PSL Meeting Presentation

Exploring Performance Optimisation Opportunities in ONETEP

Francis Russell
Joint work with Chris-Kriton Skylaris & Karl Wilkinson

Imperial College London & Southampton University

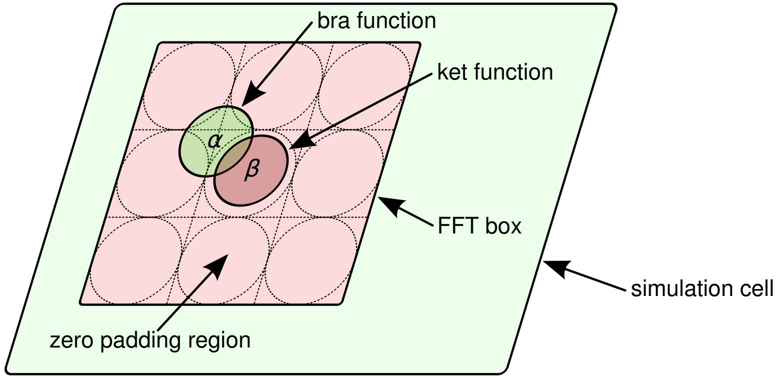
25/02/2013

DSLs for Quantum Chemistry

- I have been looking as domain specific languages (DSLs) for quantum chemistry, specifically in context of the linear scaling code ONETEP.
- In my previous presentation, I concluded that the domain of variability in the input to a domain-specific language for ONETEP was quite small.
- The domain of variability in the code we can generate is a far more interesting target - primarily from the perspective of increasing performance.
- We decided to look at the potential for optimisation in ONETEP's more computationally intensive routines to determine if they optimisations we'd like to generate even exist.

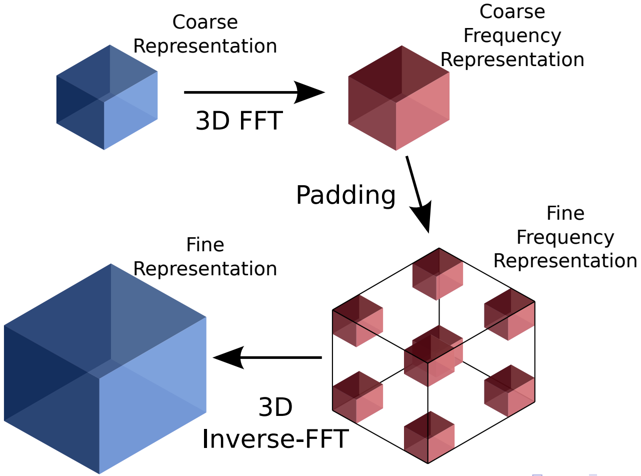
FFT-Boxes

- Within a simulation cell, ONETEP performs many Discrete Fourier transforms within regions called FFT-boxes.
- The most computationally intensive involve the interpolation of values in these boxes.



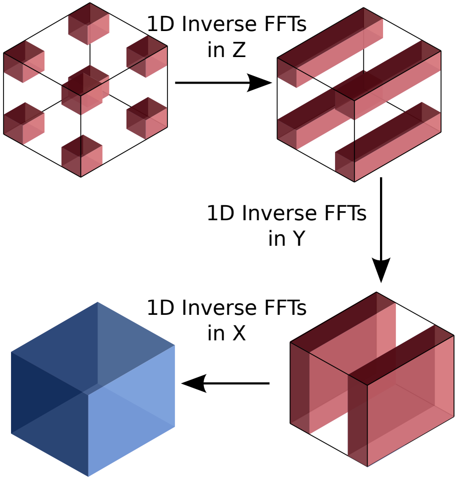
Fourier Interpolation of FFT-boxes

The interpolation process in ONETEP involves transforming a 3-dimensional block of data to block 8x the original size. The resolution has been doubled in each dimension.



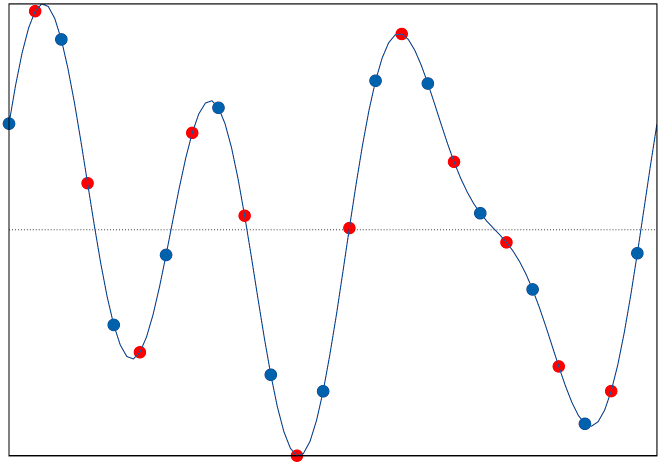
ONETEP's Approach to Interpolation

ONETEP performs transforms in each dimension so each FFT only operates on 50% zeroes instead of the 87.5% of the naïve strategy.



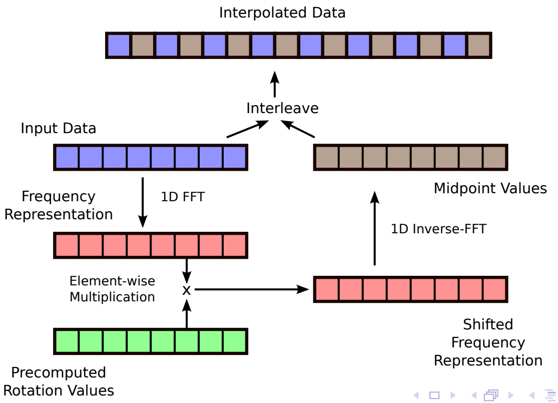
Phase-Shift Approach to Interpolation

We have a signal discretely sampled at the blue points, but we want to know the value at both the blue and red points. We can find these if we shift the signal by half a sample.



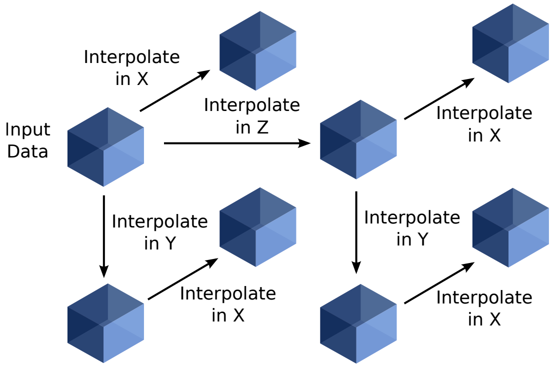
Phase-Shift Interpolation in 1D

- We use a 1D FFT to compute the frequency representation, apply a phase shift, then trasform back to compute the values of the midpoints.
- We interleave the original data and midpoints to produce the interpolated values.
- We never operate on zeros (except those in the original input).



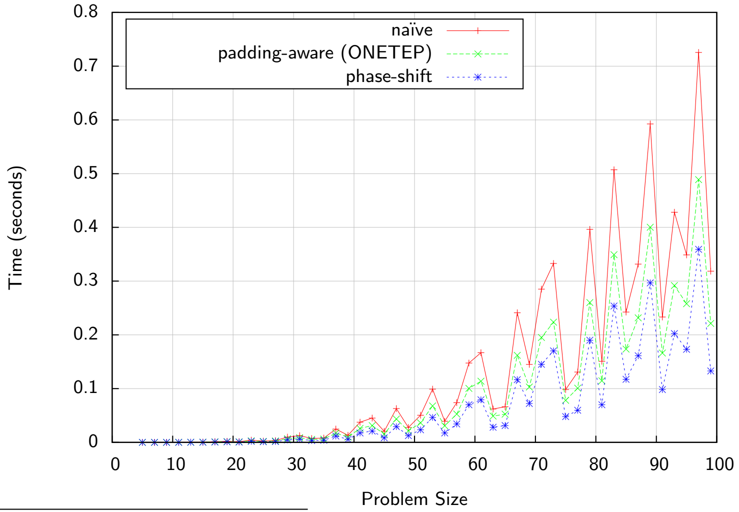
Phase-Shift Interpolation in 3D

- Interpolation is done by dimension, with the most cache inefficient done first.
- We interleave the original data and the 7 interpolated blocks as a final step.



Performance Results with FFTW[1]

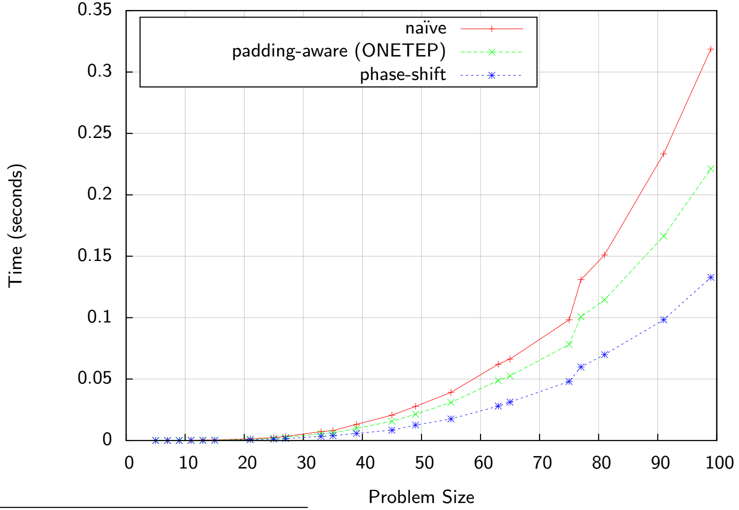
Results are difficult to interpret because of FFTW's performance at different problem sizes due to the factorisation it chooses.



[1] Intel Core i7-2600, 3.4GHz, 8MB L2 cache, FFTW 3.3

Performance Results with FFTW[2]

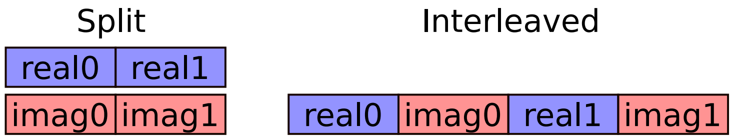
Filtering the results to those which FFTW likes best (products of small primes). Specifically, sizes of the form $2^a 3^b 5^c 7^d 11^e 13^f$ where $e + f < 2$.



[2]Intel Core i7-2600, 3.4GHz, 8MB L2 cache, FFTW 3.3

Performance within ONETEP

- In our standalone benchmarks, our results give around a 35% reduction in execution time over ONETEP's approach for FFTW's preferred sizes.
- In practice, we found the actual reduction to be a lot less and overall reduction in execution time to usually be less than 5%.
- When doing Fourier interpolation, ONETEP spends a lot of time in its modifying data layouts before calling FFTW.
- Specifically, converting from a split to interleaved representation of complex numbers. Sometimes this is due to two sets of real operands being merged so interpolation can be done on both simultaneously.



Making the new interpolation work better with ONETEP

- I discovered that FFTW provides a “guru” interface that makes it possible to pass in data in data in the split format.
- I wrote a new implementation of the interpolation routines that could accept and return data in this format without remarshalling.
- Also wrote a routine that returned the product of the real and complex parts as part of the re-interleave step, avoiding the need for the calling routine in ONETEP to have to iterative over the data again.

ONETEP Interpolation Routine Timings[3]

Test	Original (s)	Modified (s)	Reduction
test01	49.47	35.27	28.70%
test02	61.51	44.56	27.56%
test03	40.67	29.66	27.07%
test04	90.99	61.51	32.40%
test05	47.78	34.46	27.88%
test06	3.73	2.73	26.81%
test07	8.72	6.46	25.92%
test08	93.33	66.83	28.39%
test09	3.47	2.35	32.28%
test10	361.81	265.67	26.57%
test11	33.27	23.04	30.75%
test12	57.65	40.19	30.29%
test13	17.37	12.30	29.19%
test14	40.55	29.74	26.66%
test15	26.05	18.55	28.79%
test16	33.93	25.67	24.34%
test18	107.61	77.48	28.00%
test19	73.72	52.60	28.65%
test20	95.72	62.37	34.84%
test21	2.68	2.13	20.52%
test22	53.47	39.34	26.43%
test23	22.55	15.40	31.71%
test24	22.81	15.42	32.40%

[3] ONETEP 3.3.9.5, Intel Core i7-2600, 3.4GHz, 8MB L2 cache, FFTW 3.3

ONETEP Total Execution Time Timings[4]

Test	Original (s)	Modified (s)	Reduction
test01	131.43	111.06	15.49%
test02	136.79	118.22	13.58%
test03	619.04	606.73	1.99%
test04	169.56	139.72	17.60%
test05	107.47	93.61	12.90%
test06	27.02	26.35	2.49%
test07	49.99	45.57	8.85%
test08	296.68	247.57	16.55%
test09	15.82	14.49	8.44%
test10	1126.06	955.09	15.18%
test11	90.85	77.61	14.57%
test12	154.00	131.92	14.34%
test13	231.26	223.75	3.25%
test14	272.05	255.62	6.04%
test15	97.64	83.10	14.90%
test16	98.63	87.28	11.51%
test18	343.04	294.95	14.02%
test19	180.72	156.56	13.37%
test20	260.15	221.94	14.69%
test21	11.45	10.95	4.35%
test22	125.30	109.90	12.29%
test23	46.18	38.92	15.72%
test24	46.46	38.85	16.39%

[4]ONETEP 3.3.9.5, Intel Core i7-2600, 3.4GHz, 8MB L2 cache, FFTW 3.3

Running on CX1

- We benchmarked on CX1 cluster at Imperial using the Intel Math Kernel Libraries instead of FFTW.
- Performance was terrible, significantly worse than vanilla ONETEP ($\approx$ 20% slowdowns).
- It looks like MKL really dislikes working with split layout data.
- If so, this means that ONETEP's data marshalling that we worked to remove actually helps with IMKL.
- I updated the interpolate library to support staging data so that the FFT routines could work on entirely *contiguous*, *interleaved* data.
- The other implementation would gather and scatter directly from and to the input and output arrays using the FFT routines.
- When planning the interpolate, we benchmark and select the best technique. When using IMKL, we almost always do the scatter/gather ourselves.

Running on CX1

We don't have performance results for this new implementation. . . yet.