DFToy:
A new proxy app for DFT calculations

Arjen Tamerus [1] Phil Hasnip [2]

[1]University of Cambridge [2]University of York

Plane-wave DFT: current barriers

With the changes that the Exascale era has brought to HPC architectures, codes need to adapt to fully utilise the hardware. In CASTEP specifically, we have identified the following issues:

- High communication overhead: a significant portion of calculation time is spent transforming between real and reciprocal space - a global, all-to-all Fourier transform.
- Unfavourable memory access patterns: non-coalesced memory access and indirection.
- Reliance on double precision: increases memory pressure, and limits GPU compute to datacentre-level chips.
- High memory allocation/transfer/bandwidth pressure - limits effective use of accelerators and stresses interconnects.

In future work, DFToy will be a useful platform to experiment with alternative, novel algorithms to work around these issues - e.g. using mixed-precision approaches, more efficient FFT algorithms, and realspace-only algorithms to avoid having to map between real and reciprocal space.

But why?

Why do we *need* a DFT proxy app? There are several issues with traditional DFT applications that make them less-than-ideal for benchmarking or novel development:

- They tend to be **highly complex**, comprising 500k+ lines of code written by multiple contributors over multiple decades - lots of legacy, rigid structures.
- Fortran is often the language of choice. Fortran is great, but **inflexible** - writing e.g. efficient GPU code can be a pain.
- Licensing can be an issue - it's often **non-free** even if the code may be free-as-in-beer for academics.
- **Expertise** is required to create, tweak and optimise a benchmark for a new system.

DFToy offers a simplified, free-as-in-beer-and-speech, no-knowledge-required benchmarking platform - just compile and run. It's modular design simplifies the implementation of novel algorithms or linking to alternative libraries.

Resources

Interested? Scan the QR code below for the repository and other materials.

CAMBRIDGE OPEN ZETTASCALE LAB

This work is part of the Cambridge Open Zettascale Lab.

The DFT Scaling crisis

Materials modelling codes are some of the largest consumers of core hours on many TOP500-ranked supercomputers. Codes like VASP, CASTEP, Quantum Espresso and others frequently top the rankings when it comes to core hours consumed on the UK's national supercomputer ARCHER2[1], and many other systems show similar statistics.

These density functional theory based codes have applications in material discovery, the development of new battery materials, medical molecules, superconductor research and many more fields beside those. There is therefore no reason to expect the popularity and resource usage of these code to diminish anytime soon - if anything, the opposite is more likely[3].

Unfortunately, traditional DFT codes have some computational characteristics that limit their scalability to new Exascale or even large Petascale computing clusters. A further complicating factor is that these codes often have had decades of development on classical HPC infrastructures - making them robust, but also highly complex and hard to adapt to modern architectures.

Introducing DFToy

DFToy is:

1. An easy-to-run, simple-to-understand benchmark.
2. A small, modular, easy-to-extend platform for developing novel algorithms.
3. A model for the parallel scaling behaviour of plane-wave DFT codes on various architectures.

DFToy was conceived as a more effective way to develop and experiment with DFT algorithms. By stripping away the legacy and complexities of established codes and isolating the core computational load into a small code, we create a much more accessible platform.

In its current state, DFToy implements a full DFT-alike calculation using an iterative conjugate-gradient solver. It captures the most significant computational characteristics - the all-to-all communications of the FFTs used for realspace-to-reciprocal-space conversion and vice versa, and the $\mathcal{O}(n^3)$ scaling dense matrix multiplications.

Future work includes:

- Assessing DFToy's use as a procurement and performance benchmark.
- Porting the code to accelerators and implementing novel, better-scaling algorithms.
- Creating a parallel behaviour model of DFToy that auto-tunes the code for optimal performance.
- Taking our findings from DFToy experiments and porting them to a real DFT code (CASTEP).

References

[1] Archer2 usage statistics - https://www.archer2.ac.uk/support-access/status.html#usage-statistics.

[2] Dftoy - https://github.com/arjentamerus/dftoy.

[3] Leopold talirz, edoardo aprà, jonathan e. moussa, & samuel poncé. (2023). Italirz/atomistic-software: v2023.1.28 (v2023.1.28). zenodo. https://doi.org/10.5281/zenodo.7578861.

This work was performed using resources provided by the Cambridge Service for Data Driven Discovery (CSD3) operated by the University of Cambridge Research Computing Service (www.csd3.cam.ac.uk), provided by Dell EMC and Intel using Tier-2 funding from the Engineering and Physical Sciences Research Council (capital grant EP/T022159/1), and DiRAC funding from the Science and Technology Facilities Council (www.dirac.ac.uk).

The simple maths

DFToy's main computational load is finding the minimum energy of a fake 'atom' of "nonexistium". This is done by constructing the Hamiltonian matrix $\hat{H}$ from the semi-arbitrarily defined Kinetic energy and local and non-local potentials:

$$\hat{H} = \hat{T} + \hat{V}_{loc} + \hat{V}_{nl}$$

As DFToy is not concerned about real-world materials - we just want to model the scaling behaviour - we can significantly simplify the Hamiltonian's components. Each of the Hamiltonian's components is implemented as an $NxNxN$ matrix, representing a sampled force field with radius $r = 1$ and sampling resolution s, giving us $N = 2s + 1$ accounting for a zero point. We can change the problem size by increasing the sampling resolution, i.e. the number of wavevectors w in our calculation ($w = (2s + 1)^3$).

We find our solution by finding the lowest M energy states - making this effectively an Eigensolver implemented with an iterative CG algorithm.

Comparative benchmarks

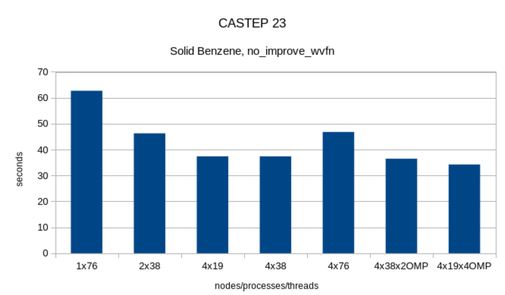

Figure 1. CASTEP scaling.

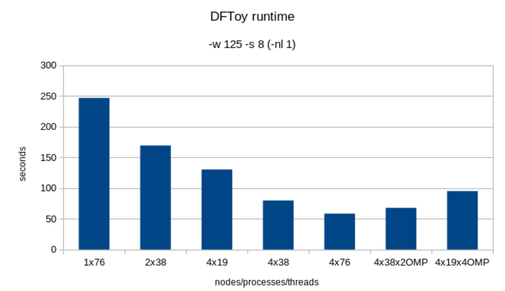

Figure 2. DFToy scaling.