Supplemental Material for: "Driving Rate Dependence of Avalanche Statistics and Shapes at the Yielding Transition"

Chen Liu,[1, 2] Ezequiel E. Ferrero,[1, 2] Francesco Puosi,[3, 1, 2] Jean-Louis Barrat,[1, 2] and Kirsten Martens[1, 2]

[1] *Université Grenoble Alpes, LIPHY, F-38000 Grenoble, France*
[2] *CNRS, LIPHY, F-38000 Grenoble, France*
[3] *Université de Lyon, Laboratoire de Physique, ENS Lyon, CNRS, 46 Alleé d'Italie, F-69007 Lyon, FranceUniv. Grenoble Alpes, LIPHY, F-38000 Grenoble, France*

(Dated: January 22, 2016)

We describe in detail the model used in the manuscript and explain our numerical implementation set to run in parallel on GPUs. We provide also some details about the post-processing and analysis of the raw simulation results.

THE MESOPLASTIC MODEL

We study the scalar elasto-plastic model in two ($2d$) and three dimensions ($3d$) under the presence of an imposed shear-rate, following the modifications proposed by Nicolas et al. [1] to the model of Picard and coworkers [2, 3].

An amorphous system is represented by a coarse-grained scalar field $\sigma(\boldsymbol{r}, t)$, denoting the instantaneous deviatoric shear stress of the system at spatial position $\boldsymbol{r}$ and time t upon the application of a simple shear. An over-damped dynamics is imposed for this scalar quantity following some basic rules: (i) The stress loads locally in an elastic manner while it is *below* a certain yield stress $\sigma_{\mathtt{Y}}(\boldsymbol{r})$. (ii) When the local stress overcomes the local yield stress, a *plastic event* occurs. Dissipation occurs locally, expressed as a progressive drop of the local stress, together with a redistribution of the stresses in the rest of the system, provided by a long-range elastic perturbation. This process stops when a criterion for the accumulated local strain is met, the region recovers its elastic properties and a new local yield threshold is chosen from a given distribution.

The shear stress perturbation caused on the system is computed within the framework of tensorial linear elasticity assuming an isotropic incompressible material [2]. A Green's function $G(\boldsymbol{r}, \boldsymbol{r}')$ relates the stress variation $\delta\sigma$ at each point in space with the corresponding component of the plastic strain $\gamma^{pl}(\boldsymbol{r}'; t)$ associated with a plastic event occurring at $\boldsymbol{r}'$. The perturbation given by the elastic propagator $G(\boldsymbol{r}, \boldsymbol{r}')$ can be approximated by the far field expression [2, 3] of the continuum mechanics solution [4]

$$\delta\sigma(\boldsymbol{r}, t) = \mu \int d\boldsymbol{r}' G(\boldsymbol{r}, \boldsymbol{r}')\gamma^{pl}(\boldsymbol{r}'; t) \tag{1}$$

where μ is the shear modulus. This kernel decays with the distance as $1/r^d$ ($r \equiv |\boldsymbol{r} - \boldsymbol{r}'|$) and changes sign according to the angle sustained between the shear direction and the interaction vector $\theta \equiv \arccos((\boldsymbol{r}-\boldsymbol{r}')\cdot\boldsymbol{r}_{\dot\gamma^{(\mathtt{ext})}})$, with a particular quadrupolar symmetry. For example, in $2d$ we have $G(\boldsymbol{r}, \boldsymbol{r}') \equiv G(r, \theta) \sim \frac{1}{\pi r^2} cos(4\theta)$ in polar coordinates. The self interaction $G_0 \equiv G(\boldsymbol{r}, \boldsymbol{r})$ is chosen to be a negative constant that rules the local dissipation rate.

Dynamics at zero temperature

We can define the model as a d-dimensional scalar field $\sigma(\boldsymbol{r})$, $\boldsymbol{r} \equiv (x_1, x_2, \ldots, x_d)$ subjected to the following evolution in real space

$$\frac{d}{dt}\sigma(\boldsymbol{r}, t) = \mu\dot\gamma^{(\mathtt{ext})} + \frac{d}{dt}\delta\sigma(\boldsymbol{r}, t) \tag{2}$$

$$= \mu\dot\gamma^{(\mathtt{ext})} + \mu \int d\boldsymbol{r}' G(\boldsymbol{r}, \boldsymbol{r}') \frac{d}{dt}\gamma^{pl}(\boldsymbol{r}'; t) \tag{3}$$

where we have imposed a global elastic loading $\dot\gamma^{(\mathtt{ext})}$ on top of the perturbation induced by plastic events.

The picture is completed by a dynamical law for the plastic strain γ^{pl}. Following [3], we use a law relating the plastic strain velocity of a region undergoing a plastic deformation to the instantaneous local stress.

$$\frac{d}{dt}\gamma^{pl}(\boldsymbol{r}, t) = \frac{1}{\mu\tau} n(\boldsymbol{r}, t)\sigma(\boldsymbol{r}, t) \tag{4}$$

Here τ is just a mechanical relaxation time that fixes the time units, $n(\boldsymbol{r})$ is a local "state variable" which takes values $n = \{0, 1\}$ indicating whereas the system at position $\boldsymbol{r}$ is plastically active ($n = 1$) or not ($n = 0$).

The concept of "active" and "inactive" regions is intimately related to the discretization of space. Let us say for the moment that different "patches" of the system hold at each time a single value of $n(\boldsymbol{r})$, that is modified according to the following local rules:

$$n(\boldsymbol{r}, t) : \begin{cases} 0 \to 1 & \text{if} \quad \sigma > \sigma_y \\ 0 \leftarrow 1 & \text{when} \quad \int dt' |\partial_t\sigma(t')/\mu + \dot\gamma^{pl}(t')| \geq \gamma_c \end{cases} \tag{5}$$

Model parameters

As in Ref [1] we choose randomly from a distribution the local yield thresholds σ_y. More precisely, we imagine a potential energy landscape (PEL) with energy barriers $E_y \equiv \sigma_y^2/4\mu$. This landscape is composed of metabasins of exponentially distributed depths E_y. Small jumps between PEL basins are neglected and only larger jumps corresponding to the irreversible rearrangements at low enough temperature are considered. In other words, a lower cutoff $E_y^{\mathtt{min}} = \mu\gamma_c^2/4$ is introduced in the energy barrier distribution

$$P(E_y) = \Theta(E_y - E_y^{\mathtt{min}})\lambda e^{-\lambda(E_y - E_y^{\mathtt{min}})}. \tag{6}$$

The parameters λ and γ_c determine the average yield strain $\langle\gamma_y\rangle$. We have choose here $\gamma_c = 0.035$ and $\lambda = 700$ such that $\langle\gamma_y\rangle \simeq 7,7\%$, which is a realistic value. The distance (in terms of strain) among metabasins minima can be expressed in units of the strain γ_c used to define the cutoff $E_y^{\mathtt{min}}$. For simplicity, this distance is choosen to be equal to γ_c. Therefore, once it yields, a block will remain plastic until it has accumulated a total strain equal or greater than γ_c.

The time and stress units, τ and μ are chosen to be the unity without loose of generality.

NUMERICAL APPROACH

Our system is described by a $3d$ (or else $2d$) scalar field $\sigma(\boldsymbol{r})$, and a state variable $n(\boldsymbol{r})$ for each block of a spatially discretized space. This is, each spatial block of volume $v_0 = \delta x\delta y\delta z$, centered at position $\boldsymbol{r} = \{x, y, z\}$ is represented by a single value of the scalar fields in the nodes of a cubic lattice $\{x \pm \delta x/2, y \pm \delta y/2, z \pm \delta z/2\} \rightarrow (i, j, k)$.

In practice, we have a $L_x \times L_y \times L_z$ array of real variables σ_{ijz} representing the local stresses and a boolean array n_{ijz} with identical dimensions holding the binary state of the blocks. Further, we discretize also the time t in Eq.2, choosing a small discrete time step $dt \ll 1$ that we keep constant during all the simulation process.

In order to simulate the equation of motion (2) for the local stresses, beforehand we choose a *mechanically stable* initial configuration $\sigma_{ijz}(t = 0)$. Such a configuration has to ensure that the sum of all stresses in each column or row of the cubic box should be equal. Typically we choose $\sigma_{ijk} = 0$ for all $\{i, j, k\}$, and consistently, all state variables to be initially on the "inactive" state $n_{ijk} = 0$.

Once we have an initial configuration at hand, we evolve Eq.2 with a simple Euler integration method. We avoid any kind of numerical integration problems by choosing an integration time step dt small enough. We have used in this work $dt = 10^{-2}$ and 10^{-3}, which are sufficient to avoid integration problems. Notice nevertheless, that certain minor details of the resulting curves can depend on dt, as for example the lower cutoff of P_S and P_T. This effect of a finite integration step is more prominent at large driving rates as can be seen in Fig.2b-inset of the manuscript.

$$\sigma_{ijk}(t+dt) = \mu\dot{\gamma}^{(\mathtt{ext})}dt + \mu dt \sum_{i'j'k'} G_{(ijk),(i'j'k')} n_{i'j'k'}(t) \frac{\sigma_{i'j'k'}(t)}{\mu\tau} \tag{7}$$

After each integration step for σ_{ijk} we update the state variables n_{ijk} according to the rules defined in Eq.5. In there, n changes from 0 to 1 as soon as the local stress overcomes the local threshold. If so, an auxiliary variable is set to accumulate (from zero) the local strain during its evolution. When the accumulated total strain reaches a given value γ_c the active phase stops, and n goes back to 0. The dynamics goes on, updating consistently σ_{ijk} and n_{ijk}, until a stop criterion is matched (total simulation time, total strain deformation, etc.).

Boundary conditions and spectral method for the dynamics

Numerical simulations are done in finite size systems. Contrary to fully-connected models, in this case space is defined and we are forced to define boundary conditions. Since we are interested in bulk quantities, we can choose periodic boundary conditions (p.b.c.) in all directions without loose of generality. In problems where the analysis of wall effects on the system rheology is particularly relevant, the numerical approach is different from the one we present here (see for example [2, 5]). The choice of p.b.c. will also simplify the numerical implementation. In particular, it allows for the use of a technique called pseudo-spectral method, that we describe in the following.

The second term on the RHS of Eq.2 is an integral over all space, since the kernel $G(\boldsymbol{r}, \boldsymbol{r}')$ is long-range. If we Fourier transform with respect to the variable $\boldsymbol{r}'$, the integral over space is simplified to independent products for each Fourier mode $\boldsymbol{q}$

$$\int d\boldsymbol{r}' G(|\boldsymbol{r} - \boldsymbol{r}'|) n(\boldsymbol{r}')\sigma(\boldsymbol{r}') \qquad \longrightarrow \qquad G_{\boldsymbol{q}}\tilde{n}(\boldsymbol{q})\tilde{\sigma}(\boldsymbol{q}) \tag{8}$$

This transforms a non-local, time-consuming, sum over spatial coordinates into a local operation in the Fourier modes that can be trivially performed in a parallel scheme. Therefore we can evolve the local stresses in Fourier space according to the transformed equation of motion

$$\frac{\partial\sigma_{\boldsymbol{q}}}{\partial t} = \mu\dot{\gamma}\delta(\boldsymbol{q}) + \frac{1}{\tau} G_{\boldsymbol{q}}\tilde{n}(\boldsymbol{q})\tilde{\sigma}(\boldsymbol{q}) \tag{9}$$

Of course, since after each update of $\sigma(\boldsymbol{r})$ we need to update also $n(\boldsymbol{r})$, it is necessary to transform stresses back to real space. So the process includes two Fourier transforms (one forward and one backward) at each time-step of the dynamics. Nevertheless, with this technique we reduce the computing time for the convolution (Eq.8) from $O(N^2)$ to $O(N\log(N))$. In addition, this operation is highly optimized in standard libraries (e.g., FFTW3, cuFFT) that make the method even more suitable for a parallel implementation.

PARALLEL IMPLEMENTATION ON GPU

In the last years the use of GPUs to accelerate simulations has burst out in many areas of physics and science in general.

Following previous GPU implementations from some of us [6**?** , 7], we have implemented a GPU-based parallel implementation of the elasto-plastic model described above. Our codes are written in C++ and C for CUDA [8]. For this project we had worked with NVIDIA GPUs. The CUDA programming framework makes it simpler the access to many low level directives, preserving a more compact and easy to read code[9]. Simulations were ran on Kepler architecture (GK208) GPUs, the Tesla K20.

Update routine, CUDA kernels and main stream

We use an algorithm developed from scratch to implement our model. Self-developed CUDA kernels and available optimized parallel libraries are used alternately. Among the libraries we can name: the GPU-suited Fast Fourier Transform library *cuFFT* [10] from the *NVIDIA CUDA Toolkit*, the *STL*-like *Thrust* library [11] of parallel standard algorithms (reductions, scannings, searches, etc.) and the counter-based Random Number Generator named PHILOX, from the *Random123* open-source library [12].

As anticipated in the previous section, the equation of motion for the local stresses is resolved in several steps:

1) Computation of $\dot{\gamma}^{pl}(\boldsymbol{r})$ in real space, basically the product $\sigma(\boldsymbol{r})n(\boldsymbol{r})$ times constants (Eq.4).

2) Discrete Fourier transform (DFT) of $\dot{\gamma}^{pl}(\boldsymbol{r})$.

3) $G_{\boldsymbol{q}}$ times $\dot{\gamma}^{pl}_{\boldsymbol{q}}$ pointwise multiplication (Eq. 8)

4) Euler step integration in Fourier space using the increment of Eq.7.

5) Inverse discrete Fourier transform of the resulting $\tilde{\sigma}(\boldsymbol{q})$ giving as a result the scalar stress field at the incremented time $\sigma(\boldsymbol{r}, t+dt)$.

6) Update of state variables $n(\boldsymbol{r})$ according to Eq.5.

From a computational viewpoint, steps 1, 3, 4 and 6 can be trivially computed in parallel, since we need only to read and write arrays locally with no interdependence. This is easily implemented in massively parallel routines with a SIMD (Single Instruction Multiple Data) approach. We use either self-developed CUDA kernels or Thrust well-settled functions for each of this steps. At steps 2 and 5, we make use of the cuFFT library, especially powerful in the transformations of real or complex arrays with an x dimension being strictly a power of two.

Besides the evolution of the system, we need also to calculate certain physical quantities with some frequency as time evolves. We are interested, for example, in instantaneous global values as the average stress and average activity. To account for this measurements, we make intensive use of *Thrust* routines, as the parallel *reduce*.

Our CUDA kernels are moderately optimized, trying to keep aligned and coalesced memory access, avoiding threads divergence and atomic functions. Further optimizations are still possible, but we choose to preserve code readability over elaborated tricks that obfuscate the code for a negligible speedup. As defensive programming techniques we use assertions and each routine is independently tested before implementation.

The structure of the main stream is simply as follows:

- Initialization
- Time loop:
  - System Update (as itemized above)
  - if (condition) Measures
  - if (condition) Print results
- Final averages, printing and cleaning.

We have created a C++ class to host our functions and keep a clean `main()` routine where we set up the physical protocol for the simulation. CUDA kernels, used by the class functions, are described in a separate file for further clearness.

Validation of the overall program is made by computing a full flow curve in a wide range of shear-rates and comparing with independent serial implementations of the same model. These tests also serve us to know that with the parallel implementation and the use of a GPU we obtain a speed up of 100x and beyond respect to a single-CPU serial version of the same algorithm.

Our source codes are freely available to download, modify and use under GNU GPL 3.0 at [13].

POST-PROCESSING AND DATA ANALYSIS

To obtain compound averages as the distributions of stress drop sizes or the avalanche shapes presented in the manuscript, we make use of *Python* scripts [14] doing a post-processing of the stress time series output by our algorithm. Since data files attain a considerable size (up to a couple of gigabytes), care is taken in using fast load functions and list operations rather than array operations, to reduce the processing time; which is, in any case, always much shorter than the simulation time to obtain the raw data.

Power-laws exponent fitting and error estimation procedure

In Fig. 1 and Fig. 2 of the manuscript scaling regimes at low shear rate span over wide ranges (about four decades for P_S and two decades for P_T or P_x). As the scaling regimes can be quite clearly identified by eye, we perform power-law fits in manually selected regions, avoiding the lower and upper cutoffs. Of course, this introduce a small variation of the fitted exponents and an uncertainty, that is considered in the error bar estimation of the measured exponents.

Having determined the values of τ and $\tau^{'}$ from P_S and P_T respectively, we fix them and proceed to scaling analyses of data at different L and $\dot{\gamma}$ to estimate the exponents d_f and α that provide the best collapse of the upper (exponential) cutoff of the distributions. We can qualify the collapse either by eye or by computing the relative deviations of the rescaled curves. This allows us to have an error of estimation for exponent values within the indicated error bars.

We have also tried a more sophisticated and systematic way of estimating the exponent of power law histogram as described in the work by Planet et al. [15]. However, it does not give us better estimations than the ones we extract from the ad hoc fittings.

In Table 1 of the manuscript we present all our fitted exponents with their corresponding error bars. The only non measured exponent is z, that is computed from the relation $z = d_f/\delta$.

Stress-drop shape averages and comparison with analytical predictions

Each stress-drop has a certain duration T. To analyze the stress-drop shapes, we consider only the stress-drops of durations T that belong to the scaling regime of P_T. This is, in the window of T in which P_T is a power law. We are interested in the average shape of stress drops at different T. In order to improve the statistics of the shape averages we do a logarithmic binning on T, with a rather small binning step, and consider the mid value in logarithmic scale of each bin as representative of the duration of all stress-drops found within the bin. In this way, we are able to collect a good amount of stress drops with the "same" duration. The average shape of the stress-drops within a bin centered at T, is computed as follows: Each stress-drop i can be represented as $V_i(t)$. First we rescale all stress-drops on their duration T_i defining $\tilde{V}_i(\tilde{t}) \equiv V_i(\tilde{t}T_i)$, where $\tilde{t} = t/T_i \in [0,1]$. Then, we define the average shape of the stress-drops as $V_T(\tilde{t}) \equiv \frac{1}{N_T}\sum_i \tilde{V}_i(\tilde{t})$, with N_T the number of stress-drops found in the interval.

Once we get $V_T(\tilde{t})$ for different T, we fit them with the equation proposed in [16], $V_T(\tilde{t}) \propto B(\tilde{t}(1-\tilde{t}))^c(1-a_s(\tilde{t}-0.5))$, leaving B, c and a_s as free parameters. From this fitting, we find the power law dependence of B on T, the exponent $c \approx \delta - 1$, and the behavior of $a_s(T)$, presented in Fig.3(b) of the manuscript.

Multi-parameter dependence of the shape asymmetric values a_s and a_g

We have characterized the "degree of asymmetry" of a stress drop shape with the parameters a_s and a_g defined in the manuscript.

We propose a common functional dependence for both of them with $\dot{\gamma}$, L, T as $a_x \approx CL^{-b}\dot{\gamma}^{-\epsilon}T^{-m}$, whit C being a prefactor. A first step is to rescale a_x getting $\tilde{a}_x \equiv a_x L^b \dot{\gamma}^{\epsilon}$. If our assumption is right and the value of b and ϵ are well chosen, $\tilde{a}_x(T)$ for different $\dot{\gamma}$ and L can be collapsed together on the same curve $\tilde{a}_x = CT^{-m}$. Indeed, this is what we observe. The rescaled and merged data sets are used to process a unique power law fitting. We fit $\log_{10}(\tilde{a}_x)$ versus $\log_{10}(T)$ with $y = kx + \kappa$ by a least-square method, where $k = -m$ and $C = 10^{\kappa}$. For a chosen pair (b,ϵ), the least-square fitting method gives us the error $E(b,\epsilon)$ qualifying the fit. We span our choices over a wide domain of (b,ϵ) and repeat the procedure for each pair, obtaining a surface $E(b,\epsilon)$. The minimum of $E(b,\epsilon)$ provides therefore the best possible choice of (b,ϵ). After fixing them, we extract the parameters m and C from the corresponding fit of the rescaled data.

This proposed functional dependence and the above explained procedure allow us to display a_s or a_g in a continuum plane L-$\dot{\gamma}$, as shown in the inset of Fig.3(a) in the manuscript.

REFERENCES

[1] A. Nicolas, K. Martens, and J. L. Barrat, Epl **107**, 6 (2014).
[2] G. Picard, A. Ajdari, F. Lequeux, and L. Bocquet, The European Physical Journal E **15**, 371 (2004).
[3] G. Picard, A. Ajdari, F. Lequeux, and L. Bocquet, Phys. Rev. E **71**, 010501 (2005).

[4] J. D. Eshelby, Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences **241**, 376 (1957).

[5] A. Nicolas and J.-L. Barrat, Phys. Rev. Lett. **110**, 138304 (2013).

[6] K. Martens, L. Bocquet, and J.-L. Barrat, Phys. Rev. Lett. **106**, 156001 (2011).

[7] E. E. Ferrero, S. Bustingorry, and A. B. Kolton, Phys. Rev. E **87**, 032122 (2013).

[8] NVIDIA Corporation, "CUDA C Programming Guide Version 7.0," (2015).

[9] Nevertheless, all our routines could be straight-forward translated to a open standard platform, as OpenCL, and get in dependency on the hardware vendor.

[10] NVIDIA Corporation, "CUDA library for the Fast Fourier Transform, DU-06707-001_v7.0," (2015).

[11] J. Hoberock and N. Bell, "Thrust: A parallel template library," (2010), version 1.7.0.

[12] J. K. Salmon, M. A. Moraes, R. O. Dror, and D. E. Shaw, in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11 (ACM, New York, NY, USA, 2011) pp. 16:1–16:12.

[13] https://bitbucket.org/ezeferrero/epm.

[14] Python Software Foundation. Python Language Reference, version 2.7.6. Available at http://www.python.org.

[15] R. Planet, S. Santucci, and J. Ortín, Phys. Rev. Lett. **105**, 029402 (2010).

[16] L. Laurson, X. Illa, S. Santucci, K. T. Tallakstad, K. J. Maloy, and M. J. Alava, Nature Communications **4** (2013).