



Original software publication

nlchains: A fast and accurate time integration of 1-D nonlinear chains on GPUs

L. Pistone*, M. Onorato

Dipartimento di Fisica, Università di Torino, via Pietro Giuria 1, 10125, Torino, Italy



ARTICLE INFO

Article history:

Received 27 February 2019

Received in revised form 29 May 2019

Accepted 31 May 2019

Keywords:

FPU

Nonlinear chain

GPU

CUDA

ABSTRACT

We present nlchains, a software for simulating ensembles of one-dimensional Hamiltonian systems with nearest neighbor interactions. The implemented models are the α - β Fermi–Pasta–Ulam–Tsingou model, the discrete nonlinear Klein–Gordon model with equal or site-specific masses, the Toda lattice and the discrete nonlinear Schrödinger equation. The integration algorithm in all cases is a symplectic sixth order integrator, hence very accurate and suited for long time simulations. The implementation is focused on performance, and the software runs on graphical processing unit hardware (CUDA). We show some illustrative simulations, we estimate the runtime performance and the effective scaling of the cumulative error during integration. Finally, we give some basic pointers to extend the software to specific needs.

© 2019 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

Code metadata

Current code version	v1.1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2019_56
Legal Code License	MIT
Code versioning system used	git
Software code languages, tools, and services used	C++14, CUDA, MPI, CMake
Compilation requirements, operating environments & dependencies	Linux, CUDA SDK, GCC, Boost, Armadillo (and one compatible LAPACK library)
Link to developer documentation/manual	https://bitbucket.org/pisto/nlchains/src/master/README.md
Support email for questions	lorenzo.pistone@unito.it

Software metadata

Current software version	v1.1.0
Permanent link to executables of this version	https://bitbucket.org/pisto/nlchains/downloads/nlchains-gite6c2001-opt64
Legal Software License	MIT
Computing platforms/Operating Systems	Linux
Installation requirements & dependencies	CUDA \geq 10.0, NVIDIA card (compute capability \geq 3.5, 6.0 or 7.0), Ubuntu 18.04 environment (see <code>Dockerfile-ubuntu1804-cuda100</code> in the source tree)
If available, link to user manual – if formally published include a reference to the publication in the reference list	https://bitbucket.org/pisto/nlchains/src/master/README.md
Support email for questions	lorenzo.pistone@unito.it

1. Introduction and motivation

Nonlinear chains, that is systems of points with nonlinear nearest-neighbor interactions, have an important role in the un-

* Corresponding author.

E-mail address: lorenzo.pistone@unito.it (L. Pistone).

derstanding of the basic processes of nonlinear physics and statistical mechanics. Despite being in general toy models, or at best crude approximations of real physical systems, they show a rich phenomenology, and even basic questions such as the thermalization dynamics or heat transport law often do not have a definitive answer. Not by chance, one of the first numerical experiment in physics was the Fermi–Pasta–Ulam–Tsingou, that is the renowned FPUT experiment [1,2]. The aim of the experiment was to observe the thermalization of a linear chain of masses and springs, when a small nonlinearity is added. The apparent paradox of the FPUT experiment is that thermalization in the FPUT system cannot be attained in a short time, and so it could not be observed with the computers of the fifties.

Recently, we and collaborators have applied tools of Wave Turbulence [3,4], a statistical description of weakly interacting wave systems, to seek universal traits in the thermalization dynamics of nonlinear chains, [5–9] and also [10,11]. To this end, we needed a software to run large simulation of these systems. These numerical experiment are often time-consuming because thermalization oftentimes requires a long time compared to the wave periods of the corresponding linearized system, and also because large ensembles of realizations of the same chain are needed to extract meaningful statistics. For these reason, the code had to be written with performance but also accuracy in mind, because the Hamiltonian structure of the system needs to be preserved for long times. We could not find a properly published software that could fit our needs. Recently, an implementation of the FPUT system (possibly extendible to other model) has been published [12], however it was not designed with performance as a strong point, but rather it has a pedagogical value, stressing on fast prototyping and code readability. Our contribution here is the software that we used in our research, that is the implementation on graphical processing unit hardware (GPUs) of the time integration for several one-dimensional nonlinear chains. The principal value of this software is the effort that we have spent in tuning its performance.

2. The implemented algorithms

In Table 1 we list the models that are already implemented in *nlchains*. The q_j and p_j variables are the conjugate coordinates and momenta, while ψ_j is a complex variable, for j an index that runs from 0 to $N - 1$, with N the length of the chain. We have included the original α - β Fermi–Pasta–Ulam–Tsingou model (FPUT), the discrete nonlinear Klein–Gordon model with equal masses (DNKG) and disordered, site-specific masses (dDNKG), the Toda lattice and the discrete nonlinear Schrödinger equation (DNLS). The specific version of the Toda lattice potential has been chosen so that it is tangent to the α -FPUT model, that is the potential energy between two adjacent masses $V(q_{j+1} - q_j) = V(\Delta q_j)$ is $V_{\text{Toda}}(\Delta q_j) = V_{\text{FPUT}}(\Delta q_j) + O(\Delta q_j^4)$. Extending the software to other systems with similar potentials is expected to be easy. The parameters α , β , m and m_i can be set by the user.

2.1. The integration scheme

As mentioned in the introduction, the choice in the algorithm was driven by the need of a good performance but also accuracy in the conservation of the Hamiltonian structure of the nonlinear chains. We chose the 6th order symplectic Yoshida integrator [13]. This algorithm suited our needs for the following reasons. It allows for very long time simulation, as being symplectic it avoid secular growth of conserved quantities such as the Hamiltonian. It is explicit, which makes it direct and easy to implement. We chose the sixth order, as it resulted in an optimal trade-off between computational speed and accuracy in

our field of research. Higher order schemes of the same type can be implemented with trivial modifications of the source code (details will be given in a later section).

For reference, we report here the principle of this integration scheme. Since all the systems considered are Hamiltonian, the formal solution to the time evolution of some initial state $z = (q_0, \dots, q_{N-1}, p_0, \dots, p_{N-1})$ (or $z = (\psi_0, \dots, \psi_{N-1})$ for the DNLS model) is given by the Poisson bracket

$$\dot{z} = \{z, H(z)\}. \quad (1)$$

Now the above equation can be formally solved by introducing the differential operator $D_H = \{\cdot, H(\cdot)\}$, to obtain

$$\dot{z} = D_H z, \quad z(\delta t) = e^{\delta t D_H} z(0). \quad (2)$$

In general it is not possible to give an explicit form of $e^{\delta t D_H}$, as that coincides with integrating the system. However, since the Poisson brackets are bilinear, it is possible to take advantage of the fact that the Hamiltonians in consideration are the sum of different terms. If one splits the Hamiltonians in two contributions, $H = H_A + H_B$, then the differential operator also splits in two corresponding parts, $D_H = D_A + D_B$, to get

$$z(\delta t) = e^{\delta t D_H} z(0) = e^{\delta t (D_A + D_B)} z(0). \quad (3)$$

In general, D_A and D_B do not commute, so it is not possible to write $e^{\delta t D_H} = e^{\delta t D_A} e^{\delta t D_B}$, however it can be shown that it is possible to generate a scheme of the type

$$e^{\delta t D_H} = e^{c_1 \delta t D_A} e^{d_1 \delta t D_B} e^{c_2 \delta t D_A} e^{d_2 \delta t D_B} \dots e^{c_k \delta t D_A} e^{d_k \delta t D_B} + O(\delta t^x) \quad (4)$$

where the coefficients c_k and d_k are real and x is some positive integer, the order of the integration scheme. Given that, when the splitting $H = H_A + H_B$ is chosen carefully so that H_A and H_B are individually integrable, it is possible to generate a fully explicit integrations scheme.

For the real models, the most natural choice is to split the Hamiltonian into kinetic and potential energy, that is $H_A = \frac{1}{2} p_j^2$ and $H_B = V(q_{j+1} - q_j)$. For the DNLS model, we follow [14] and we split the Hamiltonian in the linear and nonlinear contributes, that is $H_A = \sum \frac{1}{2} \beta |\psi_j|^4$ and $H_B = \sum |\psi_{j+1} - \psi_j|^2$. The solution to the c_k and d_k of Eq. (4) that we hard-coded in the software is taken from [13], it is of the 6th order and the values of the coefficients c_k and d_k are

$$\begin{aligned} \{c_0, \dots, c_7\} = \{ & 0.392256805238780, 0.510043411918458, \\ & -0.471053385409757, \\ & 0.068753168252518, 0.068753168252518, \\ & -0.471053385409757, \\ & 0.510043411918458, 0.392256805238780 \} \end{aligned} \quad (5)$$

and

$$\begin{aligned} \{d_0, \dots, d_7\} = \{ & 0.784513610477560, 0.235573213359357, \\ & -1.177679984178870, \\ & 1.315186320683906, -1.177679984178870, \\ & 0.235573213359357, \\ & 0.784513610477560, 0 \}. \end{aligned} \quad (6)$$

Note that the coefficient d_7 is 0, hence it is possible to merge the last integration of one step $e^{\delta t c_7 D_A}$ with the first integration of the next step $e^{\delta t c_0 D_A}$. We expect that if the user needs higher order integrators (e.g. 8th), modifying the software to this end should be quite trivial.

In concluding this section, we remark that for all the models except DNLS, the integration is performed in physical space. For the DNLS model however, because the linear and nonlinear

Table 1

The list of implemented models in *nlchains*. For real models, q_j and p_j are the conjugate coordinate and momentum at index $j \in [0, N)$ with N the length of the chain ($\tilde{q}_j = p_j$). For DNLS, ψ_j is a complex variable, and i is the imaginary unit.

Subprogram	Equation of motion and Hamiltonian density
DNKG	$\dot{p}_j = q_{j-1} - 2q_j + q_{j+1} - mq_j + \beta q_j^3$ $H_j = \frac{1}{2}p_j^2 + \frac{1}{2}(q_{j+1} - q_j)^2 + \frac{1}{2}mq_j^2 + \frac{1}{4}\beta q_j^4$
dDNKG	$\dot{p}_j = q_{j-1} - 2q_j + q_{j+1} - m_j q_j + \beta q_j^3$ $H_j = \frac{1}{2}p_j^2 + \frac{1}{2}(q_{j+1} - q_j)^2 + \frac{1}{2}m_j q_j^2 + \frac{1}{4}\beta q_j^4$
FPUT	$\dot{p}_j = (q_{j-1} - 2q_j + q_{j+1}) (\alpha (q_{j+1} - q_{j-1}) + 1) +$ $\quad + \beta ((q_{j+1} - q_j)^3 - (q_j - q_{j-1})^3)$ $H_j = \frac{1}{2}p_j^2 + \frac{1}{2}(q_{j+1} - q_j)^2 + \frac{1}{3}\alpha (q_{j+1} - q_j)^3 + \frac{1}{4}\beta (q_{j+1} - q_j)^4$
Toda	$\dot{p}_j = \frac{1}{2\alpha} (e^{2\alpha(q_{j+1}-q_j)} - e^{2\alpha(q_j-q_{j-1})})$ $H_j = \frac{1}{2}p_j^2 + \frac{1}{4\alpha^2} (e^{2\alpha(q_{j+1}-q_j)} - 2\alpha (q_{j+1} - q_j) - 1)$
DNLS	$i\dot{\psi}_j = -(\psi_{j-1} - 2\psi_j + \psi_{j+1}) + \beta \psi_j \psi_j ^2$ $H_j = \psi_{j+1} - \psi_j ^2 + \frac{1}{2}\beta \psi_j ^4$

sub-Hamiltonians are diagonal in Fourier and physical space respectively, the integration scheme become essentially a refined split-step scheme [15], and the time evolution of the linear part is performed in Fourier space, as suggested (but not implemented) in [14]. This has the side effect that a large number of Fast Fourier Transforms (FFTs) are performed in integrating the DNLS model, that is 14 FFTs per step: the number of non-zero d_k coefficients, doubled to account for a direct and inverse FFT to go back and forth from physical space. As we will explain later, this has important consequences in the numerical accuracy and speed of the algorithm.

2.2. Other calculated quantities

In order to observe recurrence and equipartition, the software also calculates with a user-settable interval the average energy per eigenstate of the linearized system (that is $\alpha = 0$ and $\beta = 0$ in Table 1). For the DNKG, FPUT and Toda systems, it can be shown that the eigenstates of the linearized systems are the so called normal modes,

$$a_k = \frac{1}{\sqrt{2\omega_k}} (\tilde{p}_k - i\omega_k \tilde{q}_k) \quad (7)$$

where the \tilde{q}_k and \tilde{p}_k are the discrete Fourier transform of the q_j and p_j , and ω_k is the dispersion relation of the system, that is $\omega_k = \sqrt{m + 4 \sin(\pi k/N)^2}$. For the DNLS model, the normal modes a_k are simply the discrete Fourier transform $\tilde{\psi}_k$ of the physical space variables ψ_j , and the dispersion relation is $\omega_k = 4 \sin(\pi k/N)^2$. In all these cases the energy per mode is defined as

$$e_k = \omega_k \langle |a_k|^2 \rangle \quad (8)$$

with $\langle \cdot \rangle$ being the average over the ensemble. For the dDNKG model, the eigenstates v_k and corresponding eigenvalues ω_k^2 are calculated numerically from the matrix representation of the system of ordinary differential equations that correspond to the equations of motion of the chain. The energy per mode is then obtained from and explicit projection of the coordinates and momenta vectors $q = (q_0, \dots, q_{N-1})$ and $p = (p_0, \dots, p_{N-1})$ over the eigenstates,

$$e_k = \langle v_k \cdot (\omega_k^2 q_k + p_k) \rangle / 2. \quad (9)$$

From the average energy per mode, the software calculates and outputs the associated information entropy,

$$S_{\text{inf}} = \sum_{k=0}^{N-1} e'_k \log(e'_k), \quad e'_k = \frac{N}{E} e_k, \quad (10)$$

where N is the length of the chain and $E = \sum e_k$ the total linear energy, and the Wave Turbulence entropy

$$S_{\text{WT}} = - \sum_{k=0}^{N-1} \log(e'_k). \quad (11)$$

Traditionally, Eq. (10) has been used to monitor the route to thermalization of the FPUT problem, but in the framework of Wave Turbulence only Eq. (11) has the properties of an entropy function, that is it can be proven that statistically it is a monotonic decreasing function in time. However, it can be shown [8] that both these two entropies are greater than zero out of thermal equilibrium, and zero at perfect equipartition, and they are essentially equivalent for the purpose of monitoring the route to thermalization.

3. Software architecture

The software is packaged in a stand-alone Linux executable. Build requirements and compilation steps are documented in the readme (file README.md) that comes with the sources. An important detail of the building process is that all models except DNLS are optimized at compile-time for a specific chain length. A warning is generated if a build of *nlchains* is launched for a value of the chain length that does not correspond to the optimized value, but the computation is carried out anyway. For more details we refer the reader to Section 5.

The simulation is set up with a number of command line arguments. The invocation in the shell is in the following form:

```
[<MPI launcher>] nlchains <model> \\  
                                <common options> \\  
                                <model specific options>
```

The executable can be run as-is to run on a single GPU on the current host, or through MPI to split the ensemble of realizations on different GPUs. The user should assign to each compute node the same number of MPI processes as the number of GPU attached to the node itself. When running through MPI, the software expects the presence of a shared filesystem, because full-state dumps are written on disk with the MPI shared I/O facilities.

The GPU code is written in CUDA, and the host code is written in C++14, hence a NVIDIA GPU card (minimum compute capability 3.0) is required. Please note that the software is implemented in floating point double precision, and so a card of the Tesla line is suggested, because consumer-level cards are largely limited in the double precision performance.

Table 2

List of the output files. The value of *prefix* is set with the command line option `-p` (see Table 3)

Filename	Description
<i>prefix-step</i>	Full dump of the ensemble state
<i>prefix-linenergies-step</i>	Energy per eigenstate
<i>prefix-entropy</i>	List of entropies
dDNKG only:	
<i>prefix-eigenvectors</i>	eigenvectors
<i>prefix-omegas</i>	pulsation of the eigenvectors

For the purpose of performance comparison, and as a fallback in the case that no GPU is available on the host, all the models also have a CPU-only implementation. To launch it, simply append to the model name `-cpu`, for example `DNLS-cpu`. The implementation is single-threaded but multiple cores and nodes can be used for parallelization in the same way that the GPU implementation can be launched on multiple GPUs. The implementation also makes use of advanced SIMD instructions present in modern processors (SSE, AVX and AVX-512), and as such it should be regarded competitive in performance. This implementation is used here to compare to the performance of the GPU implementation (see Section 5.2). However, it is not the main message of this paper to present a CPU implementation. The interested reader is referred to the implementation notes attached to this paper as supplementary material for further information.

3.1. File formats

The format of the inputs and outputs is raw binary. The list of output files is shown in Table 2.

The initial state of the ensemble, and the full state dumps are in the same format, that is as a C++ array `double[C][N][2]`, where *C* is the ensemble size, *N* is the chain length, and the last two-elements array contains the conjugate coordinate q_j and momentum p_j of the element in the chain. For the DNLS model, the format is `std::complex<double>[C][N]`, that is the real and imaginary part of the element in the chain take the places of the conjugate coordinate and momentum.

The energy per linear mode dumps is simply a plain array of doubles, `double[N]`. Note that when a discrete Fourier transform is involved in the calculation of the linear energies (all cases except dDNKG), we use the FFTW [16] convention (FFTW_FORWARD) for the sign of the exponent.

The list of entropies has the format `double[][3]`, where each triplet contains the absolute time (the step number times the timestep), the Wave Turbulence entropy of Eq. (11) and the information entropy of Eq. (10).

For the DNKG model, two additional files are created: the eigenvectors file in format `double[N][N]` (the first index being the index of the eigenvector), and the square root of the corresponding eigenvalues (corresponding to the pulsation of the eigenstates) as a `double[N]`.

3.2. Command line options

The first argument to *nchains* is one of the subprogram listed in Table 1. If no other argument is printed, the list of arguments applicable to the model is printed, together with a short description. In Table 3 we describe the most important command line options that are common to all models and are necessary to run a simulation. Other common switches are available (such as for terminating when a given entropy threshold is reached), but for brevity refer to the readme available in the source tree.

Table 3

Command line options that are common to all models.

Argument	Description
<code>-i</code>	initial state of the ensemble
<code>-n</code>	length of the chain
<code>-c</code>	number of realizations in the ensemble
<code>--dt</code>	time step
<code>-b</code>	number of time steps per kernel invocation
<code>-s</code>	number of time steps to run
<code>-p</code>	prefix for output filenames

The model parameters α , β , m and m_i (see Table 1) can be set respectively with `--alpha`, `--beta`, `-m`; for the DNKG model `-m` takes a value, for the dDNKG it takes a filename with a list of mass values in the format `double[N]`.

Note that the time granularity of the dumps and the entropy calculation is equal to the value of `-b`. A larger interval for the space-consuming full state and linear energy dumps can be set with the option `--dump_interval`.

4. Illustrative examples and accuracy

As an example of the use case of this software, we show in Fig. 1 the interesting case of the Toda lattice dynamics versus the α -FPUT dynamics, with the same initial state in terms of linear wave modes and the corresponding entropy curves given by Eq. (11). The initial state of the ensemble (empty circles in the top half of Fig. 1) has been initialized with random values for the energy per linear mode, the same set for all realizations in the ensemble and rescaled to have $E = 1$, but each realization had a different phase of the normal modes a_k . This scheme ensures that all realization have the same initial linear energy. The value of α is 0.5, so that the nonlinear terms of the Hamiltonians are small compared to the linear part, because in this regime it is easier to observe the equipartition of the linear modes. The timestep is set to $\delta t = 0.1$ and the total number of steps is $2 * 10^7$. We can see that for the Toda lattice the entropy (dashed line in the bottom half of Fig. 1) quickly settles to a value not far from the initial one, and that clearly signals that the system is not thermalized, while for the α -FPUT system (solid line) the system eventually reaches equipartition. This can be appreciated from direct inspection of the energy per mode at the final state: in the top half of Fig. 1, the filled circles are the energy per mode of the α -FPUT chain at equipartition (minus some statistical fluctuation due to the finite size of the ensemble), while empty squares are the final state for the Toda lattice. For more elaborate examples we refer the interested reader to the papers [7,8].

In Fig. 2 we show the scaling of the accuracy of the simulation as a function of the step size. In order to measure the accuracy, we control the value of a known exact integral of motion, that is the value of the Hamiltonian $H(t)$, or the total energy. The simulation is then run for a fixed total time $T = 100000$ (arbitrary units), but with a different time step, and hence a different total number of steps. The initial state of each simulation is initialized in a similar way to the data of Fig. 1. The accuracy is then calculated as the average relative deviation from the initial value of the energy,

$$\epsilon = \langle |H(T) - H(0)| / H(0) \rangle. \quad (12)$$

Since we use a sixth order symplectic integrator, it is expected that the scaling of the error is of the type $\epsilon = O(\delta t^6)$. We see in Fig. 2 that we get the expected scaling of the error for the models FPUT, DNKG, Toda and dDNKG, up to a saturation around $\epsilon \sim 10^{-13}$. For the model DNLS, we get a saturation much earlier, at around $\epsilon \sim 10^{-9}$. This can be explained by the fact that the DNLS algorithm as we mentioned earlier requires 14 Fast Fourier Transforms (FFTs) for each single time step, hence

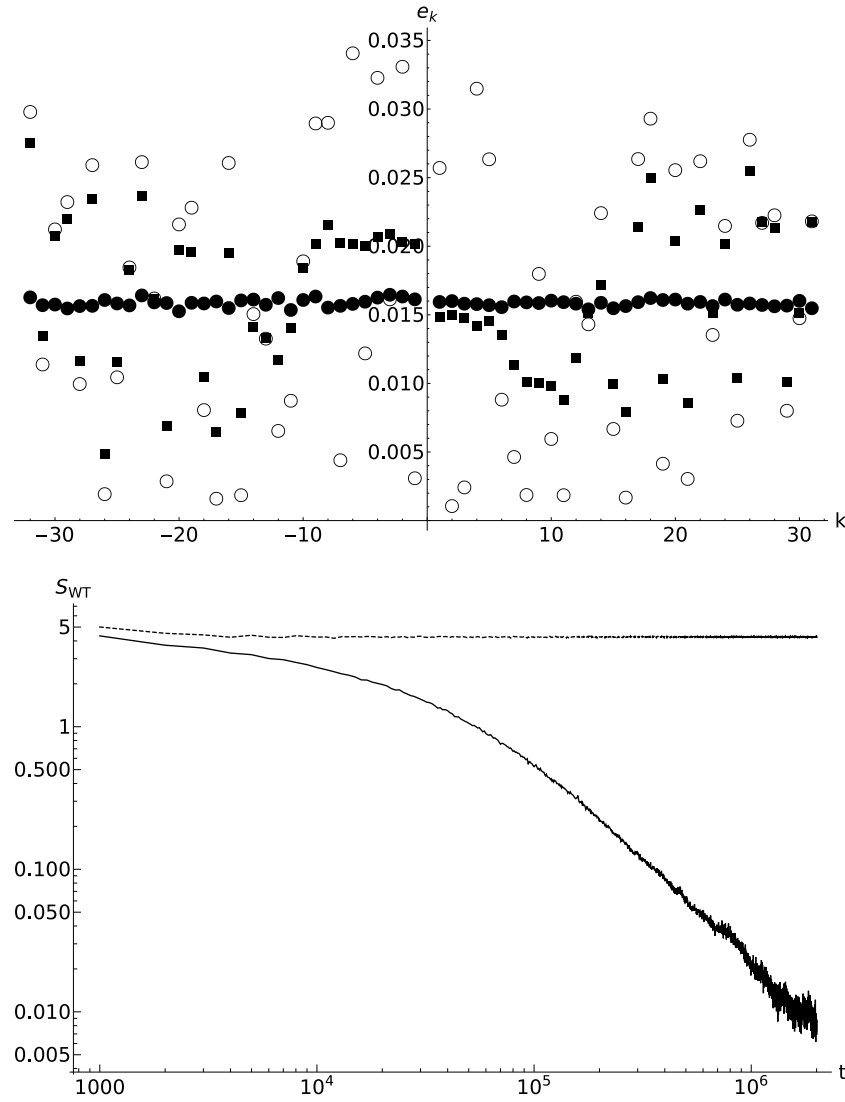


Fig. 1. A run from the same initial state of the Toda and α -FPUT model, for the same value of α , for a chain with length 64 and an ensemble size of 4096. Top figure, the energy per mode, Eq. (7): initial state \circ , thermalized α -FPUT chain \bullet , final state for the Toda lattice \blacksquare . Bottom figure, the entropy curves, solid and dashed for the α -FPUT and Toda models respectively.

the data undergoes many more additions and multiplications compared to the other models, and numerical errors due to the finite accuracy of machine numbers accumulates faster. In fact, the behavior of ϵ when δt is small is of the type $\epsilon = O(\delta t^{-1})$, that is it is proportional to the number of FFTs. This is further corroborated by the fact that when the nonlinear parameter is set to zero, that is when the symplectic integration scheme is no longer a source of error as it becomes exact, the scaling of the error is still $\epsilon = O(\delta t^{-1})$ even for large values of δt .

All the data shown in this Section is attached as supplementary material.

5. Implementation and software extensibility

The software does not have an interface for adding new models, and essentially all the available settings can be accessed through command line options. This is a design choice because performance has been the top priority in developing this code: in the case of GPU programs good performance is in general attained with tight coupling between host and GPU code, and avoiding unnecessarily generalization (such as allowing to specify new models through virtual functions). In lieu of a runtime

flexibility, *nlchains* is designed to be easily modifiable and extendible. In support of the source code comments, a manual with extensive implementation notes, a description of the internal utility interfaces and examples for their usage is provided in the supplementary material of this paper (also as a stand-alone document in the source code tree, [supporting-material/documentation/implementation-notes.pdf](#)). We refer the user who needs to adapt *nlchains* to his or her needs to this manual, to avoid cluttering this paper with implementation details.

5.1. Notes on the implementation for the end user

We mention here only some implementation details that are useful even for the user who does not intend to modify the functionality of *nlchains*.

All models except DNLS have three different implementations of the GPU kernels. One implementation is optimized for a chain length less than 32 (`move_chain_in_thread`), another one is generic for all chain lengths (`move_split`), and the last one (`move_chain_in_warp`) must be tuned at compile time for a specific chain length greater or equal to 32. The instruction on

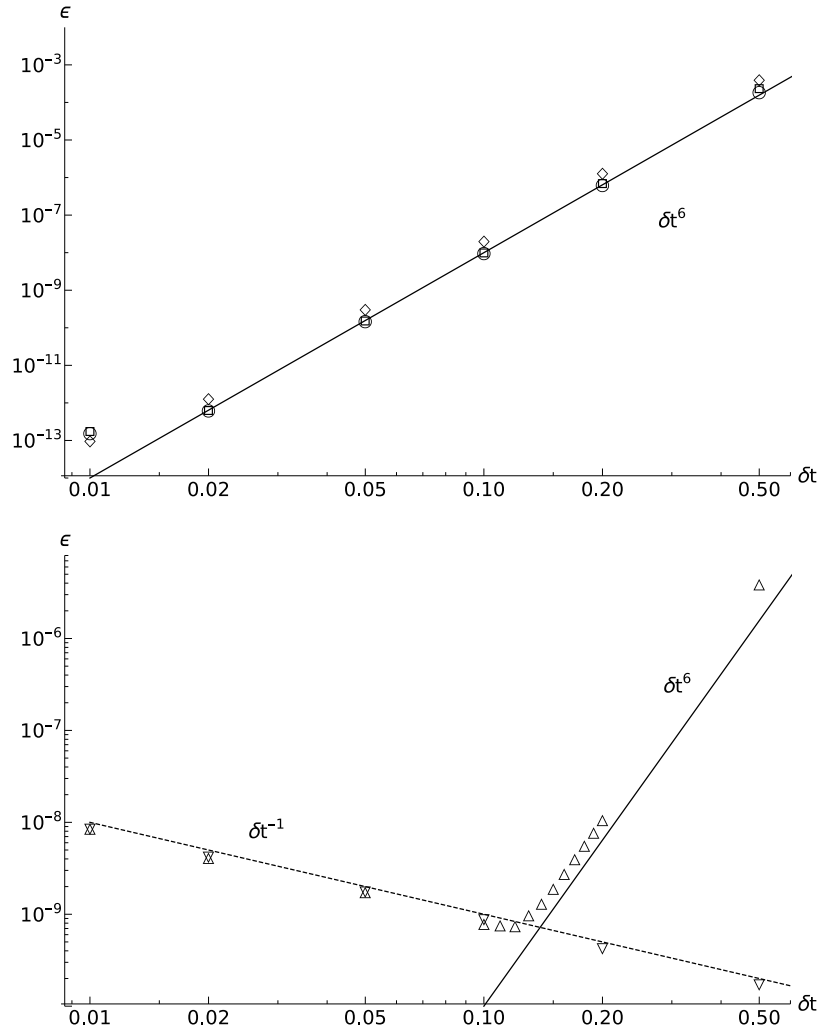


Fig. 2. The accuracy of the integration algorithms and their scalings. Top figure: FPUT \circ , Toda \square , DNKG \diamond , dDNKG $+$. Bottom figure: DNLS \triangle , DNLS with zero nonlinearity ∇ . Solid lines are the scaling δt^6 , dashed line δt^{-1} .

how to select the target chain length optimization are detailed in the readme within the sources. This optimized implementation is much faster compared to the generic one (referred to as the “split” kernel for how the q_j and p_j variables are kept in separate buffers), because the ensemble state is kept in registers for all the duration of the kernel, rather than being read and written in global memory. Since register memory is limited, the maximum target chain length is around 1024, though it is not possible to give a precise upper bound as that depends on the GPU hardware and the compiler version. The user should always compare the runtime of the optimized version and the generic one by using the command line option `--split_kernel`.

We showed earlier how the integration of the DNLS model essentially turns into a refined split-step scheme with a large number of FFTs involved. We use the library cuFFT [17] for this purpose. In order to save some of the numerous round trips of the ensemble to and from registers and global memory, the linear and nonlinear operators have been implemented as cuFFT callbacks. In general, this leads to a large performance gain, however there might be circumstances where the non-callback version may be faster. The use of callbacks can be suppressed selectively with the command line options `--no_linear_callback` and `--no_nonlinear_callback`, and we encourage to benchmark the various combinations with and without callbacks for a defined ensemble size and hardware combination.

Table 4

Performance measurements on a single K40m card (clocked at 875MHz and 3004MHz for core and memory respectively) for a dataset with a chain length of 64 and 1024 copies, and a kernel batching size (option `-b`) of 100000 (1000 for the DNLS model). File dumping has been disabled in these runs.

Chain	Steps/second
DNKG (optimized kernel)	163904
dDNKG (optimized kernel)	150003
FPUT (optimized kernel)	83254
Toda (optimized kernel)	37431
FPUT (split kernel)	4012
DNLS (with cuFFT callbacks)	2303
DNLS (without cuFFT callbacks)	1149

5.2. Performance measures

Performance of the software depends on a large number of factors. For reference, in Table 4 we show some rough estimates of the number of steps per second for most of the implementations present in the software, for a fixed chain length size of 64 and ensemble size of 1024. These performance figures should roughly scale linearly with the number of GPUs, and linearly with the inverse of the chain length and ensemble sizes. As mentioned earlier, the big performance bottleneck for the FPUT model is the memory access in the case of the “split” kernel, hence it is crucial to recompile the software for a desired chain length. Such

Table 5

Performance measurements on a Intel Xeon Processor E5-2680, both single threaded and 24 threads, for a dataset with a chain length of 64 and 1024 copies (1008 for the multithreaded test), batching size (option `-b`) of 10000 (1000 for the DNLS model). File dumping has been disabled in these runs.

Chain	Steps/second (1 thread)	Steps/second (24 threads)
DNKG	1229	24510
dDNKG	1280	25623
FPUT	802	17123
Toda	197	4165
DNLS	90	1965

difference is expected also in the models DNKG, dDNKG and Toda. The DNLS model is the slowest of all, due to the large number of FFTs involved. However, it is possible to appreciate the speedup due to the use of cuFFT callbacks.

As previously mentioned, we have implemented the integrators as traditional CPU code, in order to compare the advantages of the GPU implementation. The code makes use of recent vectorization (SIMD) capabilities of recent CPUs, in particular it can use the AVX and AVX-512 instruction sets. The CPU implementation has been tested on an Intel Xeon Processor E5-2680 (which has a release date similar to the NVIDIA K40m card used in the GPU benchmarks), and as such it utilizes the AVX instruction set. It can be parallelized in the same way of the GPU implementation (see Section 3). The results are shown in Table 5, both for a single thread run and a 24-threads run (all the cores of the CPU in use), with the same parameters of the data shown in Table 4. We observe the following. The relative order in terms of performance of the models is essentially the same as the GPU implementation. The multithreaded performance is roughly linear in the number of threads, though it shows the typical signs of saturation: a 20-fold increase is observed when running the simulation with 24 threads. Finally, we see that the speedup of the (optimal) GPU implementation over the multithreaded implementation is significant. For the real models, we observe a speedup of 6.7x (DNKG), 5.9x (dDNKG), 4.9x (FPUT) and 9.0x (Toda). For the DNLS model, the speedup is much more modest (1.2x). This is due to the fact that the GPU implementation suffers from the high-latency global memory operations, while the CPU implementation operates on a single chain at once, and hence the CPU cache is utilized. Unfortunately it was not possible for us to test the code on more recent GPU hardware which should provide a much better memory technology.

6. Impact

The purpose of this software is to provide the scientific community a specialized tool for simulating nonlinear chains (and possibly other simple Hamiltonian, nearest-neighbor systems). While the simulation algorithms is not new, and trivial implementations of the Yoshida sixth order integration algorithms are not particularly difficult, to our knowledge there has been so far no effort to code a simulation with a strong focus on performance, modernity and extendibility of the code, nor software that makes use of heterogeneous hardware architectures such as GPUs.

This software has been essential in our research work [7,8]. Naive implementations of this kind of simulations can have a runtime of weeks. Exploiting the parallelization possibilities of GPU hardware, and a painstaking work of tuning and optimization of the code made the run of a simulation of a typical size a matter of at most hours.

7. Conclusions

In this paper we presented *nlchains*, a specialized software for simulating a number of one-dimensional Hamiltonian systems with nearest neighbor interactions on GPU hardware. The software has been coded during the study of the thermalization of these systems, but other applications are possible, as the speed and accuracy are very good. We have described briefly the usage of the software, and mentioned a few implementation details that can guide the interested user in adapting the software to his or her needs. We provided also a few simulation results, most importantly the scaling of the cumulative errors in the simulations, which matches very well the expected scaling $O(\delta t^6)$ with δt being the step size. To our knowledge, this is also the first implementation of the Yoshida 6th order symplectic integrator for the discrete nonlinear Schrödinger equation as described in [14] with the suggested optimization of leveraging the Fast Fourier Transform for the linear operator of the algorithm.

Declaration of competing interest

The authors wish to confirm that there are no known conflicts of interest associated with this publication.

Acknowledgments

We thank the OCCAM facility, Università di Torino, for providing the hardware necessary to the development of this software. M. O. has been funded by “Progetto di Ricerca d’Ateneo CSTO160004” and by the “Departments of Excellence 2018–2022” Grant awarded by the Italian Ministry of Education, University and Research (MIUR) (L.232/2016).

Appendix A. Supplementary data

Supplementary material related to this article can be found online at <https://doi.org/10.1016/j.softx.2019.100255>.

References

- [1] Fermi E, Pasta J, Ulam S. Studies of nonlinear problems. Tech. rep., I, Los Alamos Scientific Laboratory Report No. LA-1940; 1955.
- [2] Gallavotti G. The Fermi-Pasta-Ulam problem: a status report, vol. 728. Springer; 2008.
- [3] Falkovich G, Lvov V, Zakharov V. Kolmogorov spectra of turbulence. Berlin: Springer; 1992.
- [4] Nazarenko S. Wave turbulence, vol. 825. Springer; 2011.
- [5] Onorato M, Vozella L, Proment D, Lvov Y. Route to thermalization in the α -Fermi-Pasta-Ulam system. Proc Natl Acad Sci USA 2015;112(14). <http://dx.doi.org/10.1073/pnas.1404397112>.
- [6] Lvov YV, Onorato M. Double scaling in the relaxation time in the β -Fermi-Pasta-Ulam-Tsingou model. Phys Rev Lett 2018;120(14):144301.
- [7] Pistone L, Onorato M, Chibbaro S. Thermalization in the discrete nonlinear Klein-Gordon chain in the wave-turbulence framework. Europhys Lett 2018;121(4):44003.
- [8] Pistone L, Chibbaro S, Bustamante M, Lvov Y, Onorato M. Universal route to thermalization in weakly-nonlinear one-dimensional chains. 2018, ArXiv preprint. [arXiv:1812.08279](https://arxiv.org/abs/1812.08279).
- [9] Bustamante MD, Hutchinson K, Lvov YV, Onorato M. Exact discrete resonances in the Fermi-Pasta-Ulam-Tsingou system. 2018, ArXiv preprint. [arXiv:1810.06902](https://arxiv.org/abs/1810.06902).
- [10] Spohn H. The phonon Boltzmann equation, properties and link to weakly anharmonic lattice dynamics. J Stat Phys 2006;124.
- [11] Lukkarinen J. Kinetic theory of phonons in weakly anharmonic particle chains. In: Thermal transport in low dimensions. Springer; 2016, p. 159–214.
- [12] Kashyap R, Sen S. PULSEDYN—A Dynamical simulation tool for studying strongly nonlinear chains. Comput Phys Comm 2019.

- [13] Yoshida H. Construction of higher order symplectic integrators. *Phys Lett A* 1990;150(5):262–8.
- [14] Boreux J, Carletti T, Hubaux C. High order explicit symplectic integrators for the Discrete Non Linear Schrödinger equation. 2010, ArXiv preprint. [arXiv:1012.3242](https://arxiv.org/abs/1012.3242).
- [15] Weideman J, Herbst B. Split-step methods for the solution of the nonlinear Schrödinger equation. *SIAM J Numer Anal* 1986;23(3):485–507.
- [16] Frigo M, Johnson SG. The design and implementation of FFTW3. *Proc IEEE* 2005;93(2):216–31.
- [17] NVIDIA Corporation. cuFFT. 2018, <https://docs.nvidia.com/cuda/cufft/>.