Microscopic and mesoscopic simulations of amorphous systems using LAMMPS and GPU-based algorithms

E. Ferrero and F. Puosi

Laboratoire Interdisciplinaire de Physique (LIPhy), UJF and CNRS

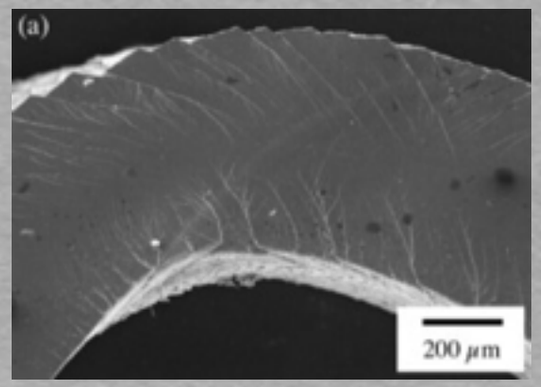
14/05/2014

Journée des utilisateurs CIMENT

Amorphous materials

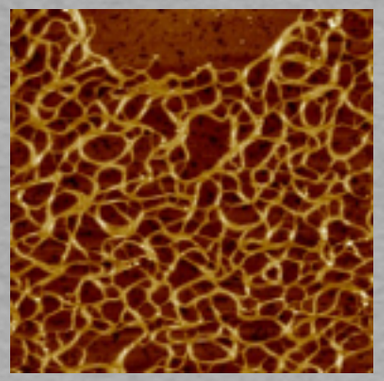
Hard glasses **Soft glasses**

Metallic/oxide glasses

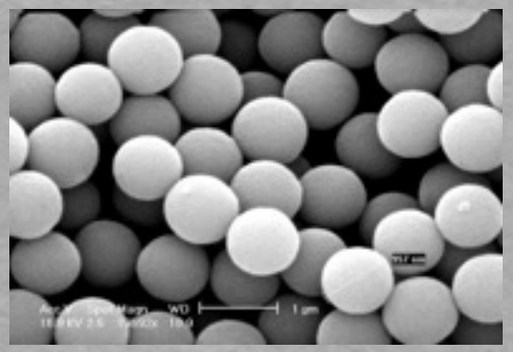


Length scale ≤ 1 nm

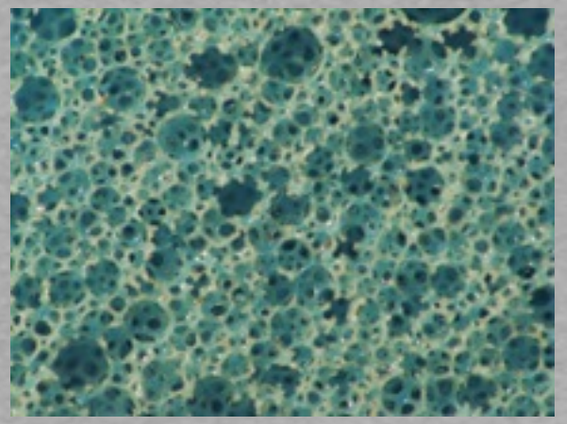
Polymers



Colloids



Foams

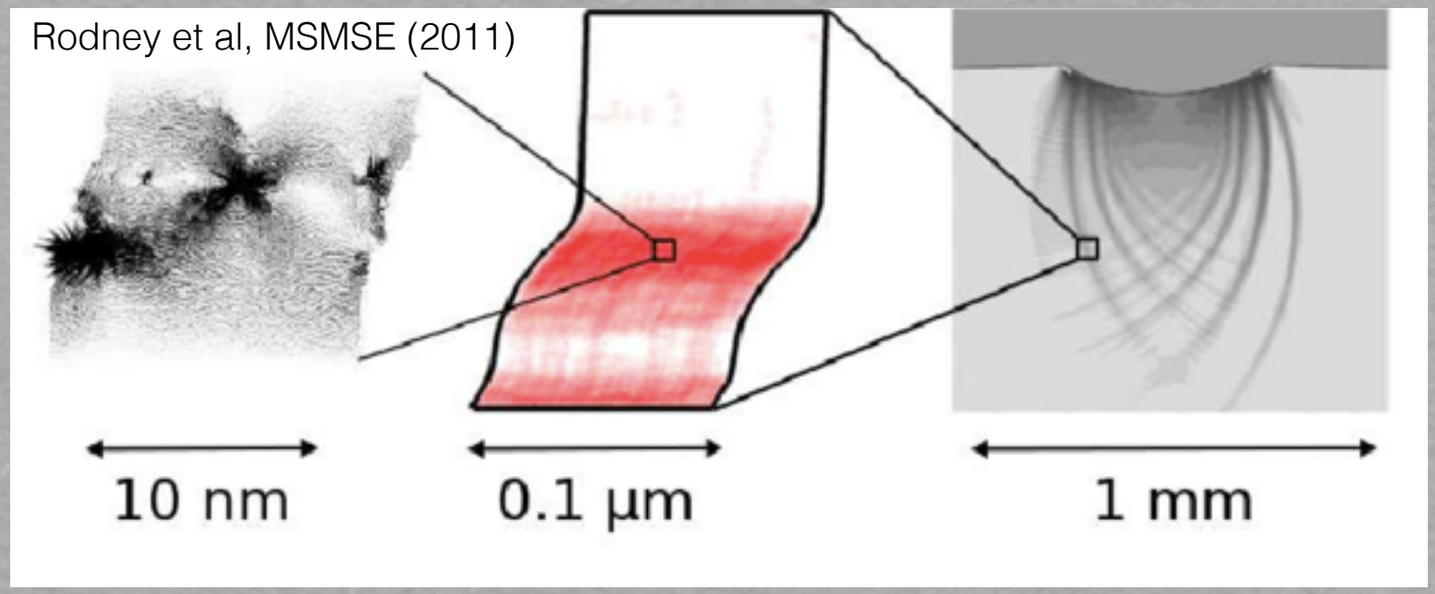


Length scale ≥ 0.1 μm

Particles are arranged in a disordered way, like in **liquids**,
but they are “jammed” like in **solids**

Flow of amorphous media is an outstanding question in **rheology** and **plasticity**

Length scales in the deformation of amorphous systems

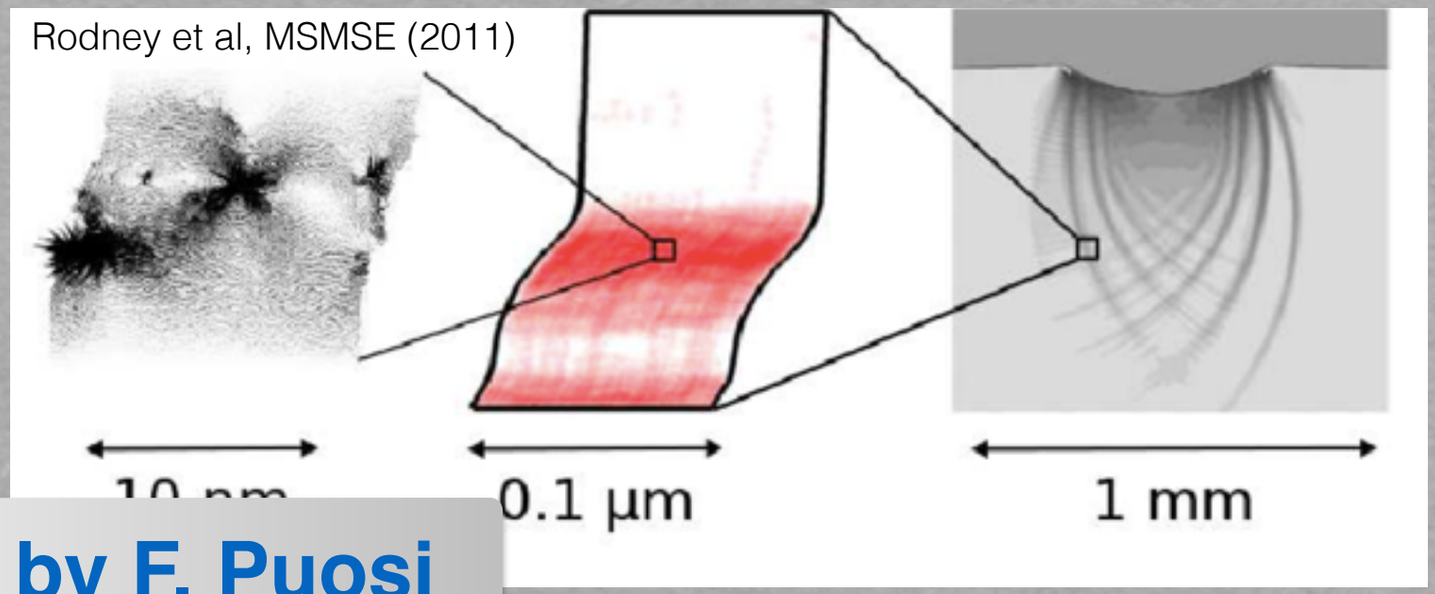


- **Microscopic scale**: scale of individual objects (droplets, colloids, particles). Dynamics is extremely complex and numerically highly time consuming at this scale.

- **Mesoscopic scale**: scale where the elementary local “plastic” events that constitute the flow of amorphous materials do occur.

- **Macroscopic scale**: scale relevant to engineering problems. Phenomenological laws describe the flow behavior.

Length scales in the deformation of amorphous systems



1st part by F. Puosi

- **Microscopic scale**: scale of individual objects (droplets, colloids, particles). Dynamics is extremely complex and numerically highly time consuming at this scale.

- **Mesoscopic scale**: scale where the elementary local "plastic" events that constitute the flow of amorphous materials do occur.

2nd part by E. Ferrero

- **Macro** jineering problems. Phenomenological laws describe the flow behavior.

Molecular Dynamics (MD) basics

- N particles (atoms, groups of atoms, meso-particles) in a box (rigid walls, periodic boundary conditions)
- All physics in energy potential (forces)
  - pair-wise interactions (LJ, Coulombic), many-body forces (EAM, Tersoff, REBO), molecular forces (springs, torsions)…
- Integrate Newton's equation of motion
  - Velocity-Verlet formulation:
    - update V by1/2 step (using F)
    - update X (using V)
    - build neighbor lists (occasionally)
    - **compute F** (using X)
    - apply constraints & boundary conditions
    - update V by1/2 step (using new F)
    - output and diagnostics
  - CPU time break-down:
    - **inter-particle forces = 80%**
    - neighbor lists = 15%
    - everything else = 5%
- Properties via time-averaging ensemble configurations

What is LAMMPS

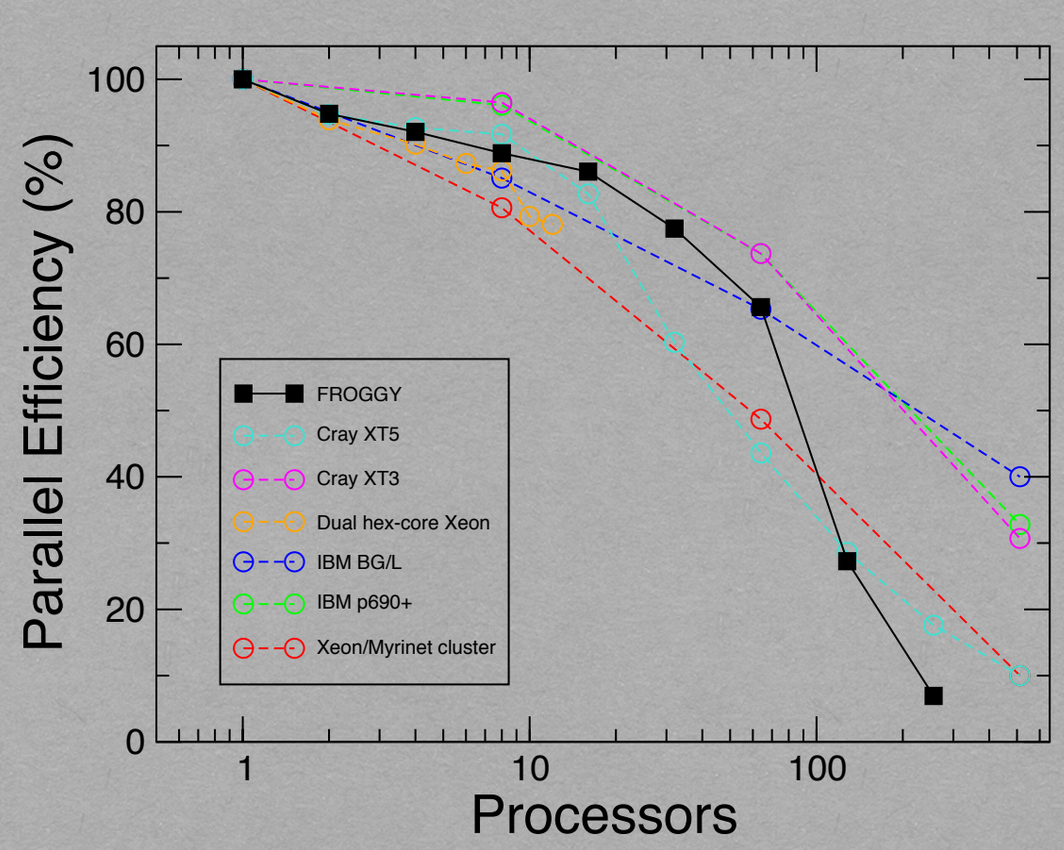
Large-scale Atomic/Molecular Massively Parallel Simulator
http://lammps.sandia.gov

- Classical MD code
- Open source (GPL), highly portable C++
- Simulations at varying length and time scales:
  electrons ⇒ atomistic ⇒ corse-grained ⇒ continuum
- Spatial-decomposition of simulation domain for parallelism
- GPU and OpenMP enhanced
- Can be coupled to other scales: QM, kMC, FE…

LAMMPS on Froggy

Benchmark: **atomic fluid**

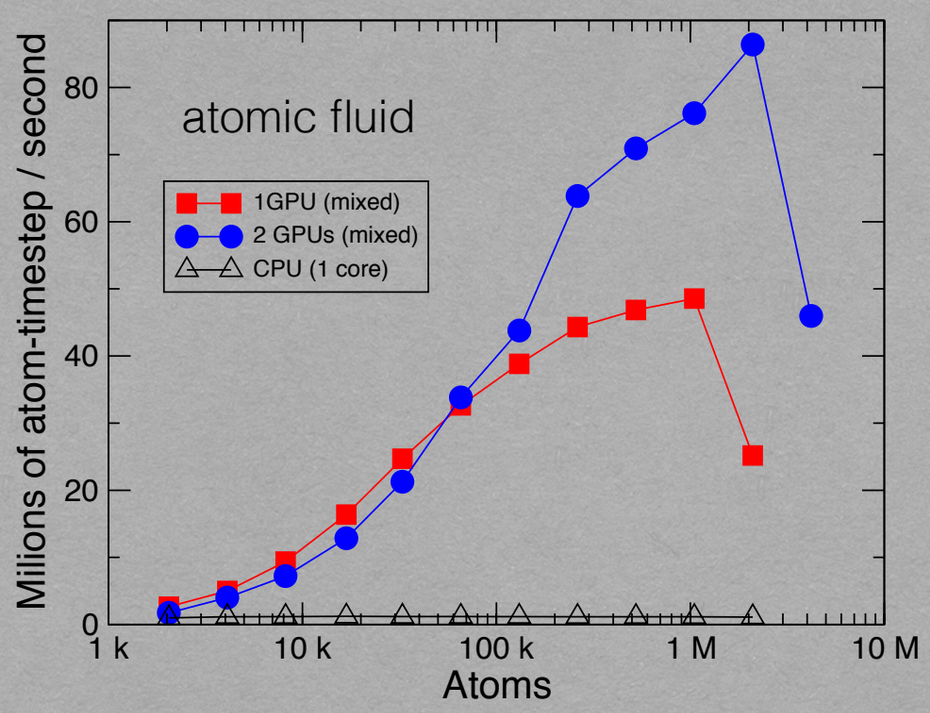
- 32000 atoms for 1000 time-steps
- reduced density 0.8442 (liquid)
- force cutoff 2.5 sigma (~2.5 diameters)
- neighbor skin 0.3 sigma (neighbor atom 55)
- NVE time integration



GPU acceleration in LAMMPS

USER-CUDA package

- GPU version of pair styles, fixes and computes
- an entire LAMMPS calculation run entirely on the GPU for many time steps
- only one core per GPU
- better speed-up if the number of atom per GPU is large



GLASSDEF project

European Research Council

Microscopic aspects:

- **Parallel Replica Dynamics in glasses**: a method to extend timescales accessible to Molecular Dynamics simulation (FP, D. Rodney and J.-L. Barrat)

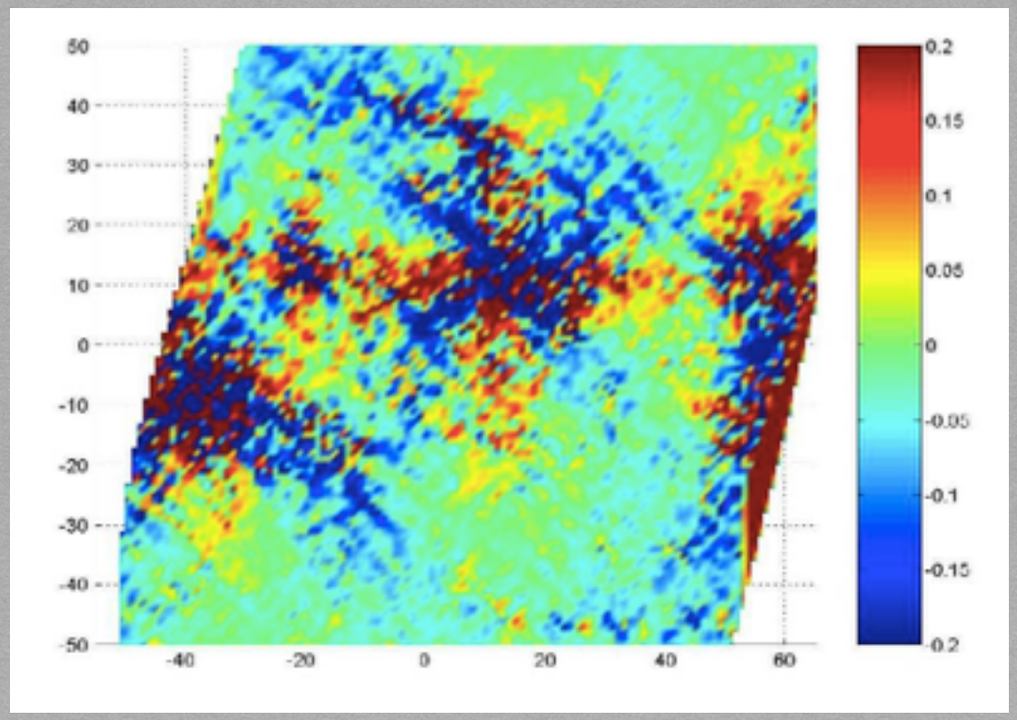
- **Plasticity in 2D sheared amorphous solids**: investigation of the spatio temporal correlations of plastic activity in athermal amorphous solids (FP, A. Nicolas, J. Rottler and J.-L. Barrat)

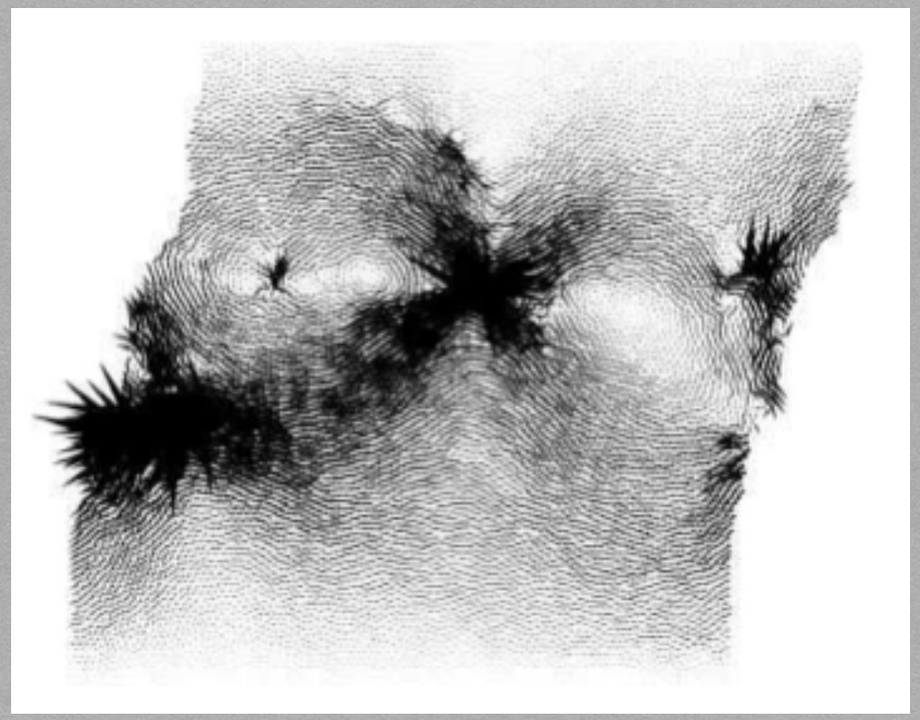
- **Microscopic test of a mean field model for the flow of amorphous systems** (FP, J. Olivier, K. Martens and J.-L. Barrat)

- **Avalanches in amorphous systems**: scaling description of avalanches in the shear flow of athermal amorphous solids at finite shear rates (FP, E. Ferrero, C. Liu, L. Marradi, K. Martens, A. Nicolas and J.-L. Barrat)

Elementary plastic events

At low temperature, the onset of plastic deformation in glasses is due to the accumulation of **elementary plastic events**, consisting of localized in space and time atomic rearrangements involving only a few tens of atoms, the so-called Shear Transformations (STs)





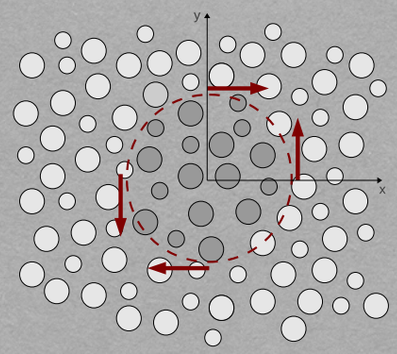
Tanguy et al., EPJE (2006)

Open question:
How do the elastic response to a ST build in time?

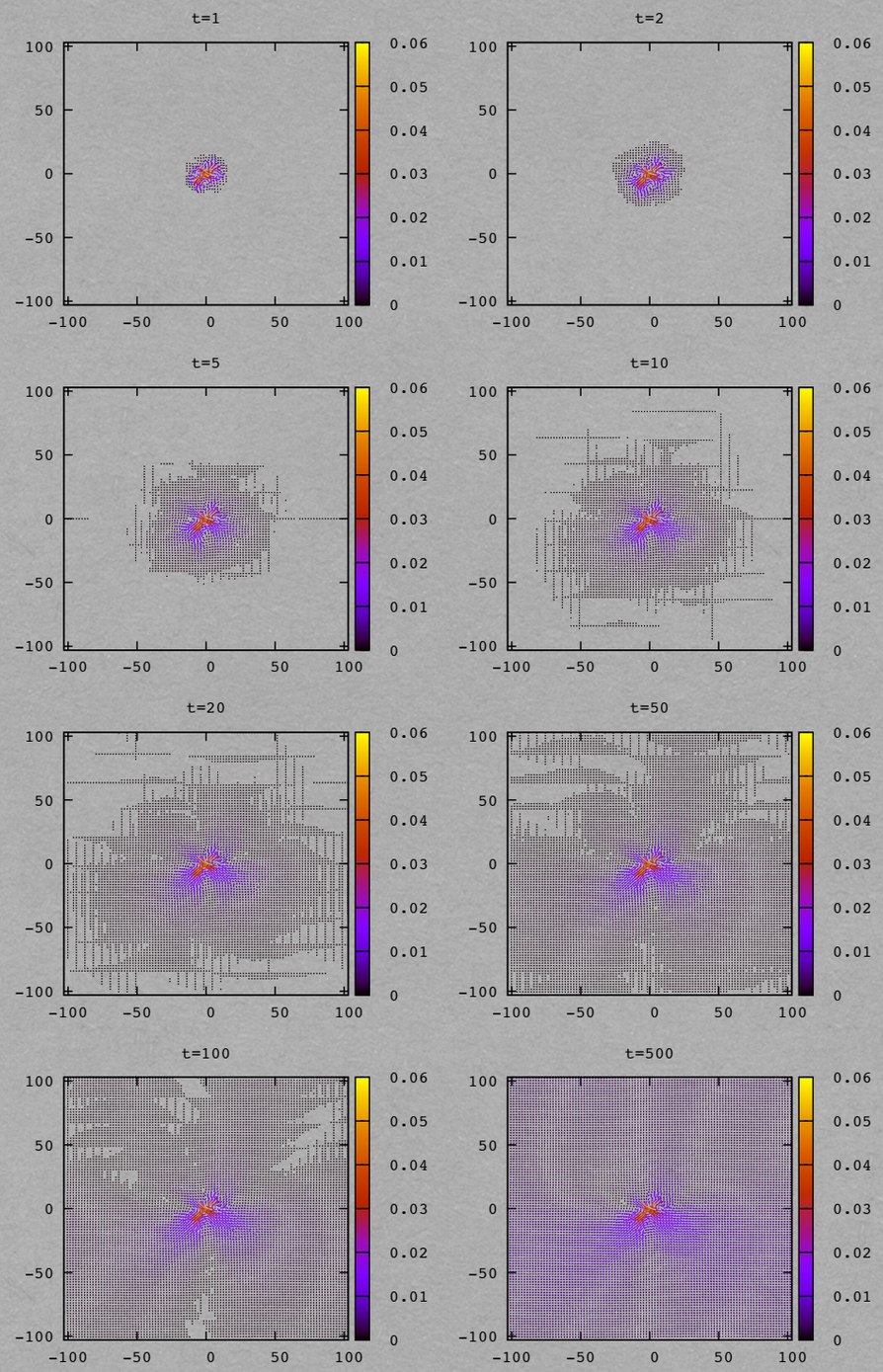
Fictitious STs in amorphous solids

2D binary mixture of athermal Lennard-Jones particles.

- We apply an instantaneous local shear transformation and we observe the response of the system



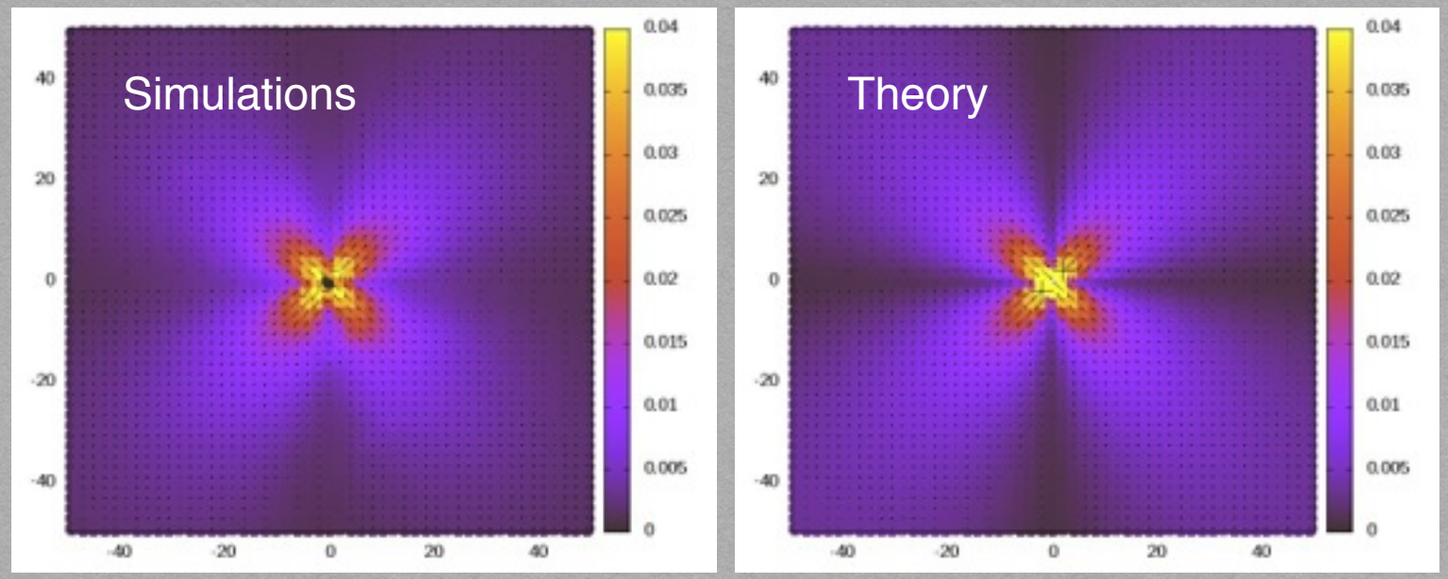
Disorder average of the response: average over different spatial realizations of the ST.



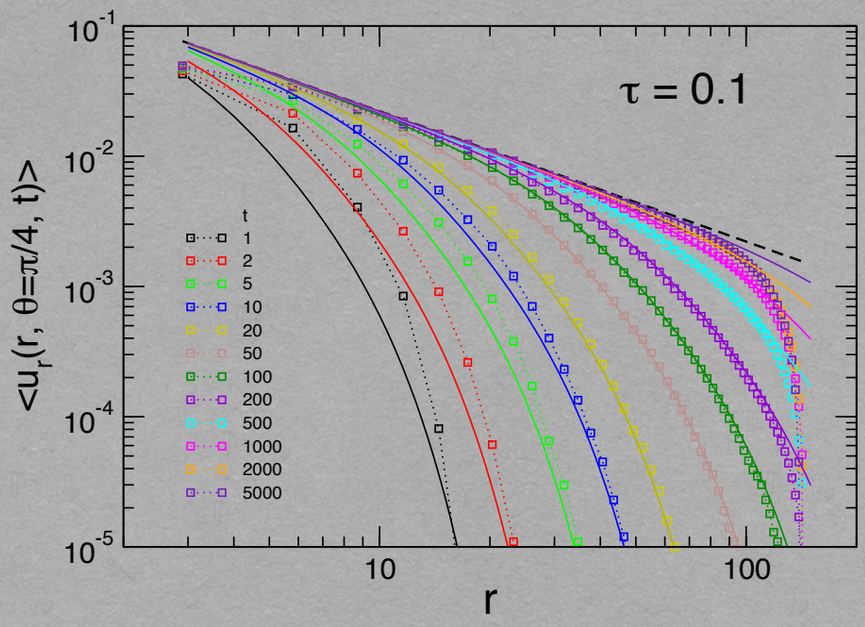
F. Puosi, J. Rottler and J.-L. Barrat, PRE (2014)

Comparison with Continuum Elasticity Theory

Equilibrium response:
Eshelby inclusion problem



Transient regime:
solution of a **diffusion equation** for the displacement field in an homogeneous medium in the presence of a perturbation due to a set of two force dipoles in the origin.

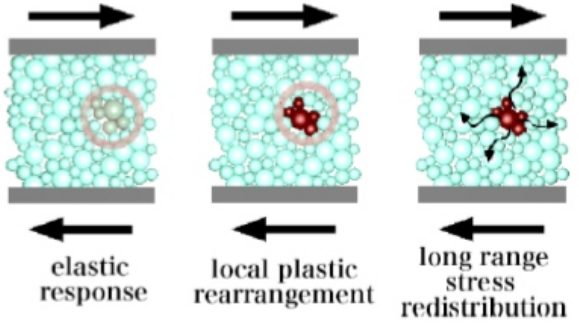


F. Puosi, J. Rottler and J.-L. Barrat, PRE (2014)

… a “mesoscopic” approximation

A 2D scalar field for the “local” stress

$$\frac{d}{dt}\sigma(r,t) = \mu\dot{\gamma} - 2\mu \int dr' G(r,r') \frac{d}{dt}\epsilon^{pl}(r';t)$$



propagator $G^{\infty}(r,\theta) = 2cos(4\theta)/\pi r^2$ “local” in fourier space $\hat{G} = -4\frac{q_x^2 q_y^2}{q^4}$

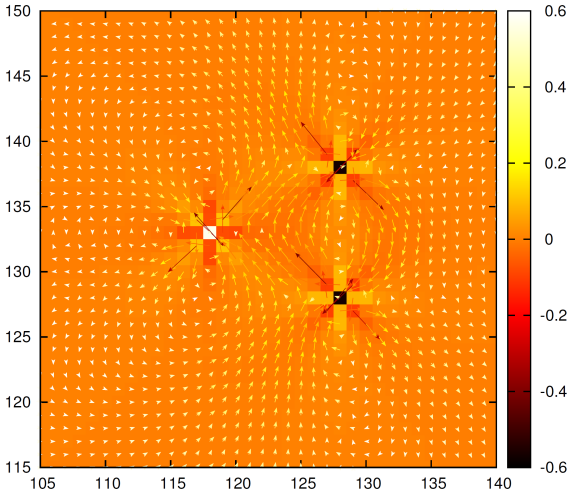
plastic strain change: $\frac{d}{dt}\epsilon^{pl}(r,t) = \frac{1}{2\mu\tau} n(r,t)\sigma(r,t)$

Case of thermally activated events $\dot{\gamma} = 0$

Plastic activation $(n: 0 \rightarrow 1)$

$$p_{\mathtt{on}} = \tau_{\mathtt{on}}^{-1} \min[e^{-(\sigma^2 - \sigma_{\mathtt{Y}}^2)/2T}, 1]$$

Elastic recovery $(n: 1 \rightarrow 0)$ $\quad p_{\mathtt{off}} = \tau_{\mathtt{off}}^{-1}$



+ (eventually) independent tracers moving according to the resulting deformation fields.

Workflow (physical protocol)

```
InitializeEverything();

for (i=0; i<t_max; i++){
    UpdateStateVariablesActivated();
    ComputePlasticStrainChange();
    TransformToFourierSpace();
    Convolution();
    #ifdef TRACERS
        CalculateDisplacementField();
    #endif
    AntitransformFromFourierSpace();
    EulerIntegrationStep();
    #ifdef TRACERS
        UpdateTracersPositions();
    #endif
    if (condition(i)){
    DoSomeMeasurementsAndAccumulate();
    }
}

PrintResults();
```

CUDA_Kernels , cudaMemSet

CUDA_Kernel

CUDA_Kernel or Thrust call

cuFFT

CUDA_Kernel

CUDA_Kernel or Thrust call

cuFFT

CUDA_Kernel

CUDA_Kernel

Thrust calls + C host code

C host code

CUDA basic kernel example

Euler integration step

pre-computed in Fourier space

$$\sigma_i(t+\delta t) = \sigma_i(t) + (\mu\dot{\gamma} - 2\mu \overbrace{\sum_j G_{ij} n_j \Delta\epsilon_j}) \delta t$$

```
__global__ void Kernel_EulerIntegrationStep(REAL *a, const REAL *b, REAL gamma, REAL dt)
{
    int idx = blockIdx.x*blockDim.x+threadIdx.x;
    int idy = blockIdx.y*blockDim.y+threadIdx.y;
    if ( idx < LX && idy < LY){
        int index = idx*LY + idy;
        //There is a factor of 2 missing here (multipling b), that would cancell with the factor of \
        1/2 that is missing in the calculus of b as well. Everyone happy.
        a[index]+=(gamma+b[index])*dt;
    }
}
```

Kernel Call

```
void EulerIntegrationStep(float shear_rate){
    dim3 dimBlock(TILE_X, TILE_Y);
    dim3 dimGrid(LX/TILE_X, LY/TILE_Y);
    assert(dimBlock.x*dimBlock.y<=THREADS_PER_BLOCK);
    assert(dimGrid.x<=BLOCKS_PER_GRID && dimGrid.y<=BLOCKS_PER_GRID);
    Kernel_EulerIntegrationStep<<<dimGrid, dimBlock>>>(d_sigma, d_dsigma_r, shear_rate, TIME_STEP);
    //TODO: Replace this by a Thrust transformation.
}
```

cuFFT library call

Plan declaration

```
cufftHandle plan_c2r;
cufftHandle plan_r2c;
```

```
    // cuFFT plan
    #ifdef DOUBLE_PRECISION
    cufftPlan2d(&plan_c2r, LX, LY, CUFFT_Z2D);
    cufftPlan2d(&plan_r2c, LX, LY, CUFFT_D2Z);
    #else
    cufftPlan2d(&plan_c2r, LX, LY, CUFFT_C2R);
    cufftPlan2d(&plan_r2c, LX, LY, CUFFT_R2C);
    #endif
```

Plan usage

```
void TransformToFourierSpace(){
    #ifdef DOUBLE_PRECISION
    CUFFT_SAFE_CALL(cufftExecD2Z(plan_r2c, d_epsilon_dot_r, d_epsilon_dot));
    #else
    CUFFT_SAFE_CALL(cufftExecR2C(plan_r2c, d_epsilon_dot_r, d_epsilon_dot));
    #endif
}
```

CUDA kernel RNG use

Philox: A counter-based RNG from the “Random 123 library” (2011)

Fast, easy to parallelize, use minimal memory/cache resources (ideal for GPU), require very little code. Have passed all of the rigorous SmallCrush, Crush and BigCrush tests in the extensive TestU01 suite.

```
__device__
void PhiloxRandomPair(const unsigned int index, const unsigned int time, float *r1, float *r2)
{
    RNG2 rng;
    RNG2::ctr_type c_pair={{}};
    RNG2::key_type k_pair={{}};
    RNG2::ctr_type r_pair;
        //key
        k_pair[0]= philox_seed; //eventually do: PHILOX_SEED + sample;
        //counters
        c_pair[0]= time;
        c_pair[1]= index;
        //pair of uniform random numbers generation
        r_pair = rng(c_pair, k_pair);
        //normalize to (0,1]
        *r1= u01_open_closed_32_53(r_pair[0]);
        *r2= u01_open_closed_32_53(r_pair[1]);
}
```

```
__global__ void Kernel_UpdateStateVariablesActivated(const REAL* d_sigma, bool* d_strain_sign, bool* d_state,\
                                                      const REAL temp, unsigned int time)
{
    int idx = blockIdx.x*blockDim.x+threadIdx.x;
    int idy = blockIdx.y*blockDim.y+threadIdx.y;
    if ( idx < LX/2 && idy < LY){
        int index = idx*LY + idy;
        bool actualstate;
        REAL actualsigma, lelast;
        float r1, r2;
        PhiloxRandomPair(index, time, &r1, &r2);

        for (int j=0;j<2;j++)
        {
            REAL rannum = r1*(j==0)+r2*(j==1);
```

Thrust Library basic usage

- High-Level Parallel Algorithms Library
- Parallel Analog of the C++ Standard Template Library (STL)
- Portability and Interoperability (CUDA, TBB, OpenMP,...)

```
REAL ComputeAverageStress(){
    thrust::device_ptr<REAL> sigma_ptr (d_sigma);
    return thrust::reduce(sigma_ptr,sigma_ptr+NN, (REAL) 0, thrust::plus<REAL>())/(REAL)NN;
}
```

(REAL is either float or double)

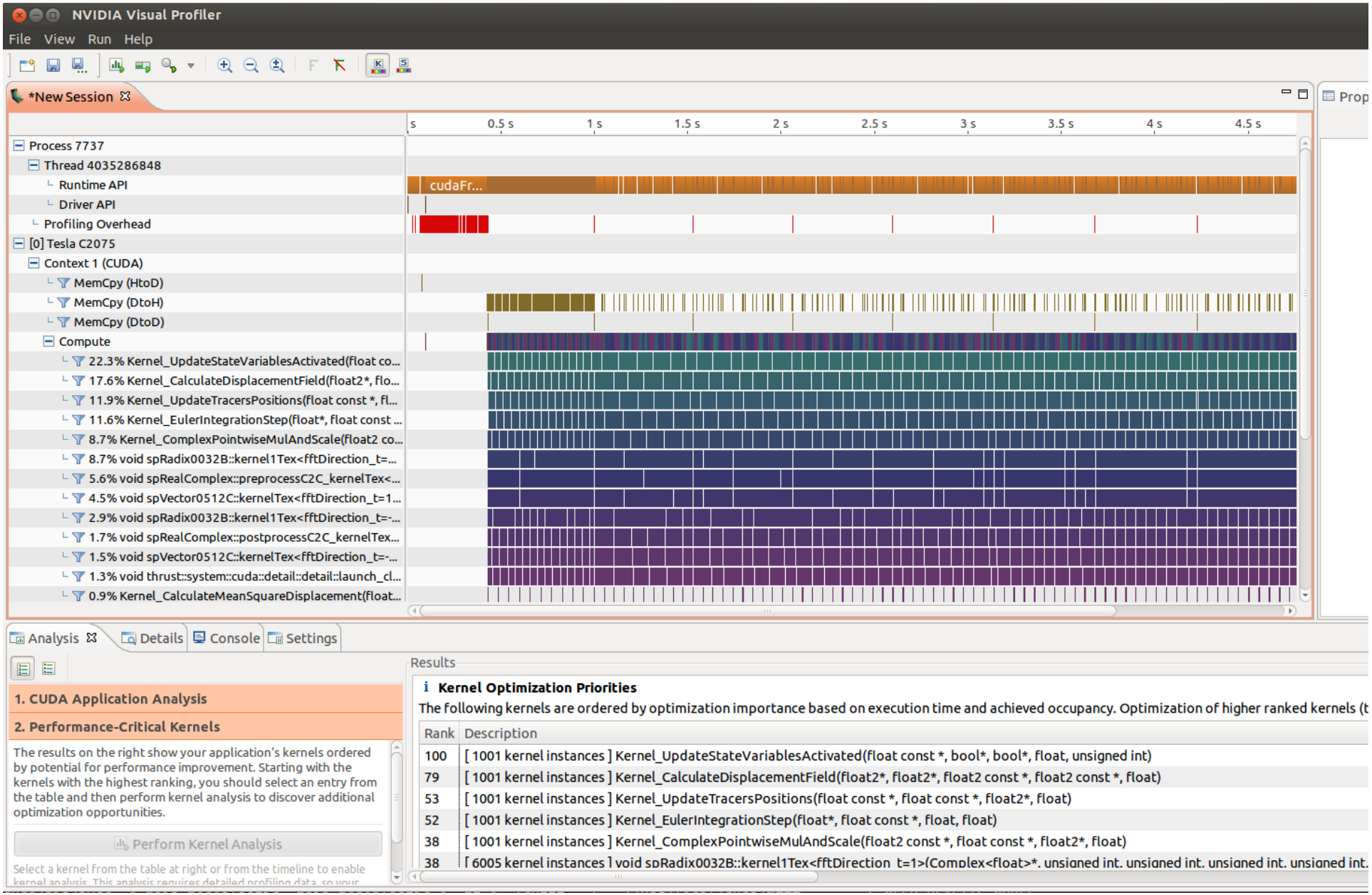
Thrust Library “advanced” usage

$$S(q,t) = \frac{1}{N_{trac}} \left\langle \sum_{n=1}^{N_{trac}} \exp[i\vec{q}\cdot(\vec{r_i}(t+t_0) - \vec{r_i}(t_0))] \right\rangle_{t_0}$$

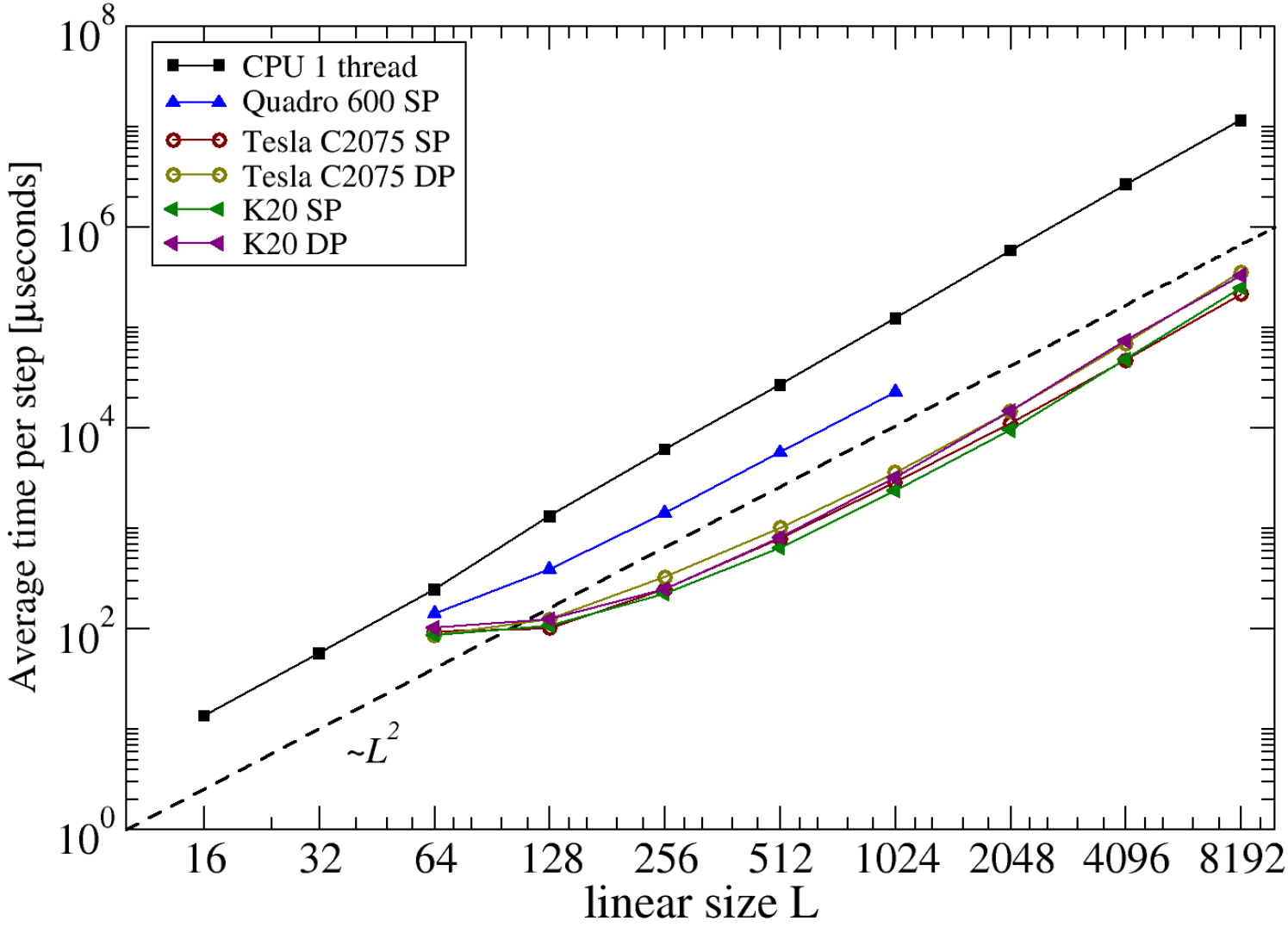
$$q_x = \frac{2\pi}{L_x} i \quad q_y = \frac{2\pi}{L_y} j$$

```
void CalculateAccumulateTrajectoriesStructureFactor(const unsigned int index_t, const unsigned int index_tw, \
                                                    const unsigned int twtd){

    dim3 dimBlock(QTILE_X, QTILE_Y);
    dim3 dimGrid(QNX/QTILE_X, QNY/QTILE_Y);
    assert(dimBlock.x*dimBlock.y<=THREADS_PER_BLOCK);
    assert(dimGrid.x<=BLOCKS_PER_GRID && dimGrid.y<=BLOCKS_PER_GRID);
    thrust::device_ptr<unsigned int> qsquares_ptr(d_qsquares);
    thrust::device_vector<unsigned int> q2(NN);
    thrust::device_ptr<REAL> sf_ptr(d_sf_aux);
    thrust::device_ptr<REAL> sf_out_ptr(d_sf_out);
    thrust::device_ptr<unsigned int> qq_ptr(d_qsquares_out);
    thrust::pair<thrust::device_ptr<unsigned int>, thrust::device_ptr<REAL> > new_end;
    thrust::device_ptr<unsigned int> occurence_ptr(d_qsquares_occurence);

    for (unsigned int i=0; i<index_tw+1; i++){
    Kernel_CalculateTrajectoriesSF<<<dimGrid, dimBlock>>>(d_tracers,&d_tracerstw[i*NTRACERS], d_sf_aux,\
                                                          LX, LY);
    //Do a copy of q^2 in the i*LY+j order
    thrust::copy(qsquares_ptr,qsquares_ptr+NN,q2.begin());
    //Sort the recentry calculated structure factor according to q2, this will sort the keys as well
    thrust::sort_by_key(q2.begin(),q2.end(),sf_ptr);
    //reduce_by_key. Warning: It DOESN'T support in place keys, nor values. Estimated time to realize this: 12hs
    new_end = thrust::reduce_by_key(q2.begin(),q2.end(),sf_ptr,qq_ptr,sf_out_ptr);
    assert(new_end.first-qq_ptr == qqssize);
    thrust::transform(sf_out_ptr,sf_out_ptr+qqssize, occurence_ptr, sf_out_ptr, thrust::divides<REAL>());

    thrust::copy(sf_out_ptr,sf_out_ptr+qqssize,h_sf);
        for(int k=0;k<qqssize;k++) h_sf[(index_t-i*twtd)*qqssize+k] += h_sf[k];
        h_sf_counter[index_t-i*twtd] += 1;
    }
}
```

Profiling with NVVP



Performance



Speedups
~ 50x

(CPU code not optimal)

A lot more to exploit, starting with CPU-GPU concurrency

Unexpected similarity between Fermi and Kepler GPUs

- cuda 6.0 vs cuda 5.0 ?
- ECC enabled/disabled ?
- very low occupancy in both cases ?

Froggy use

```
# This will give you access to the 2 GPUS of one node.
# More exactly, it gives you all the CPU cores that are associated to the 2 GPUS of one node.
[bzizou@froggy1 ~]$ oarsub -I --project test -l /nodes=1/gpu=2 -t gpu
[ADMISSION RULE] Set default walltime to 1800.
[ADMISSION RULE] Modify resource description with type constraints
[COMPUTE TYPE] Setting compute=NO
[GPUNODE] Adding gpu node restriction
OAR_JOB_ID=349173
Interactive mode : waiting...
Starting...
Connect to OAR job 349173 via the node frogkepler3

# You can get the id of the gpus with the oarprint command on the gpuset property:
[bzizou@frogkepler3 ~]$ oarprint gpuset
1
0
```

- Nice nodes, nice GPUs, clear tutorials, everything runs smoothly.
- GPUs are evolving fast. Opportunity to upgrade the cluster compute power by replacing ONLY the accelerator.
- Oar GPU target can perhaps be improved.
  - Possible cross access socket-GPU when CUDA_SET_DEVICE() is used by the application.
  - Allow possibility of targeting the same GPU with many processes. *timesharing=user,**
  - In general 1CPU+1GPU jobs --> we “waste” 15CPUs. A GPU process should block the full node if it wants to have exclusive access to the GPU.
- Annoying, but necessary thing: Max Wall-time = 96hs
  - Stop/Restart coding, is difficult when we compute many things on-the-fly.
  - Automatic checkpoints (BLCR like...)? Is it possible for a GPU context?