A Systematic Approach to Improving Data Locality Across Fourier Transforms and Linear Algebra Operations

Doru Thom Popovici, Andrew Canning, Zhengji Zhao, Lin-Wang Wang, John Shalf
Lawrence Berkeley National Lab
USA

报告人：冯国峰

日　期：2021/6/23

Vision

The performance of most scientific applications depends on efficient mathematical libraries.

Ideally, computational libraries and frameworks should offer developers two key benefits. First, they should provide interfaces to capture complicated algorithms that are otherwise difficult to write by hand. Second, they should automatically map algorithms to efficient implementations that perform at least as well as expert hand-optimized code.

Vision

Hard ware is changing:

CPU->many-core cpu->GPU->specialized hardware

Coordinate among the computing cores:

break the barriers between library calls and come up with a common representation that exposes the dataflow of computation and access patterns

Problem(abstract)

First, they should provide interfaces to capture complicated algorithms that are otherwise difficult to write by hand. (mostly achieved)

Second, they should automatically map algorithms to efficient implementations that perform at least as well as expert hand-optimized code. (mostly fall short on)

Problem(abstract)

Reason: Due to the complexity, users typically resort to implementations for the problem, in the form of black box(so we can’t use inter-procedure optimization) library calls to high performance libraries. The cost of this approach is that neither a compiler nor an expert can optimize across the various stages.

Problem(abstract)

Compilers: Principles, Techniques & Tools

Intraprocedural optimization: Most compiler optimizations, are performed on procedures one at a time.

Interprocedural optimization: operates across an entire program.

Related work

Various Vendor-tuned libraries:

FFTW, MKL, BLAS, cuFFT, cuBLAS

Problem(specific)

Density Functional Theory Calculations take up to approximately more than 25% of the computational cycle, are the most important scientific application on modern supercomputers.

Requires a mix of mathematical operations.

Problem(specific)

$$\hat{H}\psi_i(r) = \left[-\frac{1}{2}\nabla^2 + V(r) + \sum_l |\phi_l >< \phi_l| \right] \psi_i(r) = \epsilon_i \psi_i(r), \quad (1)$$

Most time is spent on

FFT, iFFT and H times

psi.

How to make it faster?

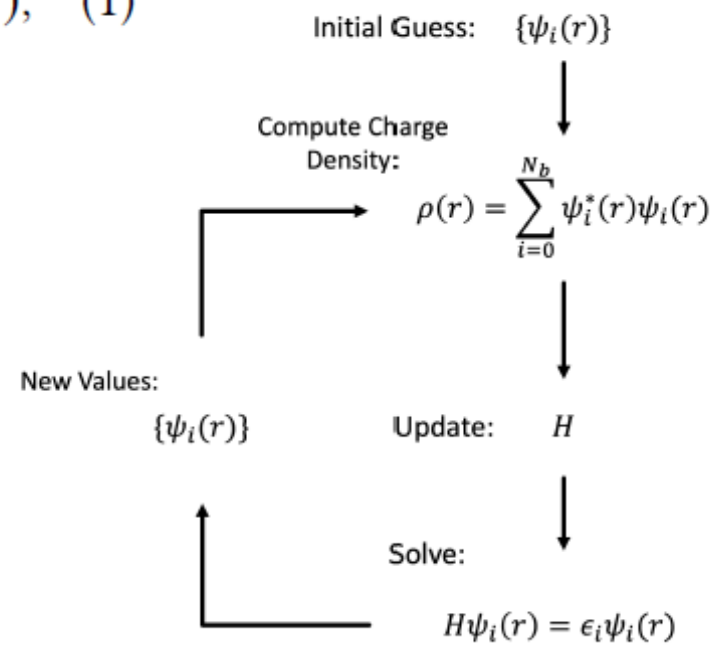

Figure 1: Schematic depiction of the self consistent field (SCF) method implemented in electronic structure codes.

Problem(specific)

$$\hat{H}\psi_i(r) = \left[-\frac{1}{2}\nabla^2 + V(r) + \sum_l |\phi_l >< \phi_l| \right] \psi_i(r) = \epsilon_i \psi_i(r), \quad (1)$$

Most time is spent on FFT, iFFT and H times psi.

How to make it faster?

Idea 1: new theories/algorithm

eg: ISDF, ACE, …

Idea 2： more efficient implementation.(eg: Parallelization, data access pattern…)

Problem(specific)

$$\hat{H}\psi_i(r) = \left[-\frac{1}{2}\nabla^2 + V(r) + \sum_l |\phi_l >< \phi_l| \right] \psi_i(r) = \epsilon_i \psi_i(r), \quad (1)$$

Storing the full matrix H requires large amount of memory and the computation will be extremely expensive.

The expensive matrix computation is replaced with memory bound operations (the performance is limited by the data movement).

Solution

Memory bound problems:

Idea: improve data locality

Method: merge memory bound operations

Requirement: use common notations to express

Solution

Systematic Way of Merging the DFT and non-DFT Stages:

Most DFT-based operations follow a special pattern, namely a forward DFT is applied on the input data, followed by a point-wise multiplication and an inverse DFT.

Applying optimizations across the multiple stages is enabled by the ability to represent the DFT and the additional computation with the same high level mathematical representation.

Solution

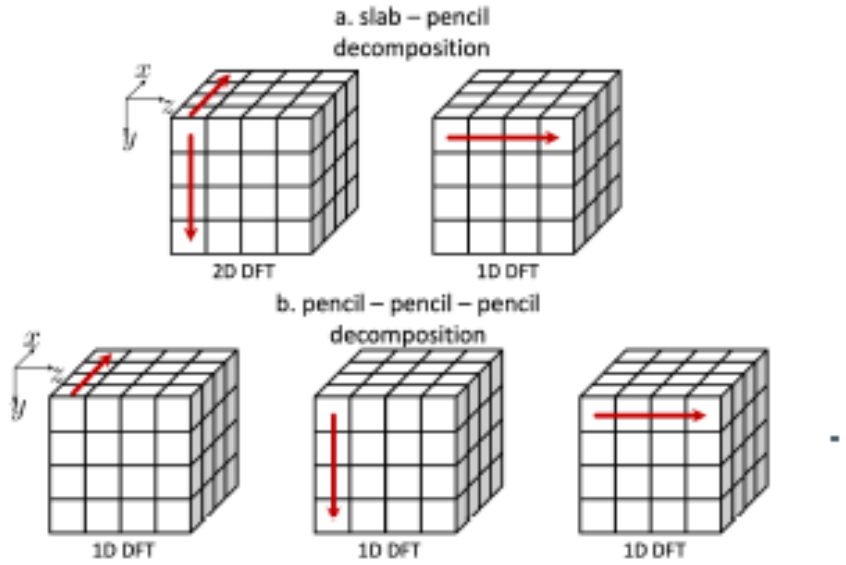

Figure 4: Algorithms for computing the 3D DFT: (a) the slab-pencil algorithm decomposes the 3D DFT into 2D and 1D DFTs. (b) the pencil-pencil-pencil algorithm decomposes the DFT into three 1D DFTs in the corresponding dimensions.

Algorithm 1: Computing $H\Psi$ as outlined in Equations 1 and 2.

Input:

$\Psi^{(b)(\nu_0\nu_1\nu_2)}$ - input planewaves

$\nabla^{(\nu_0\nu_1\nu_2)}$ - kinetic energy

$\Phi^{(lr)(\mu_0\mu_1\mu_2)}$ - non - local projectors

$V^{(\mu_0\mu_1\mu_2)}$ - ionic potential

Output:

$\Psi^{(b)(\nu_0\nu_1\nu_2)}$ - output planewaves

begin

$T_0^{(b)(\mu'_0\mu'_1\mu'_2)} = \Psi^{(b)(\nu_0\nu_1\nu_2)} \cdot P^{(\nu_0\nu_1\nu_2)(\mu'_0\mu'_1\mu'_2)}$

$T_1^{(b)(\mu_0\mu_1\mu_2)} = T_0^{(b)(\mu'_0\mu'_1\mu'_2)} \cdot iDFT_{3D}^{(\mu'_0\mu'_1\mu'_2)(\mu_0\mu_1\mu_2)}$

$T_2^{(b)(lr)} = T_1^{(b)(\mu_0\mu_1\mu_2)} \cdot \left(\Phi^{(lr)(\mu_0\mu_1\mu_2)}\right)^T$

$T_1^{(b)(\mu_0\mu_1\mu_2)} = T_1^{(b)(\mu_0\mu_1\mu_2)} \odot V^{(\mu_0\mu_1\mu_2)}$

$T_1^{(b)(\mu_0\mu_1\mu_2)} += T_2^{(b)(lr)} \cdot \Phi^{(lr)(\mu_0\mu_1\mu_2)}$

$T_0^{(b)(\mu'_0\mu'_1\mu'_2)} = T_1^{(b)(\mu_0\mu_1\mu_2)} \cdot DFT_{3D}^{(\mu_0\mu_1\mu_2)(\mu'_0\mu'_1\mu'_2)}$

$\Psi^{(b)(\nu_0\nu_1\nu_2)} = \Psi^{(b)(\nu_0\nu_1\nu_2)} \odot \nabla^{(\nu_0\nu_1\nu_2)}$

$\Psi^{(b)(\nu_0\nu_1\nu_2)} += T_0^{(b)(\mu'_0\mu'_1\mu'_2)} \cdot U^{(\mu'_0\mu'_1\mu'_2)(\nu_0\nu_1\nu_2)}$

end

Solution

Merging the Packing with the 3D DFT:

$$T_0^{(b)(\mu_0'\mu_1'\mu_2')} = \Psi^{(b)(\nu_0\nu_1\nu_2)} \cdot P^{(\nu_0\nu_1\nu_2)(\mu_0'\mu_1'\mu_2')}$$
$$T_1^{(b)(\mu_0\mu_1\mu_2)} = T_0^{(b)(\mu_0'\mu_1'\mu_2')} \cdot iDFT_{3D}^{(\mu_0'\mu_1'\mu_2')(\mu_0\mu_1\mu_2)}$$

$$T_0^{(b)(\mu_0'\mu_1'\mu_2')} = \Psi^{(b)(\nu_0\nu_1\nu_2)} \cdot P^{(\nu_0\nu_1\nu_2)(\mu_0'\mu_1'\mu_2')}$$
$$T_{00}^{(b)(\mu_0\mu_1'\mu_2')} = T_0^{(b)(\mu_0'\mu_1'\mu_2')} \cdot iDFT_{1D}^{\mu_0'\mu_0}$$
$$T_{01}^{(b)(\mu_0\mu_1\mu_2')} = T_{00}^{(b)(\mu_0\mu_1'\mu_2')} \cdot iDFT_{1D}^{\mu_1'\mu_1}$$
$$T_1^{(b)(\mu_0\mu_1\mu_2)} = T_{01}^{(b)(\mu_0\mu_1\mu_2')} \cdot iDFT_{1D}^{\mu_2'\mu_2},$$

$$T_{02}^{(b)(\mu_0\nu_1\nu_2)} = \Psi^{(b)(\nu_0\nu_1\nu_2)} \cdot \left(P^{\nu_0\mu_0'} \cdot iDFT_{1D}^{\mu_0'\mu_0}\right)$$
$$T_{03}^{(b)(\mu_0\mu_1\nu_2)} = T_{02}^{(b)(\mu_0\nu_1\nu_2)} \cdot \left(P^{\nu_1\mu_1'} \cdot iDFT_{1D}^{\mu_1'\mu_1}\right)$$
$$T_1^{(b)(\mu_0\mu_1\mu_2)} = T_{03}^{(b)(\mu_0\mu_1\nu_2)} \cdot \left(P^{\nu_2\mu_2'} \cdot iDFT_{1D}^{\mu_2'\mu_2}\right),$$

Solution

Merging the Packing with the 3D DFT:

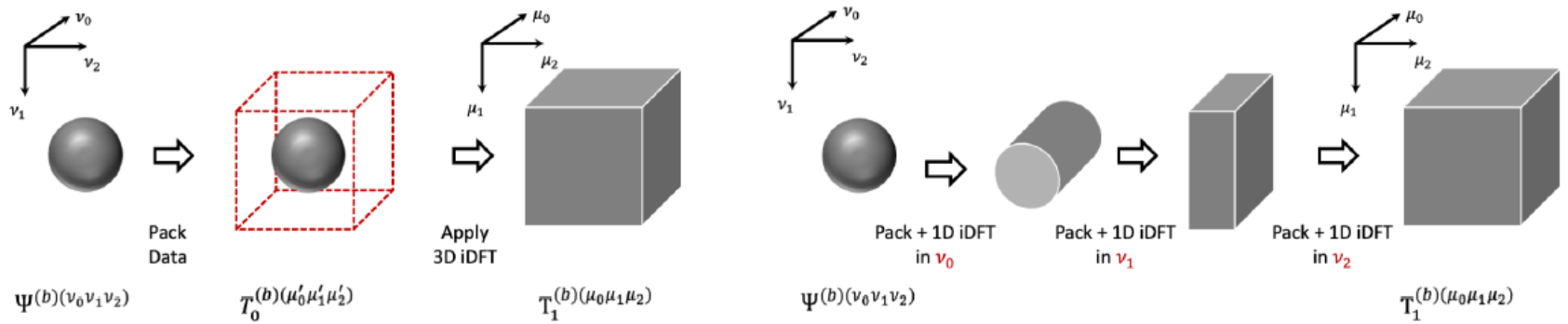

Figure 5: The Fourier computation requires each wavefunction to be padded with zeroes. Version (a) does the full packing and then applies the 3D DFT on the padded cube. Version (b) splits the padding and the 3D DFT and interleaves the operations.

Solution

Merging the DFT with the Linear Algebra:

$$
\begin{aligned}
T_1^{(b)(\mu_0\mu_1\mu_2)} &= T_{03}^{(b)(\mu_0\mu_1\nu_2)} \cdot \left(P^{\nu_2\mu_2'} \cdot iDFT_{1D}^{\mu_2'\mu_2} \right) \\
T_2^{(b)(lr)} &= T_1^{(b)(\mu_0\mu_1\mu_2)} \cdot \Phi^{(lr)(\mu_0\mu_1\mu_2)} \\
T_1^{(b)(\mu_0\mu_1\mu_2)} &= T_1^{(b)(\mu_0\mu_1\mu_2)} \odot V^{(\mu_0\mu_1\mu_2)} \\
T_1^{(b)(\mu_0\mu_1\mu_2)} &+= T_2^{(b)(lr)} \cdot \Phi^{(lr)(\mu_0\mu_1\mu_2)} \\
T_{03}^{(b)(\mu_0\mu_1\nu_2)} &= T_1^{(b)(\mu_0\mu_1\mu_2)} \cdot \left(DFT_{1D}^{\mu_2\mu_2'} \cdot U^{\mu_2'\nu_2} \right).
\end{aligned}
$$

The last stage of the padded 3D DFT, the real space computation and the first stage of the unpadded DFT

Solution

Merging the DFT with the Linear Algebra:

There are extra operations between the DFT stages, namely the projection of the $T1$ tensor on the non-local projectors stored in Φ, and the scaling of the projectors in Φ by the $T2$ tensor and the update to $T1$. $T2$ must be fully computed before the scaling of the projectors, which makes merging all stages hard.

A solution is to create two groups of operations.

Solution

Merging the DFT with the Linear Algebra:

A solution is to create two groups of operations.

The first group of operations:

$$T_1^{(b)(\mu_0\mu_1\mu_2)} = T_{03}^{(b)(\mu_0\mu_1\nu_2)} \cdot \left(P^{\nu_2\mu_2'} \cdot iDFT_{1D}^{\mu_2'\mu_2} \right)$$

$$T_2^{(b)(lr)} = T_1^{(b)(\mu_0\mu_1\mu_2)} \cdot \left(\Phi^{(lr)(\mu_0\mu_1\mu_2)} \right)^T .$$

Solution

Merging the DFT with the Linear Algebra:

$$T_1^{(b)(\mu_0\mu_1\mu_2)} = T_{03}^{(b)(\mu_0\mu_1\nu_2)} \cdot \left(P^{\nu_2\mu_2'} \cdot iDFT_{1D}^{\mu_2'\mu_2}\right)$$

$$T_2^{(b)(lr)} = T_1^{(b)(\mu_0\mu_1\mu_2)} \cdot \left(\Phi^{(lr)(\mu_0\mu_1\mu_2)}\right)^T.$$

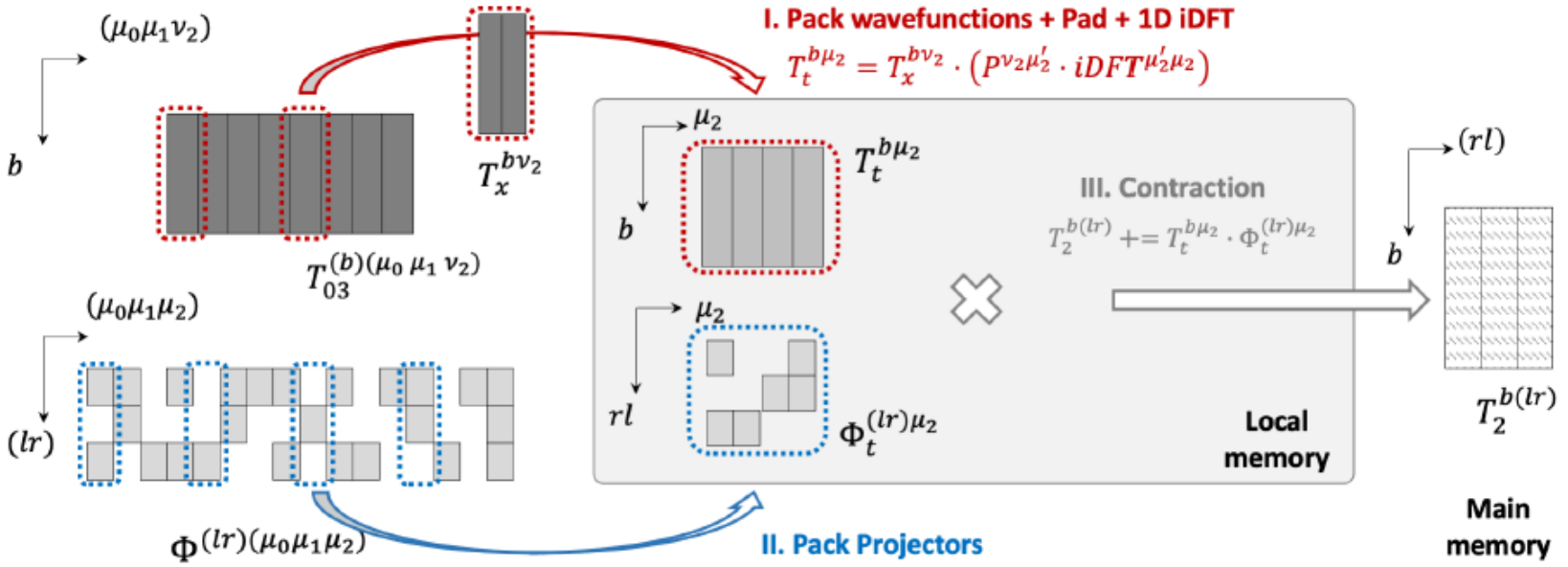

Figure 6: Merging the padded 1D iDFT with the projection of the Fourier transformed data points on the non-local projector $\Phi^{(lr)(\mu_0\mu_1\mu_2)}$. The tensors are depicted as matrices using a column major layout. The padded 1D iDFT reads strided batches of data given the size of the dimensions $(\mu_0\mu_1)$, applies zero-padding followed by the batched 1D iDFT (I). The same stride is used to read the corresponding projector data points (II). The contraction is performed and partial results update the tensor T_2 (III).

Solution

Merging the DFT with the Linear Algebra:

A solution is to create two groups of operations.

The second group of operations:

$$T_1^{(b)(\mu_0\mu_1\mu_2)} = T_{03}^{(b)(\mu_0\mu_1\nu_2)} \cdot \left(P^{\nu_2\mu_2'} \cdot iDFT_{1D}^{\mu_2'\mu_2}\right) \tag{19}$$

$$T_1^{(b)(\mu_0\mu_1\mu_2)} = T_1^{(b)(\mu_0\mu_1\mu_2)} \odot V^{(\mu_0\mu_1\mu_2)}$$

$$T_1^{(b)(\mu_0\mu_1\mu_2)} += T_2^{(b)(lr)} \cdot \Phi^{(lr)(\mu_0\mu_1\mu_2)}$$

$$T_{03}^{(b)(\mu_0\mu_1\nu_2)} = T_1^{(b)(\mu_0\mu_1\mu_2)} \cdot \left(DFT_{1D}^{\mu_2\mu_2'} \cdot U^{\mu_2'\nu_2}\right).$$

Solution

Merging the DFT with the Linear Algebra:

$$T_1^{(b)(\mu_0\mu_1\mu_2)} = T_{03}^{(b)(\mu_0\mu_1\nu_2)} \cdot \left(P^{\nu_2\mu_2'} \cdot iDFT_{1D}^{\mu_2'\mu_2}\right) \quad (19)$$

$$T_1^{(b)(\mu_0\mu_1\mu_2)} = T_1^{(b)(\mu_0\mu_1\mu_2)} \odot V^{(\mu_0\mu_1\mu_2)}$$

$$T_1^{(b)(\mu_0\mu_1\mu_2)} += T_2^{(b)(lr)} \cdot \Phi^{(lr)(\mu_0\mu_1\mu_2)}$$

$$T_{03}^{(b)(\mu_0\mu_1\nu_2)} = T_1^{(b)(\mu_0\mu_1\mu_2)} \cdot \left(DFT_{1D}^{\mu_2\mu_2'} \cdot U^{\mu_2'\nu_2}\right).$$

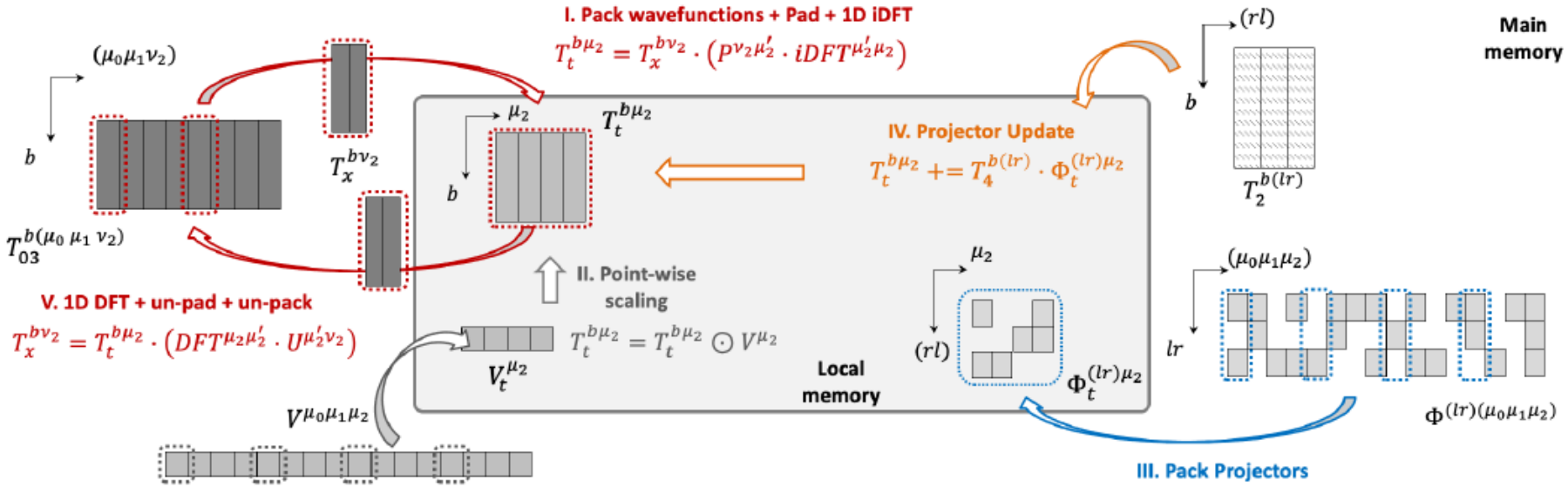

Figure 7: Merging the inverse and forward 1D DFTs with the point-wise computation with V and the update of the scaled projector Φ. Batches of the data are read, padded and Fourier transformed (I). The point-wise scaling with the ionic potentials is then performed (II). The corresponding components of the projectors are packed in local memory (III). The projectors are scaled by the data stored in T_2 and the values added to the local buffer (IV). Once all updates are performed, the data points are Fourier transformed and only the relevant data points are stored back into the same locations as the input batches (V).

Solution

Implementation:

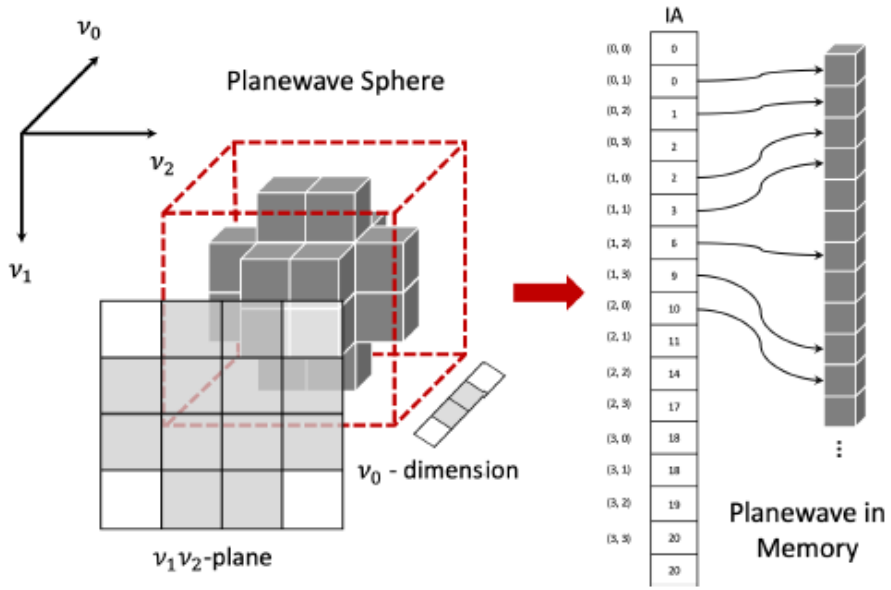

Figure 8: The CSR-like format to store the planewaves. The sphere is projected on the $\nu_1\nu_2$ plane, and the plane is flattened and used as an offset indexing array.

Each mathematical kernel is implemented from scratch, for both the CPU and the GPU

Analysis and Results

We run all codes on the following architectures

- AMD Ryzen 7 3800X CPU 8/16/32MB cores/threads/L3 cache with 64 GB of DRAM
- AMD EPYC 7302 CPU 2/16/32/128MB sockets/cores/threads/L3 cache with 2 TB of DRAM
- NVIDIA V100 GPU - 80 streaming multi-processors with 16 GB of DRAM
- NVIDIA A100 GPU - 108 streaming multi-processors with 40 GB of DRAM

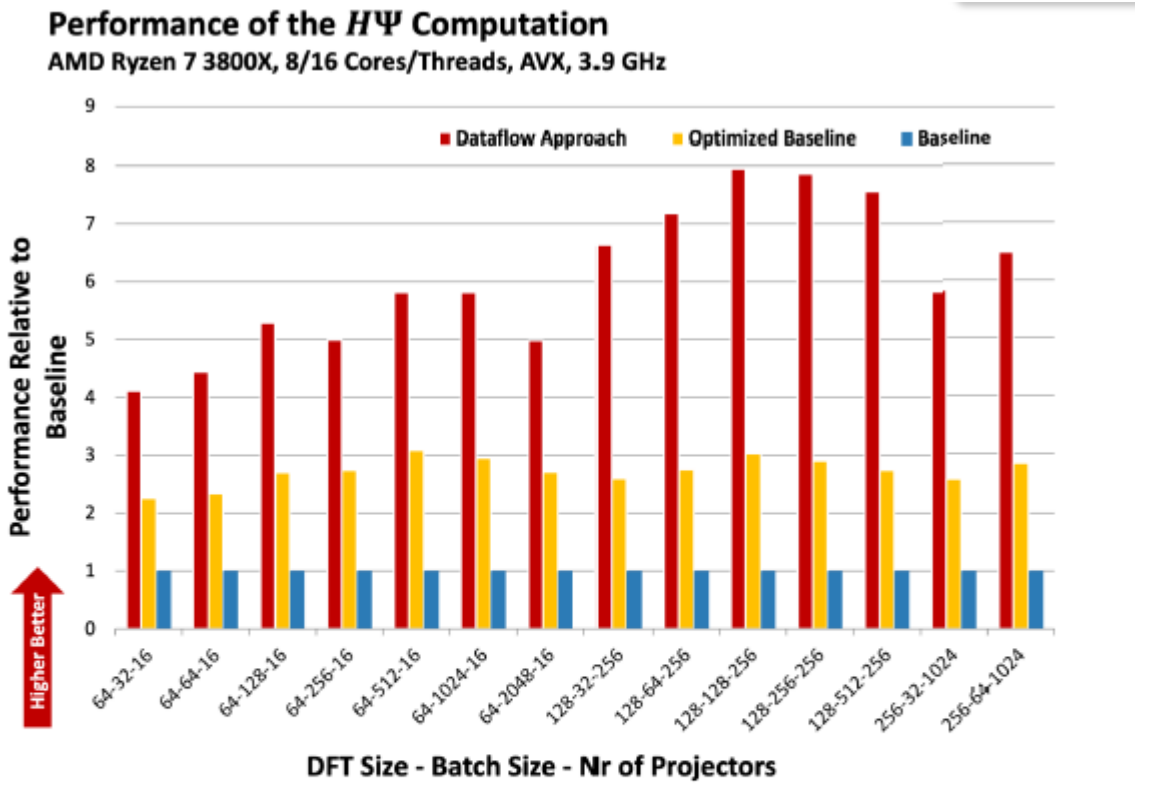

Figure 10: $H\Psi$ performance on an AMD Ryzen 7 3800X. The blue bar represents the conventional library codes, the yellow bar represents an improved baseline using our own libraries called sequentially, and the red bar represents our optimized dataflow implementation.

Analysis and Results

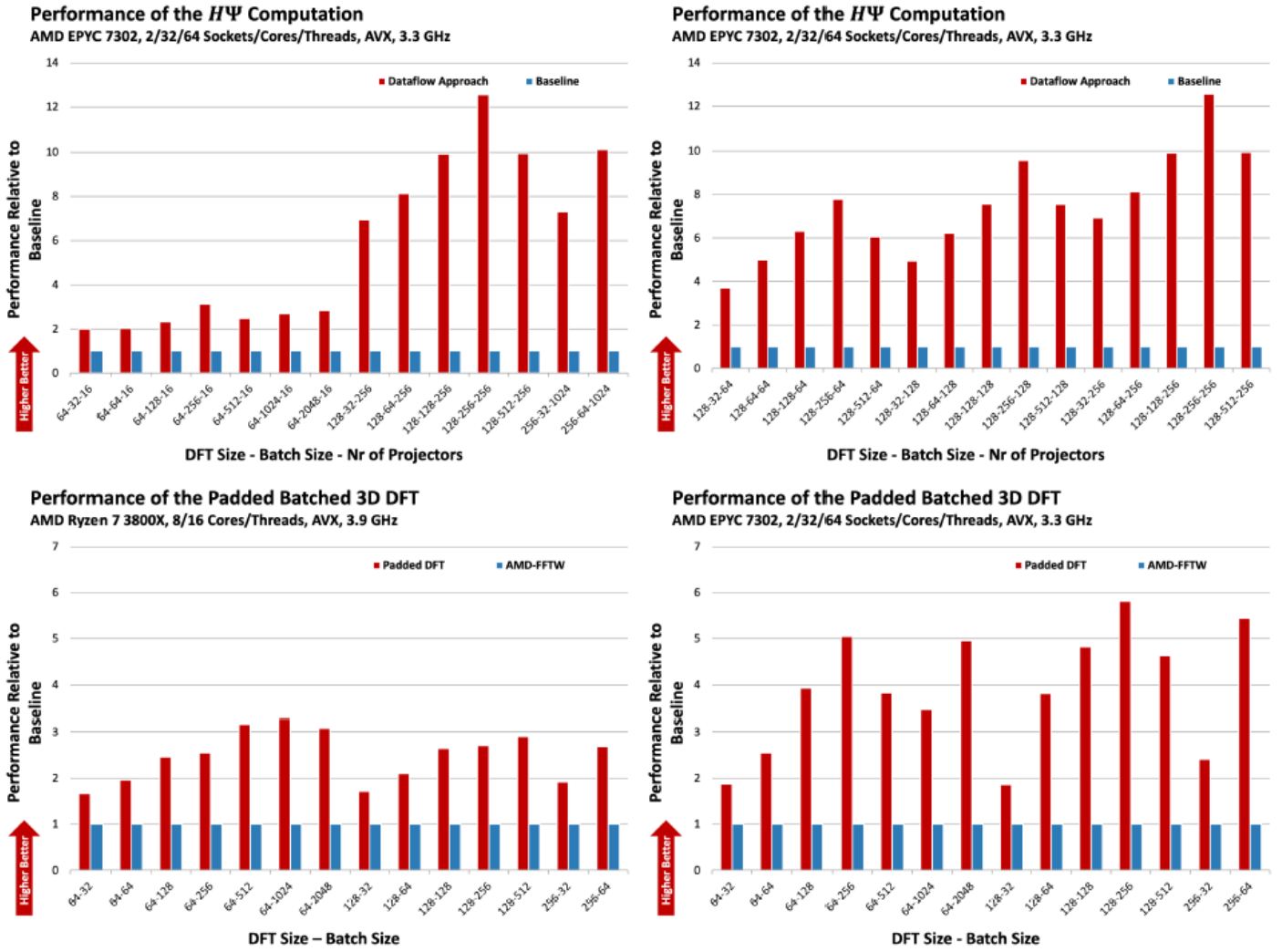

Figure 11: Performance results on the AMD Ryzen and EPYC CPUs. The top left plot shows performance of $H\Psi$ on the AMD EPYC, where we compare our optimized dataflow approach (red) against the baseline (blue). The top right plot shows performance of the same $H\Psi$ on the AMD EPYC, where we keep the Fourier transform size fixed, and increase the bands and atoms. The bottom plots show the performance difference between our batched DFT implementation (red) and the AMD version of FFTW (blue) on the Ryzen and EPYC CPUs.

Analysis and Results

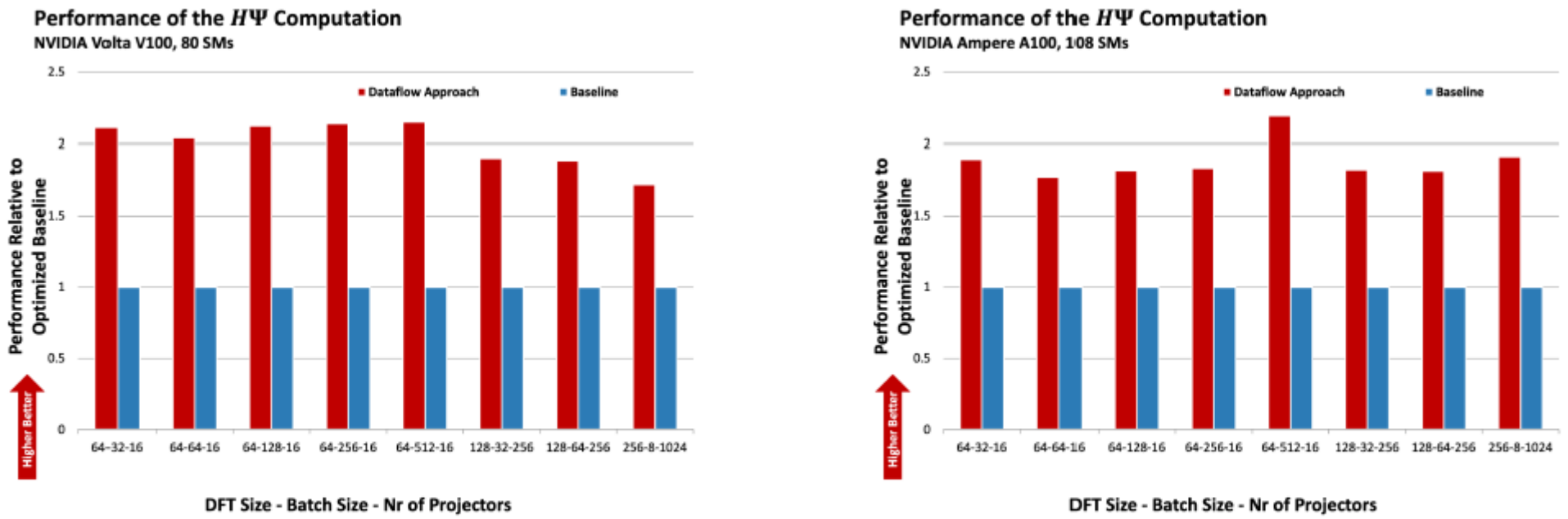

Figure 12: Performance results of the $H\Psi$ computation on the NVIDIA V100 (left) and A100 (right) GPUs. We compare our optimized dataflow approach (red bar) against the baseline (blue) that uses cuFFT library calls.

Conclusion

Create a systematic approach for merging the DFT stages with the surrounding sparse and dense linear algebra computations.

Reducing the amount of data moved to and from main memory, and in increasing the data locality on systems with large on chip memory. (the small amount of shared memory and a last level cache shared between thousand of threads are not optimal for merging operations tackled in this paper)

Future

Other (FFT-based) applications.

New hardware support (SambaNova…).

Automate the optimization.(codegenerators: TACO(The Tensor Algebra Compiler)…)

Reference

Aho A V. *Compilers: principles, techniques and tools* (for Anna University), 2/e[M]. Pearson Education India, 2003.

Doru Thom Popovici. 2018. *An Approach to Specifying and Automatically Optimizing Fourier Transform Based Operations*. Ph.D. Dissertation. Electrical and Computer Engineering, Carnegie Mellon University.

Thanks for listening!

Q & A