Guineapig++

Comment pourrait-on accélérer le code?

```
void beam_interaction(int n_slice, PHI_FLOAT* parameter )
 {

   // du debut du croisement a la coincidence des faisceaux
   for (int k = 0;  k < n_slice;  k++)
     {
       iteration_on_overlaping_slices(0,k,parameter);
     }

   // de la coincidence a la fin du croisement
  for (int k = n_slice; k < 2*n_slice-1; k++)
     {
       iteration_on_overlaping_slices(k-n_slice+1, n_slice-1,
parameter);
     }
 }
```

```
void iteration_on_overlaping_slices(int sliceOfBeam1, int sliceOfB
{
  for(int i0 = 0; i0 < grid_.get_timestep(); i0++)
    {
      int j= sliceOfBeam2;
       for(int i = sliceOfBeam1; i <= sliceOfBeam2; i++)
           {
              make_step(i, j, parameter);
               j--;
           }
      }
}
```

```
void  make_step(int i1, int i2, PHI_FLOAT* parameter)
{

                                        ⇨  distribute_particles(i1, i2,…)
grid_.all_distribute(i1,i2,…);          ⇨  distribute_particles_for_background(i1, i2,…)
                                        ⇨  distribute_virtual_photons(i1, i2, …)

for(int i = 1;  i <= switches.get_extra_grids(); i++)
   {
      gridsPtr_[i]->distribute_particles(i1, i2, …);
   }

 time_.add_timer(1);

  grid_.step_lumi(min_z, secondaries_, bhabhaSamples_, time_counter, switches,
                        pair_parameter, jet_results_, generateur_);

 time_.add_timer(2);

 grid_.update_slice_charge(i1, i2);

 time_.add_timer(3);
```

```
grid_.computeFields(switches.get_integration_method(),switches.get_charge_sign(),parameter);

 time_.add_timer(4);

  for (int i = 1; i <= switches.get_extra_grids(); i++)
    {
      gridsPtr_[i]->computeFields(switches.get_integration_method(), switches.get_charge_sign(),pa
    }

  time_.add_timer(5);

  int nbeam = 1;
  grid_.moveAllParticles(gridsPtr_,beam1,nbeam,i1,photon_[0],switches.get_interpolation(),
                         switches.get_do_eloss(),switches.get_emin(),switches.get_do_prod(),
                         switches.get_extra_grids(),photon_data_,photon_results_,generateur_);

  time_.add_timer(6);

  nbeam = 2;
  grid_.moveAllParticles(gridsPtr_,beam2,nbeam,i2,photon_[1],switches.get_interpolation(),
                         switches.get_do_eloss(),switches.get_emin(),switches.get_do_prod(),
                         switches.get_extra_grids(),photon_data_,photon_results_,generateur_);

  time_.add_timer(7);
  Etc ……..
}
```

nSlices= 32 timeStep= 5

	n= 10000 nx= ny= 64		n= 100000 nx= ny= 64		n= 10000 nx= ny= 128	
timer no 1 :	2	1	14	13	5	4
timer no 2 :	0	0	38	34	1	1
timer no 3 :	0	0	0	0	0	0
timer no 4 :	40	8	40	8	329	54
timer no 5 :	0	0	0	0	0	0
timer no 6 :	0	0	5	5	0	0
timer no 7 :	0	0	5	5	0	0
timer no 8 :	0	0	0	0	0	0
timer no 9 :	0	0	0	0	0	0
timer sum :	44.14	12.01	102.76	66.91	336.97	61.75

This routine calculates the luminosity from the collision of the two slices i1 and i2

```
void step_lumi(float min_z, PAIR_BEAM& bhabhas, BHABHASAMPLES& bhabhaSamples, …)
{
    float sum = 0.0;
    list<PARTICLE_POINTER>::iterator pointer1,pointer2;
    float poidsCompose;
    for (int i1 = 0; i1 < n_cell_x; i1++)
     {
        for (int i2 = 0; i2 < n_cell_y; i2++)
         {
            int j= i1*n_cell_y+i2;
            sum += rho1[j]*rho2[j];
            for (pointer1=grid_pointer1_[j].begin(); pointer1 != grid_pointer1_[j].end(); pointer1++)
             {
                for( pointer2 = grid_pointer2_[j].begin(); pointer2 != grid_pointer2_[j].end(); pointer
                 {
                   poidsCompose = (*pointer1).weight() * (*pointer2).weight();
                  collide_ee(i1, i2, min_z, pointer1, pointer2,switches, poidsCompose, bhabhas,
                                bhabhaSamples, pair_parameter, time_counter, jet_results, hasard)
                 }
             }
         }
     }
}
```