Unit 6: Code Structure Considerations

As noted in Unit 3, most modern computing architectures are now designed in such a way that single precision floating-point operations are (usually) significantly faster than double precision calculations. For some chips, half-precision is even faster. Additionally, reducing the precision also allows for increased speed of data movement.

In a mixed-precision code, data will need to be transformed between the precisions being used. The costs of this conversion will form an overhead and, hence, the gains from using single precision must outweigh these overheads.

Let y=f(x) be a function that takes double precision data, performs double precision calculations and outputs double precision data. Let y_sp=f_sp(x_sp) be the single precision version of f(x) with single precision input and output, and internally performs single precision calculation. Additionally, x_sp=conv_dp_sp(x) converts double precision data to single precision data and x=conv_sp_dp(x_sp) converts single precision data to double precision data. Ideally, the number of conversions should be minimised subject to the desired level of accuracy in the overall computation being maintained. We will start by considering the simplest form mixed-precision computation structure:

```
function y = f_mp(x)

input :: double precision data x
output :: double precision data y

x_sp = conv_dp_sp(x);
y_sp = f_sp(x_sp);
x = conv_sp_dp(x_sp);
```

For the mixed-precision function to be beneficial over using the double precision version, we require the execution time (or energy consumption) to be smaller when using the mixed-precision version.

Case Study: Fast Fourier Transforms

Let x be a vector of length n containing complex numbers with both real and imaginary components stored in double precision and let x_sp be a vector of length n containing complex numbers with both real and imaginary components stored in single precision. Additionally, let y=fft(x) be the fast Fourier transform performed with double precision arithmetic and y_sp=fft_sp(x_sp) be the fast Fourier transform performed with single precision arithmetic. Using an efficient algorithm for the double precision fast Fourier transform with $n = 2^p$, where p is a positive integer, the number of floating point operations is

$$\sim\frac{34}{9}n\log_2 n.$$

Assuming that the single precision fast Fourier transform algorithm takes half the time of the double precision algorithm, we have

$$time(\text{fft}) \approx \frac{34}{9} c_1 n \log_2 n \text{ and } time(\text{fft\_sp}) \approx \frac{17}{9} c_1 n \log_2 n$$

where c_1 is a real constant. Additionally, let $time(\text{convert\_dp\_sp}) = c_2 n$ and $time(\text{convert\_sp\_dp}) = c_3 n$, where c_2 and c_3 are real constants. Therefore, we require

$$time(\text{fft\_mp}) = (c_2 + c_3)n + \frac{17}{9} c_1 n \log_2 n < \frac{34}{9} c_1 n \log_2 n.$$

This is equivalent to

$$c_2 + c_3 < \frac{17}{9} c_1 \log_2 n.$$

With respect to execution time, for large enough n it will always be a benefit to use the mixed-precision fast Fourier transform over the double precision version. In [1], the ratio of the single precision (wall clock) execution time to the double precision version of the FFT algorithm decreases towards 0.5 as the problem size increased. For larger problems, using the Intel Xeon (IvyBridge E5-2697v2 2.7GHz), it was always worth using the mixed-precision version of the fast Fourier transform instead of the double precision version. In the following table, we provide the total wall clock execution times for DL_POLY's [2]

- the fast Fourier transform function fft;
- the function spme that calls the fast Fourier transform, performs some other double precision functions and calculations and, if required, performs the precision conversions.

We use different sizes of DL_POLY's test problem Problem02.

No. atoms	MPI processes	fft			spme		
		Mixed precision	Double precision	Ratio	Mixed precision	Double precision	Ratio
51737	1	4.15	5.26	0.79	8.59	9.67	0.89
	2	1.97	2.31	0.85	4.19	4.55	0.92
	4	0.87	1.10	0.79	2.05	2.29	0.90
	8	0.45	0.52	0.87	1.00	1.08	0.92
413896	8	5.10	6.77	0.75	9.99	11.8	0.85
	16	2.75	4.13	0.67	5.48	6.93	0.79
	32	1.38	2.30	0.60	2.81	3.79	0.74
6467125	49	27.7	46.1	0.60	45.0	63.7	0.71
	96	14.6	29.1	0.50	23.1	38.7	0.60
	144	9.83	17.2	0.57	15.0	22.6	0.66

Further details and results are available in [1].

[1] "Using mixed precision within DL_POLY's force and energy evaluations: long-range interactions and fast Fourier transforms", *H. S. Thorne*, Tech. Rep. RAL-TR-2018-03, 2018.

[2] "The DL POLY 4 User Manual (version 4.08)", *I. Todorov and W. Smith*, March 2016.

Mixed-precision and loops

There are cases where the code structure will (normally) make it prohibitively difficult to develop an efficient mixed-precision code. Consider the following nested double precision and mixed precision loops:

```
For i=1,…,n
  …
  For j=1,…,n_i
    …
    … calculate x using double precision
    …
    … calculate y using double precision
    a(j) = a(j) + y
    …
  End for
End for
```

```
For i=1,…,n
  …
  For j=1,…,n_i
    …
    … calculate x using double precision
    …
    x_sp = conv_dp_sp(x)
    …
    … calculate y_sp using single precision
    a(j) = a(j) + conv_sp_dp(y_sp)
    …
  End for
End for
```

Unless the amount of work required to compute y in double precision is very large compared to the rest of the loop and the use of single precision produces a significant decrease, the cost of the conversions between precisions will be greater than the savings introduced by computing y_sp instead of y.

For some codes, it may be appropriate to have an additional single precision update array, a_update, and perform the following alternative code:

```
a_update(:) = 0.0
For i=1,…,n
  …
  For j=1,…,n_i
    …
    … calculate x using double precision
    …
    x_sp = conv_dp_sp(x)
    …
    … calculate y_sp using single precision
    a_update(j) = a_update(j) + y_sp
    …
  End for
End for
For i=1,…,n
  a(i) = a(i) _ conv_sp_dp(a_update(i))
End for
```

If the cost of allocating and deallocating the additional single precision array and the precision of a_update allows for the accurate accumulation of summations, then this may be preferable because the number of conversions from single to double precision is significantly reduced. In practice, y_sp and a_update(:) are likely to vary in value in such a way that the accuracy of the computation is lost (see Unit 5) and, hence, this is not practical. See [3] for full details of a loop structure where mixed-precision version exhibit all of these problems.

[3] "Using mixed precision within DL POLY's force and energy evaluations: short-range interactions", *H. S. Thorne,* Tech. Rep. RAL-TR-2018-004, 2018.