CENTER FOR
BRAIN SIMULATION

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

CoreNeuron

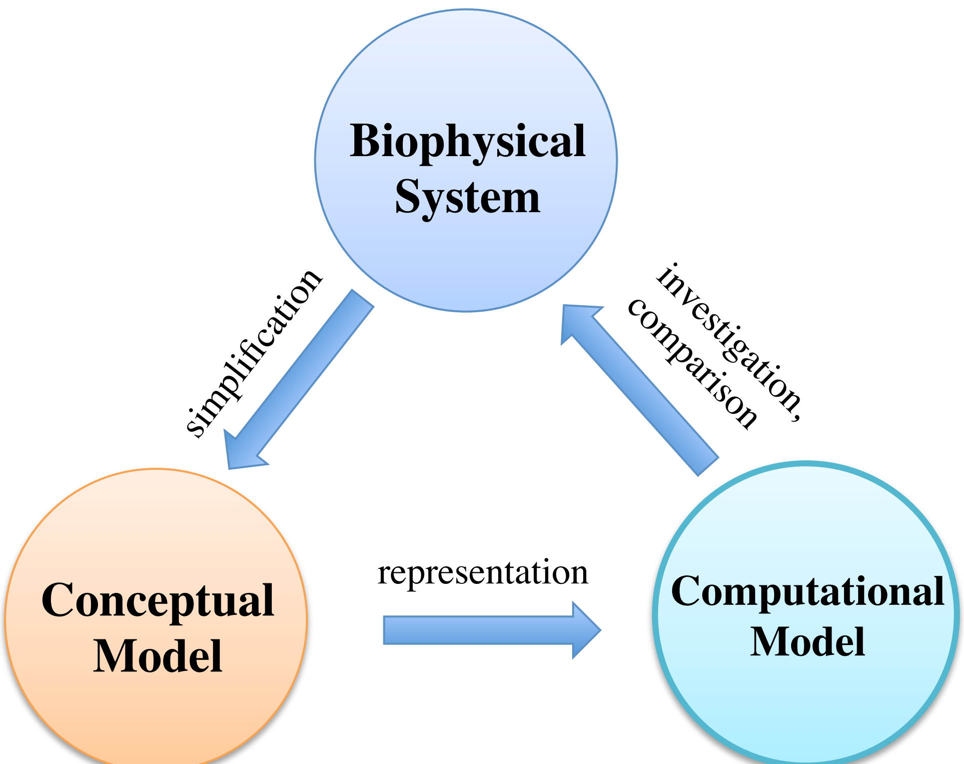
Journey to Eurohack 2015

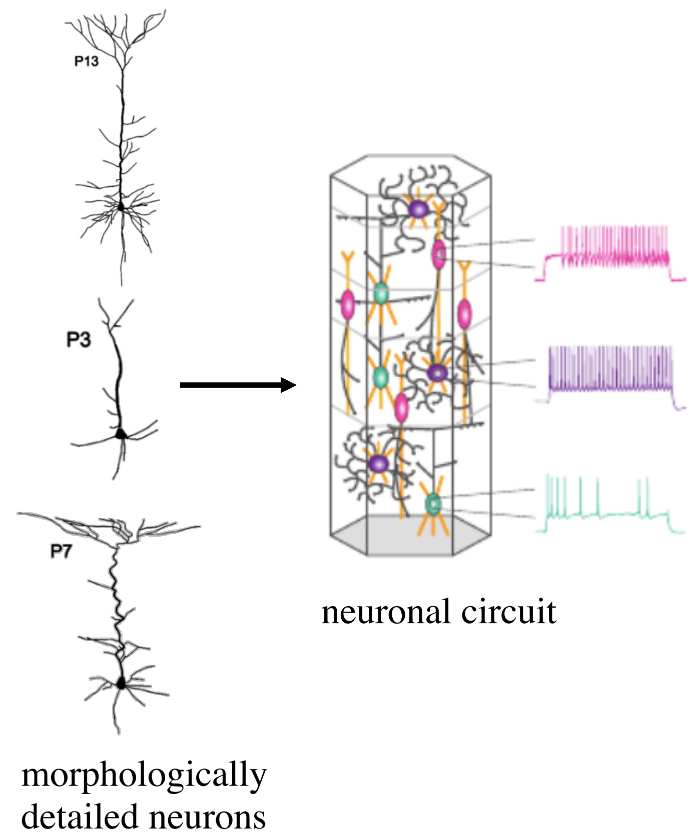
Blue Brain Project team, Ben, Jakob

Blue Brain Project

Comprehensive approach to systematically create unifying models of brain circuits by

- reverse engineering biological components
- construction of math models of the biophysics





Preparation

48 Racks, MIRA, ANL
3.14m threads, 260m neurons
May 2015

28 Racks, JUQUEEN, Juelich
1.8m threads, 160m neurons
Feb 2015

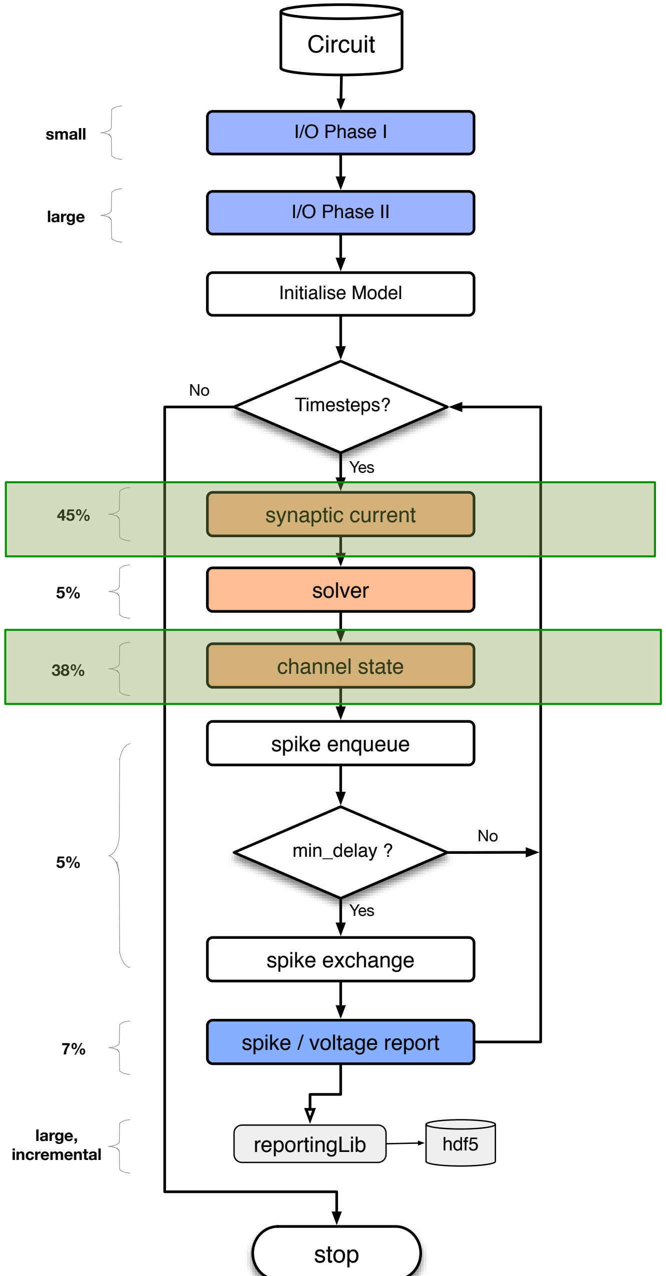
4 Racks, BBP IV, Lugano
262k threads, 10m neurons
Dec 2014

Preparation : Before the workshop

- Preparing simulation dataset
- Compilation with Cray, Intel and PGI
    - fixing Random123 issue with cray
- Profiling on host / sandybridge
    - verify the cpu performance with

What to offload?

CoreNeuron Simulator



- **Stimulus, Solver: 5-8%**
- **Channel State update: 90%**

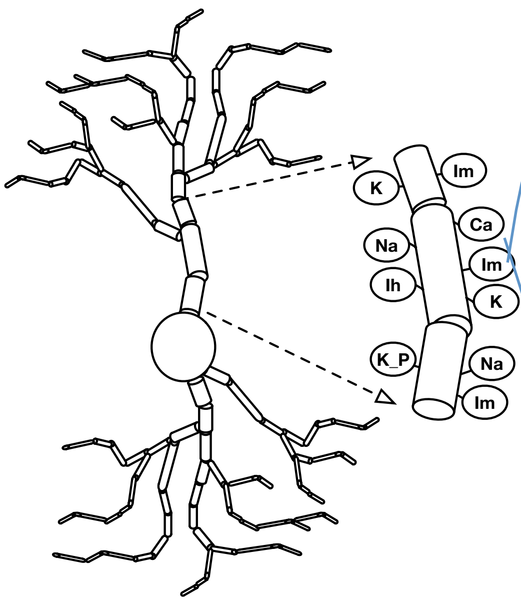
Porting Challenges

Porting Challenges: Data Structure

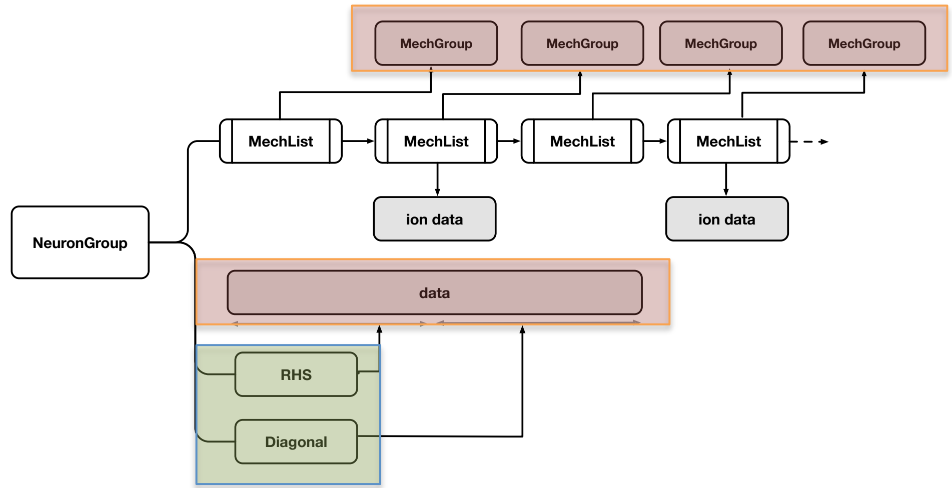
```
typedef struct NrnThread {
    double _t;
    double _dt;
    double cj;
    NrnThreadMembList* tml;
    int ncell; /* analogous to old rootnodecount */
    int end;    /* 1 + position of last in v_node a
    int id; /* this is nrn_threads[id] */
    int _stop_stepping; /* delivered an all threa

    double* _actual_rhs;
    double* _actual_d;
    double* _actual_a;
    double* _actual_b;
    double* _actual_v;
    double* _actual_area;
    int* _v_parent_index;
    Node** _v_node;
    Node** _v_parent;
    char* _sp13mat; /* handle to general sparse ma
    Memb_list* _ecell_memb_list; /* normally n
    void* _vcv; /* replaces old cvode_instance and n

#if 1
    double _ctime; /* computation time in seconds (
#endif

    NrnThreadBAList* tbl[BEFORE_AFTER_SIZE]
    hoc_List* roots; /* ncell of these */
    Object* userpart; /* the SectionList if this

} NrnThread;
```

```
typedef struct Memb_list {
    Node** nodelist;
#if CACHEVEC != 0
    /* nodeindices contains all nodes this extension is responsible for,
    * ordered according to the matrix. This allows to access the matrix
    * directly via the nrn_actual_* arrays instead of accessing it in the
    * order of insertion and via the node-structure, making it more
    * cache-efficient */
    int *nodeindices;
#endif /* CACHEVEC */
    double** data;
    Datum** pdata;
    Prop** prop;    //DIMENSION IS nodecount
    Datum* _thread; /* thread specific data (when static is no good) */
    int nodecount;
} Memb_list;


typedef union Datum {    /* interpreter stack type */
    double  val;
    Symbol  *sym;
    int i;
    double  *pval;  /* first used with Eion in NEURON */
    HocStruct Object **pobj;
    HocStruct Object *obj;  /* sections keep this to construct a name */
    char    **pstr;
    HocStruct hoc_Item* itm;
    hoc_List* lst;
    void* _pvoid;   /* not used on stack, see nrnoc/point.c */
} Datum;
```

Challenges: Data Structure & Lots of Kernels

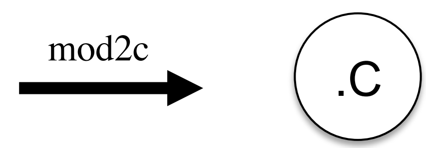


Biologist view:
compartment model



Memory View: In memory representation of Neurons

```
DERIVATIVE states {
    LOCAL mAlpha, mBeta, mInf, mTau, lv, qt

    qt = 2.952882641412121
    lv = v

    if(lv == -32){
        lv = lv+0.0001
    }

    mAlpha = mAlphaf(lv)
    mBeta = mBetaf(lv)
    mInf = mAlpha/(mAlpha+mBeta)
    mTau = (1/(mAlpha+mBeta))/qt
    m' = (mInf-m)/mTau

    v = lv
}
```



Hackathon Development

Two teams

MiniApp Benchmark

- take a kernel
- port to OpenAcc
- Benchmark on GPU
- What performance we expect?
- optimizations

Data Struct Workflow

- extract data structure
- Push to GPU (OpenACC)
- Modify on GPU
- Copy back to CPU (OpenACC)
- Validate pointers, data!

```
_PRAGMA_FOR_VECTOR_LOOP_
for( i = 0; i < count; i++) {

    int idx = node_index[i];
    v = vec[idx];

    p3[i] = data[ion_index[i]];

    double gNaTs2 = p0[i]*p1[i]*p1[i]*p1[i]*p2[i];
    double ina = gNaTs2*(v-p3[i]);

    data[ion_index1[i]] += gNaTs2;
    data[ion_index2[i]] += ina;

    vec_rhs[idx] -= ina;
```

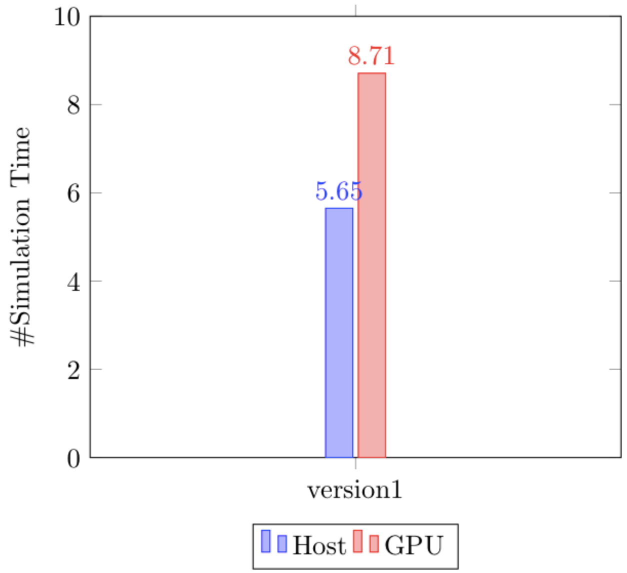
wrap OpenACC and auto-vectorisation related pragmas

OpenACC API's to copy the complex data structure

```

for (tml = nt->tml; tml; tml = tml->next) {

    /*copy all double data for thread */
    d__data = (double *) acc_copyin(nt->_data,
        nt->_ndata*sizeof(double));

    /*update d_nt._data to point to device copy */
    acc_memcpy_to_device(&(d_nt->_data), &d__data,
        sizeof(double*));

}
```

Results

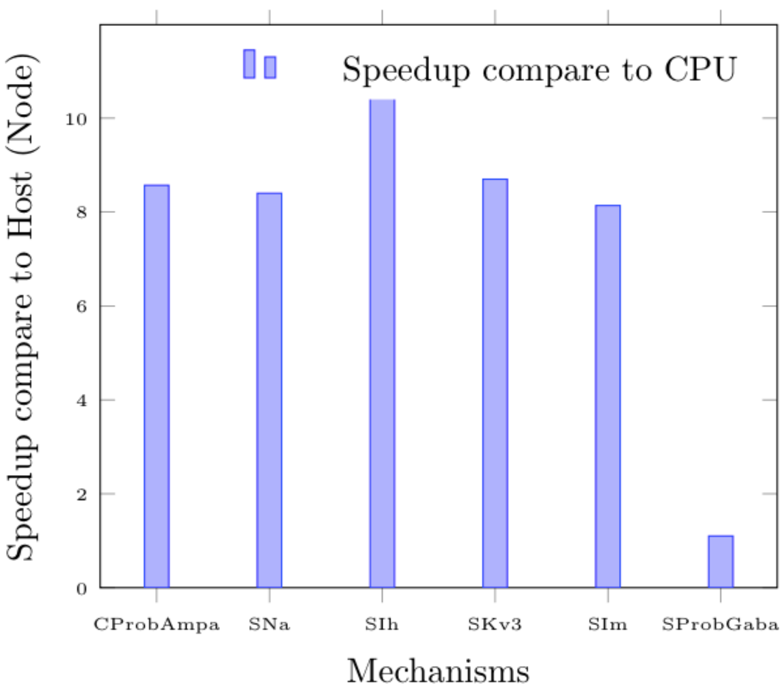
Node to node comparison



(whole simulation)

- Single thread offloading kernels from CPU
- Host node 1.5x faster compare to single GPU

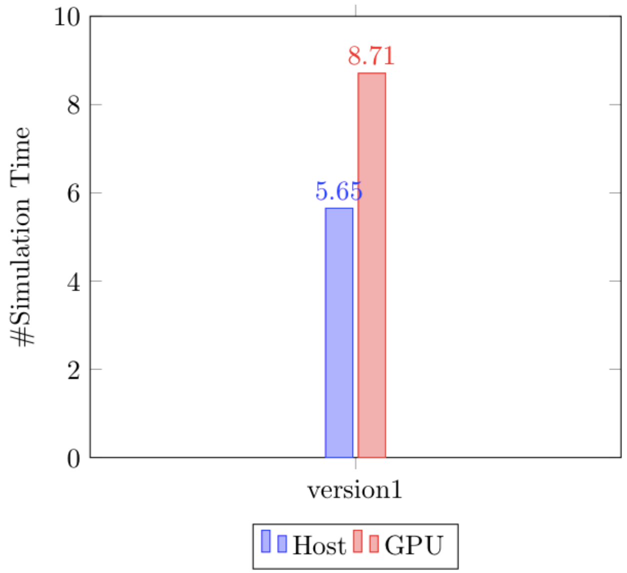
Individual kernels



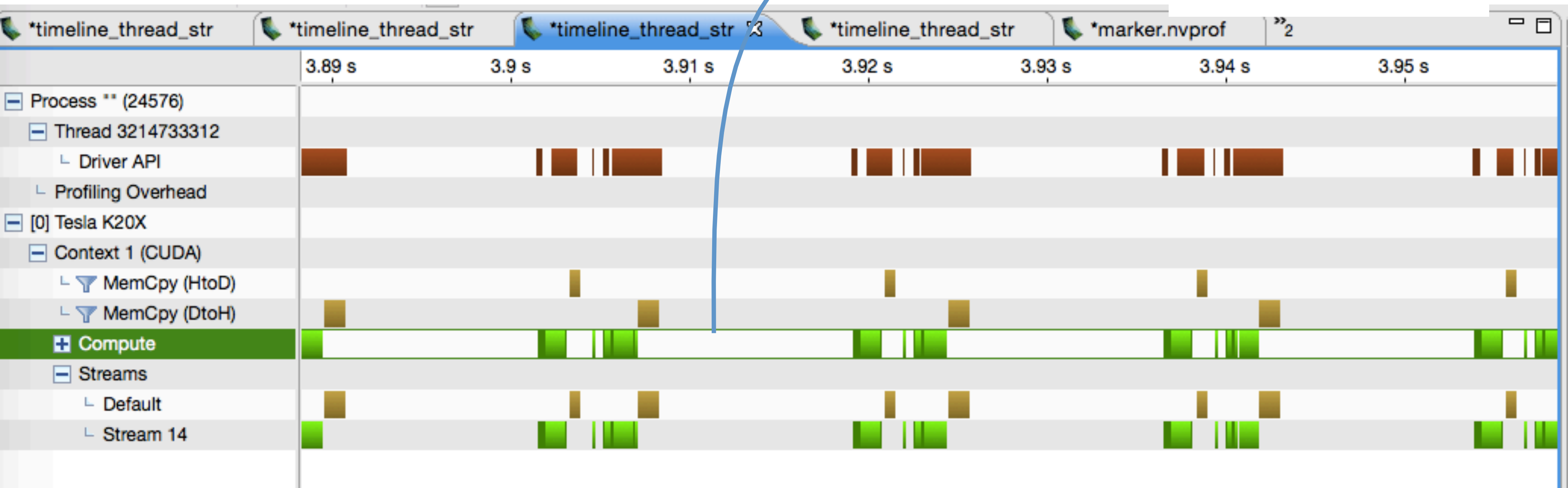
CENTER FOR
BRAIN SIMULATION

Why GPU code slower?

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE



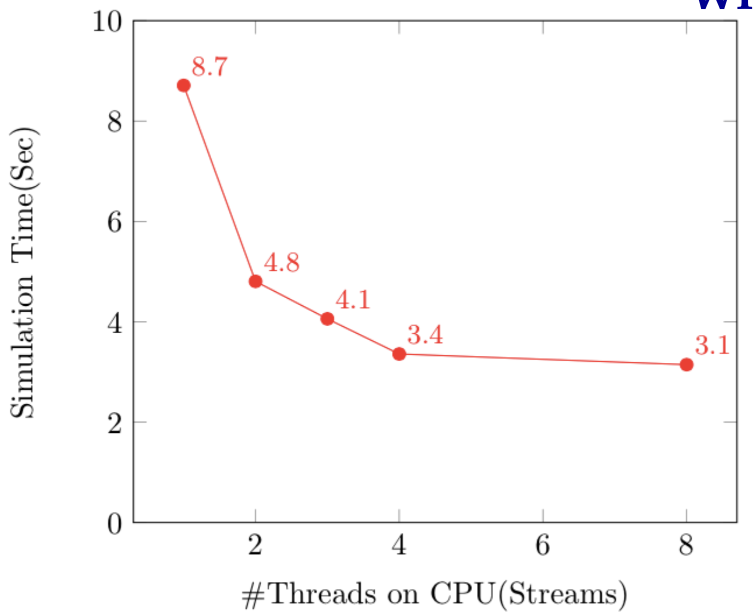
- Single thread offloading kernels from CPU
- Host node 1.5x faster compare to single GPU

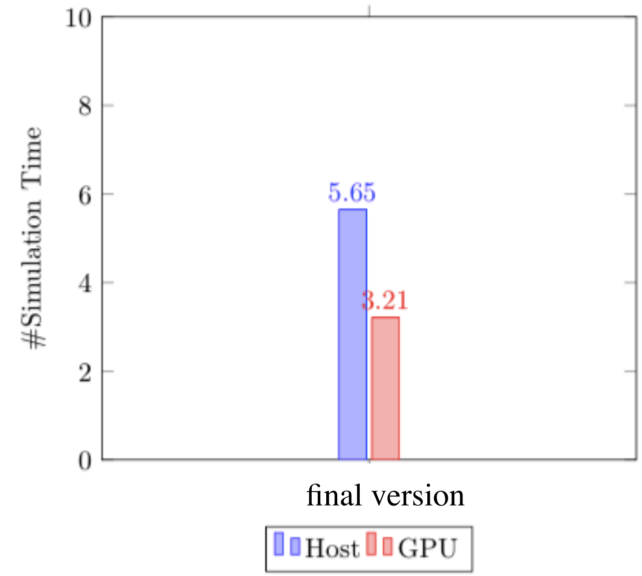


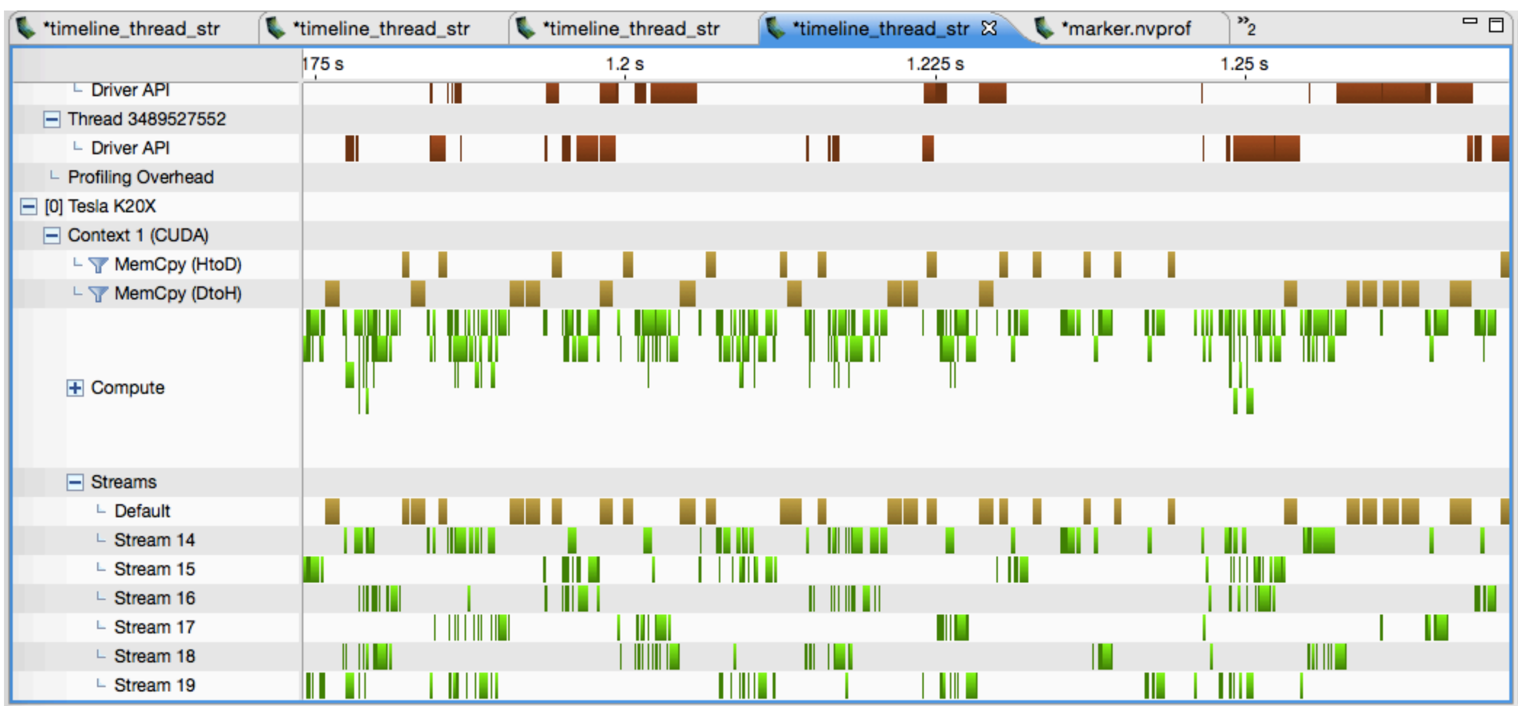
CENTER FOR
BRAIN SIMULATION

Multiple threads offloading to GPU with streams

ÉCOLE POLYTECHNIQUE







Compiler Issues

- Cray
  - copy(vec[0:asdfgh]
- PGI
  - vectorization & inlining
- Streams
  - only default stream being used for memcpy?

Next Steps

Ported kernels with OpenACC shows very good speedup compare to CPU (node to node comparison). In order to improve performance:

- Stimulus injection on GPU
- Solver
- Other small routines for spike activity