Recent developments in the Thick-GEM charging-up calculations and Garfield++ simulations acceleration.

Pedro M. M. Correia

University of Aveiro

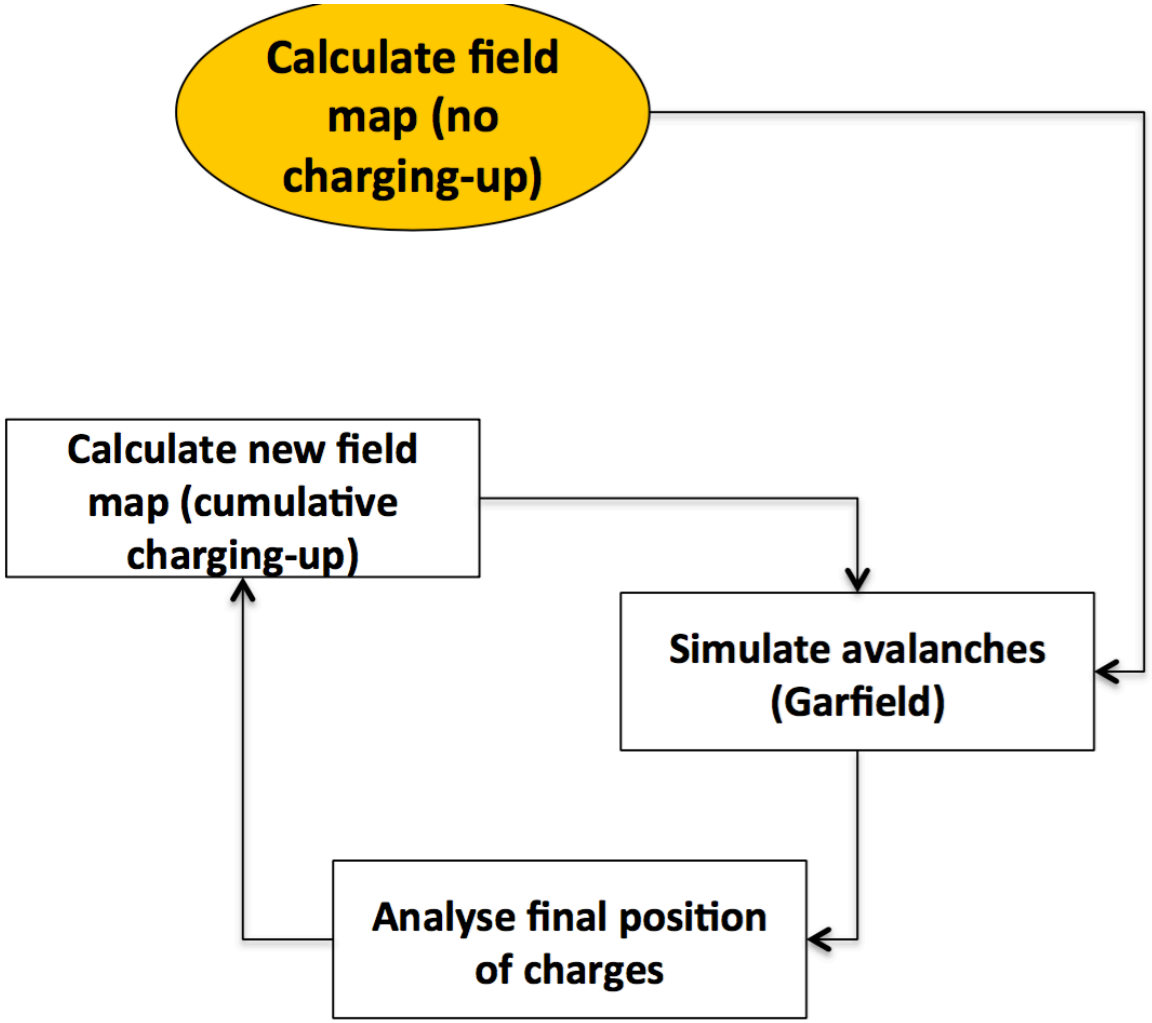
pmcorreia@ua.pt

Outline

- Charging-up calculations with Thick-GEMs
  - Recent developments
  - Discussion of a possible Garfield++ new class
- OpenMP integration with Garfield++
  - Code example
  - SpeedUp
- Conclusions

Charging-up calculation

- Iterative-based calculation
- Calculation of field maps require several FEM calls
- Dynamic method is available (http://dx.doi.org/10.1088/1748-0221/9/07/P07025)
- Only static method will be discussed for simplicity

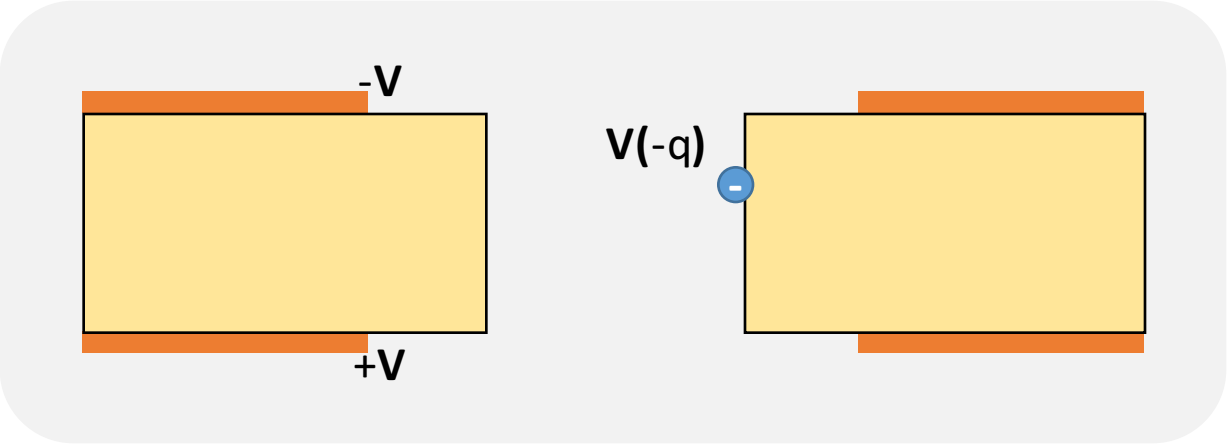


Principle of superposition

- The need of several executions of FEM is tedious
- Solution: Calculate **Potential field map** + **Charged field maps** at the beginning of the simulation
- Garfield++ reads potentials as a vector of **nodes** vs **E. Potential**
- For each node, the new **E. Potential** for each iterations is calculated inside Garfield++!

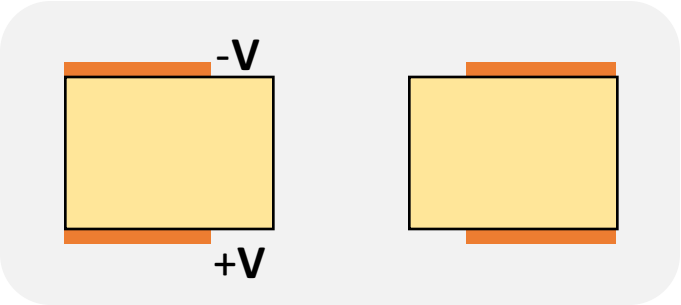
Dimensions not to scale

Complete field map



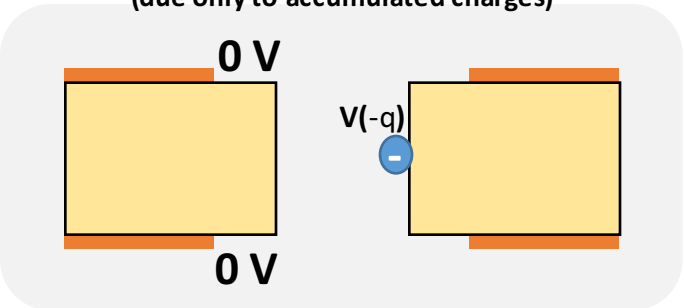
=

Potential field map



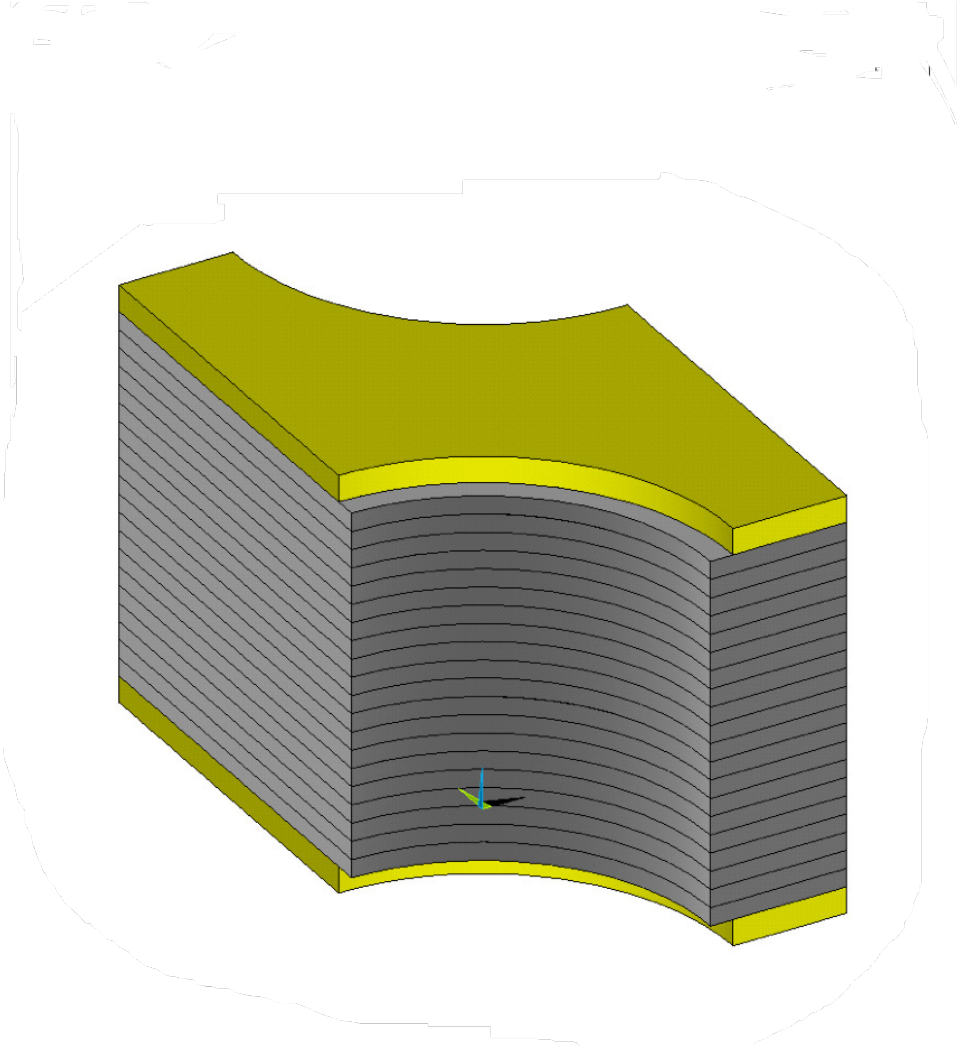
+

Charged field map
(due only to accumulated charges)



Principle of superposition

- Ex: Thick-GEM, insulator with 20 slices
- For each voltage between electrodes, field map is calculated as usually (without charging-up)
- For each slice, the field map correspondent to 1 electron accumulated on the correspondent slice surface is calculated
- 22 field maps due to charges + 1 (at least) potential field map are needed for full simulation



```
PRNSOL_700V.lis
PRNSOL_750V.lis
PRNSOL_800V.lis
PRNSOL_900V.lis
PRNSOL_slice10.lis
PRNSOL_slice11.lis
PRNSOL_slice12.lis
PRNSOL_slice13.lis
PRNSOL_slice14.lis
PRNSOL_slice15.lis
PRNSOL_slice16.lis
PRNSOL_slice17.lis
PRNSOL_slice18.lis
PRNSOL_slice19.lis
PRNSOL_slice1.lis
PRNSOL_slice20.lis
PRNSOL_slice2.lis
PRNSOL_slice3.lis
PRNSOL_slice4.lis
PRNSOL_slice5.lis
PRNSOL_slice6.lis
PRNSOL_slice7.lis
PRNSOL_slice8.lis
PRNSOL_slice9.lis
PRNSOL_sliceRimBottom.lis
PRNSOL_sliceRimTop.lis
```

Garfield++ Method

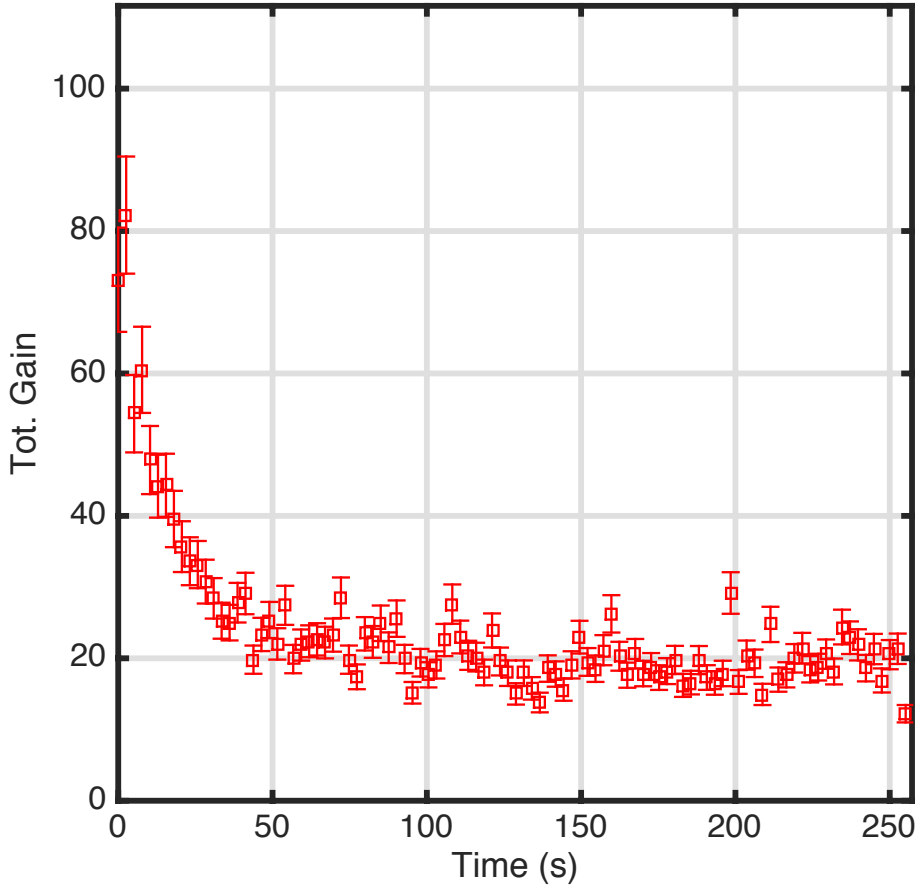
- A new Garfield++ class has been developed (~500 lines of code, up to now).
- Responsible to find the field maps (only **ANSYS** at the moment)
- Writes a temporary field map depending on the number of accumulated charges for each iteration
- It allows restart simulation at a specific iteration if previous field maps are stored
- (code for demonstration only)

```
#include "ChargingUpAnsys.hh"

using namespace std;
using namespace Garfield;
int main(int argc, char * argv[]) {
    double ChargesVector[ nSlices ];

    //(...);

    ChargingUpAnsys file(mapfilesdir, nSlices, ChargesVector, gasstr, vgem, npe);
    if (!file.checkSlicesFieldMaps()){
        std::cout<<"Error 1, files does not exist"<<endl;
        exit(0);
    }
    file.loadSlicesFieldMaps();


    double simulatedCharges[ nSlices ];

    //for loop over iterations
    // (...) after avalanches calculation and calculation of the
    // number of accumulated electrons+ions

    file.UpdateFieldMap(simulatedCharges);
    file.SaveKaptonChargesFile(iter);
    file.printCurrentCharges();
    //end of the for loop

}
```

Gain Results

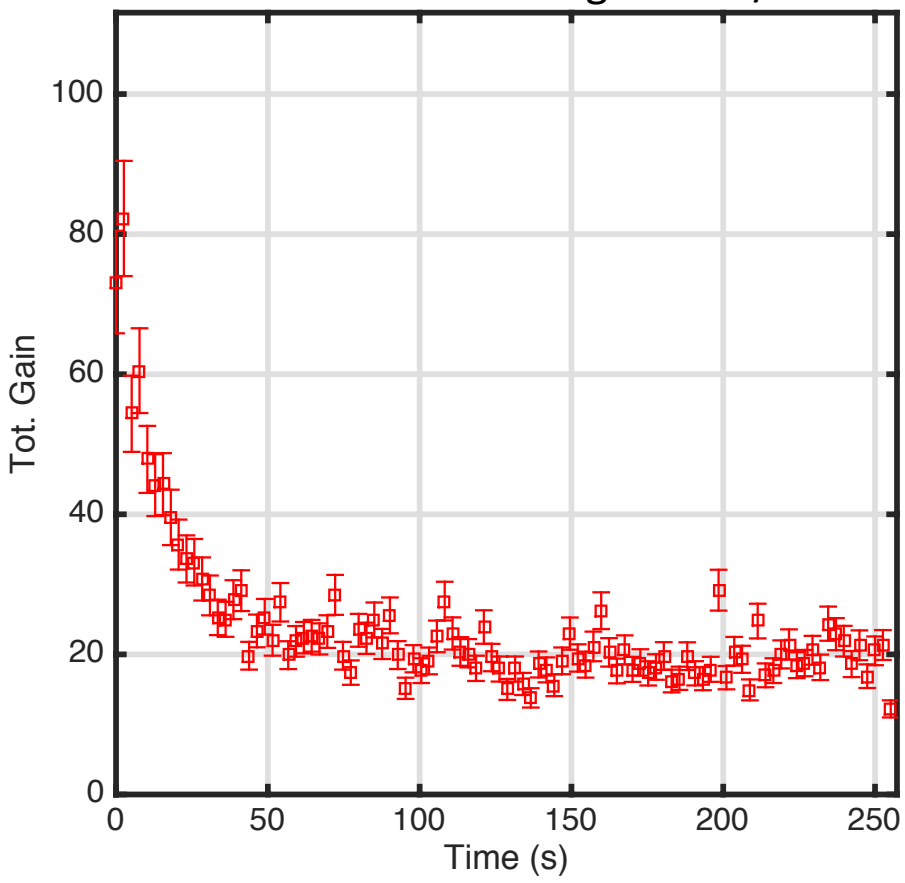
- Thick-GEM
  - Insulator with 20 slices
  - Pitch 0.4 mm
  - Thickness 0.2 mm
  - Hole Diameter 0.2 mm
  - Rim 40 µm
  - V_{THGEM} = 1000 V
  - Standard temp. and pressure.

Assuming 10 kHz/mm^2

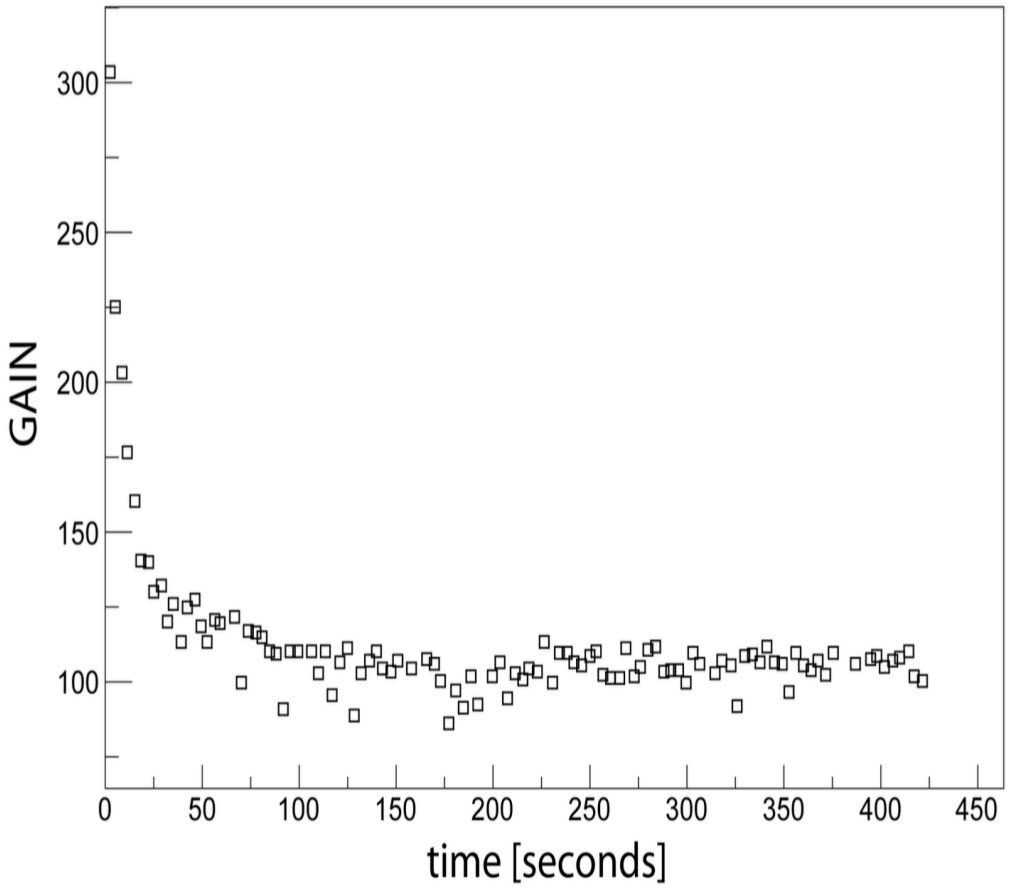


Gain Results

Assuming 10 kHz/mm^2



Experimental data. Same geometry. Unknown **VTHGEM** and **Rate**
"The gain in Thick GEM multipliers and its time-evolution", 2015
http://dx.doi.org/10.1088/1748-0221/10/03/P03026



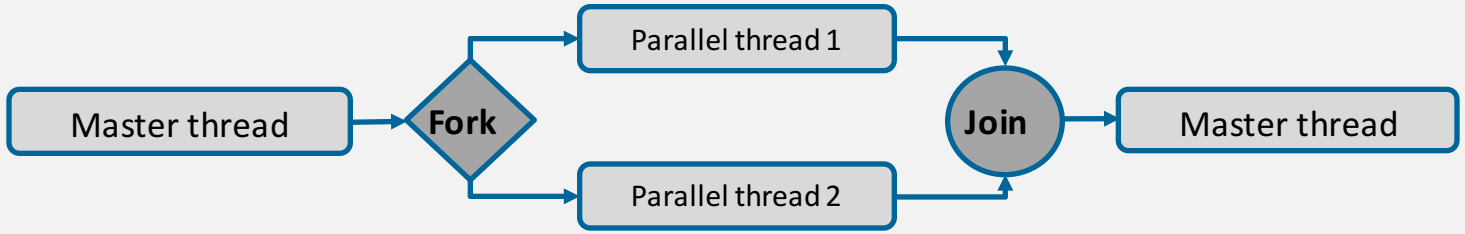
Despite gain differences, time-behaviour is the same.
Still does not explain the long-term gain increase observed after some hours.

Integration of OpenMP in Garfield++

- OpenMP is a parallelization tool first released in 1997, can be used with **Fortran**, **C** and **C++**
- Used for multithread parallelization of parts of the code
- Set of simple compiler directives and code lines are needed to parallelize simple applications:
- Ex. Compilation flag for gnu:
  - **-fopenmp**

- Uses the Fork-join[1] model of parallel execution

[1]M. E. Conway. *A multiprocessor system design*. In Proceedings, November 12-14 1963



- For-loop example C++:

```
//Typical C++
for (int i = 0; i < 8;
++i) {
 do_some_task(i);
}
```

```
//C++ with OpenMP
#include <omp.h>

omp_set_num_threads(8);
#pragma omp parallel for
for (int i = 0; i < 8; ++i) {
 do_some_task(i);
}
```

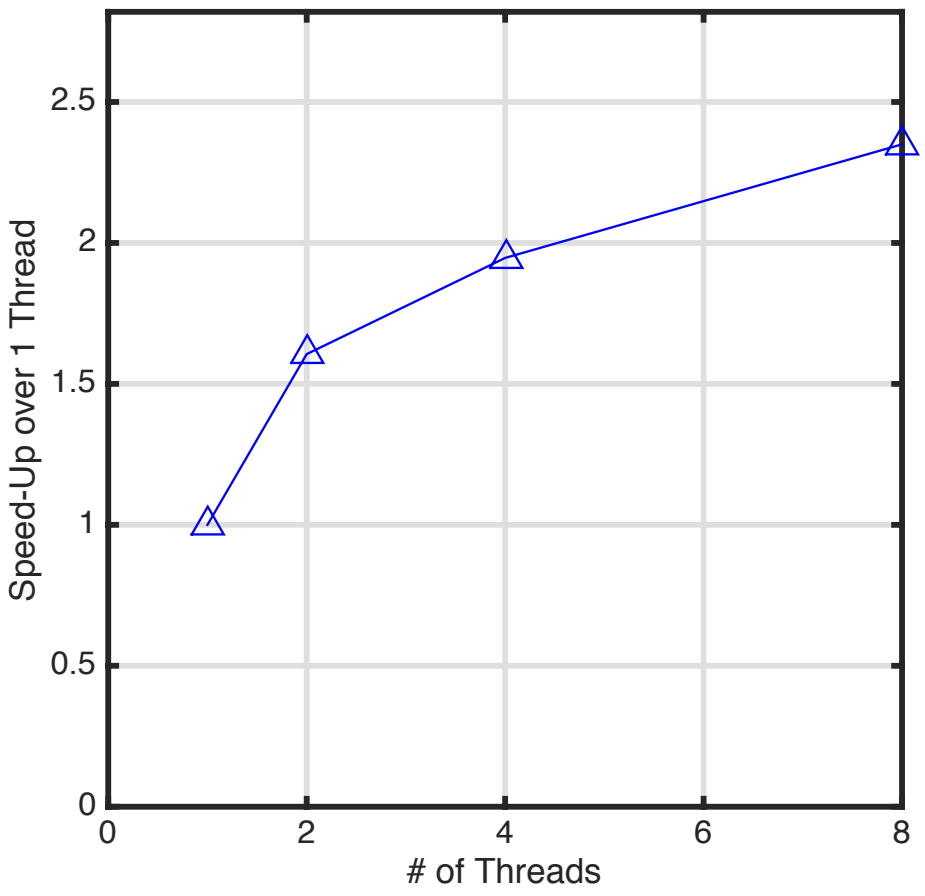
Integration of OpenMP in Garfield++

- Careful must be taken when objects are created outside the parallel region.
- Some Garfield objects need to be defined for each thread (as arrays of pointers), otherwise race conditions will happen between different threads
- Parallel region is defined with the line #pragma omp parallel for
- Omp_set_lock(&writelock) is called to lock everything that is not thread-safe or needs to be synchronized (ex: std::cout)

```
#include <omp.h>

omp_lock_t writelock;

int main(int argc, char * argv[]) {

  //(...)
  MediumMagboltz* gas = new MediumMagboltz();
  gas->SetComposition("ar", 70., "co2", 30.);
  gas->SetMaxElectronEnergy(1000.);
  //gas->DisableEnergyRangeAdjustment();

  ComponentAnsys123 *fm = new ComponentAnsys123();
  //(...)

  int nThreads=omp_get_max_threads();
  AvalancheMicroscopic *aval[nThreads];
  Sensor *sensor[nThreads];
  cout<<"Number of threads available are "<<omp_get_max_threads()<<endl;
#pragma omp parallel for
  for (int i = 0; i<npe;i++) {
      omp_set_lock(&writelock);
      cout<<"Number of running threads "<<omp_get_num_threads()<<endl;
      omp_unset_lock(&writelock);
      int thread_num=omp_get_thread_num();
      aval[thread_num] = new AvalancheMicroscopic();
      sensor[thread_num] = new Sensor();
      aval[thread_num]->SetSensor(sensor[thread_num]);

      //Avalanches loop
      aval[thread_num]->AvalancheElectron(x0, y0, z0, t0, e0, 0., 0., 0.);
      //loop over npe for each thread
      omp_destroy_lock(&writelock);
  }
  return 0;
}
```

SpeedUp
OpenMP in Garfield++

- Lxplus processor:
  - Intel Xeon E312xx (Sandy Bridge)
  - 8 threads
  - 2.6 GHz
  - 4 Mb Cache Size
- Optimization of the code would allow even faster simulations.
- Speedup is not linear with the increase of available threads.



Conclusions

- Charging-up gain calculations are now performed completely inside Garfield++
  - A new class has been written, only needs the initial field maps corresponding of each insulator slice
  - No need to call FEM software after each iteration -> **time and memory saving**
  - Results coherent with previous simulations; Short-time gain evolution as described by experimental work.

- Garfield++ simulations acceleration with multithread tools - **OpenMP**.
  - Race-conditions must be take into consideration, otherwise software will crash.
  - Speed-up factor > 2 can be achieved with Lxplus machines with 8 threads.