Computational Mechanics (2018) 62:1311–1326
https://doi.org/10.1007/s00466-018-1565-x

ORIGINAL PAPER

An efficient spectral crystal plasticity solver for GPU architectures

Michael Malahe[1]

Received: 13 October 2017 / Accepted: 22 February 2018 / Published online: 5 March 2018
© Springer-Verlag GmbH Germany, part of Springer Nature 2018

Abstract

We present a spectral crystal plasticity (CP) solver for graphics processing unit (GPU) architectures that achieves a tenfold increase in efficiency over prior GPU solvers. The approach makes use of a database containing a spectral decomposition of CP simulations performed using a conventional iterative solver over a parameter space of crystal orientations and applied velocity gradients. The key improvements in efficiency come from reducing global memory transactions, exposing more instruction-level parallelism, reducing integer instructions and performing fast range reductions on trigonometric arguments. The scheme also makes more efficient use of memory than prior work, allowing for larger problems to be solved on a single GPU. We illustrate these improvements with a simulation of 390 million crystal grains on a consumer-grade GPU, which executes at a rate of 2.72 s per strain step.

Keywords Crystal plasticity · Texture · GPU · Spectral methods · High-performance computing

Mathematics Subject Classification 74C10 · 65T50 · 65Y05 · 65Y10

1 Introduction

Predicting the macroscopic behavior of metals under deformation is important for the development of new designs and manufacturing processes [16]. Accurate modeling of this behaviour depends on being able to capture the evolution of the microstructure of the metal to correctly capture anisotropies in its plastic properties. In polycrystalline metals, crystal plasticity (CP) formulations have been successful in predicting the evolution of the microstructure [3], and have been integrated into finite element models (CPFE) [13], but are relatively computationally expensive when compared with phenomenological models due to the need to iteratively solve very stiff algebraic systems to determine the microscopic single crystal responses [8].

These microscopic responses can be homogenized into a macroscopic response using a number of methods, with varying degrees of computational complexity, such as the Taylor model [24], self-consistent models [14,18] and the LAMEL model [27]. The Taylor model is one of the simplest, in which it is assumed that all grains in the aggregate undergo the same deformation, and is significantly less computationally expensive than the higher-order models. Due to this, it has found widespread use in crystal plasticity frameworks. However, it has some limitations, such producing sharper textures than those predicted by a more accurate homogenization-based finite element method [23]. Despite the Taylor model's efficiency, it still relies on being able to determine a large number of crystal responses, on the order of hundreds per representative volume element (RVE) [23], which remains a challenge for large-scale simulations.

One approach to improving the computational feasibility of the crystal plasticity model is to use pre-computed crystal response databases that use an iterative solver to capture the mapping from an imposed strain and current crystal orientation to the resulting lattice spin, stress and total slip rate for a single crystal. For a large input parameter space, these databases may become too large to be practical, and work has been done to compress them using first a generalized spherical harmonic (GSH) basis [9,15] and then a more efficient Fourier basis using the discrete Fourier transform (DFT) [12]. This approach is referred to as spectral crystal plasticity (SCP), and it was found that retaining only very few of the largest Fourier coefficients, on the order of a thousand, was sufficient for successfully reproducing the behaviour of interest [10]. This lead to an order of magnitude increase

✉ Michael Malahe
michael.malahe@uct.ac.za

[1] Centre for Research in Computational and Applied Mechanics, University of Cape Town, 5th floor Menzies Building, Private Bag X3, Rondebosch 7701, South Africa

in efficiency over a standard iterative CP solver [17], leading to SCP solvers being integrated into finite element solvers [11,28,29] and a viscoplastic fast Fourier transform (VPFFT) full-field solver [6].

Further developments in this direction recognized the inherent parallelizability of the SCP approach, and a shared memory implementation achieved an additional order of magnitude improvement in speed, followed by a graphics processing unit (GPU) implementation that produced a third order of magnitude speedup [17,22]. The primary workload in these cases is in evaluating the inverse DFT, and the good performance of the GPU implementation was attributed to casting the inverse DFT as a matrix–matrix multiplication and using a divide-and-conquer style algorithm to reduce the computational complexity [17,22].

The current work takes these developments a step further, by introducing a number of improvements in computational efficiency and reductions in the memory requirements for the GPU implementation. We find that the most significant improvement in arithmetic efficiency comes from performing fast range reductions on trigonometric arguments in evaluating the inverse DFT. The most significant improvement in terms of both memory transaction efficiency and in reducing memory requirements comes in evaluating terms in the inverse DFT directly, rather than as the matrix–matrix multiplication proposed in prior work. We find that the combined improvements lead to a tenfold increase in computational efficiency, and a reduction in memory requirements by a factor of roughly one thousand. This allows for the largest SCP simulations presented to date to be efficiently handled on a single consumer-grade GPU.

In Sect. 2 we present the formulation of both the base crystal plasticity model and the spectral database approach. In Sect. 3 we present the important details of our implementation. In Sect. 4 we present and discuss results confirming agreement with the base iterative solver and highlighting the effects of the various optimizations. Finally, in Sect. 5 we present a summary of our findings, and suggest directions for future work.

2 Formulation

2.1 Crystal plasticity model

We use a version of the crystal plasticity model developed by Peirce, Asaro and Needleman [3,20] in the context of face-centered cubic (FCC) polycrystals. Consider a closed reference domain $\Omega \subset \mathbb{R}^3$ under a motion $\mathbf{x} = \boldsymbol{\chi}(\mathbf{X}, t)$, where $\mathbf{X}$ is a material point of Ω, mapped by $\boldsymbol{\chi}$ to a spatial point $\mathbf{x}$. The deformation gradient at a time t is then given by the material gradient of the deformation $\boldsymbol{\chi}(\cdot, t)$ at time t:

$$\mathbf{F} = \nabla \boldsymbol{\chi}. \tag{1}$$

The domain is comprised of disjoint microscopic crystal grains Ω_i, such that

$$\Omega = \bigcup_i \Omega_i, \tag{2}$$

where it is assumed that all of the grains have equal volume. Each macroscopic continuum point $\mathbf{X}$ is associated with the grains in some finite region containing $\mathbf{X}$, such that $\mathbf{X} \in \Omega_{\mathbf{X}} \subset \Omega$. It is assumed that the deformation gradient experienced by the microscopic grains in $\Omega_{\mathbf{X}}$ is identical to the macroscopic deformation gradient at $\mathbf{X}$. We now consider the response of each grain within $\Omega_{\mathbf{X}}$ to this deformation gradient $\mathbf{F}(\mathbf{X})$.

The deformation gradient is multiplicatively decomposed into an elastic part $\mathbf{F}^*$, and a plastic part $\mathbf{F}^p$, through

$$\mathbf{F}^* \equiv \mathbf{F}\mathbf{F}^{p^{-1}}. \tag{3}$$

The second Piola–Kirchhoff stress $\boldsymbol{\sigma}^*$ in each grain is related to the elastic strain $\mathbf{E}^*$ in each grain by

$$\boldsymbol{\sigma}^* = \mathcal{L}[\mathbf{E}^*], \tag{4}$$

where $\mathcal{L}$ is a fourth order elasticity tensor. The Cauchy stress $\boldsymbol{\sigma}$ is implicitly given by

$$\boldsymbol{\sigma}^* = \mathbf{F}^{*^{-1}} (\det(\mathbf{F}^*)\boldsymbol{\sigma}) \mathbf{F}^{*^{-T}}. \tag{5}$$

The elastic strain is assumed to be very small, given by

$$\mathbf{E}^* = \frac{1}{2}\left(\mathbf{F}^{*^T}\mathbf{F}^* - \mathbf{1}\right), \tag{6}$$

where $\mathbf{1}$ is the second-rank identity tensor. The elasticity tensor accounts for the cubic symmetry of the FCC single crystal, and is given by [5]

$$\mathcal{L}_{ijkl} \equiv C_{12}\delta_{ij}\delta_{kl} + C_{44}\left(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}\right) + \bar{C}\sum_{r=1}^{3}\delta_{ir}\delta_{jr}\delta_{kr}\delta_{lr}, \tag{7}$$

$$\bar{C} = C_{11} - C_{12} - 2C_{44}, \tag{8}$$

where C_{11}, C_{12}, and C_{44} are constants. The velocity gradient is given by

$$\mathbf{L} = \dot{\mathbf{F}} \cdot (\mathbf{F})^{-1}, \tag{9}$$

which can be expanded as

$$\mathbf{L} = \mathbf{L}^* + \mathbf{F}^* \cdot \mathbf{L}^p \cdot (\mathbf{F}^*)^{-1}, \tag{10}$$

where the elastic and plastic components respectively have been defined by

$$\mathbf{L}^* := \dot{\mathbf{F}}^* \cdot (\mathbf{F}^*)^{-1}, \tag{11}$$

$$\mathbf{L}^p := \dot{\mathbf{F}}^p \cdot (\mathbf{F}^p)^{-1}. \tag{12}$$

The plastic velocity is expressed as a linear combination of Schmid tensors given by [21]

$$\mathbf{L}^p = \sum_{\alpha=1}^{12} \nu^\alpha \mathbf{S}_0^\alpha, \tag{13}$$

$$\mathbf{S}_0^\alpha := \mathbf{m}_0^\alpha \otimes \mathbf{n}_0^\alpha, \tag{14}$$

where α labels a particular slip system, in which $\mathbf{S}_0^\alpha$ is the Schmid tensor for that system defined by $\mathbf{m}_0^\alpha$ and $\mathbf{n}_0^\alpha$, which are the time-independent slip direction and slip plane normal respectively in the intermediate lattice space, and ν^α is the slip rate on the system. The slip rate is modeled with a rate-dependent power law relation [3]:

$$\nu^\alpha = \nu_0 \left| \frac{\tau^\alpha}{s^\alpha} \right|^{1/m} \operatorname{sgn}(\tau^\alpha), \tag{15}$$

where ν_0 is a reference slip rate, m is the rate sensitivity of slip, τ^α is a resolved shear stress, and s^α is the slip system deformation resistance. The resolved shear stress is approximated by [8]

$$\tau^\alpha \approx \boldsymbol{\sigma}^* : \mathbf{S}_0^\alpha, \tag{16}$$

and the slip system deformation resistance is taken to evolve by a Voce-type saturation hardening law [25]:

$$\dot{s}^\alpha = h_0 \left(1 - \frac{s^\alpha}{s_s}\right)^a G, \tag{17}$$

$$G := \sum_{\beta=1}^{12} |\nu^\beta|, \tag{18}$$

where h_0, a and s_s are slip hardening parameters that are taken to be identical for all of the slip systems. In the case where all slip systems start with the same initial deformation resistance, we simply denote this common deformation resistance by s. The texture evolution is dictated by the lattice rotation tensor $\mathbf{R}^*$, which evolves as

$$\dot{\mathbf{R}}^* = \mathbf{W}^* \mathbf{R}^*, \tag{19}$$

where $\mathbf{W}^*$ is the lattice spin tensor given by

$$\mathbf{W}^* = \mathbf{W} - \mathbf{W}^p, \tag{20}$$

where $\mathbf{W}$ is the applied spin tensor given by

$$\mathbf{W} = \frac{1}{2}\left(\mathbf{L} - \mathbf{L}^T\right), \tag{21}$$

and $\mathbf{W}^p$ is the plastic spin tensor given by

$$\mathbf{W}^p = \frac{1}{2} \sum_{\alpha=1}^{12} \nu^\alpha \left(\mathbf{m}^\alpha \otimes \mathbf{n}^\alpha - \mathbf{n}^\alpha \otimes \mathbf{m}^\alpha\right). \tag{22}$$

The spatial slip directions and normals at a given time are given by

$$\mathbf{m}^\alpha = \mathbf{F}^* \mathbf{m}_0^\alpha, \tag{23}$$

$$\mathbf{n}^\alpha = \mathbf{F}^{*^{-T}} \mathbf{n}_0^\alpha. \tag{24}$$

2.2 Homogenization

Homogenization within the polycrystal is done using a Taylor-type model [3]. In each macroscopic volume element, there are N crystals, with a Cauchy stress of $\boldsymbol{\sigma}^{[k]}$ in the kth crystal. The macroscopic stress response is taken as the volume average of the crystal responses, which simply becomes the average since we have assumed the grains have equal volume:

$$\bar{\boldsymbol{\sigma}} = \frac{1}{N} \sum_{k=1}^{N} \boldsymbol{\sigma}^{[k]}. \tag{25}$$

2.3 Spectral database formulation

Following the approach of Knezevic et al. [10], we proceed to generate a database of single crystal responses. Each entry in the database corresponds to a simulation involving a single crystal with an orientation $\mathbf{g} = (\phi_1, \Phi, \phi_2)$, where ϕ_1, Φ, ϕ_2 are the Bunge-Euler angles, each in the range $[0, 2\pi)$. The crystal is simulated under a constant applied stretching tensor $\mathbf{D} = \frac{1}{2}\left(\mathbf{L} + \mathbf{L}^T\right)$ until it has reached a true strain equal to some pre-chosen fixed true strain increment $\Delta\varepsilon$. The database entry then associates each input, a combination of crystal orientation and applied stretching tensor, with an output, the state of the crystal after stepping to the given strain increment.

Assuming that elastic strains are negligible, the applied stretching tensor reduces to $\mathbf{D} \approx \mathbf{D}^p = \frac{1}{2}\left(\mathbf{L}^p + \mathbf{L}^{pT}\right)$. Making the standard assumption for metals that the plastic deformation is isochoric, we also have that $\mathbf{D}$ is traceless.

The outputs representing the final state of the crystal are chosen as the minimal set required to update the slip system deformation resistance, stress and texture for any given increment, which amounts to 9 total scalars. The sum of effective shearing rate magnitudes, G, is required to evaluate

the deformation resistance using Eq. 17. The 5 independent components of the deviatoric and symmetric Cauchy stress following a single strain step, $\boldsymbol{\sigma}'$, are combined with the deformation resistance to evaluate the Cauchy stress at a given time. Finally, the 3 independent components of the anti-symmetric plastic spin, $\mathbf{W}^p$, are required to evaluate Eq. 19 to update the texture.

The inputs are parametrized by four scalars. The crystal orientations are parametrized directly by the three Bunge-Euler angles. We parametrize the strain rate tensor $\mathbf{D}$ by first scaling it and rotating it into its principal frame:

$$\dot{\varepsilon} = \|\mathbf{D}\|_F, \tag{26}$$

$$\mathbf{D}_0 = \mathbf{D}/\dot{\varepsilon}, \tag{27}$$

$$\boldsymbol{\Lambda}_D = \mathbf{Q}^T \mathbf{D}_0 \mathbf{Q}. \tag{28}$$

The diagonal matrix $\boldsymbol{\Lambda}_D$ containing the eigenvalues of $\mathbf{D}_0$ now has unit norm and zero trace, and can therefore have its entries parametrized by a single angular variable $\theta \in [0, 2\pi)$ as [26]

$$\begin{aligned}\lambda_1 &= \sqrt{\frac{2}{3}}\cos(\theta - \pi/3),\\ \lambda_2 &= \sqrt{\frac{2}{3}}\cos(\theta + \pi/3),\\ \lambda_3 &= -\sqrt{\frac{2}{3}}\cos(\theta).\end{aligned} \tag{29}$$

With these four input scalars, each in the range $[0, 2\pi)$, we construct a uniform 4-dimensional grid of evenly-spaced inputs with N_g grid points per dimension. For each of the inputs, the crystal plasticity model is integrated over the fixed strain increment and the values $\boldsymbol{\sigma}'/(s|\dot{\varepsilon}|^m)$, $\mathbf{W}^p/\dot{\varepsilon}$ and $G/\dot{\varepsilon}$ are stored in the database. This large raw database is then compressed by taking the 4-D DFT of each of the output variables and retaining only the largest few Fourier coefficients:

$$\boldsymbol{\sigma}'_{\mathbf{j}}/(s|\dot{\varepsilon}|^m) = \frac{1}{N_g^4}\sum_{\mathbf{k}} \hat{\boldsymbol{\sigma}'}_{\mathbf{k}} e^{2\pi i \mathbf{j}\cdot\mathbf{k}/N_g}, \tag{30}$$

$$\mathbf{W}^p_{\mathbf{j}}/\dot{\varepsilon} = \frac{1}{N_g^4}\sum_{\mathbf{k}} \hat{\mathbf{W}}^p_{\mathbf{k}} e^{2\pi i \mathbf{j}\cdot\mathbf{k}/N_g}, \tag{31}$$

$$G_{\mathbf{j}}/\dot{\varepsilon} = \frac{1}{N_g^4}\sum_{\mathbf{k}} \hat{G}_{\mathbf{k}} e^{2\pi i \mathbf{j}\cdot\mathbf{k}/N_g}, \tag{32}$$

where $\mathbf{j}$ is the spatial index on the raw grid, $\mathbf{k}$ is the spectral index, and hats denote Fourier coefficients. Note that the raw grid is *not* uniform over orientation space, and therefore is not the most efficient possible sampling of the orientation space. The uniformity over the Bunge-Euler angles is chosen to allow for an efficient DFT and inverse DFT.

For the purposes of ordering the Fourier coefficients only, the 9 output scalars in the database are scaled to have mean 0 and variance 1, in order to have them vary on comparable scales and be dimensionless. For a given Fourier index, the Fourier coefficients for these scaled values are then combined into a vector of length 9, and the magnitudes of these vectors for each index are used to set a descending order for the Fourier coefficients. This order is used for storing the Fourier coefficients corresponding to the *unscaled* data, and those corresponding to the scaled data are not stored.

The spectral database is constructed from two sweeps of the raw database. In the first sweep, the full grid is read in for each component in turn, and used to determine the coefficient magnitudes and subsequently the ordering indices for the coefficients. In the second sweep, the grid is read in the same way, but this time the largest Fourier coefficients are written to disk for each component in turn. These sweeps done to keep the memory footprint minimal.

In this work, $\min(N_g^4, 2^{16})$ coefficients are retained in this spectral database, chosen as an upper bound on the number of coefficients we expect to use in a practical computation. Typically the number of Fourier terms used in a computation, N_t, is of order one thousand.

It is now possible to fetch a single crystal response from this spectral database. Given the imposed strain rate tensor $\mathbf{D}$ we determine $\dot{\varepsilon}$ and θ from Eqs. 26–29. Given the current crystal orientation in the principal frame of $\mathbf{D}$, we determine ϕ_1, Φ and ϕ_2 directly. We then determine the nearest point on the raw grid by

$$\mathbf{j} = \text{nint}\left(\frac{N_g}{2\pi}(\phi_1, \Phi, \phi_2, \theta)\right), \tag{33}$$

where `nint` returns the vector of nearest integers. We then retrieve G using Eq. 32, and use it to update s using Eq. 17. Finally, we update the remaining quantities using Eqs. 30 and 31.

We also use the spectral interpolation scheme suggested in [10], in which the spectral database is effectively replaced by the best approximation of a spectral database with a greater number of grid points than were originally computed. This scales N_g up to $N_g N_r$, where we refer to N_r as the *refinement multiplier*.

3 Methods

3.1 Evaluation of CP model

Time integration of the base CP model for the purposes of constructing the database is done using the implicit time-stepping scheme and iterative solver described by Kalidindi et al. [8]. We also use some of the properties of annealed

oxygen-free high thermal conductivity (OFHC) copper presented in that work for our material parameters: $h_0 = 180$ MPa, $\nu_0 = 0.001\text{s}^{-1}$, $s_s = 148$ MPa, $a = 2.25$, and $m = 0.012$. For the elastic anisotropy, we use the values determined by Alers et al. [1]: $C_{11} = 168.7$ GPa, $C_{12} = 121.7$ GPa, and $C_{44} = 75.0$ GPa. We use an initial timestep of $\Delta t = 0.001$ s, a strain rate of $\dot{\varepsilon} = 0.001\ \text{s}^{-1}$ and a target strain increment of $\Delta\varepsilon = 0.02$, leading to a final time of $t_{\text{final}} = \Delta\epsilon/\dot{\varepsilon} = 20$ s. The initial conditions are $\boldsymbol{\sigma}_0 = \mathbf{0}$ MPa, $s_0 = 16$ MPa and $\mathbf{F}_0^p = \mathbf{1}$. The initial orientation is an "unrotated" one with all of the crystal plane normals aligned with the corresponding lab axes.

3.2 Spectral symmetries

We use the symmetry in the DFT of real data to avoid computing slightly fewer than half of the terms in the series, as suggested in [10]. The inverse DFT of a quantity Y on a domain with sides composed of an equal number of discrete points L is given by

$$Y_{\mathbf{j}} = \sum_{\mathbf{k}} \hat{Y}_{\mathbf{k}} e^{2\pi i \mathbf{j}\cdot\mathbf{k}/L}. \tag{34}$$

For real data, the coefficients $\hat{Y}_{\mathbf{k}}$ have the symmetry $\hat{Y}_{\mathbf{k}} = \overline{\hat{Y}_{L-\mathbf{k}}}$. Hence we can account for two terms at once by recognizing that

$$\begin{aligned}
&\hat{Y}_{\mathbf{k}} e^{2\pi i \mathbf{j}\cdot\mathbf{k}/L} + \hat{Y}_{L-\mathbf{k}} e^{2\pi i \mathbf{j}\cdot(L-\mathbf{k})/L} \\
&= \hat{Y}_{\mathbf{k}} e^{2\pi i \mathbf{j}\cdot\mathbf{k}/L} + \overline{\hat{Y}_{\mathbf{k}}} e^{-2\pi i \mathbf{j}\cdot\mathbf{k}/L} \\
&= \hat{Y}_{\mathbf{k}} e^{2\pi i \mathbf{j}\cdot\mathbf{k}/L} + \overline{\hat{Y}_{\mathbf{k}} e^{2\pi i \mathbf{j}\cdot\mathbf{k}/L}} \\
&= 2\,\text{Re}\left(\hat{Y}_{\mathbf{k}} e^{2\pi i \mathbf{j}\cdot\mathbf{k}/L}\right).
\end{aligned}$$

3.3 Memory optimizations

In this and the following sections we make reference to a number of specific GPU hardware concepts to motivate and demonstrate certain optimizations. For an introduction to these concepts, we refer the reader to [4] for an overview. On the software side, we specifically target Nvidia's Compute Unified Device Architecture (CUDA) application programming interface (API), which is covered in [19].

In prior work, the inverse Fourier transform is implemented as a matrix–matrix multiplication to allow for the use of divide-and-conquer algorithms, such as the Strassen algorithm, in order to reduce the total number of arithmetic operations [13,17,22]. The approach is given by

$$\begin{pmatrix} \boldsymbol{\sigma}'_{\mathbf{j}}/(s|\dot{\varepsilon}|^m) \\ \mathbf{W}^p_{\mathbf{j}}/\dot{\varepsilon} \\ G_{\mathbf{j}}/\dot{\varepsilon} \end{pmatrix} = \begin{pmatrix} \hat{\boldsymbol{\sigma}}'_{\mathbf{k}} \\ \hat{\mathbf{W}}^p_{\mathbf{k}} \\ \hat{G}_{\mathbf{k}} \end{pmatrix}_{9\times N_t} \cdot \begin{pmatrix} e^{\frac{2\pi i(\mathbf{j}_1\cdot\mathbf{k}_1)}{L}} & \cdots & e^{\frac{2\pi i(\mathbf{j}_{N_c}\cdot\mathbf{k}_1)}{L}} \\ \vdots & \ddots & \vdots \\ e^{\frac{2\pi i(\mathbf{j}_1\cdot\mathbf{k}_{N_t})}{L}} & \cdots & e^{\frac{2\pi i(\mathbf{j}_{N_c}\cdot\mathbf{k}_{N_t})}{L}} \end{pmatrix}_{N_t\times N_c}, \tag{35}$$

where N_t is the number of Fourier terms used, and N_c is the number of crystals. In both storing and loading the intermediate matrix of complex exponentials however, significant pressure is placed on the global memory system, causing the solver to be bound by memory transactions. This is a much greater cost than any arithmetic savings from the use of efficient matrix–matrix multiplication schemes, and additionally places a severe limit on the maximum problem size that can be handled by the solver due to there being an additional N_t complex numbers worth of memory required for each additional crystal. We avoid this with a matrix-free approach in which we simply evaluate the Fourier series directly on a per-crystal basis.

In order to further reduce global memory transactions, we read as many Fourier terms into shared memory as possible. For a general D-dimensional database, each Fourier term is represented in memory as a structure containing D integers for the coordinate $\mathbf{k}$ and 9 floating point values for the coefficients $\hat{\mathbf{W}}^p_{\mathbf{k}}$, $\hat{\boldsymbol{\sigma}}'_{\mathbf{k}}$, and $\hat{G}_{\mathbf{k}}$. In our single-precision implementation, a shared memory array of these terms is then read in from a global memory array as if it were an array consisting only of elements of some 32-bit type. In our implementation we chose integers for this type, but any 32-bit type would have the same effect. Each thread j then reads the integers at the array locations i that satisfy $i \bmod B = j$, where B is the block size, which ensures global memory coalescing and avoids shared memory bank conflicts. This can also be achieved with a 64-bit type, for example in a double-precision implementation, provided the device supports 8-byte bank mode and has it activated.

While this makes the scheme significantly faster than one in which each thread reads each term from global memory, it puts a large load on the shared memory system. Depending on the device, it's possible for the solver still to be bound by memory transactions, even with this scheme. We found this to be the case for the GeForce GTX 980 Ti used in this work, and circumvented it by using two crystal grains per thread. The effect for each thread is that the amount of shared memory loading per crystal is halved. In our case this was sufficient to prevent the shared memory system from being saturated, and led to the solver being arithmetic-bound. On other hardware this may not be necessary at all, or more crystals per thread may be required. The drawback of having multiple crystal grains per thread is that the number of registers per thread increases, which reduces the maximum number of resident warps per multiprocessor (i.e. occupancy), potentially exposing instruction latency. However, since the arithmetic for each

Table 1 Demonstration of fast modulo using bitwise AND for $\mathbf{j}\cdot\mathbf{k} = 107$ and $L = 32$

Quantity	Value	Binary representation							
$\mathbf{j}\cdot\mathbf{k}$	107	0	1	1	0	1	0	1	1
$L-1$	31	0	0	0	1	1	1	1	1
$(\mathbf{j}\cdot\mathbf{k})\&(L-1)$	11	0	0	0	0	1	0	1	1

crystal is independent, the device is still able to achieve very high instruction-level parallelism (ILP), which sufficiently hides the instruction latency even though the occupancy is low. The effectiveness of using instruction-level parallelism to reduce latency is borne out for example in [7].

3.4 Arithmetic optimizations

With the solver now being arithmetic-bound, we seek out additional arithmetic optimizations. The most significant target is the trigonometric functions involved in calculating each $e^{2\pi i(\mathbf{j}\cdot\mathbf{k})/L}$ term. There is the potential for the argument $2\pi\mathbf{j}\cdot\mathbf{k}/L$ to be very large, at most $2\pi DL$, requiring either hardware or software to perform a range reduction when evaluating the trigonometric functions. We circumvent this by performing the range reduction explicitly before passing the argument into the trigonometric function. Since both $\mathbf{j}\cdot\mathbf{k}$ and L are integers, using the periodicity of the exponential with a pure imaginary argument, we note that

$$e^{2\pi i(\mathbf{j}\cdot\mathbf{k}/L)} = e^{2\pi i(((\mathbf{j}\cdot\mathbf{k} \bmod L)+nL)/L)} \tag{36}$$

$$= e^{2\pi i((\mathbf{j}\cdot\mathbf{k} \bmod L)/L)}e^{2\pi i n} \tag{37}$$

$$= e^{2\pi i((\mathbf{j}\cdot\mathbf{k} \bmod L)/L)}, \tag{38}$$

where n is an integer, and therefore can replace $\mathbf{j}\cdot\mathbf{k}/L$ with the potentially smaller $(\mathbf{j}\cdot\mathbf{k} \bmod L)/L$.

While this speeds up the computation significantly for large arguments, the integer division required in the modulo operation may still be a sizable expense depending on the architecture. For the compute capability 5.2 hardware used in this work, this was the case, so we used the fact that if L is a power of two, then

$$(\mathbf{j}\cdot\mathbf{k}) \bmod L = (\mathbf{j}\cdot\mathbf{k})\&(L-1), \tag{39}$$

where & is a bitwise AND operation. Table 1 demonstrates the logic for this optimization for $\mathbf{j}\cdot\mathbf{k} = 107$ and $L = 32$. This allows for higher throughput bitwise operations rather than a relatively expensive integer division.

3.5 Overview

We now provide an overview of the two phases of the method, the *offline* phase in which the spectral database is created, and the *online* phase in which it is used, along with their respective parameters.

For the offline phase, all of the material parameters are required, and any database created will only be usable for a material described by the same parameters. The method parameters required at this stage pertain to how the iterative CP model is evaluated (Sect. 3.1), and how the database is constructed (Sect. 2.3). For the CP model evaluation, there is the initial slip resistance $s_0 = 16$ MPa, an initial timestep $\Delta t = 0.001$ s, a strain rate $\dot{\varepsilon} = 0.001\ \mathrm{s}^{-1}$, and a target total strain increment $\Delta\varepsilon = 0.02$. These final two parameters dictate the final time for each single crystal simulation going into the database, which is $t_{\mathrm{final}} = \Delta\epsilon/\dot{\varepsilon} = 20$ s.

For the construction the raw database, we must choose the number of grid points in each direction N_g, after which we perform an iterative CP computation for each point on the grid, and for each point we store the values of $\boldsymbol{\sigma}'/(s|\dot{\varepsilon}|^m)$, $\mathbf{W}^p/\dot{\varepsilon}$ and $G/\dot{\varepsilon}$ at the final time to disk.

For the construction of the spectral database from the raw database, the only method parameter is the limit on the number of Fourier terms to store for each component, which in our case we set to be $\min(N_g^4, 2^{16})$. After the spectral database is constructed it is stored to disk.

Now for the online phase, the only material parameters required are those for updating the slip resistance using Eq. 17: h_0, a and s_s. The problem parameters provided are the imposed velocity gradient $\mathbf{L}(t)$, the number of crystals N_c, their initial orientation distribution, and an initial slip resistance s_0. The method parameters provided are the strain increment per step $\Delta\varepsilon$, the number of Fourier terms N_t, and the refinement multiplier N_r. We additionally choose a spectral database to use, which is implicitly a choice of the number of grid points in the raw database it was generated from, N_g.

We first generate the crystals with a given orientation distribution and set their initial slip resistances to $s = s_0$. At each timestep, we then determine $\dot{\varepsilon}$ and θ from $\mathbf{D} = \frac{1}{2}\left(\mathbf{L}(t) + \mathbf{L}(t)^T\right)$ using Eqs. 26–29, and additionally determine $\mathbf{W} = \frac{1}{2}\left(\mathbf{L}(t) - \mathbf{L}(t)^T\right)$. Then from our desired strain increment $\Delta\varepsilon$, we determine the corresponding timestep $\Delta t = \Delta\varepsilon/\dot{\varepsilon}$. Now for each crystal, we determine its current orientation in the principal frame of $\mathbf{D}$ to retrieve ϕ_1, Φ and ϕ_2 for the database lookup. We then determine the nearest point $\mathbf{j}$ on the raw grid using Eq. 33. This is used to retrieve G using Eq. 32, which is used to update the slip resistance s using Eq. 17. The slip resistance is required to correctly scale $\boldsymbol{\sigma}$ and $\mathbf{W}^p$ when they are fetched using Eqs. 30 and 31 respectively. These quantities are in the principal frame of $\mathbf{D}$ after being fetched, so are transformed out of that frame into the lab frame. Finally for the texture, the lattice spin tensor $\mathbf{W}^*$ is updated using Eq. 20, which is then used to update the lattice rotation tensor $\mathbf{R}^*$ using Eq. 19. The rotation tensor is decomposed into its Bunge–Euler angle representation,

which is then stored for the next timestep along with s as the only two per-crystal quantities that are stored between timesteps.

4 Results

4.1 Simple shear and plane strain compression

We examine two test cases, and compare them with the literature as verification of the scheme. The first is a simple shear test, given by

$$\mathbf{L} = \begin{pmatrix} 0 & \dot{\varepsilon} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \tag{40}$$

with $\dot{\varepsilon} = 1.0$, and the second is a plane strain compression test, given by

$$\mathbf{L} = \begin{pmatrix} \dot{\varepsilon} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -\dot{\varepsilon} \end{pmatrix}, \tag{41}$$

with $\dot{\varepsilon} = 1.0\ \text{s}^{-1}$. In both cases the configuration is composed of 65,536 randomly-oriented crystal grains forming a single Taylor polycrystal. The random orientations are drawn in such a way that their corresponding rotations are uniformly distributed within SO(3), using the method in [2]. The simulation is run from $t = 0$ to $t = 1000$ s. For the iterative solver, we use an initial timestep of $\Delta t = 0.001$ s. For the spectral solver, we use a strain increment of 0.02, corresponding with a timestep of $\Delta t = 0.02$ s. The spectral database is constructed from an $N_g = 128$ raw database, and is further refined by a factor of $N_r = 128$. The number of Fourier terms used is either $N_t = 1024$ or $N_t = 8192$, chosen to correspond directly with the results in the literature.

We compare stress-strain curves for the two solvers for the simple shear case in Fig. 1, and we compare textures at the final timestep in Fig. 2. The same comparisons are made for the plane strain compression case in Figs. 3 and 4. The stress-strain curves are also compared with those from the same tests conducted in the work by Mihaila et al. [17]. Their case differs from our case slightly, by using an isotropic elasticity tensor, a rate sensitivity of slip of $m = 0.01$ instead of $m = 0.012$, and a database dimension of $N_g = 120$ instead of dimension $N_g = 128$. Additionally they use the spectral refinement method, but the refinement multiplier value they use is not reported. The curves show good agreement generally, although the spectral solver with 8192 Fourier terms produces a more accurate result for the simple shear case in the Mihaila et al. work, and produces a more accurate result for the plane strain compression case in our work.

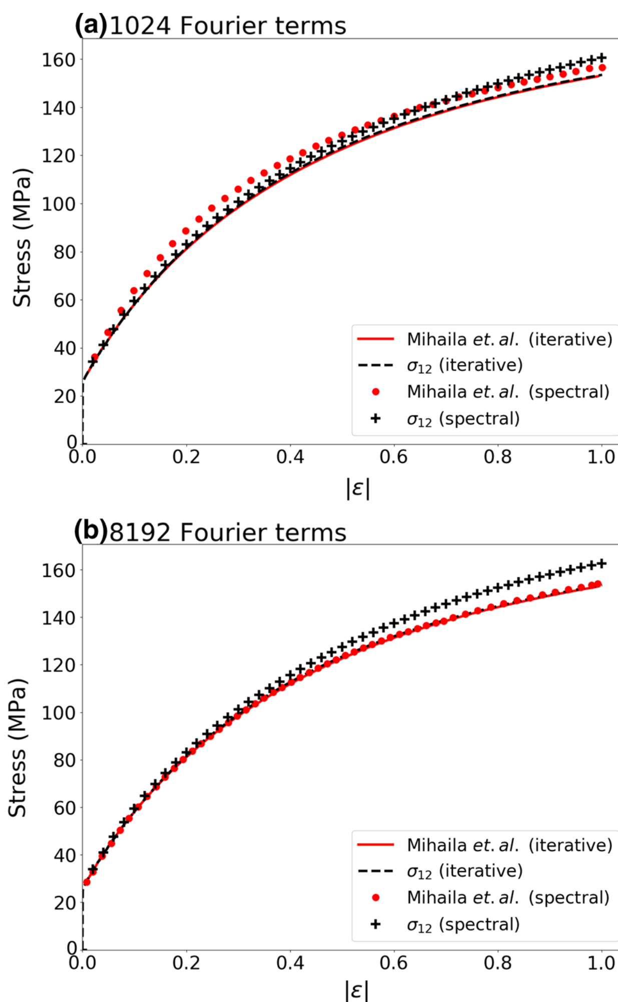

Fig. 1 Comparison of true stress strain curves for the iterative and spectral schemes for the simple shear case with the results in [17]. **a** Cases with 1024 Fourier terms. **b** Cases with 8192 Fourier terms

4.2 Accuracy scaling

In this section we measure the scaling of the solver accuracy with the number of Fourier terms, database grid length and refinement multiplier. We take as a reference simulation the simple shear case with $N_t = 65{,}536$ Fourier terms, $N_c = 65{,}536$ crystals, a database grid length of $N_g = 128$ and a refinement multiplier of $N_r = 128$. For subsequent simulations, the error is defined as the L2 norm of the difference between the full history of the Cauchy stress and that in the reference simulation, normalized to the norm of the reference simulation. We then vary each of the three parameters separately, fixing the other two to be the same as in the reference simulation, to determine the influence of that parameter on the overall error. The exception is the database grid length test, which was was done with $N_t = 2048$ terms, the total number of terms in the smallest database.

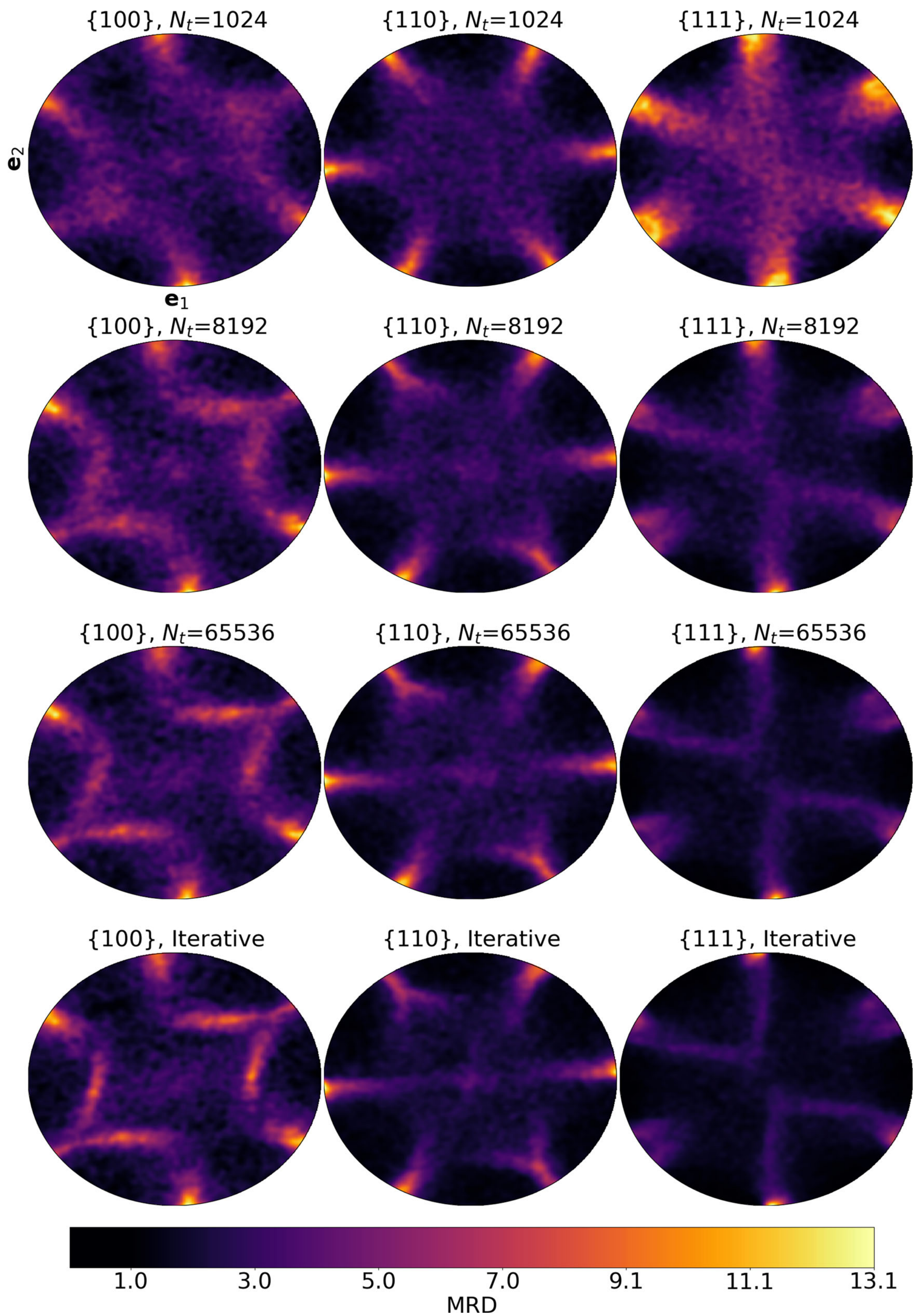

Fig. 2 Comparison of final textures by equal-area projection of crystal orientations for the spectral and iterative schemes for the simple shear case. The density is expressed as multiples of random distribution (MRD), with MRD = 1 corresponding to a "random" distribution that is uniform over SO(3). The first three rows are from the spectral scheme for a varying number of Fourier terms as indicated, and the final row is from the iterative scheme

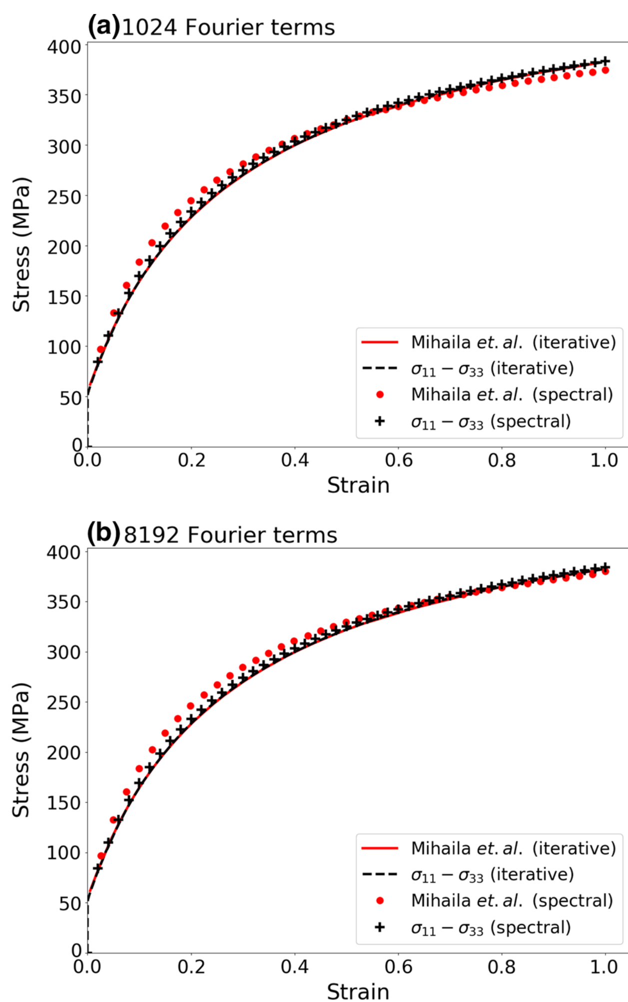

Fig. 3 Comparison of true stress strain curves for the iterative and spectral schemes for the plane strain compression case with the results in [17]. **a** Cases with 1024 Fourier terms. **b** Cases with 8192 Fourier terms

Figure 5 shows the dependence of the error on the database grid length, and Fig. 6 shows the dependence of the error on the refinement multiplier. The effect of the refinement multiplier on the stress strain curves and final textures is illustrated for the plane strain compression case in Figs. 7 and 8. Finally, Fig. 9 shows the dependence of the error on the number of Fourier terms used. The error decreases linearly with the database grid length and refinement multiplier, and decreases roughly as $\frac{1}{\sqrt{N_t}}$ with the number of Fourier terms.

4.3 Efficiency

We define the efficiency of the scheme as relative to an idealized implementation in which memory transactions have no cost and there is zero latency. Specifically, we do a summation of the reciprocal throughput of the arithmetic instructions required in the scheme to determine the idealized reciprocal throughput. This assumes that each warp scheduler is issuing one instruction per cycle, so the dual-issuing available on some architectures may allow for exceeding this idealized throughput. Dual-issuing is possible when two independent instructions that utilize different execution units are available from the same warp. This effect is heavily dependent on architecture and exact compiler output, and is hard to control at the source code level, so we choose not to attempt to model it in our "ideal" throughput target. This also allows for a cleaner comparison with prior implementations, as fewer assumptions about their code and computational environment need to be made.

The primary workload is the computation of the Fourier terms, so the throughput is measured as the number of Fourier terms per second. We define N_s to be the number of scalars associated with each Fourier coordinate in the spectral database. For this work, $N_s = 9$ accounts for the 5 necessary components of the symmetric and deviatoric $\boldsymbol{\sigma}'$, the 3 components of the anti-symmetric $\mathbf{W}_p$, and the scalar G.

Algorithm 1 `AddFourierTerm`

```
Require: j, u, k
Ensure: v
  I = j · k                              ▷ (D×IFMA)
  J = I mod L                            ▷ (1×AND)
  x = 2π J/L                             ▷ (1×I2F+1×FMUL)
  z = exp(ix)                            ▷ (1×SINCOS)
  for i_s = 1, ..., N_s do
     v[i_s] = v[i_s] + Re(u[i_s] × z)    ▷ (2×FFMA)
  end for
```

In Algorithm 1, we give pseudocode for adding a single Fourier term, with Fourier coefficients for each scalar stored in a length-N_s complex vector $\mathbf{u}$, and the corresponding real-valued sum of Fourier terms accumulated in the vector $\mathbf{v}$. We annotate each expression with the corresponding machine instruction(s) in our implementation. The instructions used are `(I/F/D)FMA` (integer/float/double fused multiply add), `AND` (bitwise and), `I2(F/D)` (conversion from integer to float/double), `(F/D)MUL` (float/double multiplication), and `SINCOS` (combined sine and cosine).

We present the cost of each instruction in terms of its reciprocal throughput (number of clock cycles per operation) for various compute capabilities along with the total cost per Fourier term in Table 2. For a given GPU, the ideal throughput is then given by $1.982 \times F \times N_{\mathrm{SM}}/T$, where F is the clock frequency, N_{SM} is the number of streaming multiprocessors (SMs) and T is the ideal reciprocal throughput per Fourier term. The factor of 1.982 takes into account the savings from using the symmetries in the discrete Fourier transform. This factor varies slightly depending on the total number of coefficients and the order of the coefficients, so here we've chosen the value measured for our most efficient simulation.

For example, the ideal throughput for the GTX 980 Ti used in this work, which has compute capability 5.2, has

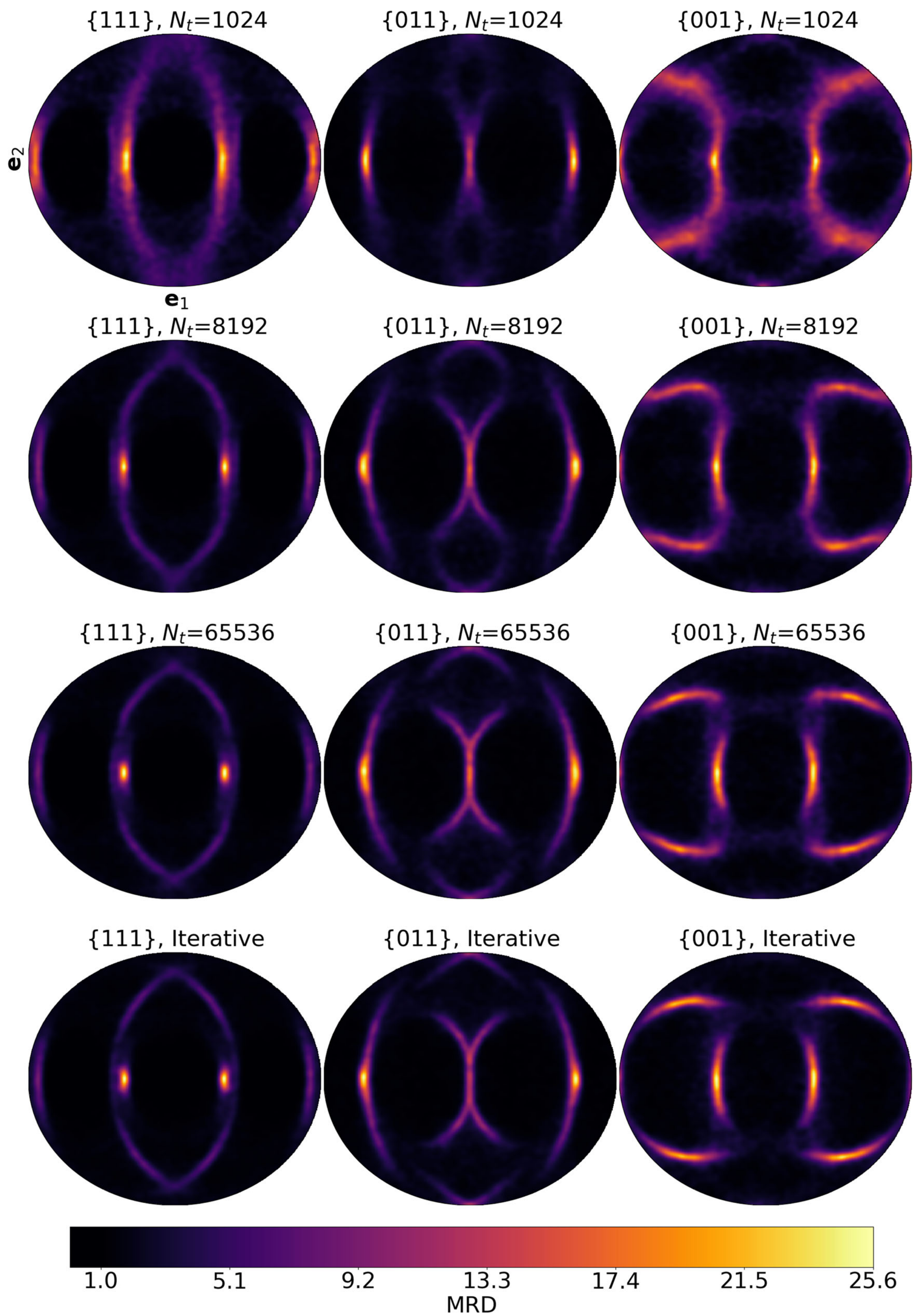

Fig. 4 Comparison of final textures by equal-area projection of crystal orientations for the spectral and iterative schemes for the plane strain compression case. The density is expressed as multiples of random distribution (MRD), with MRD = 1 corresponding to a "random" distribution that is uniform over SO(3). The first three rows are from the spectral scheme for a varying number of Fourier terms as indicated, and the final row is from the iterative scheme

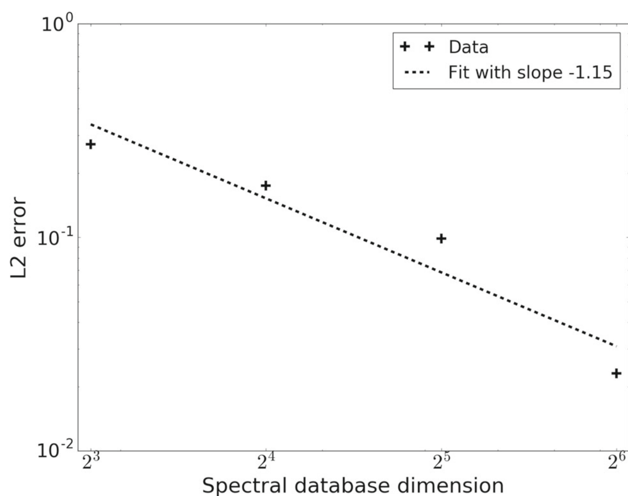

Fig. 5 L2 error as a function of database dimension. The slope of the fit is reported with respect to the log–log scale. The error decreases linearly with the database dimension

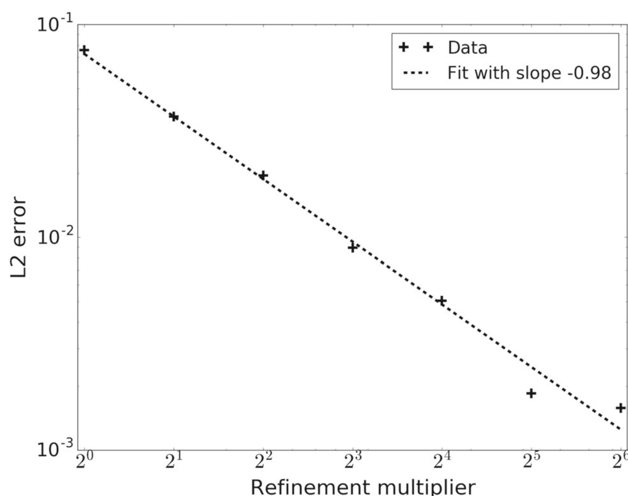

Fig. 6 L2 error as a function of refinement multiplier. The slope of the fit is reported with respect to the log–log scale. The error decreases linearly with the refinement multiplier

a clock frequency of $F = 1.2025$ GHz and $N_{\mathrm{SM}} = 22$ is $1.982 \times 1.2025 \times 10^9 \times 22/0.03472 = 1.510 \times 10^{12}$ terms/s. The efficiency is then defined as the ratio of the achieved throughput to this ideal throughput. The efficiency of this implementation is compared with that of prior implementations in Table 3. The ideal throughput is exceeded in this work due to exposing enough instruction-level parallelism that dual-issuing of instructions from warps is in fact occurring. For example, a `SINCOS` may be dispatched to the special function unit (SFU) at the same time as an `FFMA` is dispatched to the floating point unit (FPU). In terms of actual utilization, the profiling shows that 95% of the device's compute resources are utilized, comprised of 85% arithmetic, 9% memory operations and 1% control flow.

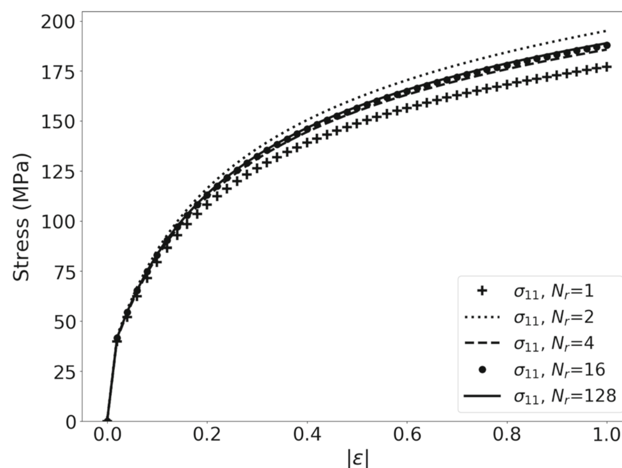

Fig. 7 Comparison of true stress strain curves for the spectral scheme for the plain strain compression case for a range of refinement multipliers

Since one of the implementations we compare with makes use of double precision floating point numbers, while ours makes use of single precision, we should note at this point that the floating point precision used in the computations has an effect on both accuracy and throughput. CUDA conforms to the IEEE-754-2008 standard in its representation of floating point numbers, in which there are 24 significant bits for single precision numbers and 53 significant bits for double precision numbers [19]. This affects different mathematical functions differently, but typically the errors are measured in single-digit multiples of the *unit in the last place* (ulp) of the argument, which is the gap between the two floating point numbers nearest the argument. For example, for the `sincos` operations used in this work, the maximum absolute error is 2 ulp for single precision and 1 ulp for double precision, which for arguments in the range $[-\pi, \pi]$ leads to a maximum absolute error of 4.17×10^{-7} for single precision and 4.44×10^{-16} for double precision [19]. Broadly speaking, all of the single precision operations will have errors on the order of 10^{-7}, while double precision operations will have errors on the order of 10^{-16}, which will affect any accumulated errors in the computation accordingly.

In terms of throughput, each CUDA architecture has different performance characteristics for each floating point precision type. Typically each architecture is tailored to perform well for a particular floating point precision. For example, the compute capability 5.2 architecture used in this work has 32 times faster single precision mutiply-add performance compared with its double precision performance, since it is designed for single precision performance. This is different from the 3.5 architecture used in the double precision implementation in [22], which has 16 times faster double precision multiply-add performance compared with the 5.2 architecture. These differences have been recorded and accounted for in Tables 2 and 3.

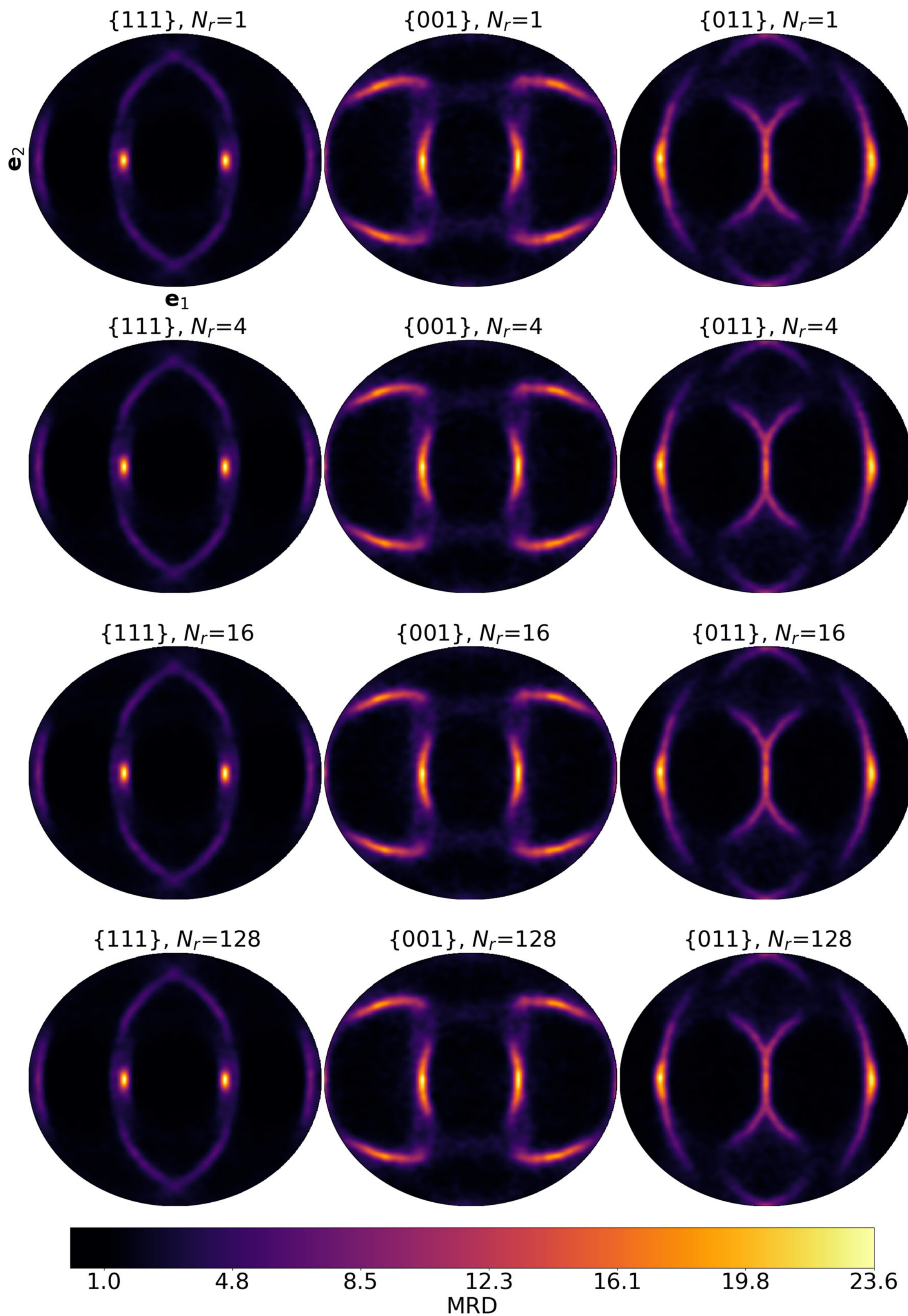

Fig. 8 Comparison of final textures by equal-area projection of crystal orientations for the spectral scheme over a range of refinement multipliers for the plane strain compression case. The density is expressed as multiples of random distribution (MRD), with MRD = 1 corresponding to a "random" distribution that is uniform over SO(3)

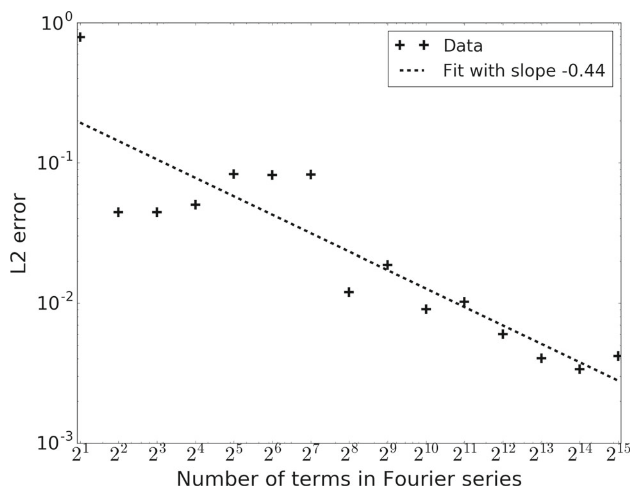

Fig. 9 L2 error as a function of number of Fourier terms. The slope of the fit is reported with respect to the log–log scale. The error decreases roughly as the reciprocal square root of the number of terms

4.4 Efficiency scaling

The peak efficiency presented in Table 3 is achieved with a large number of Fourier terms and a large number of crystals. We examine the effect of varying these factors independently on the efficiency of the scheme. In the first study, we fix the number of terms in the Fourier series to $2^{16} = 65{,}536$, and vary the number of crystals from 33 to $33 \times 2^{21} = 69{,}206{,}016$. This set of values is chosen so that the precise number of crystals required to saturate the GPU is within the set. For the GTX 980 Ti used in this work, this value is determined by there being 22 SMs with one block per SM, 768 threads per block and 2 crystals per thread for a total of 33,792 crystals. The throughput for this set of values is shown in Fig. 10. We see that indeed the peak throughput is nearly reached at 33,792 crystals, and is achieved slightly thereafter.

In the second study, we fix the number of crystals to 540,672, and vary the number of terms in the Fourier series. This is intended to determine for which regimes our approximation that the workload of the solver is dominated by Fourier series computations holds. The throughput as a function of the number of terms in the Fourier series is shown in Fig. 11. We see that the Fourier terms account for a little over half the workload with 256 terms, account for more than 90% of the workload with 4096 terms, and are effectively the entire workload with 8192 or more terms.

We now examine the relative contributions of our optimizations to the overall increase in efficiency over prior work. We take a simulation of 540,672 crystals with 65,536 Fourier terms, and turn off each of the two arithmetic optimizations individually to determine their effect on the efficiency. With both of them turned off, the remaining efficiency gains are due to the combined memory optimizations.

Table 2 Ideal reciprocal throughput of a single Fourier term on a single streaming multiprocessor for the CUDA compute capabilities and floating point precision (single or double) used in this and prior work

Instruction	Count	Reciprocal throughput by compute capability and floating point precision		
		2.0 (single)	3.5 (double)	5.2 (single)
`IFMA`	D/N_s	1/16	1/32	3/128*
`AND`	$1/N_s$	1/32	1/160	1/128
`I2(F/D)`	$1/N_s$	1/16	1/32	1/32
`(F/D)MUL`	$1/N_s$	1/32	1/64	1/128
`SINCOS`	$1/N_s$	1/4	5/64*	1/32
`(F/D)FMA`	2	1/32	1/64	1/128
Total		$\frac{(12+2D)/N_s+2}{32}$	$\frac{(42+10D)/N_s+5}{320}$	$\frac{(10+3D)/N_s+2}{128}$
Total	$(D = 4, N_s = 9)$	0.1319	0.04410	0.03472

Throughput values annotated with “*” correspond to operations that map to multiple native instructions on the given architecture. These values are therefore compiler-dependent, and are determined by inspecting disassembly for `nvcc` version 8.0.61. Other throughput values are taken directly from [19]. Both sets of values for the 5.2 architecture are confirmed by inspecting the full program disassembly of this implementation

Table 3 Comparison of computational throughput as measured by number of Fourier terms per second

Work	Hardware (precision)	Throughput (terms/s)		Efficiency (%)
		Achieved	Ideal	
[17]	2×Tesla C2050 (single)	4.96×10^{10}	4.838×10^{11}	10.3
[22]	Tesla K20 (double)	1.678×10^{10}	3.807×10^{11}	4.41
This work	GTX 980 Ti (single)	1.535×10^{12}	1.510×10^{12}	101.7

The efficiency is defined as relative to an ideal implementation in which there are no memory transactions, there is no instruction latency, and a single instruction is issued per cycle

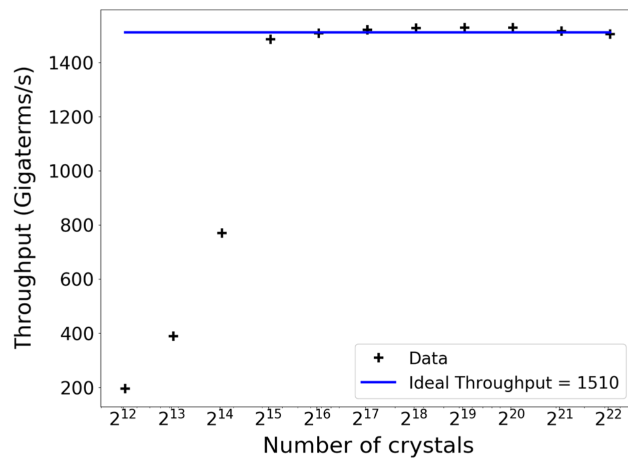

Fig. 10 Throughput as a function of number of crystals, with 65,536 terms used in the Fourier series

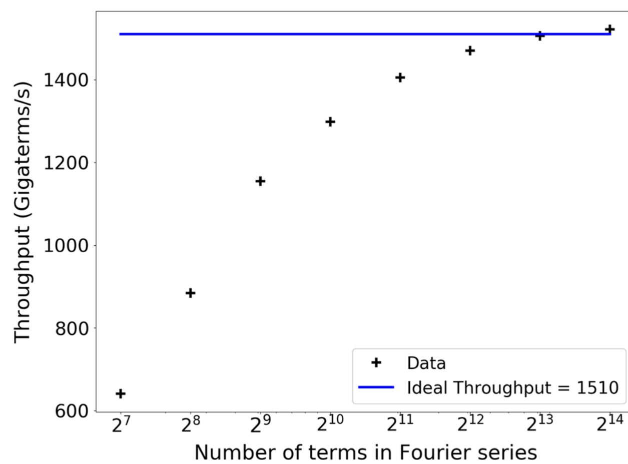

Fig. 11 Throughput as a function of number of terms in the Fourier series, with 540,672 crystals

With all the optimizations on, the throughput is 1.535×10^{12} terms per second. With only the fast modulo turned off, it's 1.123×10^{12} terms per second. With the range reduction left to the CUDA implementation, the effect on the throughput depends heavily on the refinement multiplier, as that will determine the size of the arguments to the trigonometric functions. This effect on the throughput is summarized in Fig. 12. With the refinement multiplier in the range 1–32, the throughput drops to around 7.9×10^{11} terms per second. This represents all arithmetic optimizations being turned off, so we can therefore partition the roughly tenfold increase in efficiency over prior implementations into a twofold increase from arithmetic optimizations and a fivefold increase from memory optimizations.

We note that at a refinement multiplier of $N_r = 64$ and greater, the throughput drops even further, as additional range reduction is required, dropping as low as 3.0331×10^{11} terms per second for a refinement multiplier of 128. This

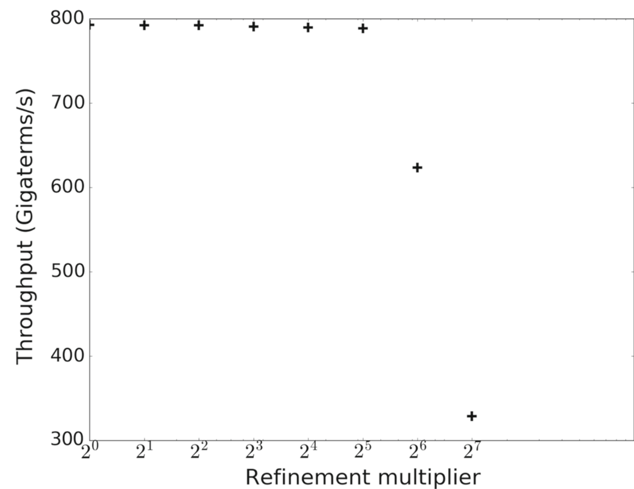

Fig. 12 Throughput as a function of the refinement multiplier for the case of no explicit range reduction of trigonometric arguments

can be explained by the CUDA range-reduction implementation needing to use the "slow path" to maintain accuracy, which activates for arguments larger than 48,039.0 for single precision [19]. The largest possible argument is $2\pi DL = 8\pi N_g N_r = 1024\pi N_r$, so solving for N_r we get that the earliest possible refinement multiplier at which the slow path is activated is around $N_r = 15$. In Fig. 12 we observe a very minor slowdown at $N_r = 16$, but the significant slowdown occurs at greater refinement multipliers as more than just the largest terms fall into the slow path.

4.5 Memory constraints

The theoretical memory requirement for each crystal is 4 floating point numbers: the three Euler angles defining the crystal orientation and the single slip system deformation resistance. For single precision this amounts to 16 bytes. We find that on a simulation with 390 million crystals, the amount of memory consumed is 6.36 GiB, which is 16.3 bytes per crystal, in line with the theoretical bound when accounting for overheads. With 1024 Fourier terms, the simulation executes at a rate of 2.72 s per strain step. The memory required per crystal places a hard limit on the possible number of crystals in a single simulation. In future implementations we may avoid this by splitting the memory in half and buffering transactions with RAM, which can be executed concurrently with the computation.

4.6 More complex material models

A natural extension for this approach would be to use a more complex hardening law. For example, one that takes into account differences between the self-hardening rates and latent hardening rates of slip systems, such as the model pre-

sented in [8]. In this case, the hardening of each of the 4 sets of 3 coplanar slip systems must be tracked independently. Replacing the single slip deformation resistance s with 4 different resistances, one for each set of coplanar slip systems, would increase the dimension of the state space from 9 (5 components of $\boldsymbol{\sigma}'$, 3 components of $\mathbf{W}^p$ and a single slip deformation resistance s) to 12. A general increase in the dimension of the state space would impact the efficiency, memory and storage requirements of the solver based on the ratio of the new state space dimension to the old state space dimension. In this case, it would first increase the amount of arithmetic required by a factor of approximately 4/3, as 4/3 times more inverse DFTs would need to be computed. Second, it would increase the size of the crystal database by a factor of 4/3. Third, the number of database terms that could be loaded into shared memory at once would be reduced by 4/3. Finally, the global memory required per crystal would increase from 4 floating point numbers (the three orientation angles and the slip resistance) to 7 floating point numbers.

In terms of homogenization, it is known that the Taylor model used here produces sharper textures than those from a more accurate homogenization-based finite element method [23]. This can be improved upon by integrating the SCP solver into a more advanced homogenization scheme. For example, this integration is done for viscoplastic self-consistent homogenization in the SCP-VPFFT scheme presented in [6].

5 Conclusions

In this work we presented an improved GPU implementation of a spectral crystal plasticity solver. The key improvements came in speeding up the inverse DFT implementation, primarily by avoiding the matrix–matrix multiplications of prior implementations, and by performing explicit fast range-reduction on arguments before they were passed to trigonometric functions. The improvements combined for a factor of 10 increase in computational efficiency, of which a factor of 5 was attributable to improvements in memory transactions, and a factor of 2 was attributable to improvements in arithmetic efficiency.

The new implementation was found to also be more efficient in its memory footprint, leading to simulations of hundreds of millions of crystal grains being feasible on a single consumer-grade GPU. This development allows for much larger-scale microstructure-sensitive simulations to be practically computed, and future work will explore the integration of this improved implementation into large-scale crystal plasticity finite element simulations.

Acknowledgements The author would like to thank Professor Daya Reddy for his support and comments on this work. The author would like to thank the National Research Foundation of South Africa for supporting the work. Computations were performed using facilities provided by the University of Cape Town's ICTS High Performance Computing team.

References

1. Alers GA, Thompson DO (1961) Dislocation contributions to the modulus and damping in copper at megacycle frequencies. J Appl Phys 32(2):283–293. https://doi.org/10.1063/1.1735992
2. Arvo J (1992) Fast random rotation matrices. Graph Gems III 5(1):117–120
3. Asaro RRJ, Needleman A (1985) Overview no. 42 Texture development and strain hardening in rate dependent polycrystals. Acta Metall 33(6):923–953
4. Brodtkorb AR, Hagen TR, Sætra ML (2013) Graphics processing unit (GPU) programming strategies and trends in GPU computing. J. Parallel Distrib. Comput. 73(1):4–13. https://doi.org/10.1016/j.jpdc.2012.04.003
5. Dederichs PH, Leibfried G (1969) Elastic green's function for anisotropic cubic crystals. Phys Rev 188(3): 1175–1188 (1969). http://prola.aps.org/abstract/PR/v188/i3/p1175_1
6. Eghtesad A, Zecevic M, Lebensohn RA, McCabe RJ, Knezevic M (2017) Spectral database constitutive representation within a spectral micromechanical solver for computationally efficient polycrystal plasticity modelling. Comput Mech, pp 1–16. https://doi.org/10.1007/s00466-017-1413-4
7. Fritzen F, Hodapp M, Leuschner M (2014) GPU accelerated computational homogenization based on a variational approach in a reduced basis framework. Comput Methods Appl Mech Eng 278:186–217. https://doi.org/10.1016/j.cma.2014.05.006
8. Kalidindi SR, Bronkhorst CA, Anand L (1992) Crystallographic texture evolution in bulk deformation processing of FCC metals. J Mech Phys Solids 40(3):537–569
9. Kalidindi SR, Duvvuru HK (2005) Spectral methods for capturing crystallographic texture evolution during large plastic strains in metals. Acta Mater 53:3613–3623. https://doi.org/10.1016/j.actamat.2005.04.017
10. Knezevic M, Al-harbi HF, Kalidindi SR (2009) Crystal plasticity simulations using discrete Fourier transforms. Acta Mater 57(6):1777–1784. https://doi.org/10.1016/j.actamat.2008.12.017
11. Knezevic M, Kalidindi SR (2017) Crystal plasticity modeling of microstructure evolution and mechanical fields during processing of metals using spectral databases. JOM 69(5):16–20. https://doi.org/10.1007/s11837-017-2289-7
12. Knezevic M, Kalidindi SR, Fullwood D (2008) Computationally efficient database and spectral interpolation for fully plastic Taylor-type crystal plasticity calculations of face-centered cubic polycrystals. Int J Plast 24:1264–1276. https://doi.org/10.1016/j.ijplas.2007.12.002
13. Knezevic M, Savage DJ (2014) A high-performance computational framework for fast crystal plasticity simulations. Comput Mater Sci 83:101–106. https://doi.org/10.1016/j.commatsci.2013.11.012
14. Kroner E (1961) On the plastic deformation of polycrystals. Acta Metall 9(2):155–161
15. Li D, Garmestani H, Schoenfeld S (2003) Evolution of crystal orientation distribution coefficients during plastic deformation. Scripta Mater 49(9):867–872. https://doi.org/10.1016/S1359-6462(03)00443-3
16. Li DS, Garmestani H, Ahzi S (2007) Processing path optimization to achieve desired texture in polycrystalline materials. Acta Mater 55(2):647–654. https://doi.org/10.1016/j.actamat.2006.04.041
17. Mihaila B, Knezevic M, Cardenas A (2014) Three orders of magnitude improved efficiency with high-performance spectral crystal

plasticity on GPU platforms. Int J Numer Meth Eng 97(11):785–798. https://doi.org/10.1002/nme.4592
18. Molinari A, Canova GR, Ahzi S (1987) A self consistent approach of the large deformation. Acta Metall 35(12):2983–2994
19. NVIDIA: CUDA C programming guide, v8.0. NVIDIA (2017)
20. Peirce D, Asaro RJ, Needleman A (1983) Material rate dependence and localised deformation in crystalline solids. Acta Metall 31(12):1951–1976
21. Rice JR (1971) Inelastic constitutive relations for solids: an internal variables theory and its application to metal plasticity. J Mech Phys Solids 19(6):433–455
22. Savage DJ, Knezevic M (2015) Computer implementations of iterative and non-iterative crystal plasticity solvers on high performance graphics hardware. Comput Mech 56(4):677–690. https://doi.org/10.1007/s00466-015-1194-6
23. Tadano Y, Kuroda M, Noguchi H (2012) Quantitative re-examination of Taylor model for FCC polycrystals. Comput Mater Sci 51(1):290–302. https://doi.org/10.1016/j.commatsci.2011.07.024
24. Taylor GI (1938) Plastic strain in metals. J Inst Met 62:307–324
25. Tome C, Canova GR, Kocks UF, Christodoulou N, Jonas JJ (1984) The relation between macroscopic and microscopic strain hardening in FCC polycrystals. Acta Metall 32(10):1637–1653
26. Van Houtte P (1994) Application of plastic potentials to strain rate sensitive and insensitive anisotropic materials. Int J Plast 10(7):719–748
27. Van Houtte P, Delannay L, Samajdar I (1999) Quantitative prediction of cold rolling textures in low-carbon steel by means of the lamel model. Textures Microstruct 31(3):109–149. https://doi.org/10.1155/TSM.31.109
28. Zecevic M, McCabe RJ, Knezevic M (2015) A new implementation of the spectral crystal plasticity framework in implicit finite elements. Mech Mater 84:114–126. https://doi.org/10.1016/j.mechmat.2015.01.018
29. Zecevic M, McCabe RJ, Knezevic M (2015) Spectral database solutions to elasto-viscoplasticity within finite elements: application to a cobalt-based FCC superalloy. Int J Plast 70:151–165. https://doi.org/10.1016/j.ijplas.2015.03.007

Computational Mechanics is a copyright of Springer, 2018. All Rights Reserved.