libKEDF: An Accelerated Library of Kinetic Energy Density Functionals

Johannes M. Dieterich,[a] William C. Witt,[a] and Emily A. Carter *[b]

Kinetic energy density functionals (KEDFs) approximate the kinetic energy of a system of electrons directly from its electron density. They are used in electronic structure methods that lack direct access to orbitals, for example, orbital-free density functional theory (OFDFT) and certain embedding schemes. In this contribution, we introduce libKEDF, an accelerated library of modern KEDF implementations that emphasizes nonlocal KEDFs. We discuss implementation details and assess the performance of the KEDF implementations for large numbers of atoms. We show that using libKEDF, a single computing node or (GPU) accelerator can provide easy computational access to mesoscale chemical and materials science phenomena using OFDFT algorithms. © 2017 Wiley Periodicals, Inc.

DOI: 10.1002/jcc.24806

Introduction

Kinetic energy density functionals (KEDFs) obtain the noninteracting kinetic energy (T_s) and kinetic potential ($\delta T_S/\delta\rho(\vec{r})$) of a system of electrons directly from its electron density.[1] They are a key ingredient in orbital-free density functional theory (OFDFT)[1,2] and in some formulations of orbital-free embedding theories (OFETs).[3–8] The most sophisticated contemporary KEDFs can obtain accurate kinetic energies for several types of materials, including solid and liquid phases of light metals[9,10] and some semiconductors.[11–13] Ongoing efforts aim to overcome the main remaining weakness of KEDFs,[14,15] which include incorrect description of highly fluctuating electronic densities as found in covalently bound molecules[12] and transition metals.[16–18]

We can compare this activity to that of another important area of research in density functional theory (DFT): the development of exchange-correlation (XC) functionals. The libxc[19] library has had great success in providing a framework for reference implementations of XC functionals. It attracted and maintains an active developer and user ecosystem. We wish to provide a similar point of convergence for the KEDF community by introducing libKEDF, an accelerated, free, and open-source library of KEDFs for developers and users alike under a BSD 3-clause license.[20] A differentiating feature of libKEDF is its emphasis on nonlocal functionals and their attendant complications (e.g., accessing fast Fourier transform [FFT] libraries).

Our goals for libKEDF are to provide a modern software library with (1) high code quality, (2) facile portability, and (3) excellent performance. We provide a clean advanced programming interface (API) for outside users, as well as developer-friendly internal interfaces and code. We view high-quality code as essential to ensuring numerical accuracy, enabling maintainability, and encouraging use and extension. Facile portability, closely linked to code quality, refers to portability across software platforms (operating systems and toolchains) and hardware platforms (processors and accelerators). This feature is crucial because, after a brief period of convergence toward x86 processors for desktops and servers, a litany of new hardware platforms has appeared alongside a variety of ways of programming them. Lastly, excellent performance is generally required for consumer codes to adopt any library. Particularly in the case of OFDFT, we argue that good performance for large systems will advance adoption for computationally expensive simulations such as OFDFT molecular dynamics (MD),[10,21,22] Monte–Carlo (MC), and nanoscale structure optimizations.[23] These gains will lend insight into important challenges in materials science.

We discuss in the following our modern KEDF implementations. We present general algorithmic and implementation aspects first, followed by a summary of performance benchmark data and discussion of relevant implementation details. We conclude by providing an outlook for future development and integration targets of libKEDF.

Algorithms and General Implementation

Nonlocal KEDFs

In this section, we discuss briefly the three nonlocal KEDFs that serve as a suite of test cases for our implementation strategy. One can find a more detailed treatment of nonlocal KEDFs in several review articles.[1,24,25] The subject has a long history, and several early publications[26,27] inspired a great deal of follow-up work, including, for example, more recent

[a] J. M. Dieterich, W. C. Witt
Department of Mechanical and Aerospace Engineering, Princeton University, Princeton, New Jersey 08544-5263

[b] E. A. Carter
School of Engineering and Applied Science, Princeton University, Princeton, New Jersey 08544-5263
E-mail: eac@princeton.edu

Contract grant sponsor: Office of Naval Research (to E. A. C.); Contract grant number: AWD1004388; Contract grant sponsor: National Science Foundation (to W. C. W.); Contract grant number: NSF 1656466

generalizations that include time-dependence[28] or finite temperature effects.[29]

The three nonlocal KEDFs we consider in this work are constructed as a sum of multiple terms,

$$T_S[\rho]=T_{\mathrm{local}}[\rho]+T_{\mathrm{semi-local}}[\rho]+T_{\mathrm{non-local}}[\rho] \tag{1}$$

in which the local term is the Thomas–Fermi (TF) functional,[30,31] exact for homogeneous electron gases, and the semilocal term is the von Weizsäcker (vW) functional,[32] exact for a single orbital. At present, the choice of nonlocal term depends on the system of interest. For the KEDFs we consider, a generic form for the nonlocal term may be written as

$$T_{\mathrm{non-local}}[\rho]=c_{\mathrm{NL}}\int\int \rho^{\alpha}(\vec{r})\; w\left(\vec{r},\vec{r}^{\,\prime}\right)\; \rho^{\beta}\left(\vec{r}^{\,\prime}\right)\; d^3r\; d^3r^{\prime} \tag{2}$$

where c_{NL}, α, and β are constants and $w\left(\vec{r},\vec{r}^{\,\prime}\right)$ is an integral kernel. Wang and Teter (WT)[27] proposed a density independent $w\left(\vec{r},\vec{r}^{\,\prime}\right)$ that is suitable for nearly free-electron bulk systems. Some years later, Wang, Govind, and Carter (WGC99)[33,34] developed a doubly density-dependent $w\left(\vec{r},\vec{r}^{\,\prime}\right)$ to accommodate more rapid density variations, such as those encountered in light metal samples with defects such as vacancies or surfaces. More recently, Huang and Carter (HC10)[11] further modified $w\left(\vec{r},\vec{r}^{\,\prime}\right)$ to better incorporate the physics of bulk semiconductors. Other nonlocal KEDFs for semiconductors derived from the WGC99 functional are not considered here as a separate class.[12,13]

In all three cases, key portions of $w\left(\vec{r},\vec{r}^{\,\prime}\right)$ depend only on the distance between $\vec{r}$ and $\vec{r}^{\,\prime}$ (i.e., $|\vec{r}-\vec{r}^{\,\prime}|$), which indicates the possibility of treating the nonlocal term as a convolution. This observation is important because convolutions in real space correspond to simple multiplications in reciprocal space, enabling one to employ FFT libraries to evaluate convolutions with $O(N\log N)$ computational scaling, where N is a measure of the system size. For the WT functional, this strategy alone is sufficient to achieve quasi-linear scaling, but additional sophistication is required for the WGC99 and HC10 functionals. In the WGC99 case, a Taylor expansion is used to approximate $w\left(\vec{r},\vec{r}^{\,\prime}\right)$ based on its form for a uniform electron density, and the Taylor expansion is evaluated to either first or second order.[33] For the HC10 functional, a density-binning strategy is employed to achieve quasi-linear scaling.[11]

General implementation

The most important feature of our library implementation is an object-oriented strategy for consolidating commonly required math operations. The main philosophy is to treat, say, any quantity represented on a Cartesian, equidistantly spaced grid having periodic boundary conditions as part of a class endowed with access to both simple operations (e.g., integrals) as well as more complicated methods (e.g., FFT-enabled convolutions). This approach facilitates efficient code re-use. Moreover, KEDF implementations need only provide subkernels specific to that particular KEDF term and can otherwise call generic routines. For both generic and specialized kernels, we balance considerations of memory footprint against computational cost when deciding whether to store intermediate quantities or compute them on the fly, keeping in mind memory resources of typical compute infrastructures. By default, memory is allocated aligned to facilitate single instruction multiple data (SIMD) instructions for computations, thus ensuring efficiency.

Our implementations for all KEDF terms (except TF) require at least one conversion to reciprocal space (and back). This implies access to three-dimensional, real-to-complex and complex-to-real FFTs. The FFT package employed depends on the acceleration strategy. For this reason, we currently include implementations of Cartesian periodic grids for shared memory parallelization and OpenCL acceleration. To add another Cartesian representation, developers must implement the required API of generic operations, potentially reusing existing kernels. Only when adding a non-Cartesian representation without these features will additional template specializations be required to implement a given KEDF term. A list of mathematical operations available for a particular grid type (e.g., integrating over a grid, summing two quantities defined on the same grid, computing the divergence of a vector field, etc.) as defined by a C++ interface must be provided. However, most KEDF terms require only a small subset of operations. For example, the vW semilocal term requires the Laplacian of the electron density, an elementwise grid multiplication, and an integration over the grid.

Our shared-memory backend makes use of OpenMP[35] for parallelization and a parallelized FFT library with the FFTW3 interface[36] for necessary transformations. OpenMP is pragma-based and allows us to reuse the code also in non-OpenMP circumstances. As an established and continuously evolving standard, OpenMP enjoys widespread adoption across software and hardware platforms and is ideally suited to exploit the increasingly parallel shared-memory platforms our codes are and will be using.

Compute accelerator technologies are integral on private compute platforms in the form of graphics processing units and now play an increasingly important role in high-performance computing. We envision libKEDF as a library providing good performance over the entire range of hardware accessible to a researcher: from personal computing devices to compute clusters. We hence require a programming standard for accelerators that has three capabilities: (1) portability across accelerator types and vendors, (2) sufficient linguistic means to extract good performance out of an accelerator, and (3) adoption across platforms and availability of FFT libraries for this standard. We found OpenCL[37] to be the only standard currently meeting all of these requirements, with support from all major vendors, obtained through an open standardization process, and availability of, for example, clFFT[38] for accelerated FFT calls. However, adding other representations of the electron density that make use of other accelerator programming interfaces such as OpenMP offloading or CUDA[39] is

Table 1. List of implemented KEDFs accessible via either the C or C++ API in libKEDF.

Kinetic energy functional	Character
Thomas–Fermi	local
von Weizsäcker	semilocal
a Thomas–Fermi + b von Weizsäcker	semilocal
Wang–Teter	nonlocal
Smargiassi–Madden (Wang–Teter)	nonlocal
Wang–Govind–Carter 98 (Wang–Teter)	nonlocal
First-order Wang–Govind–Carter 99	nonlocal
Second-order Wang–Govind–Carter 99	nonlocal
Huang–Carter 10	nonlocal

straightforward, as noted above. We do not support distributed-memory representations yet, in part because our tests currently show no need for such platforms for medium-to-large applications (*vide infra*). Similarly, all of our computations are carried out in double precision (DP), also on accelerators. Preliminary tests indicate that single precision (SP) computations will need to be carefully implemented with numerical accuracy in mind. SP to DP peak performance of accelerators varies drastically, between 16:1 for gaming cards and 2:1 for high performance computing (HPC) accelerators. The benefit of switching to SP varies accordingly, even if sufficient numerical accuracy is assured, which makes DP throughout a safer choice.

As a library component, we provide standardized ways to interface with libKEDF. Obviously, the internal C++ API also can be used to directly interact with libKEDF. However, this is not feasible for consumer codes not written in C++ themselves, and this API may be subject to change. Hence, we provide a simple, stable C API featuring functions to initialize and setup libKEDF for any of the functionals listed in Table 1, and also to evaluate the kinetic energy from an electron density or kinetic energy and kinetic potential together. We do not provide means to evaluate only the kinetic potential, since the overhead of evaluating the kinetic energy amounts typically to an integration over some intermediate data structure needed for the potential with the expense of one floating point operation per grid point. This is a negligible computational cost when compared to arriving at the potential. The C API can be used by any programming language featuring C bindings. We also maintain official implementations for Java Native Interface (JNI), Python, and Fortran for reference and use.

We ensure portability of libKEDF across operating systems and compiler toolchains. We officially support recent LLVM[40] and GNU[41] compilers on FreeBSD[42] and Linux[43] distributions. Ports to other C++14 compliant toolchains and other operating systems are straightforward. We provide a growing collection of electron densities and accompanying kinetic energies and kinetic potentials obtained from self-consistent tests with old legacy KEDF implementations to use for checking numerical accuracy. Our implementations agree with these references to within a numerical threshold of at least 1×10^{-5} a.u. for all energies and potential elements. We next report the performance of libKEDF on shared-memory and accelerator architectures.

Performance Assessment and Implementation Details

The performance of all KEDF implementations presented here (except HC10) is independent of the character of the electron density. Since HC10 employs a binning scheme for the density,[11] we showcase two test systems with different electron density characters. Test system I represents a typical semiconductor electron density, which we obtained from a converged WT density optimization, using the local density approximation (LDA)[44] for XC, of a $10 \times 10 \times 10$ supercell of cubic-diamond (cd) Si containing 8000 Si atoms at a plane-wave basis kinetic energy cutoff of 1600 eV. The real-space grid of system I is $360 \times 360 \times 360$ points. Test system II represents a typical light-metal density, obtained from a converged WT/LDA density optimization of a $25 \times 25 \times 25$ supercell of face-centered-cubic (fcc) Al containing 62,500 Al atoms at an energy cutoff of 550 eV. The real-space grid of system II is $384 \times 384 \times 384$ points. All energy cutoffs employed here ensure convergence in the total energy to less than 1 meV/atom. Both test systems represent typical medium- to large-sized OFDFT calculations as routinely carried out in our group.

We reiterate that our principal concern in this work is the computational performance of our KEDF implementations, and in particular the performance of the nonlocal term. The physical predictions made by the new implementations are identical to other implementations reported earlier.[45] Hence, while the test system choices are somewhat arbitrary and do not in every case have strict physical motivation (most strikingly, the WT KEDF does not predict binding for the Si test system[46]), even randomly generated densities would have sufficed for many of our tests. A comparison of converged density profiles obtained for system I using the three KEDFs, as well as KSDFT, can be found in Supporting Information Figure S-1.

We analyze the performance of the libKEDF library on a variety of different hardware configurations to demonstrate performance and portability. All numerical tests are conducted on five different hardware platforms as summarized in Table 2. These platforms encompass typical server and workstation hardware for both shared-memory parallelization (tiger/cpu), as well as NVIDIA accelerators (tiger/K20 and mcmillan/K40), an AMD accelerator (hyperion/S9150), and consumer GPU (hyperion/Nano). Only free and open-source software, as well as vendor-independent free and open programming standards, are used throughout the software stack, with the exception of proprietary drivers and OpenCL runtimes from both NVIDIA and AMD. All timing values are averaged over 100 independent function calls. All timings we report are realistic end-to-end wall times including, if applicable, data transfer and copy to and from accelerators.

CPU implementation

The OpenMP-parallelized implementation relies on FFTW3-compliant libraries to execute three-dimensional FFTs in an out-of-place fashion. Parallelization of the KEDF kernels is carried out over the grid points in real or reciprocal space, relying

Table 2. Hardware platforms employed.

	tiger/CPU	tiger/K20	mcmillan/K40	hyperion/ S9150	hyperion/ Nano
CPU	2x Intel E5-2670	2x Intel E5-2670	2x Intel E5-2667	2x Intel E5-2620v4	2x Intel E5-2620v4
OS	RHEL 6.8	RHEL 6.8	RHEL 7.2	UBUNTU 16.04	UBUNTU 16.04
Acceleration	OpenMP	OpenCL	OpenCL	OpenCL	OpenCL
Compiler	G++ 5.3.1	G++ 5.3.1	G++ 5.3.1	G++ 5.4.0	G++ 5.4.0
Accelerator	N/A	NVIDIA K20m	NVIDIA K40c	AMD S9150	AMD R9 Nano
Acc. memory	N/A	5GB GDDR	12GB GDDR	16GB GDDR	4GB HBM
OpenCL runtime	N/A	CUDA 7.5	CUDA 7.5	AMDGPU-PRO/ 2117.10 OpenCL	AMDGPU-PRO/ 2117.10 OpenCL
FFT Library	FFTW 3.3.3	clFFT 2.12.1	clFFT 2.12.1	clFFT 2.12.1	clFFT 2.12.1

Details include processor type (CPU), operating system (OS), accelerator type and details, and relevant software versions.

on parallelized, real-to-complex, and complex-to-real FFT invocations otherwise. This parallelization is lock-free since quasi-linear scaling KEDFs do not exhibit any data dependency between any two grid points. Integrations over grid objects are parallelized over the slowest index with the intermediate results added using atomic operations. As an example, the WT nonlocal term is obtained by first evaluating ρ^β in real space with explicit parallelization over the real space grid points. Subsequently, this intermediate is transformed to reciprocal space using a parallelized FFT library for convolution with the WT integral kernel. The convolution itself is explicitly parallelized over all reciprocal-space grid points. We then transform the result the intermediate back to real space using the FFT library. The convolved object is multiplied by ρ^α to obtain the kinetic energy density, again explicitly parallelized over all real-space grid points. Integration yields the kinetic energy.

The simple, FFT-free TF KEDF can be evaluated for both test systems within roughly 0.2 s for both the kinetic energy and the kinetic energy + potential (Tables 3 and 4). Using a 1:1 threads:cores mapping with OpenMP yields speedups of roughly 12x over serial evaluation on an otherwise empty node. Detailed scaling information can be found in the Supporting Information for this KEDF and all other terms in Figures S-3 and S-14. This scaling is satisfactory given the simplicity of the TF kernel. The vW KEDF scales slightly worse, around 10x, which can probably be attributed to the dominating FFT transformations. Nevertheless, the vW kinetic energy and corresponding energy + potential for both large test systems can be evaluated in under 0.4 s.

As discussed above, TF and vW are required ingredients for all nonlocal KEDFs considered in this work. In the following, we discuss only the characteristics of the additional nonlocal term. All timings exclude the mandatory TF and vW evaluations associated with the full KEDF.

The nonlocal terms considered here are more computationally demanding than the TF and vW terms. The WT term is approximately three times as expensive as TF and vW separately, but nevertheless can still be evaluated in 0.66–1.07 s with very good parallel scaling of 12.7–13.6x on 16 cores. A full WT KEDF energy + potential evaluation (including TF and vW) costs approximately 1.6 s for system II. Evaluating the first-order, Taylor-expanded WGC99 term is up to 65% more expensive than the WT nonlocal term. Evaluating the Taylor-expanded WGC99 nonlocal term to second order (including the first order contribution) is roughly twice as computationally expensive as the WT term. A full energy + potential second-order WGC99 KEDF evaluation will take approximately 2.8 s for test system II, a very fast result. Scaling of both WGC99 approaches is in line with the other terms, at a speedup of approximately 13x for 16 cores.

As discussed above, the HC10 nonlocal term is significantly more complex than any of the other nonlocal terms implemented because it bins the electron density and evaluates most sub-kernels for each of the bins. Hence, the computational cost scales with the number of bins and is directly related to the largest density difference between any two grid points, as well as the parametrization of the HC10 functional.[11] Our test systems are representative of two typical solids: a metal and a semiconductor. With default parametrization, the HC10 nonlocal term is almost five times as expensive as the second-order WGC99 Taylor-expanded term for these systems. Earlier tests report a significantly higher relative cost

Table 3. Cost of evaluation and parallel efficiency of OpenMP KEDF implementations on platform tiger/CPU for test system I (real-space grid size 360 × 360 × 360) using 16 OpenMP threads on all 16 cores.

KEDF or KEDF nonlocal term	Energy evaluation cost [s]	Parallel scaling on 16 cores [X]	Energy + potential evaluation cost [s]	Parallel scaling on 16 cores [X]
Thomas–Fermi	0.16	12.6	0.18	12.2
von Weizsäcker	0.28	10.1	0.31	10.3
Wang–Teter	0.66	13.5	0.91	12.7
First-order Wang-Govind-Carter 99	0.88	12.8	1.53	12.7
Second-order Wang–Govind–Carter 99	1.02	13.6	1.76	13.7
Huang–Carter	5.09	12.2	8.44	11.6

Table 4. Cost of evaluation and parallel efficiency of OpenMP KEDF implementations on platform tiger/CPU for test system II (real-space grid size 384 × 384 × 384) using 16 OpenMP threads on all 16 cores.

KEDF or KEDF nonlocal term	Energy evaluation cost [s]	Parallel scaling on 16 cores [X]	Energy + potential evaluation cost [s]	Parallel scaling on 16 cores [X]
Thomas–Fermi	0.20	12.2	0.22	12.1
von Weizsäcker	0.31	10.4	0.35	10.5
Wang–Teter	0.78	13.6	1.07	12.7
First-order Wang–Govind–Carter 99	1.02	13.1	1.76	13.0
Second-order Wang-Govind–Carter 99	1.27	12.8	2.19	12.9
Huang–Carter	5.84	12.4	9.26	12.1

of HC10.[13] One of the details of our implementation to achieve this improved performance is in the HC10 lookup table implementation. After solving the HC10 ordinary differential equation (ODE), we obtain a lookup table containing discretized solutions. This lookup table, typically in the range of multiple tens of thousands of elements, is then used to create the HC10 integral kernel for each density bin. For contemporary CPUs, this table does not fit into caches close to the CPU cores, making random access expensive. However, we observed that most requests into the lookup table happen for smaller numerical values. We therefore split the lookup table into two, one shorter, smaller-element lookup table and a higher-element one. The smaller lookup table is cut to fit better into caches. We observed a significant reduction of the cost to evaluate the HC10 functional as a result of this change. Together with other general performance improvements provided by libKEDF, we can evaluate energy + potential of test system I, representing 8000 Si atoms, with the HC10 KEDF in approximately 9 s on 16 cores.

All libKEDF implementations follow the expected linearithmic scaling as shown in the Supplementary Information (Supporting Information Fig. S-2). We evaluated all KEDFs on real-space grids with sizes up to 1280 × 1280 × 1280 on a tiger/CPU node with 128 GB RAM, unless the data set could not fit in the main memory. This is comparable to the largest size (1024 × 1024 × 1024 in real space) reported in the literature for a one-million atom liquid lithium MD simulation employing tens of thousands of CPUs.[23] Future hardware development will enable us to routinely study such systems with libKEDF.

Table 5. Number of specialized kernels needed to evaluate the energy and energy + potential of certain terms.

	Number of specialized OpenCL kernels for energy evaluation	Number of specialized OpenCL kernels for energy + potential evaluation
Thomas–Fermi	1	1
von Weizsäcker	0	1
Wang–Teter	1	2
First-order Wang–Govind–Carter 99	3	4 (2)
Second-order Wang–Govind–Carter 99	3	5 (2)
Huang–Carter	8	12 (4)

In parentheses: number of kernels reused from energy evaluation in energy + potential. Number of generic kernel implementations in the grid type currently: 20.

OpenCL accelerated implementation

All of our OpenCL-accelerated implementations are fully vendor-neutral, that is, no vendor- or device-specific blocking is implemented. Like the OpenMP implementations, all kernels are enqueued over all grid points in reciprocal space. In real space, we block enqueueing with the maximal OpenCL double vector type possible (double/2/4/8/16) for a given grid size at runtime unless overridden. clFFT[38] is used for the necessary FFTs on the OpenCL accelerator.

Implementing fully OpenCL-accelerated KEDF terms does require implementation of any specialized sub-kernels in OpenCL. However, due to our implementation strategy detailed above, most kernels only need to be implemented once generically. As we show in Table 5, even advanced nonlocal terms such as HC10 only require us to implement eight kernels for the kinetic energy and energy + potential (plus reusing four energy kernels). All other terms only require 1–3 kernels to be implemented. The implementation burden of a new accelerated KEDF term therefore is significantly reduced.

We can compare the performance obtained from these generic OpenCL kernels on our accelerated platforms to the reference benchmark, tiger/CPU (Figs. 1 and 2, raw data in Supporting Information Tables S-1 and S-2). For the trivial TF and vW KEDFs, none of the GPUs is faster than the OpenMP reference. The difference is typically below 0.2 s; only the NVIDIA K20m is around 0.3–0.5 s/call slower than the others. Given the negligible timing, this result is not a concern and most likely indicative of the cost of memory transfer to and from the GPU compared to the cost of actual computation (at full PCIe2.0 x16 speeds, pure hardware transfer cost will amount to 0.054 s for transfer of a single grid of system II plus additional software-based overhead caused by the OpenCL stack and operating system). In the case of the hyperion/Nano platform, we observe that a regular desktop GPU performs competitively with a full compute node even in DP, opening interesting avenues for libKEDF on regular desktop computers. We attribute this mainly to the different memory type accessibly to the R9 Nano compared to any of the other accelerators: high-bandwidth memory (HBM). Memory-intensive operations are accelerated and make the hyperion/Nano platform competitive with the much bigger mcmillan/K40 and hyperion/S9150 for the simple KEDFs. All KEDF kernels scale at most

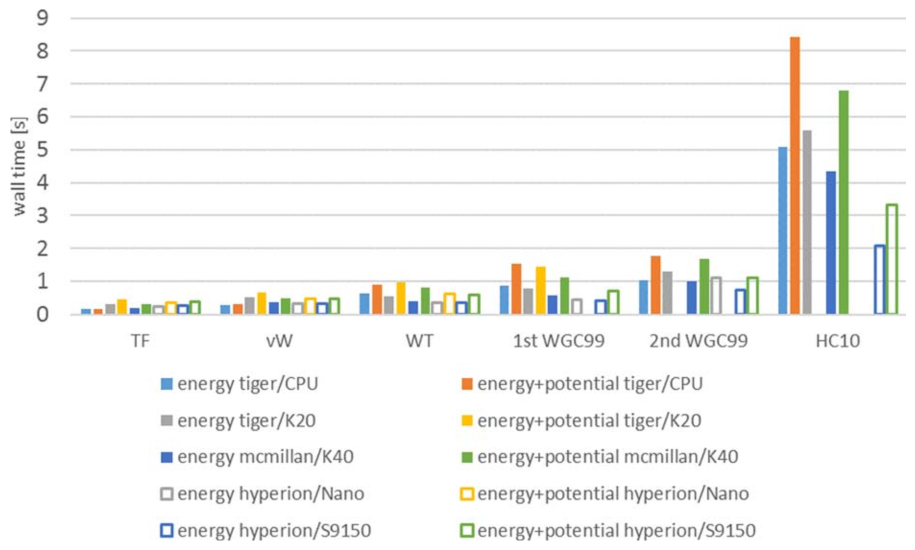

Figure 1. Wall time performance of libKEDF for implemented KEDFs to evaluate energy and energy + potential of test system I. Wall times for nonlocal terms exclude the TF and VW terms. [Color figure can be viewed at wileyonlinelibrary.com]

quasi-linearly in computational operations with the number of memory access operations, making them intrinsically memory-bound.

More interesting, then, is the performance for the much more expensive nonlocal terms. For the WT nonlocal term, the tiger/K20 platform is slightly slower than tiger/CPU for energy + potential evaluations and slightly faster for energy-only evaluations for both test systems. mcmillan/K40 is consistently faster than tiger/CPU but remains slower by up to 0.4 s than both hyperion/Nano and hyperion/S9150. A full WT KEDF energy + potential evaluation on these two AMD-accelerated test systems would then be within ±0.1 s of the tiger/CPU platforms. The tiger/K20 platform evaluates all terms in 2.8 s for test system II compared to 1.6 s on tiger/CPU, while the mcmillan/K40 platform takes 2.1 s for the same task.

For the Taylor-expanded WGC99 nonlocal terms, we are limited by the accelerator memory for the tiger/K20 and hyperion/Nano platforms. This is only a very temporary limitation as available GPU/accelerator memory will increase in upcoming hardware generations. We find that the hyperion/S9150 is faster than the CPU reference and is able to evaluate a full second-order WGC99 KEDF energy + potential of system II in 2.29 s, compared to 2.76 s for the CPU reference. The hyperion/Nano platform is approximately as fast as a full compute node for the problems fitting its on-device memory.

As discussed earlier, the HC10 nonlocal term has been prohibitively expensive in the past for medium-to-large systems like our test cases. Our OpenCL implementation is straightforward with no changes compared to the OpenMP CPU implementation except for using the regular lookup table for the

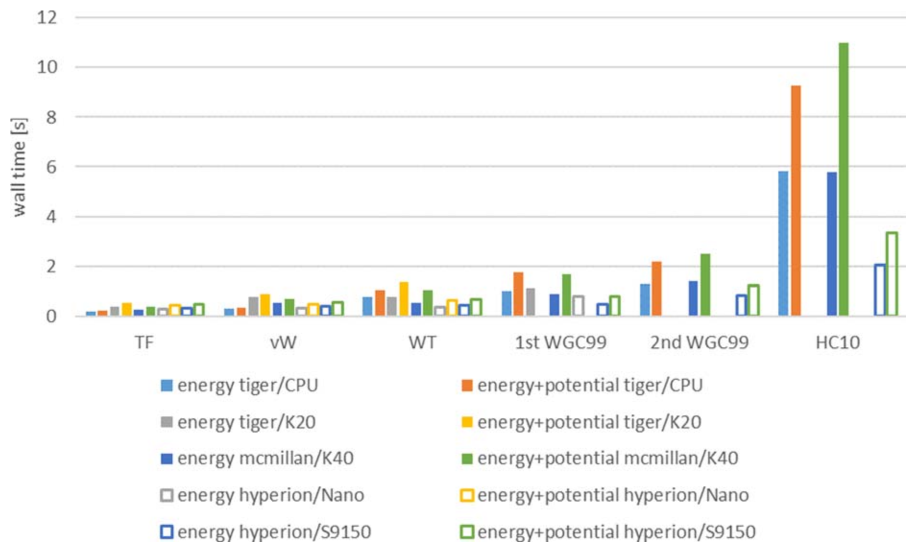

Figure 2. Wall time performance of libKEDF for implemented KEDFs to evaluate energy and energy + potential of test system II. Wall times for nonlocal terms exclude the TF and VW terms. [Color figure can be viewed at wileyonlinelibrary.com]

HC10 ODE solution and HC10 integral kernel assembly. The high memory requirement of this term makes it currently only suitable for the accelerators with more on-device memory. However, we are then able to use the hyperion/S9150 platform to gain a speedup of 2.4 to 3.6x over the CPU reference. In total, a full energy + potential with the HC10 KEDF can then be evaluated in 4.2 s compared to 8.9 s on the already fast CPU implementation for system I.

Our OpenCL KEDF/kernel implementations are in general competitive to a 16-core compute node for all tested accelerators. We find the hyperion/S9150 platform to typically outperform the reference 16-core compute node and, in the case of the HC10 term, quite significantly so. The accelerated KEDF kernels of libKEDF can even realize on a single desktop GPU in DP a performance such as that of a 16-core compute node for problems fitting into device memory.

Summary and Outlook

We have presented libKEDF, a modern implementation of KEDFs under free licensing. It is standards-compliant and portable, providing very good computational efficiency for large systems. Shared-memory parallelization makes efficient use of typical compute nodes, while OpenCL-enabled KEDF implementations work well on a range of accelerators from typical desktop cards such as the AMD R9 Nano to compute accelerators like the NVIDIA K40c and the AMD S9150. We observe that vW and TF KEDFs are slightly faster on a 16-core CPU node. Nonlocal terms can be substantially faster on some accelerators and roughly equally fast or a bit slower on others, assuming the problem fits into the available memory of the GPU/accelerator. The R9 Nano as a desktop card shows surprisingly strong results, which we attribute to its HBM architecture. This will open new avenues for the application of KEDFs since a single compute node or a desktop with a good graphics card or accelerator will suffice to compute medium-sized problems with great speed.

We are currently integrating libKEDF with our OFDFT code, PROFESS,[2,45,47] where it will eventually replace the legacy KEDF implementations present there. We will then use libKEDF in our production calculations for single-point and MD simulations. Subsequently, we will provide distributed-memory support for libKEDF together with implementations of stresses for the KEDFs.

We encourage other KEDF developers to contribute reference implementations of their KEDFs and scientists using KEDFs in their research and codes to adopt libKEDF.

Acknowledgments

The authors thank the Computational Science and Engineering Support (CSES) at Princeton University for maintaining the Terascale Infrastructure for Groundbreaking Research in Science and Engineering (TIGRESS) infrastructure used for part of the computations. The authors thank Ms. Nari Baughman for her careful editing of this manuscript and Dr. Xing Zhang for critical review. JMD wishes to extend his gratitude to Gregory Stoner, Nydia Nunez, and Dipak Bhattacharyya from AMD for providing the AMD R9 Nano and AMD FirePro S9150 used in this work. JMD wishes to thank the other members of the FreeBSD graphics and accelerators group (in particular Matthew Macy, Jean-Sébastien Pédron, Ed Maste, and Koop Mast) for their voluntary commitment to enable modern GPUs and HPC on FreeBSD. WCW acknowledges: this material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. 1656466. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Keywords: kinetic energy density functional · library · GPU · orbital-free density functional theory

How to cite this article: J. M. Dieterich, W. C. Witt, E. A. Carter. *J. Comput. Chem.* **2017**, *38*, 1552–1559. DOI: 10.1002/jcc.24806

Additional Supporting Information may be found in the online version of this article.

[1] Y. A. Wang, E. A. Carter, In Theoretical Methods in Condensed Phase Chemistry; Kluwer: Dordrecht, **2000**; pp. 117–184.
[2] G. S. Ho, V. L. Lignères, E. A. Carter, *Comput. Phys. Commun.* **2008**, *179*, 839.
[3] P. Cortona, *Phys. Rev. B* **1991**, *44*, 8454.
[4] P. Cortona, *Phys. Rev. B* **1992**, *46*, 2008.
[5] T. A. Wesolowski, A. Warshel, *J. Chem. Phys.* **1993**, *97*, 8050.
[6] P. Huang, E. A. Carter, *Annu. Rev. Phys. Chem.* **2008**, *59*, 261.
[7] C. R. Jacob, J. Neugebauer, *Wiley Interdiscip. Rev. Comput. Mol. Sci.* **2014**, *4*, 325.
[8] A. Krishtal, D. Sinha, A. Genova, M. Pavanello, *J. Phys. Condens. Matter* **2015**, *27*, 183202.
[9] G. S. Ho, C. Huang, E. A. Carter, *Curr. Opin. Solid State Mater. Sci.* **2007**, *11*, 57.
[10] A. Aguado, D. J. González, L. E. González, J. M. López, S. Núnez, M. J. Stott, In Recent Progress in Orbital-free Density Functional Theory; World Scientific: Singapore, **2013**; p. 55.
[11] C. Huang, E. A. Carter, *Physi. Rev. B* **2010**, *81*, 045206.
[12] J. Xia, E. A. Carter, *Phys. Rev. B* **2012**, *86*, 235109.
[13] I. Shin, E. A. Carter, *J. Chem. Phys.* **2014**, *104*, 18A531.
[14] T. A. Wesolowski, Y. A. Wang, Eds. Recent Progress in Orbital-free Density Functional Theory; World Scientific: Singapore, **2013**.
[15] V. V. Karasiev, D. Chakraborty, S. B. Trickey, In Many-Electron Approaches in Physics, Chemistry and Mathematics; Springer International Publishing: Cham, **2014**; p. 113.
[16] C. Huang, E. A. Carter, *Phys. Rev. B* **2012**, *85*, 045126.
[17] Y. Ke, F. Libisch, J. Xia, L.-W. Wang, E. A. Carter, *Phys. Rev. Lett.* **2013**, *111*, 066402.
[18] Y. Ke, F. Libisch, J. Xia, E. A. Carter, *Phys. Rev. B* **2014**, *89*, 155112.
[19] M. A. L. Marques, M. J. T. Oliveira, T. Burnus, *Comput. Phys. Commun.* **2012**, *183*, 2272.
[20] libKEDF github. libKEDF: An Accelerated Library of Kinetic Energy Density Functionals. Available at: https://github.com/EACcodes/libKEDF. [accessed on December 20, **2016**].
[21] M. Chen, L. Hung, C. Huang, J. Xia, E. A. Carter, *Mol. Phys.* **2013**, *111*, 3448.
[22] M. Chen, J. R. Vella, F. H. Stillinger, E. A. Carter, A. Z. Panagiotopoulos, P. G. Debenedetti, *AIChE J.* **2015**, *61*, 2841.
[23] M. Chen, X. Jiang, H. Zhuang, L. Wang, E. A. Carter, *J. Chem. Theory Comput.* **2016**, *12*, 2950.
[24] E. V. Ludeña, V. V. Karasiev, In Reviews of Modern Quantum Chemistry; World Scientific, **2002**; pp. 612–665.

[25] D. García-Aldea, J. Alvarellos, In Theoretical and Computational Developments in Modern Density Functional Theory; Nova Science Publishers, Inc.: Hauppauge, NY, **2012**; pp. 255.

[26] E. Chacón, J. Alvarellos, P. Tarazona, *Phys. Rev. B* **1985**, *32*, 7868.

[27] L.-W. Wang, M. P. Teter, *Phys. Rev. B* **1992**, *45*, 13196.

[28] D. Neuhauser, S. Pistinner, A. Coomar, X. Zhang, G. Lu, *J. Chem. Phys.* **2011**, *134*, 144101.

[29] T. Sjostrom, J. Daligault, *Phys. Rev. Lett.* **2014**, *113*, 155006.

[30] L. Thomas, *Math. Proc. Cambridge Philos. Soc.* **1927**, *23*, 542.

[31] E. Fermi, *Rend. Accad. Naz. Lincei.* **1927**, *6*, 602.

[32] C. von Weizsäcker, *Z. Phys.* **1935**, *96*, 431.

[33] Y. A. Wang, N. Govind, E. A. Carter, *Phys. Rev. B* **1999**, *60*, 16350.

[34] Y. A. Wang, N. Govind, E. A. Carter, *Phys. Rev. B* **2001**, *64*, 089903.

[35] OpenMP, The OpenMP API Specification for Parallel Programming. Available at: http://www.openmp.org/ [accessed on Decemeber 09, **2016**].

[36] M. Frigo, S. G. Johnson, *Proc. IEEE* **2005**, *93*, 216.

[37] Khronos Group, The Open Standard for Parallel Programming of Heterogeneous Systems. Available at: https://www.khronos.org/opencl/ [accessed on December 09, **2016**].

[38] clMathLibraries, clFFT: a Software Library Containing FFT Functions Written in OpenCL. Available at: https://github.com/clMathLibraries/clFFT [accessed on December 09, **2016**].

[39] NVIDIA Corporation, CUDA Parallel Computing Platform. Available at: https://www.nvidia.com/object/cuda_home_new.html [accessed on December 09, **2016**].

[40] LLVM Project, The LLVM Compiler Infrastructure. Available at: http://llvm.org/ [accessed on December 09, **2016**].

[41] Free Software Foundation, Inc., GCC, the GNU Compiler Collection. Available at: https://gcc.gnu.org/ [accessed on December 09, **2016**].

[42] The FreeBSD Project, FreeBSD The Power To Serve. Available at: https://www.freebsd.org/ [accessed on Decemeber 09, **2016**].

[43] Linux Kernel Organization, Inc., The Linux Kernel Archives. Available at: https://www.kernel.org/ [accessed on Decemeber 09, **2016**].

[44] J. Perdew, A. Zunger, *Phys. Rev. B* **1981**, *23*, 5048.

[45] M. Chen, J. Xia, C. Huang, J. M. Dieterich, L. Hung, I. Shin, E. A. Carter, *Comput. Phys. Commun.* **2015**, *190*, 228.

[46] B. Zhou, Y. A. Wang, E. A. Carter, *Phys. Rev. B* **2004**, *69*, 125109.

[47] L. Hung, C. Huang, I. Shin, G. S. Ho, V. L. Lignères, E. A. Carter, *Comput. Phys. Commun.* **2010**, *181*, 2208.

Received: 17 February 2017
Revised: 22 March 2017
Accepted: 24 March 2017
Published online on 20 April 2017