Accepted Manuscript

GPU accelerated fully space and time resolved numerical simulations of self-focusing laser beams in SBS-active media

Sarah Mauger, Guillaume Colin de Verdière, Luc Bergé, Stefan Skupin

PII: S0021-9991(12)00637-7
DOI: http://dx.doi.org/10.1016/j.jcp.2012.10.034
Reference: YJCPH 4286

To appear in: Journal of Computational Physics

Please cite this article as: S. Mauger, G.C. de Verdière, L. Bergé, S. Skupin, GPU accelerated fully space and time resolved numerical simulations of self-focusing laser beams in SBS-active media, Journal of Computational Physics (2012), doi: http://dx.doi.org/10.1016/j.jcp.2012.10.034

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.
GPU accelerated fully space and time resolved numerical simulations of self-focusing laser beams in SBS-active media

Sarah Mauger\textsuperscript{a}, Guillaume Colin de Verdière\textsuperscript{a}, Luc Bergé\textsuperscript{a}, Stefan Skupin\textsuperscript{b,c}

\textsuperscript{a}CEA-DAM, DIF, 91297 Arpajon, France
\textsuperscript{b}Max Planck Institute for the Physics of Complex Systems, 01187 Dresden, Germany
\textsuperscript{c}Friedrich Schiller University, Institute of Condensed Matter Theory and Optics, 07743 Jena, Germany

Abstract

A computer cluster equipped with Graphics Processing Units (GPUs) is used for simulating nonlinear optical wave packets undergoing Kerr self-focusing and stimulated Brillouin scattering in fused silica. We first recall the model equations in full (3+1) dimensions. These consist of two coupled nonlinear Schrödinger equations for counterpropagating optical beams closed with a source equation for light-induced acoustic waves seeded by thermal noise. Compared with simulations on a conventional cluster of Central Processing Units (CPUs), GPU-based computations allow us to use a significant (16 times) larger number of mesh points within similar computation times. Reciprocally, simulations employing the same number of mesh points are between 3 and 20 times faster on GPUs than on the same number of classical CPUs. Performance speedups close to 45 are reported for isolated functions evaluating, e.g., the optical nonlinearities. Since the field intensities may reach the ionization threshold of silica, the action of a defocusing electron plasma is also addressed.

Key words: GPU, CUDA, HMPP, Optical Self-focusing, Stimulated Brillouin Scattering

1. Introduction

Nonlinear optics is a wide field of physics, including various peculiar behaviors of light, such as optical solitons in fibers [1] or parametric amplification...
tion processes [2, 3]. Since four decades, the propagation of light in centro-
symmetric materials promoting a strong Kerr nonlinearity has been the topic
of numerous investigations. With such type of nonlinearity, powerful light
structures can grow in intensity along the optical path, as the refractive
index of the medium becomes dependent on the light intensity. Basically,
an optical wave possessing an input peak power above the so-called criti-
cal power for self-focusing rapidly increases in amplitude and collapses at a
finite propagation distance, whenever there is no saturation effect present
in the medium [4, 5]. This singular process is classically described by the
NonLinear Schrödinger (NLS) equation in higher dimensions, i.e., in (2 + 1)
dimensions for stationary [(x, y, z)] systems or (3 + 1) dimensions for time-
dependent ones [resp. (x, y, z, t)] [6, 7]. In realistic physical media, beam
collapse is halted either by nonlinear optical saturation or damping, or by
the excitation of an electron plasma. The latter situation is met in ultrafast
optics, where ultrashort pulses with femtosecond durations can easily exceed
the self-focusing power threshold for moderate pulse energies [8, 9]. In this
scope, femtosecond laser pulses shrink in the (x, y) diffraction plane due to
Kerr self-focusing, until plasma generation defocuses their most intense com-
ponents. The resulting structure is called "femtosecond filament" which can
propagate over long distances in transparent media such as air, noble gas,
liquid or glass [10, 11, 12]. Femtosecond pulsed beams do not necessarily
collapse as a whole [13, 14, 15], but they can also become unstable in the
diffraction plane and decay into multiple filaments under the action of local
noise at high power levels [16, 17, 18, 19].

For longer pulses extending to the nanosecond scale, another phenomenon
comes into play: The Stimulated Brillouin Scattering (SBS). SBS consists in
the scattering of part of the laser pump into a second electromagnetic wave,
the so-called Stokes wave, through the excitation of acoustic fluctuations in
dense media. In the absence of optical absorption, phonon waves emerge from
the nonlinear process of electrostriction and mainly scatter the Stokes wave
in the longitudinal direction opposite to the pump pulse. Extensively studied
for narrow and broadband laser pulses, SBS occurs in various settings such
as optical fibers [20, 21, 22, 23, 24, 25, 26]. In three-dimensional [(3+1)]
geometries, the coupling between Kerr and SBS nonlinearities can lead to
severe modulational instabilities and enhanced wave collapse. Such violent
dynamics are met, for example, in the optics of high power laser facilities
devoted to inertial confinement fusion [27] and they often initiate front, rear
and bulk damages in silica glasses [28, 29, 30].
Describing the filamentation of laser pulses in solid media requires a high numerical resolution. A self-focusing pump beam, coupled to a counterpropagating Stokes wave, exhibits a transverse width in the diffraction \([x, y]\) plane shrinking by a factor up to 100 compared with its incident value. Hence, its resulting intensity \(I(x, y, z, t)\) rapidly increases by four orders of magnitudes \((\times 10^4)\) by virtue of the conservation of the optical power \(P = \int I(x, y, z, t) \, dx \, dy\). Moreover, this collapse dynamics leads to strong compression events, forming subpicosecond peaks in the pulse temporal profile. For describing such singular evolution and sudden changes in the pulse shape, sufficient resolution is required in space and time. Typically, a spatial resolution of \(\sim 1 \, \mu m\) is needed in the transverse \((x, y)\) plane to correctly resolve transverse self-focusing. Concerning the temporal dimension, the resolution should lie below one femtosecond, in order to properly resolve the plasma response when the latter becomes active. For short pulses of, e.g., 100 fs duration, this requirement can easily be fulfilled using parallel codes with classical Message Passing Interface (MPI) architecture \([31]\) on medium-sized clusters. However, for longer pulses in the nanosecond range, the maximum mesh size we can reasonably exploit on 128 Central Processing Units (CPUs) is typically \(256 \times 256 \times 4096\) points in the space \((x, y, z)\) distributed over a numerical box of, e.g., \(1 \, \text{mm} \times 1 \, \text{mm} \times 5 \, \text{cm}\). This corresponds to spatial steps \(\Delta x = \Delta y \simeq 4 \, \mu m\) and \(\Delta z = 12 \, \mu m\), while the temporal increment remains limited to \(\Delta t = 60 \, \text{fs}\) \([32, 33]\). In other words, the numerical resolution is too coarse for accurately describing the pulse propagation dynamics, in particular with respect to the stage of plasma generation.

Such numerical runs typically need several days of computing time on a state-of-the-art High Performance Computing (HPC) cluster employing hundreds of CPUs. Using higher resolutions, i.e., doubling (at least) the number of points in \((x, y)\) and decreasing the time step by a factor larger than 10, would imply several months of calculations for one run on such a machine. Simply employing more CPUs (tens of thousands) is usually not an option due to the limited availability of such computational infrastructure. An alternative way to overcome this limitation is to exploit Graphics Processing Units (GPUs), which can significantly improve the code performance and thus allows to use a much higher resolution within the same computation time (i.e., less than one week). Indeed, Graphics Processing Units (GPUs) recently provided a new technology for accelerating simulation codes. Their use, originally confined to graphical applications, is now extended to parallel computation and provides boosted numerical performances in various physi-
cal contexts, e.g., among others, pulse propagation in fibers [34], Boltzmann particle transport [35], Brownian dynamics [36] as well as spin models [37]. This evolution became possible with the native support of double precision computations by GPUs along with the development of new tools, such as the Computer Unified Device Architecture (CUDA) designed by NVIDIA [38] or the Open Computing Language (OpenCL) supported by Khronos [39, 40]. The hardware structure of a GPU allows for a huge computation power supplied by a large number of elementary processing units. The internal memory bandwidth is also quite large compared to what is available in a regular CPU. Such hardware features alleviate some of the limitations observed in scientific CPU codes. The only major difficulty is to “feed” the GPU with data, keeping in mind that, e.g., its connexion to the main CPU is done through a PCI-Express link which is limited to a mere 8 GB/s. Here, we will combine GPU computation with standard MPI parallelization to increase the code performance on medium-sized clusters with GPUs available on each computer node.

This paper addresses how GPUs can be exploited in the numerical solving of NLS-type equations modeling singular behaviors in nonlinear optics. In Sec. 2 we recall the model equations. The collapse of a single optical wave as well as of two coupled waves in the SBS context are discussed in Sec. 3. Here, the numerical scheme for classical calculations on CPU cores is briefly sketched. Numerical simulations will be performed for simulating either unmodulated Gaussian laser pulses or phase-modulated ones, expected to inhibit SBS. Sec. 4 details two adaptations of the CPU code with CUDA or Hybrid Multicore Parallel Programming (HMPP) for running on GPUs. Algorithmic performances are discussed in Sec. 5. Improvements of the physical data due to better resolution is commented on in Sec. 6. Section 7 finally concludes this work.

2. Model Equations

Following the standard SBS-Kerr model [1], we consider a linear polarized electric field composed of a forward and a backward component with amplitude $U_1$ and $U_2$, respectively,

$$E(\vec{r}, t) = \sqrt{\frac{\mu_0 \omega_0}{2k_0}} \left( U_1 e^{i k_1 z - i \omega_1 t} + U_2 e^{-i k_2 z - i \omega_2 t} + c.c. \right),$$

(1)
where c.c. means complex conjugate, and $\mu_0$ is the magnetic permeability in vacuum. The forward and backward waves have center frequencies $\omega_i \simeq \omega_0 = 2\pi c/\lambda_0$ ($i = 1, 2$ respectively) and wave numbers $k_i \simeq n_0\omega_0/c = k_0$ where $c$ is the speed of light in vacuum, and $n_0$ is the linear index in silica. Using scalar approximation, the propagation model is derived from the Helmholtz equation

$$\vec{\nabla}^2 E - \frac{1}{c^2}\partial_t^2 E = \mu_0 \left( \partial_t^2 P + \partial_t J \right),$$

where $P$ is the polarization of the medium and $J$ is the current density associated with the dynamic of free electrons. The field envelopes $U_i$ have been normalized such that $I_i = |U_i|^2$ are their intensities in W/cm$^2$. Neglecting dispersion, the polarization $P$ is given by

$$P = \varepsilon_0 \chi^{(1)} E + \varepsilon_0 \chi^{(3)} E^3 + \varepsilon_0 \gamma_e \frac{\Delta \rho}{\rho_0} E,$$

with $\varepsilon_0 \mu_0 = 1/c^2$. Here, the first term refers to linear polarization with $\chi^{(1)} = n_0^2 - 1$ being the linear susceptibility. The second one describes the nonlinear optical polarization with cubic susceptibility $\chi^{(3)}$. The third contribution is the electrostrictive polarization with coefficient $\gamma_e$ for a background material density $\rho_0$. The density fluctuation $\Delta \rho$ is governed by

$$\left[ \partial_t^2 - \Gamma' \vec{\nabla}^2 \partial_t - C_S^2 \vec{\nabla}^2 \right] \Delta \rho = -\frac{\gamma_e}{2n_0c} \nabla^2 E^2,$$

where $\Gamma'$ is the phonon lifetime and $C_S = 5.97 \times 10^5$ cm/s is the acoustic velocity. The acoustic wave number and frequency are given by $q = \| \vec{k}_1 - \vec{k}_2 \| \simeq 2k_1$ and $\Omega = \omega_1 - \omega_2 \simeq \Omega_B = C_S q$, respectively, so that the density fluctuations can be written as

$$\Delta \rho = \bar{\rho} e^{i(qz - \Omega t)} + c.c.,$$

where $\bar{\rho}$ is assumed to be a slowly-varying envelope.

Compared with the standard SBS-Kerr model, we also incorporate the effect of plasma generation through the term $\mu_0 \partial_t J$ in Eq. (2). The scalar current density is given as $J = q_e \rho_e \nu_e$, where $q_e$ is the electron charge, $\rho_e$ is the free electron density, and $\nu_e$ is their velocity. Using a simple Drude model for the electron velocity we find $\partial_t J \approx q_e^2 \rho E/m_e$. Further on, the source equation for $\rho_e$ can be approximated as

$$\frac{\partial \rho_e}{\partial t} = W(I)\rho_{nt} - \frac{\rho_e}{\tau_{rec}},$$

where $\rho_{nt}$ is the nonthermal electron density.
involving the total intensity \( I = |U_1|^2 + |U_2|^2 + (U_1U_2^*e^{2ik_0} + c.c.) \). Here, \( \rho_{nt} = 2.2 \times 10^{22} \text{ cm}^{-3} \) is the density of neutral species, \( \tau_{\text{rec}} \) is the recombination time of electrons in silica and \( W(I) \) is the photo-ionization rate \( (\rho_e \ll \rho_{nt}) \). For intensities less than \( 10^{13} \text{ W/cm}^2 \), avalanche ionization can be discarded and \( W(I) \) is evaluated in the multiphoton limit \( W(I) = \sigma_K I^K \), where \( K = \text{mod}(U_i/\hbar\omega_0) + 1 \) is the minimum number of photons with energy \( \hbar\omega_0 \) necessary to extract an electron from neutral species with ionization energy \( U_i \). For silica at 355 nm, we assume \( U_i = 9 \text{ eV} \) implying \( K = 3 \) photons and \( \sigma_3 = 2.48 \times 10^{-11} \text{ ns}^{-1} \text{ cm}^6/\text{GW}^3 \) is the associated cross section. With an electron recombination time \( \tau_{\text{rec}} = 150 \text{ fs} [46] \), the electron density stays almost constant during typical durations \( T_0 > 1 \text{ ps} \), so that we apply the stationarity assumption

\[
\rho_e(t) \simeq \tau_{\text{rec}}\rho_{nt}\sigma_3 I^3. \tag{7}
\]

Losses to the optical field induced by photo-ionization can be evaluated from Poynting’s theorem leading to

\[
\mu_0 \frac{\partial J_{\text{PI}}}{\partial t} = -ik_0\beta^{(K)}I^{K-1}E, \tag{8}
\]

where \( \beta^{(K)} = \sigma_K K\hbar\omega_0\rho_{nt} \) is the multiphoton absorption (MPA) factor. Finally, terms corresponding to plasma coupling and losses are plugged into Eq. (2) and we separate forward and backward components [33]. Then, the propagation equations under investigation finally read as

\[
(\partial_z + k'\partial_t) U_1 = \frac{i\vec{\nabla}_\perp^2 U_1}{2k_0} + \frac{in_2\omega_0}{c} \left( |U_1|^2 + 2|U_2|^2 \right) U_1
- \frac{g_0}{2} QU_2 - \frac{ik_0\rho_{nt}\sigma_3\tau_{\text{rec}}}{2n_0^2\rho_c} F_1 U_1 - \frac{\beta^{(3)}}{2} G_1 U_1, \tag{9}
\]

\[
(\partial_z + k'\partial_t) U_2 = \frac{i\vec{\nabla}_\perp^2 U_2}{2k_0} + \frac{in_2\omega_0}{c} \left( |U_2|^2 + 2|U_1|^2 \right) U_2
+ \frac{g_0}{2} Q^* U_1 - \frac{ik_0\rho_{nt}\sigma_3\tau_{\text{rec}}}{2n_0^2\rho_c} F_2 U_2 - \frac{\beta^{(3)}}{2} G_2 U_2, \tag{10}
\]

\[
\tau_B \partial_t Q + Q = U_1 U_2^* + N. \tag{11}
\]

Here, \( z \) is the propagation direction, \( k' \simeq k_0/\omega_0 \), \( \vec{\nabla}_\perp^2 = \partial_x^2 + \partial_y^2 \) is the diffraction operator, \( g_0 = 4.5 \text{ cm/GW} \) is the Brillouin gain factor, and \( n_2 = 3.6 \times 10^{-16} \text{ cm}^2/\text{W} \) is the Kerr coefficient at the laser wavelength.
\[ \lambda_0 = 355 \text{ nm} \] [33]. \[ Q \equiv 2 \Gamma_B n_0 c C_S \bar{\rho} / iq \gamma_e \] stands for the rescaled acoustic envelope where \( \tau_B = 2 / \Gamma_B = 2 / q^2 \Gamma' \) is the phonon damping rate related to the Brillouin linewidth \( \Gamma_B \). The functions \( F_i \) and \( G_i \) in Eqs. (9-10) express for \( i, j = 1, 2 \) as

\[
F_i = I_i^3 + 4I_j^3 + 12I_i^2I_j + 18I_iI_j^2, \\
G_i = I_i^2 + 3I_j^2 + 6I_iI_j,
\]

and \( \rho_c \equiv \omega_0^2 m_e \epsilon_0 / q_e^2 = 8.8 \times 10^{21} \text{ cm}^{-3} \) is the critical plasma density. The saturation intensity for which the plasma defocusing starts to balance the Kerr effect for one wave component is \( I \simeq 11 \text{ TW/cm}^2 \), corresponding to the electron density \( \rho_e^{\text{max}} \simeq 10^{20} \text{ cm}^{-3} \). This threshold also applies in the presence of two optical components whenever they reach their maximum intensities at different times and distances.

In Eq. (11), \( N \) represents a thermally driven Gaussian random noise which acts as a seed for SBS. The noise \( N \) has zero mean \( \langle N(\vec{r}, t) \rangle = 0 \) and

\[
\langle N(\vec{r}, t) N^*(\vec{r}', t') \rangle = A_N \delta(\vec{r} - \vec{r}')\delta(t - t'),
\]

where \( A_N \) evaluates as \( 3.7 \times 10^{-29} \text{ GW}^2 \text{ ns/cm} \) for a temperature of \( T = 300 \text{ K} \) and other parameters are given above [21, 47]. Note that the spatial noise is only formally \( \delta \)-correlated in our macroscopic description, it has a microscopic correlation length on atomic scales.

3. One-wave and Two-wave self-focusing

Below we comment on the fully space and time resolved SBS code written in FORTRAN 90 using MPI for parallel execution on CPUs. We also briefly recall the main properties of single-wave self-focusing and scattering of Stokes pulses through the Brillouin effect.

3.1. The SBS CPU code

Equations (9), (10) and (11) are integrated numerically by means of \((z, t)\) progression. Composed of several functions, the main algorithm consists of an operator splitting scheme. In order to understand the program structure, let us first ignore transverse diffraction \((\nabla^2 U_i = 0)\) and the plasma response \((F_i = G_i = 0)\) for a better readability. The \((z, t)\) progression is done using
the method of characteristics: A new variable $s$ enters $z_j(s) = z_j^0 - \frac{s(-1)^j}{k'}$, $t_j(s) = t_j^0 + s$ for $j = 1, 2$, in such a way that Eqs. (9) and (10) reduce to

$$
k' \frac{dU_1(z_1(s), t_1(s))}{ds} = \frac{in_2 \omega_0}{c} \left[ |U_1|^2 + 2|U_2|^2 \right] U_1 - \frac{g_0}{2} QU_2, \tag{13}
$$

$$
k' \frac{dU_2(z_2(s), t_2(s))}{ds} = \frac{in_2 \omega_0}{c} \left[ |U_2|^2 + 2|U_1|^2 \right] U_2 + \frac{g_0}{2} Q^* U_1. \tag{14}
$$

The discretization for $U_1$ is then given by

$$
z_1(s + \Delta s) = z(s) + \Delta z; \quad t_1(s + \Delta s) = t(s) + \Delta t,
$$

and for $U_2$ by

$$
z_2(s + \Delta s) = z(s) - \Delta z; \quad t_2(s + \Delta s) = t(s) + \Delta t.
$$

Thus, when we advance by the time step $\Delta t$, the wave $U_1$ is shifted by $+\Delta z$ and the wave $U_2$ is shifted by $-\Delta z$. An operator splitting scheme is then used for the numerical integration of the complete system. The integration of the right-hand-side terms, namely Kerr effect and coupling to the acoustic wave, is performed separately between two shifting procedures. The longitudinal increment $\Delta z$ is related to the time step as $\Delta t = k' \Delta z \equiv n_0 \Delta z / c$.

The acoustic wave is seeded by the thermal noise $N$. This noise is implemented using two Gaussian distributed variables with zero mean and unit variance, $gauss_1$ and $gauss_2$. These numbers are calculated in each time step and for each mesh point, using the function $gasdev$ and the random number generator $ran1$ (see Ref. [48]). The noise function is numerically implemented through

$$
N = \sqrt{\frac{A_N}{2\Delta V \Delta t}} (gauss_1 + i gauss_2). \tag{15}
$$

Here, $1/\Delta V$ and $1/\Delta t$ are the discrete representations of $\delta(\vec{r} - \vec{r}')\delta(t - t')$ in Eq. (12). Because our (macroscopic) spatial mesh is much coarser than the microscopic SiO$_2$ building blocks of silica, we have to use $\Delta V = m_{\text{mol}} / \rho_0 N_A$, with molar mass $m_{\text{mol}} = 60$ g/mol, density $\rho_0 = 2.2$ g/cm$^3$, and Avogadro number $N_A = 6 \times 10^{23}$ mol$^{-1}$.

For each time step several integrations are performed. The first one solves the acoustic wave equation (11) using an Euler scheme. The optical waves are then treated by incrementing the terms corresponding to the Kerr effect.
Here, we make use of the formal solution to this part of the r.h.s. of Eqs. (13) and (14) \((i, j = 1, 2)\), which guarantees fluence conservation

\[
U_i^{\Delta s}(z, t) = \exp \left[ \frac{im_2\omega_0}{k'c} \int_s^{s+\Delta s} (|U_i|^2 + 2|U_j|^2) \, ds \right] U_i(z, t) \\
\approx \exp \left[ \frac{im_2\omega_0}{c} \left( |U_i|^2 + 2|U_j|^2 \right) \Delta z \right] U_i(z, t).
\]  

(16)

Next, we advance the Brillouin terms. In order to obtain fluence conservation for this split-operator integration as well, we employ the following semi-implicit mid-point scheme:

\[
U_i^{\Delta s}(z, t) \approx U_i^{\Delta s}(z, t) - \frac{g_0}{4} \left[ Q(z + \Delta z, t) U_2^{\Delta s}(z + \Delta z, t) - Q(z, t + \Delta t) U_2^{\Delta s}(z + \Delta z, t) \right] \Delta z \\
U_2^{\Delta s}(z + \Delta z, t) \approx U_2^{\Delta s}(z, t) + \frac{g_0}{4} \left[ Q^*(z, t) U_1^{\Delta s}(z, t) - Q^*(z + \Delta z, t + \Delta t) U_1^{\Delta s}(z, t) \right] \Delta z.
\]  

(17)

(18)

The propagation by \(\Delta t\) is finalized by shifting \(U_1\) and \(U_2\) in the \(z\) dimension by setting

\[
U_1(z + \Delta z, t + \Delta t) = U_1^{\Delta s}(z, t) \\
U_2(z, t + \Delta t) = U_2^{\Delta s}(z + \Delta z, t),
\]  

(19)

(20)

for which we make use of the FORTRAN intrinsic procedure CSHIFT. Boundary conditions in \(z\), i.e., at the beginning \(z = 0\) and the end \(z = L\), where \(L\) is the sample length, can be naturally imposed in this scheme. In this paper, we feed a pump pulse at \(z = 0\) into \(U_1\); \(U_2\) is put to zero at \(z = L\), and the remaining two boundaries are set to be transparent by simply discarding \(U_1^{\Delta s}(L, t)\) and \(U_2^{\Delta s}(0, t)\) after each time step. We carefully validated our implementation of the above scheme against special analytical solutions and already published results [47]. In particular, we verified convergence, i.e., that results do not change when we further refine the mesh (i.e., going from \(\Delta t = 0.5\) ps to \(\Delta t = 0.25\) ps when simulating SBS in a single-mode optical fiber to check against results published in [26]), or when we change the order of the split operators.
The full three-dimensional version of the SBS code takes into account transverse diffraction and plasma generation as well. Diffraction \( \sim \vec{\nabla}^2 \) is evaluated numerically exact in the Fourier domain as a third split operator, and the use of the FFTW3 library \([49]\) renders its numerical evaluation relatively fast. The plasma terms can be treated together with the Kerr terms in a straightforward manner. Periodic boundary conditions in \((x, y)\) are used, as a natural consequence of the Fast Fourier Transforms (FFTs) in the spectral scheme used for the diffraction operator. The parallelization with MPI of this three-dimensional scheme is achieved by simple domain decomposition with respect to the \(z\)-direction. Due to the homogeneity of the problem, load balancing is not an issue. Communication is necessary only between neighboring sub-domains, i.e., one \((x, y)\) slice per sub-domain boundary has to be exchanged between two MPI tasks. No parallelization of the two-dimensional FFTs in the \((x, y)\) is necessary, since all relevant data is available to the respective MPI process. It should be noted that the MPI-OpenMP hybrid implementation will use a slightly different approach (see Sec. 4): we will employ one MPI task per node (or CPU) and use a thread parallelism on the cores (either of the full node or of single associated CPU \(^1\)). Then, by using the threaded version of the FFTW3 library, we will execute the FFT in parallel within the MPI task. Each MPI task writes its own output and stores them into a file gathering all relevant numerical data. A MATLAB script allows us to visualize the physical results. We verified the convergence of the results yielded by the full three-dimensional version of the SBS code for several test cases requiring relatively small meshes. However, when simulating singular wave behaviors for which a sufficient resolution is beyond our computational resources, we observed a certain dependency of the results on the mesh, which will be addressed in Secs. 3.2, 3.3, and 6.

Since the split-operator integration of the Brillouin terms [Eqs. (17) and (18)] consumes more than one-third of the total execution time (both CPU and wall-clock), it will serve as an example throughout the paper for the different code versions. Table 1 shows the reference implementation in FORTRAN 90 used within the CPU code. No inter-process communications (MPI calls) are necessary in this part of the code, only the boundaries of the loop

\(^1\)A modern CPU has at least 4 internal cores which are able to run in parallel. A computer designed for scientific computations has at least 2 CPUs. Depending on the underlying hardware and the operating system, it is more efficient to use many CPUs per MPI task or associate one MPI task per CPU.
in z dimension (outer loop in k) reflect the domain decomposition due to parallelization.

The computation stages of the SBS CPU code are summarized in Fig. 1. Different functions correspond to the implementation of different terms of the model equations. These functions will further be used to compare their individual execution time with their GPU counterparts. ALL considers the whole execution of the time step. NOISE represents the evaluation of the acoustic wave equation and the generation of the Gaussian noise. DIFFRACRT corresponds to the computation of the Fourier transforms, the multiplication by the diffraction table and performing the inverse Fourier transform. KERSBS gathers the calculation of the Brillouin and Kerr terms. SHIFT accounts for the shifting of the $U_1$ and $U_2$ data tables. The implementation of the boundary conditions are represented by the function BOUND and finally, FLUX contains the calculation of the fluence and maximum intensity to be stored in the output files.

In the numerical simulations, the input pump pulse has Gaussian profile

$$U_1(x, y, t, z = 0) = \sqrt{I_1(0)}\exp \left[ -\frac{x^2 + y^2}{w_0^2} - \frac{t^2}{t_p^2} \right], \quad (21)$$

where $I_1(0) \equiv 2P_1(0)/\pi w_0^2$ is the incident pump intensity depending on the pump power $P_1(0)$, $w_0$ is the input beam width, and $t_p$ is the pulse duration measured at $1/e^2$ of the maximum intensity. Its Full-Width at Half-Maximum (FWHM) duration is given by $\tau_{\text{FWHM}} = \sqrt{2 \ln 2} \tau_p$. As already mentioned above, the Stokes wave $U_2$ is seeded by the noise function $N$, and we consider the silica sample as an active SBS-medium with boundary condition $U_2(x, y, t, z = L) = 0$.

A standard technique for reducing Brillouin backscattering is to employ broadband pump pulses with rapid phase fluctuations [30, 28]. This is of great practical relevance, because the backscattered fluence is a source of front damage in the material. Therefore, phase-modulated Gaussian pulses in the form

$$U_1(x, y, t, z = 0) = \sqrt{I_1(0)}\exp \left[ -\frac{x^2 + y^2}{w_0^2} - \frac{t^2}{t_p^2} + im \sin (2\pi \nu_m t) \right], \quad (22)$$

11
SUBROUTINE brillouin(U1, U2, U1_BUFFER, U2_BUFFER, Q_BUFFER, Q, delta_z, g0)

IMPLICIT NONE

USE Parameters

COMPLEX(8), intent(IN) :: U1 (dim_x, dim_y, dim_z_start-1:dim_z_end)
COMPLEX(8), intent(IN) :: U2 (dim_x, dim_y, dim_z_start:dim_z_end+1)
COMPLEX(8), intent(IN) :: Q (dim_x, dim_y, dim_z_start-1:dim_z_end+1)
COMPLEX(8), intent(IN) :: Q_BUFFER (dim_x, dim_y, dim_z_start-1:dim_z_end+1)
COMPLEX(8), intent(INOUT) :: U1_BUFFER (dim_x, dim_y, dim_z_start-1:dim_z_end)
COMPLEX(8), intent(INOUT) :: U2_BUFFER (dim_x, dim_y, dim_z_start:dim_z_end+1)

INTEGER(4) :: i, j, k
REAL(8) :: delta_z, g0, gdz, gdz2

INTRINSIC :: CONJG, MOD, INT

gdz = 0.25D0 * g0 * delta_z

DO k = dim_z_start, dim_z_end
    DO j=1,dim_y
        DO i=1,dim_x
            U1_BUFFER(i,j,k) = (U1(i,j,k) * (1.D0 - gdz2 * Q_BUFFER(i,j,k) * CONJG(Q(i,j,k))) &
                                - gdz * Q(i,j,k+1) * U2(i,j,k+1) - gdz * Q_BUFFER(i,j,k) * U2(i,j,k+1)) &
                                / (1.D0 + gdz2 * Q_BUFFER(i,j,k) * CONJG(Q_BUFFER(i,j,k+1)))
        ENDDO
    ENDDO
ENDDO

END SUBROUTINE

Table 1: Implementation of the Brillouin terms [Eqs. (17) and (18)] in FORTRAN 90. Boundaries of the outer loop in k (dim_z_start, dim_z_end) reflect the domain decomposition in z due to parallelization. The variables correspond as follows: U1_buffer → U_1^Δ_s, U2_buffer → U_2^Δ_s, U1 → U_1^Δ_s/2, U2 → U_2^Δ_s/2, Q_buffer → Q( · , t + Δt), Q → Q( · , t), g0 → g0, and delta_z → Δz.
Figure 1: (Color Online) Scheme of the SBS code. MPI phase 1 and 2 indicate where data exchange due to domain decomposition is necessary.
characterized by a modulation depth \( m \), a modulation frequency \( \nu_m \) and an \( 1/e \) spectral bandwidth \( \Delta \nu \approx 2m\nu_m \) \((m \gg 1)\) will also be used in the coming simulations. Such pulses develop a broad spectrum composed of \( \sim 2m \) longitudinal modes. When these modes are spaced by \( \nu_m > \Delta \nu_B = \Gamma_B/2\pi \) and for a small enough coherence length, each pump mode scatters its own Stokes pulse with no interaction between their neighbors \([22, 50, 51]\). For pump modes of equal initial intensity \( \sim I_1(0)/2m \), the exponential Brillouin gain can be decreased significantly compared to the unmodulated case. If the spectral bandwidth and the number of pump modes are above critical values, the creation of acoustic waves is efficiently inhibited \([33]\). Here, we will choose instead a subcritical bandwidth, using \( m = 21 \) and \( \nu_m = 2 \) GHz, for which the Brillouin coupling is still relevant, even though the Kerr terms may dominate. Phase modulations will act as rapid temporal perturbations breaking the pump and Stokes waves into turbulent distributions composed of sharply compressed structures in space and time.

3.2. One-wave collapse

We first retrieve the classical self-focusing behavior of a single wave using the SBS CPU code without stimulated Brillouin scattering, i.e., \( U_2 = 0 \). When we furthermore discard plasma generation \((F_1 = G_1 = 0)\), the propagation equation for \( U_1 \) reduces to the cubic NLS model

\[
i\partial_z U_1 + \frac{1}{2k_0} \nabla^2_{\perp} U_1 + \frac{n_2\omega_0}{c} |U_1|^2 U_1 = 0,
\]

(23)
after using the change of variables \( z \rightarrow \tilde{z} = z \) and \( t \rightarrow \tilde{t} = t - k'z \) (tilde notation is henceforth omitted).

This equation is known to have several invariants, i.e., the beam power \( P \equiv \int |U_1|^2 d\tilde{r} = P_1 \) and Hamiltonian \( H \equiv \int \left( |\nabla^2_{\perp} U_1|^2/2k_0 - n_2\omega_0 |U_1|^4/2c \right) d\tilde{r} \) with \( d\tilde{r} = dx dy \). The mean square radius \( \langle r^2 \rangle \equiv \int r^2 |U_1|^2 d\tilde{r} / P \) of the beam and these two invariants are linked through the relationship

\[
P d_z^2 \langle r^2 \rangle = \frac{4}{k_0} H.
\]

(24)

Eq. (24) shows that there exist initial conditions \((H < 0 \text{ for example})\), for which \( \langle r^2 \rangle \) tends to zero and the solution to Eq. (23) diverges in amplitude (or "collapses") at a finite distance \( z = z_c < +\infty \) \([7, 5]\). For finite \( L^2 \) norms \( P \), the inequality

\[
P \leq \langle r^2 \rangle \int |\nabla_{\perp} U_1|^2 d\tilde{r}
\]

(25)
implies that the gradient norm \( \int |\nabla_{\perp} U_1|^2 d\vec{r} \) diverges when \( \langle r^2 \rangle \to 0 \) as \( z \to z_c \). In these conditions, the solution \( U_1(z) \) defined in an appropriate Sobolev space diverges in amplitude and does no longer exist beyond \( z_c \). Using Sobolev inequality [6], the Hamiltonian can furthermore be bounded as [52]

\[
H \geq \int \frac{2}{k_0} |\nabla_{\perp} U_1|^2 |d\vec{r}| \left( 1 - \frac{P}{P_{cr}} \right),
\]

meaning that collapse occurs at finite distance only if \( P \) exceeds the critical power for self-focusing

\[
P_{cr} = \frac{3.72 \lambda_0^2}{8 \pi n_0 n_2}.
\]

For an incident power higher than \( P_{cr} \), the medium acts as a converging lens. According to Marburger’s prediction [53], an incident Gaussian wave usually self-focuses as a whole at the approximated distance

\[
z_c \simeq L_M = \frac{0.092 L_{\text{diff}}}{\left[ \left( \sqrt{P/P_{cr}} - 0.852 \right)^2 - 0.0219 \right]^{1/2}},
\]

where \( L_{\text{diff}} = 2 k_0 w_0^2 \) is the diffraction length of the input beam. For very high powers, broad beams subject to, e.g., random noise, can alternatively decay into small multiple cells due to modulational instabilities [16], which grow to form multiple filaments. The beam collapse then appears at a shorter distance given by [54, 55]

\[
z_{c_{\text{MF}}} \approx \frac{\lambda_0}{2 \pi n_2 I_1(0)}.
\]

As an example, we first focus on the propagation of a single wave with short input duration \( t_p = 500 \) fs and small width \( w_0 = 60 \) \( \mu \)m with an incident intensity \( I_1(0) \simeq 619 \) GW/cm\(^2\), corresponding to the power ratio \( P/P_{cr} = 100 \). The results for \( 256 \times 256 \times 4096 \) mesh points in \((x, y, z)\), a spatial mesh \( \Delta x = \Delta y = 0.9 \) \( \mu \)m, \( \Delta z = 1 \) \( \mu \)m, and a temporal step of \( \Delta t = 0.32 \) fs are shown in Fig. 2. For such short pulses, a fine resolution is necessary because the maximum intensity widely exceeds the ionization threshold and diverges in the absence of plasma response [see Fig. 2(a), dotted curves]. The maximum intensity of the pump reaches a level \( \geq 10 \) TW/cm\(^2\), which is close to the saturation threshold from which the plasma starts to defocus the pulse. When only the plasma gain is implemented (\( F_1 \neq 0, G_1 = 0 \)), saturation
takes place around this threshold value with a corresponding plasma density of $\rho_e^{\text{max}} \simeq 10^{20}$ cm$^{-3}$ (dashed curve). When plasma losses are taken into account as well (solid curve), the clamping intensity slightly decreases but the peak density remains close to expected values. Although the latter could be improved by considering a non-instantaneous plasma response, the clamped intensity levels are in the right order of magnitude [31].

The snapshots 2(b-e) represent surface plots of the maximum intensity profiles at $z = 0$ [2(b)], $z \simeq z_c = L_M = 0.188 \simeq 0.19$ cm for a pure (unsaturated) collapse [2(c)], when plasma gain is included [2(d)], and with plasma losses [2(e)]. One can observe that plasma generation slightly delays the self-focusing point by $\sim 0.1$ mm ($z_c \simeq 0.2$ cm), while MPA losses regularize the pulse profile. The measured FWHM duration is $\tau_{\text{FWHM}} \sim 100$ fs for the non-saturated pulse. Plasma coupling keeps a broader overall duration $\sim 300$ fs within an almost top-hat profile cut at $\sim 10$ TW/cm$^2$ by plasma saturation. Addition of MPA losses smooths this profile and we report a pulse length of $\sim 170$ fs FWHM duration.

3.3. Two-waves self-focusing with SBS

Brillouin backscattering takes place for longer (nanosecond) durations of the pump pulse, allowing the excitation of sound waves responsible for the Stokes pulse scattered in backward direction ($U_2 \neq 0$). Classical theories [3, 28] predict an exponential amplification of the Stokes intensity with gain $G_S = L/L_B$, $L_B = 1/g_0 I_1(0)$ (30) over a sample thickness $L$ in the so-called stationary limit, i.e., the time derivative is omitted in Eq. (11) [$\tau_B \partial_t Q \to 0$]. This stationary limit usually applies to long pulse durations, e.g., $\tau_{\text{FWHM}} \gg 10$ ns at 355 nm. In the present context, we shall focus more on FWHM pulse durations of a few nanoseconds, e.g., $\tau_{\text{FWHM}} = 2.5$ ns, for which the stationary limit does not apply. The Brillouin gain is then defined at leading order by [23]

$$G_T \sim 2\sqrt{\Gamma B_t p L/L_B}.$$ (31)

At high enough powers [32], e.g., $P_t(0) > 14P_{cr}$ for a 5-cm long silica sample, collapse of the pump takes place inside the material and triggers a Stokes pulse. Without the defocusing effect of the self-generated plasma, the intensity growth is limited by the numerical mesh, which inherently render the results mesh-dependent. This mesh-dependent behavior is a consequence of the
Figure 2: (Color Online) (a) Maximum intensity for a collapsing Gaussian pulse with $t_p = 500$ fs, $w_0 = 60$ µm and $P = 100 P_{cr}$ for no plasma generation (dotted curve), plasma gain only (dashed curve) and with multiphoton absorption (solid curve). (b-e) Snapshots of the $(x, t)$ intensity profiles, being maximum along $y$. (b) Input pulse at $z = 0$. (c) Collapsing beam at $z = 0.184$ cm $\lesssim z_c \approx 0.19$ cm with no plasma; (d) at $z \approx z_c = 0.2$ cm with plasma gain; (e) with plasma gain and losses.
singular wave collapse occurring in our system, which demands a very high resolution in both space and time to reach intensity values \( \gg 10 \text{ TW/cm}^2 \), as recalled in Sec. 3.2. Typically, less than 1-\( \mu \text{m} \) spatial steps and 1-fs time increments are necessary, which are currently not achievable as our temporal windows exceed 10 ns. However, in the full physical model including plasma generation, this mesh-dependency can partly be tamed for sufficiently high numerical resolution, as collapse is saturated by plasma defocusing. One difficulty is to employ a large enough number of points in \((x, y)\), so that the intensity growth due to Kerr-driven self-focusing does not get artificially saturated by the coarseness of the transverse grid. Another difficulty is to use enough points in \((z, t)\) to accurately describe longitudinal and temporal gradient growths. As an example, Fig. 3(a) shows the evolution of the pump and Stokes maximum intensities with plasma generation for the input pump power \( P_1(0) = 27 \, P_{cr} \), beam width \( w_0 = 150 \, \mu \text{m} \) and 1/e\(^2\) pulse duration \( t_p = 2.12 \, \text{ns} \). A coarse mesh of \( 128 \times 128 \times 4096 \) points cannot properly describe the spatial compression in the plane \((x, y)\): The pump wave barely increases at roughly the self-focusing distance, and maximum intensities attained by the forward and backward pulses remain underestimated (dotted curves). Reversely, by doubling the number of points in \((x, y)\) and halving those in \((z, t)\), both waves reach high intensity levels while the pump pulse diverges at a well-defined self-focusing distance (dashed curves). The reference mesh constituted of \( 256 \times 256 \times 4096 \) points in \((x, y, z)\) \((\Delta x = \Delta y = 3.9 \, \mu \text{m}, \Delta z = 12.2 \, \mu \text{m})\) guarantees that the pump wave undergoes collapse at the same nonlinear focus distance and that both wave intensities reach several \( \text{TW/cm}^2 \) (solid curves). Only the amplification distance of the Stokes pulse still appears to be mesh-dependent. The backscattered component is indeed very sensitive to fluctuations in the highest pump peaks beyond the self-focusing point. The observed mesh-dependent shift of the amplification distance of the Stokes pulse is, however, limited, so that overall propagation dynamics are reproduced at least qualitatively. Therefore, in the following, we shall retain this mesh as providing the minimum resolution necessary for our system. One can observe that the self-focusing distance \( z_c \) is larger than Marburger’s prediction \( L_M \) [e.g., for \( P_1(0) = 27 \, P_{cr} \), \( z_c \approx 4 \, \text{cm} \) while \( L_M = 2.5 \, \text{cm} \)], which is due to the depletion of the pump power by Brillouin scattering. Both pump and Stokes waves are maximum near the instant \( t \approx -1 \, \text{ns} \) and remain singly peaked, as already reported in [33].

Let us now examine the effect of phase modulation in the input pump pulse. The rapid fluctuations of the phase in Eq. (22) create a multimode
Figure 3: (Color Online) Peak intensities for 355-nm (a) unmodulated and (b) phase-modulated input Gaussian pulses. Blue curves correspond to the pump wave and green curves represent the Stokes wave. In (a) a pump pulse with power ratio $P_1(0) = 27P_{cr}$ is simulated with $256 \times 256 \times 4096$ points (solid curves) in the presence of plasma. The dotted (dashed) curve corresponds to the same pulse configuration simulated with $128 \times 128 \times 4096$ points (resp., $256 \times 256 \times 2048$ points). In (b) the dark curves show simulation results for $P_1(0) = 27P_{cr}$ including plasma generation, using $256 \times 256 \times 4096$ points. For the bright curves plasma response is discarded. (c-f) Maximum intensity profiles in the $(x, t)$ plane for the phase-modulated pump (b) at the distance $z = 2.34$ cm: (c) $I_1$, (d) $I_2$ in the absence of plasma generation; (e) $I_1$, (f) $I_2$ with plasma generation. Note that surface plots have been prepared with a reduced graphical resolution of 1.2 ps, which explains why the maximum intensities reported in (b) may not be attained.
spectrum of 1/e bandwidth $\Delta \nu \simeq 2m \nu_m$, where $2m$ is the approximate number of modes and $\nu_m$ is the modulation frequency. Here, we choose a subcritical bandwidth using $m = 21$ and $\nu_m = 2$ GHz, providing an overall bandwidth limited to $\Delta \nu \simeq 84$ GHz [33]. In this configuration, phase modulation does not suppress SBS, as evidenced in Fig. 3(b) where the Stokes wave sharply increases while the pump self-focusing distance is noticeably shortened. With such moderate modulation, the forward component undergoes modulational instabilities and breaks up into multiple peaks, first in time, then in space [Fig. 3(c)]. These modulational instabilities accelerate the filamentation dynamics and shorten the pump collapse distance. When plasma generation is discarded, the Stokes wave increases in a single peak as it counterpropagates toward the input facet of the sample [Figs. 3(d)]. With self-generated plasma present, the pulse intensities saturate at lower level. Moreover, one can observe that the singular amplification of the Stokes pulse starts later [i.e., at smaller $z$ distances for $U_2$, see Fig. 3(b)]. The turbulent pump pattern is smoothed in the sense that the number of diverging filaments is reduced, i.e., the defocusing action of the plasma favors the emergence of low-intensity satellite filaments [Figs. 3(e,f)].

Compared with Fig. 2 (see also Ref. [31]), the clamping intensities observed in the post-collapse stage do not reach, however, the appropriate saturation values $> 10$ TW/cm$^2$ for accurately describing the ionization stage. This is due to the coarse resolution accessible with the SBS CPU code when only limited computational resources are available, e.g., 4 days on 128 processors. One way to get better resolved results in reasonable times is to resort to GPU computations, which we will do in the next Section.

4. The SBS GPU code

Two versions of the SBS code re-written to make use of the computational power of graphics processors are detailed below: one using the CUDA architecture, the other relying on HMPP programming directives.

GPU programming requires to understand the architecture of a graphics processor. A GPU aggregates several multiprocessors, each containing several single processing units which concurrently work with the same code on different parts of a common data set. To map the user’s problem on a GPU, we have to exploit its two-level grid. The first level is a grid composed of several blocks, each of them being monitored by a multiprocessor. The second level is the block of threads handled by a single multiprocessor.
A thread is a process executed on one elementary unit of a multiprocessor, treating a single data element. All threads of one block are executed simultaneously without the possibility of taking branches. GPUs are thus Single-Instruction-Multiple-Thread (SIMT) parallel devices, where algorithms with fine-grained parallelism distribute calculations over thousands of simultaneous threads, each sharing instructions but operating on different data. Their performance relies on the organization of the GPU memory. Each multiprocessor is equipped with several thousands of registers, having fast and local access from each processing unit. It has access to a small amount of shared memory with very low latency, which serves for synchronization and communication between the threads in a GPU block. The global memory situated on separate chips is large (up to 4 GB on a NVIDIA S1070 [56]) and can be accessed with long latency by each thread on each multiprocessor. Read access are faster using constant and texture memory since they use a cache. Host memory of the CPU unit cannot be accessed from the graphics processor. In the hardware layout, each multiprocessor executes a block of threads concurrently, while the different blocks of a grid are assigned to separate multiprocessors. The objective of GPU calculations is to reach a high arithmetic intensity defined as the ratio of the number of floating point operations to the memory access (reads and writes) with no interaction between threads. This ratio must be of at least 2 for an efficient use of a GPU. In the case of our SBS code, this high arithmetic intensity combined with the high memory bandwidth will be exploited to handle larger meshes in reasonable computation times.

4.1. The CUDA Version

In a first approach, we adapted the SBS CPU code to CUDA version 3.2 providing FFTs with double precision complex numbers (CUFFT library). However, our implementation is also completely compatible with the newer CUDA version 4.0. Basically, CUDA is an extension of the C/C++ languages, which allows to implement sets of so-called kernels, each defined as a function executed in parallel on the GPU grid divided in thread blocks of dimension $\text{blockDim.}\{x, y, z\}$. Threads are characterized by two sets of coordinates: the position of the thread in its block ($\text{threadIdx.}\{x, y, z\}$) and that of its block inside the grid ($\text{blockIdx.}\{x, y, z\}$). The global position of the treated element is then a simple function of these two informations, which, e.g., in a 1D block is identified by the global index ($\text{threadIdx.x} + \text{blockIdx.x} \times \text{blockDim.x}$). Analogous functions can be performed in 2D and 3D.
by taking the respective grid and block dimensions into account. Here, the maximum number of threads per block is fixed by the hardware (1024 for the latest NVIDIA processors). A grid may embed up to $65535 \times 65535$ blocks ($\times 65535$ in CUDA 4.0). In the case of our SBS code, we first implemented FORTRAN 90/C interfaces for each function in the temporal loop (functions DIFFRACT, SHIFT...) in order to be able to use CUDA. Then, we rewrote the functions NOISE, DIFFRACT, KERSBS, etc. as kernels so they can be executed on the GPUs in parallel. A GPU grid must be defined before invoking a kernel. Since in the above functions all fine-grained computation units can be executed concurrently, the number of threads is directly determined by the numerical mesh. Data transfers between CPU and GPU being particularly expensive in execution time, the use of graphic processors is really efficient if the whole temporal loop is executed on the GPUs exclusively. Therefore, it is crucial that all functions inside the main loop are available as kernels. However, some data transfers remain necessary due to domain decomposition when more than one GPU card is used, but they are limited to one $(x, y)$ slice per sub-domain boundary, as in the CPU case with MPI parallelization. Moreover, writing the output files implies data transfers from the GPU to the CPU. Hence, it is important to choose a reasonable number of outputs in the temporal and longitudinal dimensions in production mode. In Tables 2 and 3 we give an example of the CUDA adaptation of the Brillouin loop discussed in the previous Section (see Table 1). To call such a routine from the main FORTRAN code, we have to use the INTERFACE feature provided by the FORTRAN 90 standard (Table 2a). By doing so, we make sure that both languages (CUDA and FORTRAN) can cooperate gracefully. The INTERFACE points to a C subprogram (Table 2b) which initializes and calls the actual CUDA code as shown in Table 3. From this example, we can infer that the implementation in CUDA considerably increases the code size. Our implementation of SBS in FORTRAN 90 has about 900 lines whereas the CUDA version consists of more than 5500 lines.

In order to make the most out of the available hardware, we also improved the CPU parallelization by means of Open Multi-Processing (OpenMP) allowing for total hybrid computations, i.e., CPU with MPI+OpenMP and
INTERFACE
SUBROUTINE brillouin_c(U1, U2, U1_BUFFER, &
U2_BUFFER, Q_BUFFER, Q, delta_z, g0)
USE Parameters
IMPLICIT NONE
COMPLEX(8), intent(IN) :: U1 (dim_x, &
dim_y, dim_z_start-1:dim_z_end)
COMPLEX(8), intent(IN) :: U2 (dim_x, &
dim_y, dim_z_start:dim_z_end+1)
COMPLEX(8), intent(IN) :: Q (dim_x, &
dim_y, dim_z_start-1:dim_z_end+1)
COMPLEX(8), intent(INOUT) :: U1_buffer (dim_x, &
dim_y, dim_z_start-1:dim_z_end)
COMPLEX(8), intent(INOUT) :: U2_buffer (dim_x, &
dim_y, dim_z_start:dim_z_end+1)
REAL(8), intent(IN) :: delta_z, g0
END SUBROUTINE brillouin_c
END INTERFACE

extern "C" void
brillouin_c(c8t * restrict p_u1, c8t * restrict p_u2,
c8t * restrict p_u1b, c8t * restrict p_u2b,
c8t * restrict p_q, c8t * restrict p qb, 
double * restrict delta_z, double * restrict g0)
{
  double gdz = (double) 0.25 * g0 * delta_z;
double gdz2 = gdz * gdz;
dim3 block, grid;
SetBlockDims(ga.dim_xy * ga.dim_z, 64, &block, &grid);
br_cuda <<< grid, block >>> (gdz, gdz2, ga.dim_xy, ga.dim_z,
(cplx8_t *) ga.p_u1_dev, (cplx8_t *) ga.p_u2_dev,
(cplx8_t *) ga.p_q_dev, (cplx8_t *) ga.p_qb_dev,
(cplx8_t *) ga.p_u1b_dev, (cplx8_t *) ga.p_u2b_dev);
CheckErr("after launch of kernel br_cuda");
cudaThreadSynchronize();
CheckErr("after cudaThreadSynchronize of kernel br_cuda");
}

Table 2: FORTRAN 90 INTERFACE (a) and C subprogram (b) necessary to call the Brillouin kernel br_cuda shown in Table 3. Specific CUDA directives appear in red.
Table 3: Implementation of the Brillouin terms [Eqs. (17) and (18)] as a kernel to be executed in the CUDA architecture.
MPI+GPU with CUDA. If the number of available GPUs is greater than the number of requested processes, computation is performed on GPUs in the MPI+CUDA mode. In such a case we allocate one MPI task per GPU. Conversely, when there is not enough available GPU units, part of the calculation is done on GPUs with MPI+CUDA and the other part is executed on the available CPUs in MPI+OpenMP mode. This option renders the SBS CUDA code operational in full machine on the most performing modern computers. We have observed that, for equidistant domain decomposition, using GPUs as computer engines introduces a large imbalance at execution time in favor of the GPU, the OpenMP processes lagging a lot behind. This fully hybrid version is thus useful only if the numerical mesh size of the simulation is the constraining parameter and not the execution speed. Otherwise, using the pure GPU version remains the option giving the highest performance. So far, we have not implemented a load balancing strategy between CPUs and GPUs. However, efficient static load balancing should be easily achievable by non-equidistant domain decomposition.

The only algorithmic difference in the implementation of the CUDA code occurs in the function NOISE. The computation of the noise amplitude involves a random number generator which produces pseudo-random numbers between 0 and 1 with the largest possible period. Besides random() supplied by Unix, many algorithms have been developed in the past to produce high quality random numbers. The CPU code of SBS uses the ran1() function [48]. While of good quality, this generator has, however, the constraint of maintaining an array of 32 internal values which prohibits its use in the massively parallel context of a GPU. To overcome this limitation, we instead chose to implement the Lavaux and Jenssens algorithm

$$x_{n+1} = (31167285 \times x_n + 1) \mod 2^{48}$$

which has a period of $2^{48}$ and needs to maintain only one integer value [57].

4.2. The HMPP Version

In a second approach to GPU acceleration, we developed a GPU version of the SBS code using HMPP directives. HMPP (for ”Hybrid Multicore Parallel Programming”, www.caps-entreprise.com/hmpp.html) is an hybrid programming tool that does not imply a complete rewriting of the code, contrary to CUDA. Here, the code (FORTRAN or C) has to be annotated with

---

2To maintain the properties of ran1() we should have introduced either locks or atomic operations, killing thereby the parallelism.
specific compiler directives, allowing the execution of specific functions, so-called codelets, on the graphics processors. A codelet enables us to declare a function which will be remotely executed. Its execution can be synchronous or asynchronous and specifications concerning data transfers can be added. Once defined, the codelets are executed by the directive callsite. Here, the calculation grid is not defined by the user but directly by HMPP. An example is given in Table 4 for the same Brillouin loop previously CUDA-adapted in Table 3.

The comparison of the two code sections shows that using HMPP directives allows us to leave the original code almost unchanged (only 198 lines of HMPP directives were added to the CPU code), whereas the adaptation to CUDA implies substantial code transformations. It is left completely up to the HMPP compiler to produce an efficient GPU code. The programmer has to focus only on data movements and make sure that the code is written in a way that the compiler can easily spot potential parallelism.

5. Algorithmic Performance and Benchmarking

In this Section, we confront performances of the CPU code with those of the GPU codes, first the CUDA version and second the HMPP version. File input/output (I/O) is omitted in the performance measurements. For the strong and weak scaling presented in Secs. 5.1 and 5.2, speedup tests have been performed on CPU/GPU nodes including two Intel Xeon X5570 processors equipped with 4 cores each and two Tesla processors of a S1070 NVIDIA graphics card, i.e., 8 CPUs (cores) and 2 GPUs per node. This cluster was also used for the simulations presented in Sec. 6. For complementary performance tests in Sec. 5.3 we employed Intel Xeon X5660 processors and NVIDIA M2050 graphics cards. Time evaluation was performed on each separate function presented in Fig. 1. Throughout this paper, computation times are given as wall clock time, i.e., in the absence of I/O, the elapsed time of one processor from start to end of the job, which includes active working periods and communication channel delays.

5.1. Strong scaling

We consider a test configuration with a constant mesh size, i.e., 128 × 128 × 256 points in \((x, y, z)\) and 14523 temporal steps, and increase the number of CPU or GPU units. The computation time (wall clock) is given in
SUBROUTINE brillouin(U1, U2, U1_BUFFER, U2_BUFFER, Q_BUFFER, Q, delta_z, g0)

IMPLICIT NONE

USE Parameters

COMPLEX(8), intent(IN) :: U1 (dim_x, dim_y, dim_z_start-1:dim_z_end)
COMPLEX(8), intent(IN) :: U2 (dim_x, dim_y, dim_z_start:dim_z_end+1)
COMPLEX(8), intent(IN) :: Q (dim_x, dim_y, dim_z_start-1:dim_z_end+1)
COMPLEX(8), intent(IN) :: Q_BUFFER (dim_x, dim_y, dim_z_start-1:dim_z_end+1)
COMPLEX(8), intent(INOUT) :: U1_BUFFER(dim_x, dim_y, dim_z_start-1:dim_z_end+1)
COMPLEX(8), intent(INOUT) :: U2_BUFFER(dim_x, dim_y, dim_z_start:dim_z_end+1)

INTEGER(4) :: i, j, k, l
REAL(8) :: delta_z, g0, gdz, gdz2

INTRINSIC :: CONJG, MOD, INT

gdz = 0.25D0 * g0 * delta_z

!$omp parallel do shared(u1_buffer, u2_buffer) private(i,j,k,l) collapse(2)
DO k = dim_z_start, dim_z_end
   !$ompcg Parallel
   DO l = 1, dim_x * dim_y
      i = MOD((l-1), dim_x) + 1
      j = INT((l-1)/ dim_x) + 1
      U1_buffer(i,j,k) = (U1(i,j,k) * (1.D0 - gdz2 * Q_buffer(i,j,k) * CONJG(Q_buffer(i,j,k))) &
      - gdz * Q(i,j,k+1) * U2(i,j,k+1) - gdz * Q_buffer(i,j,k) * U2(i,j,k+1)) / (1.D0 + gdz2 * Q_buffer(i,j,k) * CONJG(Q_buffer(i,j,k+1)))
      U2_buffer(i,j,k) = (U2(i,j,k) * (1.D0 - gdz2 * CONJG(Q_buffer(i,j,k)) * Q(i,j,k)) &
      + gdz * CONJG(Q_buffer(i,j,k+l)) * Q(i,j,k+1) + gdz * CONJG(Q_buffer(i,j,k)) * U1(i,j,k+1) &
      / (1.D0 + gdz2 * CONJG(Q_buffer(i,j,k+l)) * Q_buffer(i,j,k+1)))
   ENDDO
ENDDO

END SUBROUTINE brillouin

... then the codelet is called with the directives callsite...

CALL brillouin(U1,U2,U1_BUFFER,U2_BUFFER,Q_BUFFER,Q,delta_z,g0)

Table 4: Implementation of the Brillouin terms [Eqs. (17) and (18)] using HMPP directives (red). OpenMP directives appear in green.
Fig. 4. For up to 4 computation units we get a speedup factor larger than 8 (compared to the CPU code) over an entire run including all functions and associated transfers of data for both the CUDA and the HMPP code. Using for example only 2 GPUs supplies higher performance than 16 CPUs. The speedup factor, however, decreases to \( \sim 5 \) and \( \sim 3 \), respectively, when employing 32 processors, which signals starvation of the GPU resources. We attribute this saturation effect to the increasing fraction of execution time spent on data transfer between the GPUs as a consequence of the domain decomposition. Compared to CPU computations that avoid transfer between GPUs and CPUs, the data transfer time from GPU\(_X\) to GPU\(_Y\) is increased by a factor \( \sim 3 \) (GPU\(_X\) \rightarrow CPU\(_X\) \rightarrow CPU\(_Y\) \rightarrow GPU\(_Y\)). A domain boundary has to be written back from the sending GPU\(_X\) to the main memory, sent over the network using MPI and then transferred into the memory of the receiving GPU\(_Y\). Thus, the weight of the time spent on inter-process communication noticeably increases. Note that due to hand-made optimization in the CUDA code, speedup factors are slightly higher with the CUDA version than with the HMPP code.

5.2. Weak scaling

In the "weak scaling" configuration, we consider a numerical mesh brick of \( 128 \times 128 \times 64 \) points and 14523 temporal steps handled by one processing unit (CPU or GPU). The goal here is to check the code performance for increasing number of processing units when the amount of data handled by one unit remains constant. For our SBS code, for a fixed numerical domain in \((z, t)\), doubling the number of mesh points in \(z\) implies twice the number of points in time, because increments are linked by the relation \( \Delta z = \Delta t/k' \). Thus, doubling both the \(z\) mesh size and the number of processors should ideally double the execution time (wall clock). This is indeed confirmed by Fig. 5, which shows the computation time as a function of the number of processing units as we increase the number of points in the \(z\) dimension by the same factor. Deviations from the ideal curve can be again attributed to data transfers. Despite these shortcomings, the speedup factors remain excellent: Between \( \sim 12 \) and \( \sim 9 \) for the CUDA code and from \( \sim 9.5 \) down to \( \sim 7 \) for the HMPP code over 1 to 32 processing units. The speedup factors obtained with 32 processing units are better than in strong scaling configuration, which we attribute to a better optimization of the GPU memory for larger amounts of data. Again the CUDA code yields slightly better acceleration gains due to a more efficient hand-optimization of the nonlinearities.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>23318</td>
<td>2296</td>
<td>10.16</td>
<td>2611</td>
<td>8.93</td>
</tr>
<tr>
<td>2</td>
<td>12005</td>
<td>1208</td>
<td>9.94</td>
<td>1421</td>
<td>8.45</td>
</tr>
<tr>
<td>4</td>
<td>6784</td>
<td>655</td>
<td>10.36</td>
<td>845</td>
<td>8.03</td>
</tr>
<tr>
<td>8</td>
<td>3453</td>
<td>379</td>
<td>9.11</td>
<td>536</td>
<td>6.46</td>
</tr>
<tr>
<td>16</td>
<td>1778</td>
<td>240</td>
<td>7.41</td>
<td>362</td>
<td>4.91</td>
</tr>
<tr>
<td>32</td>
<td>931</td>
<td>168</td>
<td>5.54</td>
<td>288</td>
<td>3.23</td>
</tr>
</tbody>
</table>

Figure 4: (Color Online) "Strong scaling": Computation time (wall clock) on CPUs (red curve), on GPUs using the CUDA code (green curve) and the HMPP code (blue curve), as a function of the number of processing units (Intel Xeon X5570 and NVIDIA S1070).
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1768</td>
<td>144</td>
<td>12.27</td>
<td>187</td>
<td>9.45</td>
</tr>
<tr>
<td>2</td>
<td>3310</td>
<td>319</td>
<td>10.38</td>
<td>408</td>
<td>8.11</td>
</tr>
<tr>
<td>4</td>
<td>7096</td>
<td>654</td>
<td>10.85</td>
<td>841</td>
<td>8.41</td>
</tr>
<tr>
<td>8</td>
<td>13960</td>
<td>1639</td>
<td>8.52</td>
<td>2023</td>
<td>6.90</td>
</tr>
<tr>
<td>16</td>
<td>27687</td>
<td>3009</td>
<td>9.20</td>
<td>3836</td>
<td>7.22</td>
</tr>
<tr>
<td>32</td>
<td>55158</td>
<td>5722</td>
<td>9.64</td>
<td>7593</td>
<td>7.26</td>
</tr>
</tbody>
</table>

Figure 5: (Color Online) "Weak scaling": Computation time (wall clock) on CPUs (red curve), on GPUs using the CUDA code (green curve) and the HMPP code (blue curve), as a function of the number of processing units (Intel Xeon X5570 and NVIDIA S1070). The black dotted curve shows the ideal computation time on GPUs if no data transfer was involved (straight line with slope of 2).
Besides investigating strong and weak scalings, we also determined the maximum mesh size which could be handled by a single GPU. On a NVIDIA S1070 with 4GB of RAM, we can solve at most $256 \times 256 \times 512$ points. For comparison, on the NVIDIA M2050 (3GB with ECC on) used in Sec. 5.3 we could only handle $256 \times 256 \times 256$ points. For this mesh on the S1070, we report a relative acceleration factor of about 1.1 compared to a mesh of $128 \times 128 \times 256$ points. This factor is defined by the quotient of the increased fraction of the number of points (here 4) to the ratio between the wall clock times measured for each mesh (here 3.6). It falls down to unity when comparing smaller meshes, e.g., $128 \times 128 \times 256$ and $128 \times 128 \times 64$ points. Thus, we confirm that better performances are attained close to saturation of the GPU memory.

5.3. Overall comparisons

We finally compare the performance of the CPU and both GPU (CUDA and HMPP) versions of the major functions of the SBS code. Results are shown in Table 5. It displays computation times and GPU speedup gains for our previous test case (strong scaling) with a mesh of $128 \times 128 \times 256$ points and 14523 temporal steps on one processing unit (CPU or GPU). In contrast to the previous analysis for which we needed up to 32 CPUs and GPUs, we here used a smaller machine with newer hardware, namely Intel Xeon X5660 processors and NVIDIA M2050 graphics cards. For the M2050, the GPU performance is increased by a factor larger than two compared to the S1070, a direct consequence of the much higher double precision performance. As confirmed by Table 5, both CUDA and HMPP codes perform almost equal with overall speedup $\gtrsim 19$. The differences observed on the S1070 vanish, and the overall gains are doubled compared with Figs. 4 and 5. However, the performance of individual functions varies considerably between HMPP and CUDA code, and we report acceleration factors $\gtrsim 45$ compared to the CPU code in sequential mode for individual functions. In particular, the HMPP code yields an improved speedup gain for the function DIFFRACT. In this context, we emphasize that we took particular care of treating the Kerr and Brillouin terms [Eqs. (16)-(18)]: KERSBS is hand-optimized in the CUDA implementation, because the nonlinearities correspond to the most time-consuming functions in the CPU code. As a result, our CUDA implementation of KERSBS is more efficient, while the HMPP compiler stays on a safe path, discarding potential low-level optimization. We also hand-optimized the function FLUX of the CUDA implementation. This function is
<table>
<thead>
<tr>
<th>Function</th>
<th>CPU Time</th>
<th>CUDA Time</th>
<th>Gain</th>
<th>HMPP Time</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>19197 s</td>
<td>1004 s</td>
<td>19.1</td>
<td>987 s</td>
<td>19.4</td>
</tr>
<tr>
<td>NOISE</td>
<td>2796 s</td>
<td>99.0 s</td>
<td>28.2</td>
<td>176 s</td>
<td>15.9</td>
</tr>
<tr>
<td>DIFFRACT</td>
<td>4781 s</td>
<td>583 s</td>
<td>8.20</td>
<td>425 s</td>
<td>11.2</td>
</tr>
<tr>
<td>KERSBS</td>
<td>8535 s</td>
<td>190 s</td>
<td>44.7</td>
<td>262 s</td>
<td>32.6</td>
</tr>
<tr>
<td>SHIFT</td>
<td>508 s</td>
<td>93.6 s</td>
<td>5.43</td>
<td>49.8 s</td>
<td>10.2</td>
</tr>
<tr>
<td>BOUND</td>
<td>41.0 s</td>
<td>1.11 s</td>
<td>36.9</td>
<td>44.0 s</td>
<td>0.93</td>
</tr>
<tr>
<td>FLUX</td>
<td>2536 s</td>
<td>37.2 s</td>
<td>68.2</td>
<td>29.3 s</td>
<td>86.6</td>
</tr>
</tbody>
</table>

Table 5: Runtime comparisons of the CUDA and HMPP versions of the GPU code with the CPU code for different functions of the SBS code (Intel Xeon X5660 and NVIDIA M2050). Tests were performed for a mesh of $128 \times 128 \times 256$ points on one processing unit (CPU or GPU) omitting any output. ALL: Total program; NOISE: Random noise function; DIFFRACT: Diffraction evaluation including FFTs; KERSBS: Computation of Kerr and Brillouin terms; SHIFT: Table shift; BOUND: Temporal boundary conditions; FLUX: Calculation of pump/Stokes energy flux.

Actually a diagnostic routine, computing the energy flux of pump and Stokes wave through $(x, y)$ surfaces, which is necessary to get the partial energies (see also Fig. 7). Because the computation of the fluxes involves all data in the arrays $U_1$ and $U_2$, it is advantageous to do these calculations in the main simulation code in parallel.

In summary, we found that, at least for our problem, the maturity of the HMPP compiler is such that excellent code performance can easily be reached by just using an appropriate set of directives. Furthermore, HMPP allows the use of other devices such as GPUs produced by AMD by changing just a single option in one directive (target=CUDA becomes target=OPENCL). Compared to CUDA, which doubtless is the more flexible and tunable coding alternative, HMPP offers simplicity and portability easing the usage of GPUs in computational science.

6. Improvement of the physical results

Exploiting 128 GPUs makes it possible to improve the resolution using $512 \times 512 \times 8192$ points in $(x, y, z)$ for the same numerical box in similar computation time, i.e., $\sim 4$ days. The resolution is then refined with
Δx = Δy = 1.9 μm, Δz = 6.1 μm and Δt = 30 fs, which directly impacts the accuracy of the physical results in the examples shown below. GPU simulations were performed by means of the CUDA code.

6.1. One-wave collapse

Let us first study a single Gaussian pulse (U_2 = 0) undergoing only self-focusing with incident power \( P_1(0) = 27 \, P_{cr} \), for which the collapse distance predicted by Marburger [Eq. (28)] is \( L_M = 2.5 \, \text{cm} \). We compare a calculation on 128 CPUs with a mesh of \( 256 \times 256 \times 4096 \) points with two calculations employing 128 GPUs: The first one with \( 256 \times 256 \times 4096 \) points and the second one with \( 512 \times 512 \times 8192 \) points. The results are shown in Fig. 6. For the two cases with lower resolution, results of CPU and GPU codes are identical, which thereby validates the latter code in the single-wave collapse regime. For the highly resolved case, the accessible level of intensity is clearly superior, i.e., \( I_{1 \, \text{max}} \approx 13 \, \text{TW/cm}^2 \) to be compared with the 3-5 TW/cm\(^2\) obtained on the coarser mesh. In other words, on the finer mesh, the numerical scheme breaks down later. Nevertheless, the self-focusing distances stay almost the same for both meshes. The inclusion of plasma defocusing noticeably decreases the saturation intensity. With \( I_{1 \, \text{max}} = 7 \, \text{TW/cm}^2 \), the maximum electron density \( \rho_e^{\text{max}} \approx 2.8 \times 10^{19} \, \text{cm}^{-3} \) remains slightly weaker than expected values from fully time resolved plasma models [31]. We anticipate that an improved model omitting the stationary assumption would promote higher clamping intensities \( \geq 15 - 20 \, \text{TW/cm}^2 \) for peak densities closer to \( 10^{20} \, \text{cm}^{-3} \), provided that the numerical resolution is high enough.

6.2. Two-waves SBS coupling

We now focus on the results obtained with the GPU code describing the interaction between the Kerr self-focusing and Brillouin backscattering. First, we validate the full SBS GPU code in Fig. 7 for non-collapsing beams with five critical powers, using a moderate mesh of \( 128 \times 128 \times 512 \) points. Except very minor differences due to a different noise realization, all global/local dynamical aspects are identical, including the difference of partial energies \( \mathcal{E}_i \equiv \int I_i \, dx \, dy \, dt \), which is a conserved quantity of our SBS system.

Next, we simulate a phase-modulated laser pulse for which both pump and Stokes intensities reach large values while they develop turbulent dynamics with high peaks being short in time and narrow (focused) in space. Figure 8 shows numerical results for an incident phase-modulated pulse with \( P_1(0) = 27 \, P_{cr} \) and the same input modulation as in Fig. 3(b). Our former CPU
Figure 6: (Color Online) Maximum intensities for a Gaussian pulse with $P_1(0) = 27 P_{cr}$ propagating in 5 cm of silica. Dashed light blue curve (superimposed with dashed dark blue curve): CPU calculation with $256 \times 256 \times 4096$ points. Dashed dark blue curve: GPU calculation with $256 \times 256 \times 4096$ points. Light red curve: GPU calculation with $512 \times 512 \times 8192$ points without plasma contributions. Dark red curve: GPU calculation with $512 \times 512 \times 8192$ points with plasma generation.

calculation using a mesh of $256 \times 256 \times 4096$ points, recalled by blue and green curves, is compared to a GPU calculation (red and orange curves) using $512 \times 512 \times 8192$ points. The resulting behaviors are analogous in terms of the pump self-focusing distance, but the CPU-computed intensities artificially saturate at lower levels [Fig. 8(a)]. Inspection of the numerical intensity profiles reveals that, in contrast to Figs. 3(c,d), only a single peak is amplified in the pump pulse [Figs. 8(b,c)]. Thus, we again observe a mesh-dependent behavior in the SBS system when we omit plasma defocusing, as already stated in Sec. 3.3.

The influence of plasma contributions is shown in Fig. 9(b). A decrease in the pump and Stokes saturation intensities is observed, with $I_{1,\text{max}} \approx 8$ TW/cm$^2$ and $I_{2,\text{max}} \approx 5$ TW/cm$^2$. By feedback, the plasma action with $\rho_{e,\text{max}} \propto \tau_{\text{rec}} \rho_{n0} \sigma_3 I_{\text{MAX}}^3 > 10^{19}$ cm$^{-3}$ lowers the clamping intensities significantly. As a consequence, the singular growth of the single pump peak observed before is inhibited and formation of multiple filaments is favored. The detailed intensity profiles $I_1(z \approx z_c)$ and $I_2(z = 0)$ confirm this statement and show turbulent patterns compatible with those shown in Fig. 3(e,f).
Figure 7: (Color Online) (a),(d) Maximum intensities of the pump (blue) and Stokes (green) waves for an incident Gaussian wave of $5 P_{cr}$ propagating in 5 cm of silica. (b),(e) Corresponding partial energies. (c),(f) Intensity profiles of the Stokes wave at the entrance of the sample. (a,b,c) show a CPU computation whereas (d,e,f) result from the same computation on GPUs. Marginal differences visible in the intensity profiles (c), (f) are due to different realizations of the noise $N$ in the acoustic wave equation.
Figure 8: (Color Online) (a) Maximum intensities of the pump (blue and red curves) and Stokes wave (green and orange curves) for the same Gaussian phase-modulated pulse simulated in Fig. 3(b). (a) Light blue and green curves: CPU calculation using $256 \times 256 \times 4096$ points without plasma terms. Red and orange curves: GPU calculation using $512 \times 512 \times 8192$ points without plasma terms. (b) Pump and (c) Stokes maximum intensity profiles both at $z = 2.34$ cm in the plane $(x, t)$. Surface plots have been prepared with a reduced graphical resolution of 9 ps, which explains why the maximum intensities reported in (a) may not be attained.
rect confrontation with previous low-resolution CPU results (Fig. 3) shows that also in the simulations including plasma response we observe a mesh dependency of our results, even though discrepancies are smaller than in the plasmaless case (Fig. 8). A finer resolution with spatial and time steps close to 1 µm and 1 fs, thereby yielding much larger mesh sizes beyond our computational capacities, should, however, overcome this problem.

7. Conclusion

In summary, we have adapted an MPI-parallelized numerical code relying on a spectral split-operator scheme to graphics processors, employing either the CUDA or HMPP architecture. Because of the significant improvement of the code’s computing performance in terms of runtime, the GPU accelerated version allows us to run simulations using 16 times larger numerical meshes than in the CPU code over comparable execution times. Our results demonstrate that the promising GPU technology is now mature for treating efficiently large scale numerical problems in computational physics, i.e., singular wave dynamics in nonlinear optics. In the present context, our GPU accelerated code enables us to solve our physical model for single as well as two nonlinearly-coupled laser field envelopes with a finer numerical resolution. As a consequence, we are able to describe wave collapse dynamics up to optical intensities close to the threshold of ionization in transparent solids, such as silica. Thus, we can provide a better description of the complex dynamics involving plasma generation for long and intense nanosecond pulses causing optical damage by self-focusing and stimulated scattering processes. Further improvements should concern the implementation of a nonstationary plasma response and refinement of the resolution to avoid any residual saturation effects linked to the coarseness of the numerical mesh.

Acknowledgments

This work was granted access to the HPC resources of CCRT (Centre de Calcul Recherche et Technologie) under the allocation 2011-x2011106003 made by GENCI (Grand Equipement National de Calcul Intensif). Part of the CUDA and HMPP ports where done on a cluster made available through the OpenGPU project, funded by the System@tic cluster of competitiveness. CAPS society is acknowledged for its help in implementing the FFTs in the HMPP code.
Figure 9: (Color Online) (a) Maximum intensities of the pump (blue and red curves) and Stokes wave (green and orange curves) for the same Gaussian phase-modulated pulse simulated in Figs. 3(b) and 8, including the plasma response. (a) Light blue and green curves: CPU calculation using $256 \times 256 \times 4096$ points. Red and orange curves: GPU calculation using $512 \times 512 \times 8192$ points. (b) Pump and (c) Stokes maximum intensity profiles both at $z = 2.34$ cm in the plane $(x, t)$. Surface plots have been prepared with a reduced graphical resolution of 9 ps, which explains why the maximum intensities reported in (a) may not be attained.
References


