Program acceleration with GPU using CUDA

Dr. Talgat Manglayev

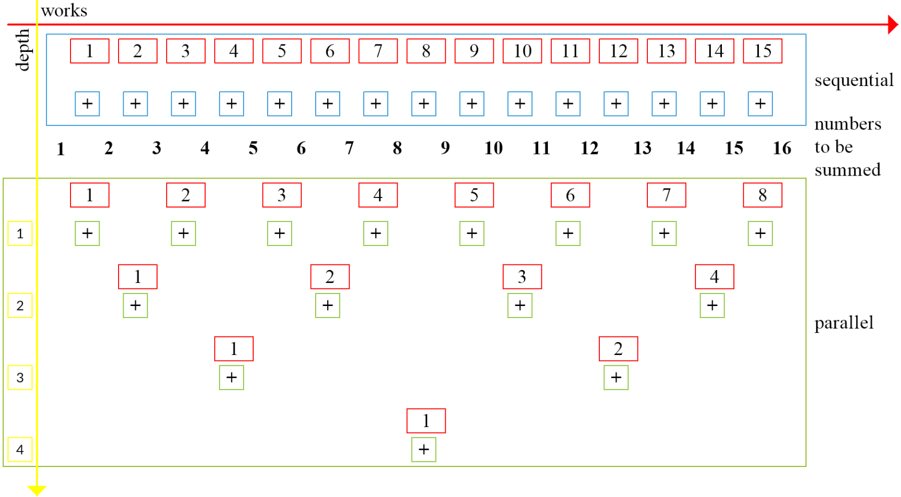
University of Helsinki, Department of Physics

UNIVERSITY OF HELSINKI

TABLE OF CONTENTS

- Introduction
  - About work and depth
  - Performance and communication cost
  - Sparse Matrix to Vector Multiplication
- CUDA
  - GPU acceleration
  - Connect CUDA to existing application
  - Add custom library to CUDA application
  - CUDA programming model
- Vlasiator
  - Science case: Vlasiator
  - Vlasiator: state-of-the-art
  - Fluid vs. kinetic
  - Putting the Vlasov into Vlasiator
- Conclusion

Sequential and parallel programming for summing numbers from 1 to 16



Work and Depth (Step)

WORK AND **DEPTH (STEP)** are abstract measures to define virtual performance model.

WORK - the total number of operations executed by a computation.

DEPTH (STEP) - the longest chain of sequential dependencies in the computation.

Work and Depth (Step)

Roughly speaking, when executing a set of tasks in parallel,

The total work is the sum of the work of the tasks

The total depth is the maximum of the depth of the tasks.

When executing tasks sequentially, both the work and the depth are summed.

TABLE OF CONTENTS

- Introduction
  - About work and depth
  - Performance and communication cost
  - Sparse Matrix to Vector Multiplication
- CUDA
  - GPU acceleration
  - Connect CUDA to existing application
  - Add custom library to CUDA application
  - CUDA programming model
- Vlasiator
  - Science case: Vlasiator
  - Vlasiator: state-of-the-art
  - Fluid vs. kinetic
  - Putting the Vlasov into Vlasiator
- Conclusion

Performance and Communication Cost

For computation with work - W and depth - D

$$\frac{W}{P} \leq T < \frac{W}{P} + D \tag{1}$$

P - number of processors, T - Time

E.B. Guy "Programming Parallel Algorithms", Communications of the ACM (March, 1996).

Performance and Communication Cost

(Latency - the time between making a remote request and receiving the reply)

work - W depth - D and latency - L

$$\frac{W}{P} \leq T < \frac{W}{P} + L * D \qquad (2)$$

P - number of processors, T - Time

E.B. Guy "Programming Parallel Algorithms", Communications of the ACM (March, 1996).

Performance and Communication Cost

Bandwidth - the rate at which a processor can access memory.

E.B. Guy "Programming Parallel Algorithms", Communications of the ACM (March, 1996).

Performance and Communication Cost

Let us consider work of primitive operation

$$W(a+b) = 1 + W(a) + W(b) \tag{3}$$

E.B. Guy "Programming Parallel Algorithms", Communications of the ACM (March, 1996).

Performance and Communication Cost

Parallelism rules apply to each

$$W(\{e_1(a) : a \, in \, e_2\}) = 1 + W(e_2) + \sum_{a \, in \, e_2} W(e_1(a)) \qquad (4)$$

$$D(\{e_1(a) : a \, in \, e_2\}) = 1 + D(e_2) + \max_{a \, in \, e_2} D(e_1(a)) \qquad (5)$$

E.B. Guy "Programming Parallel Algorithms", Communications of the ACM (March, 1996).

TABLE OF CONTENTS

- Introduction
  - About work and depth
  - Performance and communication cost
  - Sparse Matrix to Vector Multiplication
- CUDA
  - GPU acceleration
  - Connect CUDA to existing application
  - Add custom library to CUDA application
  - CUDA programming model
- Vlasiator
  - Science case: Vlasiator
  - Vlasiator: state-of-the-art
  - Fluid vs. kinetic
  - Putting the Vlasov into Vlasiator
- Conclusion

Sparse Matrix to Vector Multiplication

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} -1 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \cdot (-1) + 0 \cdot 1 + (-1) \cdot 2 \\ 2 \cdot (-1) + 0 \cdot 1 + 0 \cdot 2 \\ 0 \cdot (-1) + (-1) \cdot 1 + 0 \cdot 2 \end{bmatrix}$$

Sparse Matrix to Vector Multiplication

Given Array

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

- Non-zero elements $\begin{bmatrix} 1 & -1 & 2 & -1 \end{bmatrix}$
- Column id of non-zero elements $\begin{bmatrix} 0 & 2 & 0 & 1 \end{bmatrix}$
- Column id of non-zero elements array, where value is the first non-zero element in each row of given array $\begin{bmatrix} 0 & 2 & 3 \end{bmatrix}$

Sparse Matrix to Vector Multiplication

Given Array Vector

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} -1 \\ 1 \\ 2 \end{bmatrix}$$

- Column id of non-zero elements $\begin{bmatrix} 0 & 2 & 0 & 1 \end{bmatrix}$
- Non-zero elements $\begin{bmatrix} 1 & -1| & 2| & -1 \end{bmatrix}$
- Array from Vector Values with id from Column id $\begin{bmatrix} -1 & 2 & -1 & 1 \end{bmatrix}$

Sparse Matrix to Vector Multiplication

Given Array Vector

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} -1 \\ 1 \\ 2 \end{bmatrix}$$

- ► Column id of non-zero elements $\begin{bmatrix} 0 & 2 & 0 & 1 \end{bmatrix}$
- ► Non-zero elements $\begin{bmatrix} 1 & -1| & 2| & -1 \end{bmatrix}$
- ► Array from Vector Values with id from Column id $\begin{bmatrix} -1 & 2 & -1 & 1 \end{bmatrix}$

Sparse Matrix to Vector Multiplication

$$\begin{bmatrix}1 & -1 & 2 & -1\end{bmatrix} \cdot \begin{bmatrix}-1 & 2 & -1 & 1\end{bmatrix} = \begin{bmatrix}1 \cdot (-1) + (-1) \cdot 2 \,|\, 2 \cdot (-1) \,|\, (-1) \cdot 1\end{bmatrix}$$

$$\begin{bmatrix}1 & 0 & -1 \\ 2 & 0 & 0 \\ 0 & -1 & 0\end{bmatrix} \cdot \begin{bmatrix}-1 \\ 1 \\ 2\end{bmatrix} = \begin{bmatrix}1 \cdot (-1) + 0 \cdot 1 + (-1) \cdot 2 \\ 2 \cdot (-1) + 0 \cdot 1 + 0 \cdot 2 \\ 0 \cdot (-1) + (-1) \cdot 1 + 0 \cdot 2\end{bmatrix}$$

Sparse Matrix to Vector Multiplication

Parallel multiplication and addition operations

$$\begin{bmatrix} 1 & -1 & 2 & -1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 2 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 \cdot (-1) + (-1) \cdot 2 | 2 \cdot (-1) | (-1) \cdot 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} -1 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \cdot (-1) + 0 \cdot 1 + (-1) \cdot 2 \\ 2 \cdot (-1) + 0 \cdot 1 + 0 \cdot 2 \\ 0 \cdot (-1) + (-1) \cdot 1 + 0 \cdot 2 \end{bmatrix}$$

C++ and CUDA Complexity

$$\left[1 \cdot (-1) + (-1) \cdot 2 \middle| 2 \cdot (-1) \middle| (-1) \cdot 1\right]$$

C++ complexity

$O(n^2)$ (because nested loop)

CUDA complexity

Computation $O(logn)$

Sum of work $O(n)$

TABLE OF CONTENTS

- Introduction
  - About work and depth
  - Performance and communication cost
  - Sparse Matrix to Vector Multiplication
- CUDA
  - GPU acceleration
  - Connect CUDA to existing application
  - Add custom library to CUDA application
  - CUDA programming model
- Vlasiator
  - Science case: Vlasiator
  - Vlasiator: state-of-the-art
  - Fluid vs. kinetic
  - Putting the Vlasov into Vlasiator
- Conclusion

GPU acceleration

GPU acceleration key features

- independent data structure for parallel execution
- large enough data to cover communication cost

GPU acceleration

- independent data structure for parallel execution
- large enough data to cover communication cost

Finding element of Fibonacci sequence

0 1 1 2 3 5 8 13 21 ...

GPU acceleration

- independent data structure for parallel execution
- large enough data to cover communication cost

$$\begin{bmatrix} 1\cdot(-1)+0\cdot 1+(-1)\cdot 2 \\ 2\cdot(-1)+0\cdot 1+0\cdot 2 \\ 0\cdot(-1)+(-1)\cdot 1+0\cdot 2 \end{bmatrix}$$

GPU acceleration

- independent data structure for parallel execution
- large enough data to cover communication cost

$$\begin{bmatrix} 1 \cdot (-1) + 0 \cdot 1 + (-1) \cdot 2 \\ 2 \cdot (-1) + 0 \cdot 1 + 0 \cdot 2 \\ 0 \cdot (-1) + (-1) \cdot 1 + 0 \cdot 2 \end{bmatrix}$$

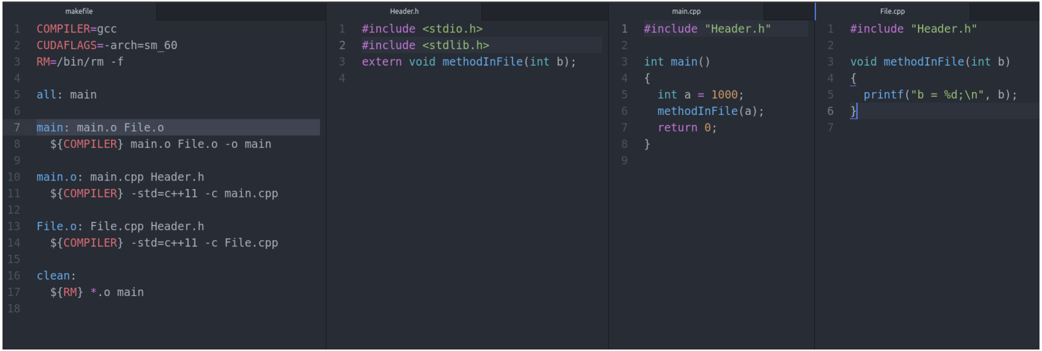
TABLE OF CONTENTS

- Introduction
  - About work and depth
  - Performance and communication cost
  - Sparse Matrix to Vector Multiplication
- CUDA
  - GPU acceleration
  - Connect CUDA to existing application
  - Add custom library to CUDA application
  - CUDA programming model
- Vlasiator
  - Science case: Vlasiator
  - Vlasiator: state-of-the-art
  - Fluid vs. kinetic
  - Putting the Vlasov into Vlasiator
- Conclusion

Connect CUDA to existing application

- Create a new method in .cu file
- Call this file from .cpp file

https://github.com/Talgat-qypshaq/cuda-sandbox/tree/master/test-c



makefile

```
COMPILER=gcc
CUDAFLAGS=-arch=sm_60
RM=/bin/rm -f

all: main

main: main.o File.o
  ${COMPILER} main.o File.o -o main

main.o: main.cpp Header.h
  ${COMPILER} -std=c++11 -c main.cpp

File.o: File.cpp Header.h
  ${COMPILER} -std=c++11 -c File.cpp

clean:
  ${RM} *.o main
```

Header.h

```
#include <stdio.h>
#include <stdlib.h>
extern void methodInFile(int b);
```

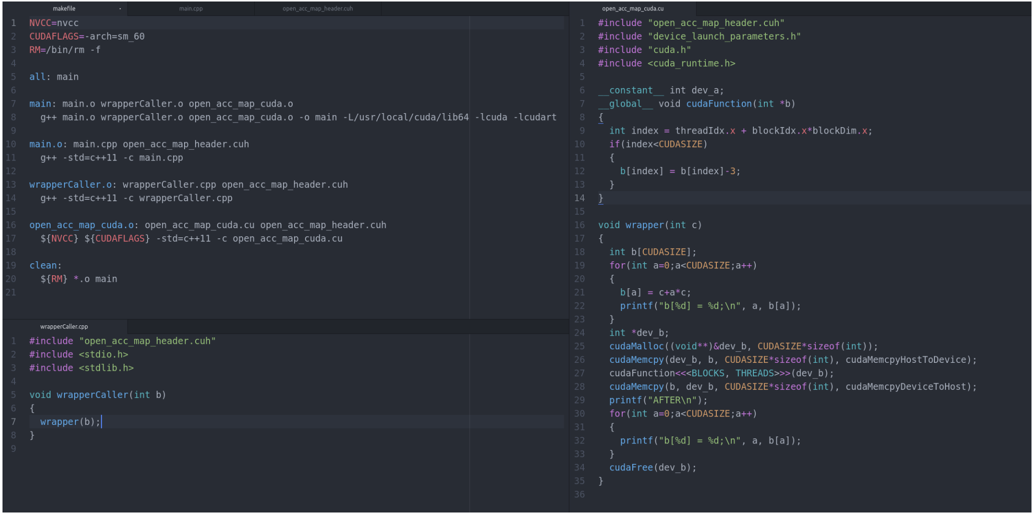
main.cpp

```
#include "Header.h"

int main()
{
  int a = 1000;
  methodInFile(a);
  return 0;
}
```

File.cpp

```
#include "Header.h"

void methodInFile(int b)
{
  printf("b = %d;\n", b);
}
```

Connect CUDA to existing application



```makefile
NVCC=nvcc
CUDAFLAGS=-arch=sm_60
RM=/bin/rm -f

all: main

main: main.o wrapperCaller.o open_acc_map_cuda.o
  g++ main.o wrapperCaller.o open_acc_map_cuda.o -o main -L/usr/local/cuda/lib64 -lcuda -lcudart

main.o: main.cpp open_acc_map_header.cuh
  g++ -std=c++11 -c main.cpp

wrapperCaller.o: wrapperCaller.cpp open_acc_map_header.cuh
  g++ -std=c++11 -c wrapperCaller.cpp

open_acc_map_cuda.o: open_acc_map_cuda.cu open_acc_map_header.cuh
  ${NVCC} ${CUDAFLAGS} -std=c++11 -c open_acc_map_cuda.cu

clean:
  ${RM} *.o main
```

```cpp
#include "open_acc_map_header.cuh"
#include <stdio.h>
#include <stdlib.h>

void wrapperCaller(int b)
{
  wrapper(b);
}
```

```cpp
#include "open_acc_map_header.cuh"
#include "device_launch_parameters.h"
#include "cuda.h"
#include <cuda_runtime.h>

__constant__ int dev_a;
__global__ void cudaFunction(int *b)
{
  int index = threadIdx.x + blockIdx.x*blockDim.x;
  if(index<CUDASIZE)
  {
    b[index] = b[index]-3;
  }
}

void wrapper(int c)
{
  int b[CUDASIZE];
  for(int a=0;a<CUDASIZE;a++)
  {
    b[a] = c+a*c;
    printf("b[%d] = %d;\n", a, b[a]);
  }
  int *dev_b;
  cudaMalloc((void**)&dev_b, CUDASIZE*sizeof(int));
  cudaMemcpy(dev_b, b, CUDASIZE*sizeof(int), cudaMemcpyHostToDevice);
  cudaFunction<<<BLOCKS, THREADS>>>(dev_b);
  cudaMemcpy(b, dev_b, CUDASIZE*sizeof(int), cudaMemcpyDeviceToHost);
  printf("AFTER\n");
  for(int a=0;a<CUDASIZE;a++)
  {
    printf("b[%d] = %d;\n", a, b[a]);
  }
  cudaFree(dev_b);
}
```

TABLE OF CONTENTS

- Introduction
  - About work and depth
  - Performance and communication cost
  - Sparse Matrix to Vector Multiplication
- CUDA
  - GPU acceleration
  - Connect CUDA to existing application
  - Add custom library to CUDA application
  - CUDA programming model
- Vlasiator
  - Science case: Vlasiator
  - Vlasiator: state-of-the-art
  - Fluid vs. kinetic
  - Putting the Vlasov into Vlasiator
- Conclusion

Add custom library to CUDA application

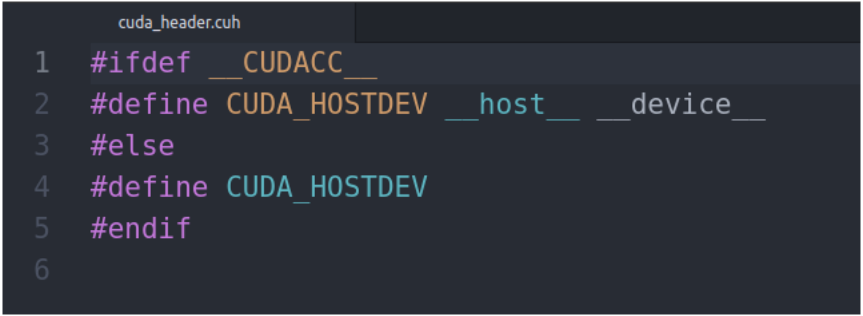
- Add condition to distinguish between device and host
- Rewrite custom library

https://github.com/Talgat-qypshaq/vlasiator/tree/openacc2

Add custom library to CUDA application

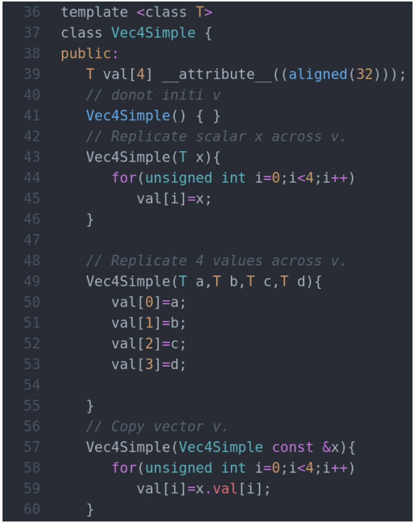
- Add condition to distinguish between device and host
- Rewrite custom library

https://github.com/Talgat-qypshaq/vlasiator/tree/openacc2

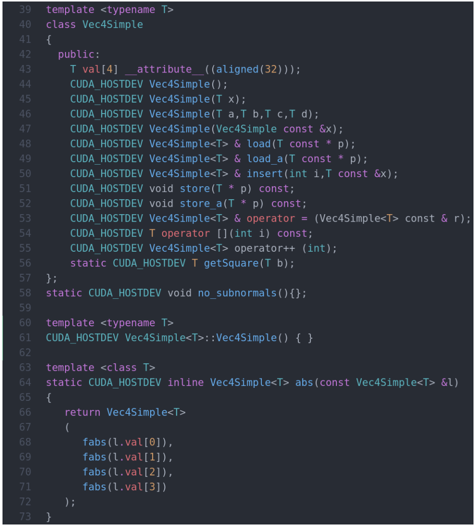


```
#ifdef __CUDACC__
#define CUDA_HOSTDEV __host__ __device__
#else
#define CUDA_HOSTDEV
#endif

```

Add custom library to CUDA application



```
template <class T>
class Vec4Simple {
public:
   T val[4] __attribute__((aligned(32)));
   // donot initi v
   Vec4Simple() { }
   // Replicate scalar x across v.
   Vec4Simple(T x){
      for(unsigned int i=0;i<4;i++)
         val[i]=x;
   }

   // Replicate 4 values across v.
   Vec4Simple(T a,T b,T c,T d){
      val[0]=a;
      val[1]=b;
      val[2]=c;
      val[3]=d;

   }
   // Copy vector v.
   Vec4Simple(Vec4Simple const &x){
      for(unsigned int i=0;i<4;i++)
         val[i]=x.val[i];
   }
```

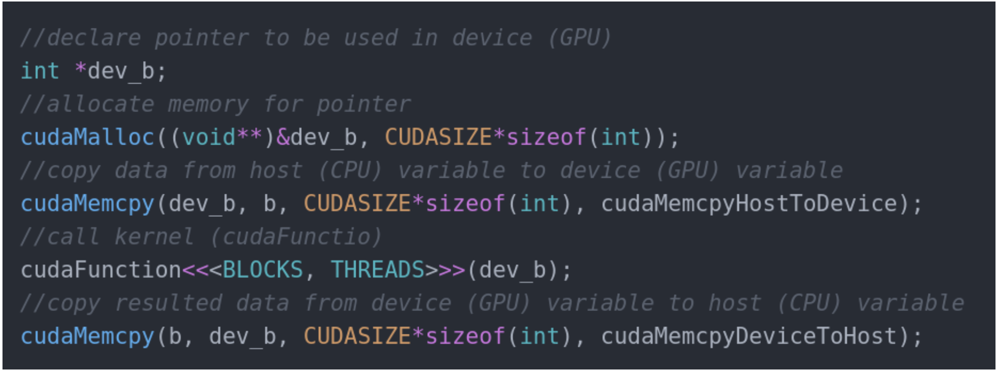


```
template <typename T>
class Vec4Simple
{
  public:
    T val[4] __attribute__((aligned(32)));
    CUDA_HOSTDEV Vec4Simple();
    CUDA_HOSTDEV Vec4Simple(T x);
    CUDA_HOSTDEV Vec4Simple(T a,T b,T c,T d);
    CUDA_HOSTDEV Vec4Simple(Vec4Simple const &x);
    CUDA_HOSTDEV Vec4Simple<T> & load(T const * p);
    CUDA_HOSTDEV Vec4Simple<T> & load_a(T const * p);
    CUDA_HOSTDEV Vec4Simple<T> & insert(int i,T const &x);
    CUDA_HOSTDEV void store(T * p) const;
    CUDA_HOSTDEV void store_a(T * p) const;
    CUDA_HOSTDEV Vec4Simple<T> & operator = (Vec4Simple<T> const & r);
    CUDA_HOSTDEV T operator [](int i) const;
    CUDA_HOSTDEV Vec4Simple<T> operator++ (int);
    static CUDA_HOSTDEV T getSquare(T b);
};
static CUDA_HOSTDEV void no_subnormals(){};

template <typename T>
CUDA_HOSTDEV Vec4Simple<T>::Vec4Simple() { }

template <class T>
static CUDA_HOSTDEV inline Vec4Simple<T> abs(const Vec4Simple<T> &l)
{
   return Vec4Simple<T>
   (
      fabs(l.val[0]),
      fabs(l.val[1]),
      fabs(l.val[2]),
      fabs(l.val[3])
   );
}
```

TABLE OF CONTENTS

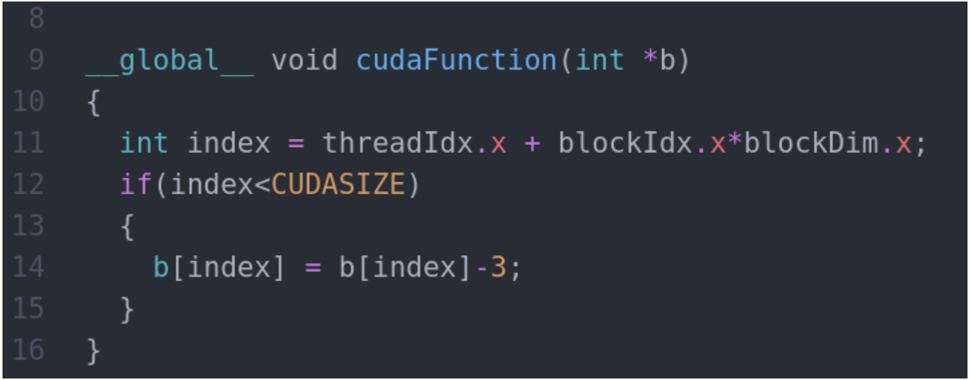
- Introduction
  - About work and depth
  - Performance and communication cost
  - Sparse Matrix to Vector Multiplication
- CUDA
  - GPU acceleration
  - Connect CUDA to existing application
  - Add custom library to CUDA application
  - CUDA programming model
- Vlasiator
  - Science case: Vlasiator
  - Vlasiator: state-of-the-art
  - Fluid vs. kinetic
  - Putting the Vlasov into Vlasiator
- Conclusion

CUDA programming model (call CUDA function)



```
//declare pointer to be used in device (GPU)
int *dev_b;
//allocate memory for pointer
cudaMalloc((void**)&dev_b, CUDASIZE*sizeof(int));
//copy data from host (CPU) variable to device (GPU) variable
cudaMemcpy(dev_b, b, CUDASIZE*sizeof(int), cudaMemcpyHostToDevice);
//call kernel (cudaFunctio)
cudaFunction<<<BLOCKS, THREADS>>>(dev_b);
//copy resulted data from device (GPU) variable to host (CPU) variable
cudaMemcpy(b, dev_b, CUDASIZE*sizeof(int), cudaMemcpyDeviceToHost);
```

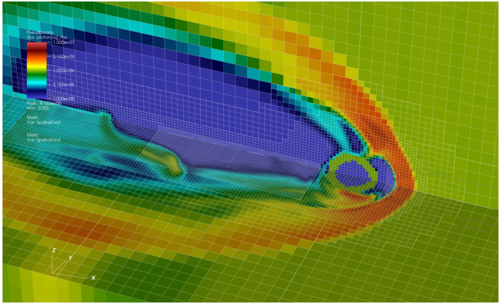
CUDA programming model (CUDA function)



```

__global__ void cudaFunction(int *b)
{
  int index = threadIdx.x + blockIdx.x*blockDim.x;
  if(index<CUDASIZE)
  {
    b[index] = b[index]-3;
  }
}
```

TABLE OF CONTENTS

- Introduction
  - About work and depth
  - Performance and communication cost
  - Sparse Matrix to Vector Multiplication
- CUDA
  - GPU acceleration
  - Connect CUDA to existing application
  - Add custom library to CUDA application
  - CUDA programming model
- Vlasiator
  - Science case: Vlasiator
  - Vlasiator: state-of-the-art
  - Fluid vs. kinetic
  - Putting the Vlasov into Vlasiator
- Conclusion



Science case: Vlasiator

- Simulating the plasma in Earth's magnetosphere & space weather effects
- Space weather drivers from plasma physics:
  - Solar wind
  - Solar storms (CME impact on this Monday, see https://twitter.com/erikapal/status/1446905406991728645?s=20)
  - Aurora
  - Adverse effect
    - Ground Induced Currents
    - Power grid problems
    - Satellite problems
    - Not very common, but significant: a very bad storm compared to COVID-19 for probability of occurrence and cost of effects

Vlasiator: state-of-the-art

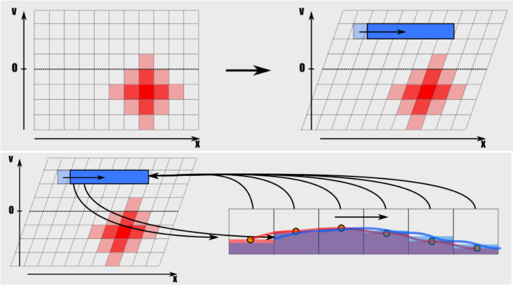
- C̃FD but with three more dimensions (… and Maxwell equations)
  - Huge computational demands and efficient computation
  - Developed with HPC in mind, in no small part with Sebastian von Alfthan
- Enabling technologies so far:
  - "full-stack" parallelizations (Vectorization, threading, MPI…)
  - Sparse arrays (velocity space)
  - Mesh refinement (of spatial grid) Enabled 3D production
- Open source (GPL2): https://github.com/fmihpc/vlasiator

Fluid vs. kinetic

- Fluid model: at one point in space, particles move with (mostly) a single velocity
- Kinetic model: at one point in space, particle motion is modelled separately for each velocity of particles
  - Essentially a 6D problem (plus time dependency)
  - Particles in space plasmas collide rarely -> need for a kinetic model
- $f(r, v; t)$ = number of particles per space (r) and velocity cell (v) (at time t)
  - evolved by the Vlasov equation
  - We use a cubical Cartesian grid to discretize this

Putting the Vlasov into Vlasiator

- Time evolution of f split to two parts, leapfrogging:
  - translation
  - acceleration (v-space translation)
- Semi-Lagrangian formalism (SLICE-3D, Zerroukat and Allen 2012)
  - Operation decomposed to single-cell columns
  - Conservative remapping, 5^{th} order in v-space, 3^{rd} in r-space
  - Domain decomposition in r-space -> v-space local to process (This could be nice to accelerate with GPUs)



Vlasov methods in space physics and astrophysics, Palmroth+2018

https://link.springer.com/article/10.1007/s41115-018-0003-2 Vlasiator-specifics at:https://link-springer-com.libproxy.helsinki.fi/article/10.1007/s41115-018-0003-2Sec28

Vlasiator simulation

The Earth’s magnetosphere responding to the flux of solar wind

Thank you!