мощных
- т.о. ускорение в 3 раза - это приемлемо

Основные принципы технологии

Или как писать программу, чтобы она легко переносилась?

1) Оформлять все фрагменты программы, обрабатывающие узлы сетки, частицы, элементы базиса, собств. функции и пр. в виде небольших процедур с унифицированной сигнатурой

2) Вызывать эти процедуры не в ядрах CUDA или в циклах OpenMP, а с помощью универсальной процедуры запуска

3) Архитектурно-зависимые функции (копирование данных, определение ошибки и пр.) вызывать не напрямую, а через библиотеку подстановок archAPI.h