Accelerating Numerical Solutions to Fractional Differential Equations

Adams Moulton Bashforth Method

Adam J. Gray
Supervised by: Dr Chris Tisdell

School of Mathematics and Statistics
University of New South Wales

October 22, 2014

Outline

- Examine the Adams Moulton Bashforth method for a first order initial value problem.
- Examine a fractional Adams Moulton Bashforth method.
- Examine a parallel CPU based implementation of the scheme.
- Examine a parallel GPU based implementation of the scheme.

A first order problem

We would like to solve the following initial value problem.

$$\frac{d}{dx}y(x) = f(x, y)$$

$$y(0) = y_0$$

This is equivalent to

$$y(x) = y_0 + \int_0^x f(t, y(t))dt$$

The first step

Suppose we want to approximate the solution some small time-step h away from 0.

$$y(h) = y(0) + \int_0^h f(t, y(t))dt$$

Using a simple quadrature rule (trapezoidal) we could approximate the integral to get

$$y(h) = y(0) + \frac{h}{2}\left[f(0, y(0)) + f(0, y(h))\right]$$

Predictor Approximation of y(h)

What if we approximate $y(h)$ and use that value in the previous integral? We can do this by using the rectangle quadrature rule which gives us

$$\int_0^h f(t, y(t))dt \approx hf(0, y(0))$$

and so we could say

$$y^P(h) = y_0 + hf(0, y(0))$$

and putting $y^P(h)$ in the equation before yields

$$y(h) = y(0) + \frac{h}{2}\left[f(0, y(0)) + f(0, y^P(h))\right]$$

Iterating the process

The previous slides explained how to step forward one small time-step h. We can now use the value calculated for $y(h)$ as the *initial value* and calculate $y(2h)$.

$$y(2h) = y(h) + \frac{h}{2}\left[f(h, y(h)), + f(2h, y^P(2h))\right]$$

with

$$y^P(2h) = y(h) + hf(h, y(h)).$$

We can restate this whole process iteratively:

$$\begin{aligned} y_{k+1} &= y_k + \frac{h}{2}\left[f(x_k, y_k) + f(x_{k+1}, y^P_{k+1})\right] \\ y^P_{k+1} &= y_k + hf(x_k, y_k) \end{aligned}$$

Is the only way of doing things?

- No!
- There are lots of different schemes for numerically approximating the solutions to integer order initial value problems (RK3/4, Euler, Milne, etc).
- In fact we have hidden the Euler method inside this method. The Euler method basically puts some small but non-zero h into the first principles definition of the derivative and solves from there.

Key Computational Points of this Method

- It's numerically stable for reasonable choices of f.
- It's convergent of order 2 in the sense that

$$\max_{j=0,1,\ldots,N} |y(t_j) - y_j| = O(h^2)$$

- The computational complexity is $O(N)$ (linear time).

The Fractional Problem

We wish to solve a fractional initial value problem of the form

$$\left({}_{0}^{C}\mathcal{D}_{x}^{\alpha}y\right)(x) = f(x, y)$$

along with initial conditions

$$y(0) = y_0^{(k)} \qquad\qquad 0 \leq k \leq \lceil\alpha\rceil - 1$$

$$\left({}_{0}^{C}\mathcal{D}_{x}^{\alpha}y\right)(x) = \frac{1}{\Gamma(\lceil\alpha\rceil - \alpha)}\int_0^x \frac{\frac{d^{\lceil\alpha\rceil}}{dt^{\lceil\alpha\rceil}}y(t)}{(x-t)^{\lceil\alpha\rceil-\alpha+1}}dt$$

Equivalent Integral Equation

The initial value problem

$$\left({}_{0}^{C}\mathcal{D}_{x}^{\alpha} y\right)(x) = f(x, y)$$

along with initial conditions

$$y(0) = y_0^{(k)} \qquad\qquad 0 \leq k \leq \lceil \alpha \rceil - 1$$

is equivalent to the integral equation

$$y(x) = \sum_{k=0}^{\lceil \alpha \rceil - 1} y_0^{(k)} \frac{x^k}{k!} + \frac{1}{\Gamma(\alpha)} \int_0^x (x-t)^{\alpha-1} f(t, y(t)) dt$$

The Fractional Case

$$y_{k+1} = \sum_{j=0}^{\lceil\alpha\rceil-1} y_0^{(j)} \frac{x_{k+1}^j}{j!} + \frac{1}{\Gamma(\alpha)} \left(\sum_{j=0}^{k} a_{j,k+1} f(x_j, y_j) + a_{k+1,k+1} f(x_{k+1}, y_{k+1}^P) \right)$$

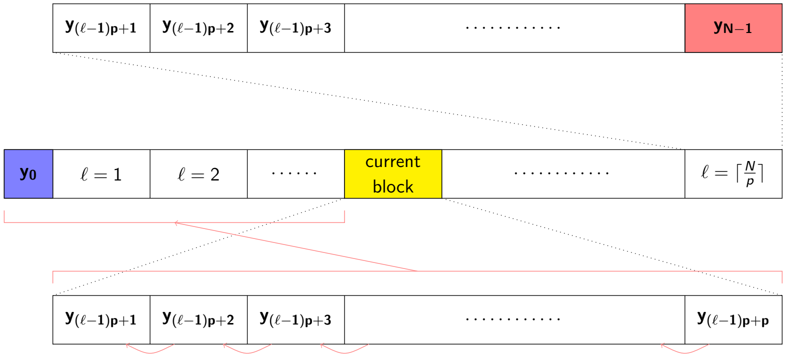
$$y_{k+1}^P = \sum_{j=0}^{\lceil\alpha\rceil-1} \frac{x_{k+1}^j}{j!} y_0^{(j)} + \frac{1}{\Gamma(\alpha)} \sum_{j=0}^{k} b_{j,k+1} f(x_j, y_j).$$

Coefficients

$$a_{j,k+1} = \frac{h^\alpha}{\alpha(\alpha+1)} \times \begin{cases} (k^{\alpha+1} - (k-\alpha)(k+1)^\alpha) \\ ((k-j+2)^{\alpha+1} + (k-j)^{\alpha+1} - 2(k-j+1)^{\alpha+1}) \\ 1 \end{cases}$$

$$b_{j,k+1} = \frac{h^\alpha}{\alpha}\left((k+1-j)^\alpha - (k-j)^\alpha\right).$$

Parallel Approach


$\mathbf{y}_{(\ell-1)p+1}$
$\mathbf{y}_{(\ell-1)p+2}$
$\mathbf{y}_{(\ell-1)p+3}$
$\cdots\cdots\cdots\cdots$
$\mathbf{y}_{N-1}$
$\mathbf{y}_0$
$\ell = 1$
$\ell = 2$
$\cdots\cdots$
current block
$\cdots\cdots\cdots\cdots$
$\ell = \lceil \frac{N}{p} \rceil$
$\mathbf{y}_{(\ell-1)p+1}$
$\mathbf{y}_{(\ell-1)p+2}$
$\mathbf{y}_{(\ell-1)p+3}$
$\cdots\cdots\cdots\cdots$
$\mathbf{y}_{(\ell-1)p+p}$

Parallel Formulation

We can requite the previous equations in the form

$$y_{j+1}^{P} = I_{j+1} + h^{\alpha} H_{j,\ell}^{P} + h^{\alpha} L_{j,\ell}^{P}$$

and

$$y_{j+1} = I_{j+1} + h^{\alpha} H_{j,\ell} + h^{\alpha} L_{j,\ell}$$

Parallel Formulation

$$I_{j+1} := \sum_{k=0}^{\lceil\alpha\rceil-1} \frac{x_{j+1}^k}{k!} y_0^{(k)}$$

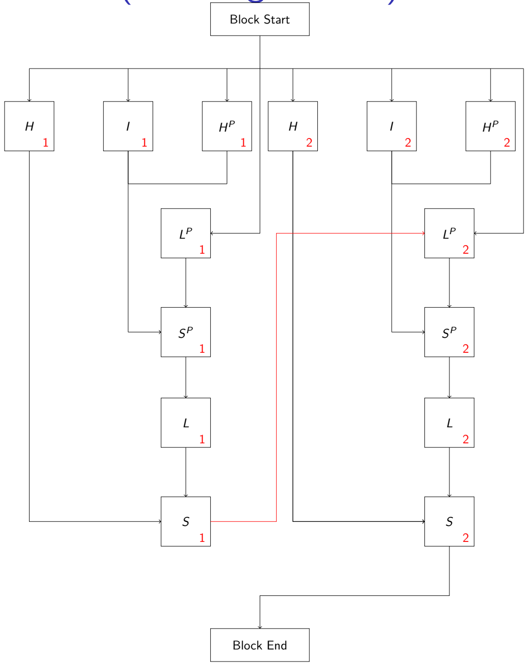
$$H_{j,\ell}^P := \sum_{k=0}^{(\ell-1)p} b_{j-k} f(x_k, y_k)$$

$$L_{j,\ell}^P := \sum_{k=(\ell-1)p+1}^{j} b_{j-k} f(x_k, y_k)$$

$$H_{j,\ell} := c_j f(x_0, y_0) + \sum_{k=1}^{(\ell-1)p} a_{j-k} f(x_k, y_k)$$

$$L_{j,\ell} := \sum_{k=(\ell-1)p+1}^{j} aj - kf(x_k, y_k) + \frac{f(x_{j+1}, y_{j+1}^P)}{\Gamma(\alpha+2.)}$$

Task Flow (in a single block)



Is this parallelization reasonable?

Principle (Amdahl)

Let P be the proportion of a computation which can be parallelized in a given progress. Then the speed-up given by Q cores is

$$S(Q) = \frac{1}{(1 - P) + \frac{P}{Q}}$$

Definition (Amdahl Efficiency)

Let $S(N, Q)$ represent the parallel speed-up for a problem with input size N and executed with Q cores. Then we define the Amdahl efficiency of parallel scheme as

$$E(Q) = \lim_{N \longrightarrow \infty} \frac{S(N, Q)}{Q}.$$

Further we say that a parallel scheme is Amdahl efficient with respect to its serial counterpart if $E(Q) = 1$.

Proposition

The parallel Adams Moulton Bashforth scheme is Amdahl efficient with respect to its serial counterpart.

Proof.

We present the parallelizable fraction of computation in each block of variables in the following table. The order column represents the number of *multiply-add* operations per task. □

Task	Computational Complexity	Number of Variables per Block	Total Complexity
H	$O(\ell p)$	p	$O(\ell p^2)$
H_p	$O(\ell p)$	p	$O(\ell p^2)$
I	$O(\alpha)$	p	$O(p)$
L	$O(p)$	p	$O(p^2)$
L_p	$O(p)$	p	$O(p^2)$

In each block H, H_p and I can be computed in parallel across all variables in the block. For the purposes of simplicity we assume that the tasks L and L_p must all be computed in serial across all variables in a block. This means that in the ℓ-th block the non-parallelizable fraction of computation is

$$\overline{P}_{block}(\ell) \sim \frac{2p^2}{2p^2(\ell+1)} \sim \frac{1}{\ell}$$

The amount of computation in the ℓ-th block is $O(\ell p^2)$ and there are $K \approx \frac{N}{p}$ blocks in the total computation. This means that the total amount of computation is

$$\sum_{\ell=1}^{K} \ell p^2 \sim \frac{K^2 p^2}{2}$$

and by taking a weighted average sum, the total fraction of non-parallelizable computation is

$$\overline{P}_{total} \sim \sum_{\ell=1}^{K} \frac{2\ell p^2}{K^2 \ell p^2} = \frac{2}{K} = \frac{2p}{N}.$$

Noticing the fact that the computation in the task I is insignificant compared to H and H_p we would suppose using $Q = 2p$ cores would be in some sense optimal. Applying Amdahl's principle with the result in **??** and with $Q = 2p$ we get

$$\begin{aligned} S(N,Q) &= S(N,2p) \\ &\sim \frac{1}{(\frac{2p}{N}) + \frac{1-\frac{2p}{N}}{2p}} \end{aligned}$$

then letting $N \longrightarrow \infty$ we get

$$E(p) = \lim_{N \longrightarrow \infty} \left(\frac{1}{(\frac{2p}{N}) + \frac{1-\frac{2p}{N}}{2p}} \right) / p = 1.$$

Practical Experimentation

I solved the initial value problem being discussed with $\alpha = \frac{1}{2}$ and $y(0) = 1$ over the interval $[0, 1]$ using a C# implementation of the above algorithm.

Practical Experimentation

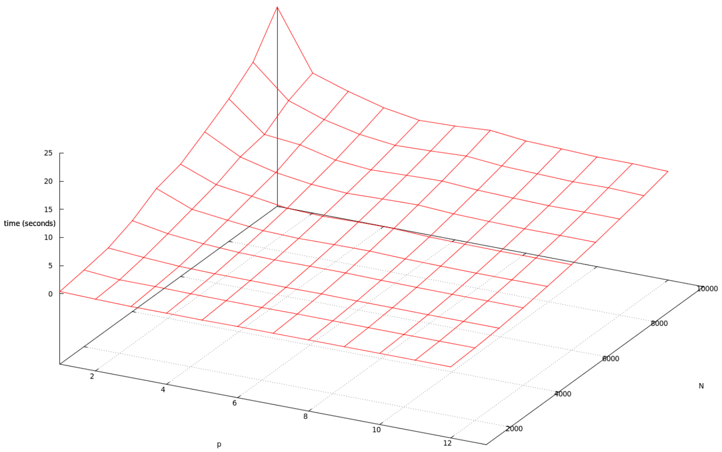


Figure: Average runtime of the scheme for different values of p and N

Redesigning the Algorithm for CUDA

Instead of trying to compute multiple variables in parallel we just return to the original (single threaded) formulation of the scheme and compute the large sums in parallel.

$$
\begin{aligned}
y_{k+1} = {} & \sum_{j=0}^{\lceil\alpha\rceil-1} y_0^{(j)} \frac{x_{k+1}^j}{j!} + \\
& \frac{1}{\Gamma(\alpha)} \left(\sum_{j=0}^{k} a_{j,k+1} f(x_j, y_j) + a_{k+1,k+1} f(x_{k+1}, y_{k+1}^P) \right) \\
y_{k+1}^P = {} & \sum_{j=0}^{\lceil\alpha\rceil-1} \frac{x_{k+1}^j}{j!} y_0^{(j)} + \frac{1}{\Gamma(\alpha)} \sum_{j=0}^{k} b_{j,k+1} f(x_j, y_j).
\end{aligned}
$$

Quick Notes for CUDA

- CUDA is particularly good for SIMD type computations (same instructions multiple data). We also refer to this as data parallelism. (We have a large amount of that in this particular case)
- We by calling a function in CUDA we can actually be invoking 1000s of instances of the function.
- Each instance is passed *grid reference* so it knows what part of the data to work on. Groups of instances (up to 1024) can do some limited communication and perform a small set of atomic operations on shared memory. This allows us add up the sub results.

Practical Experimentation

I solved the same initial value problem as I did in the C# case.

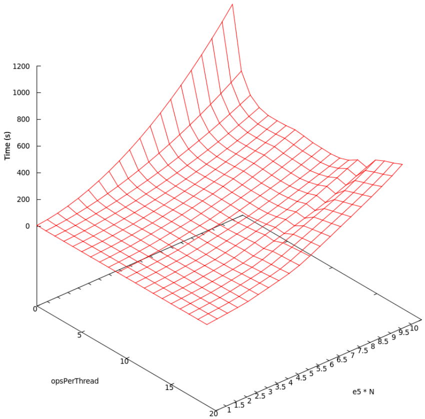
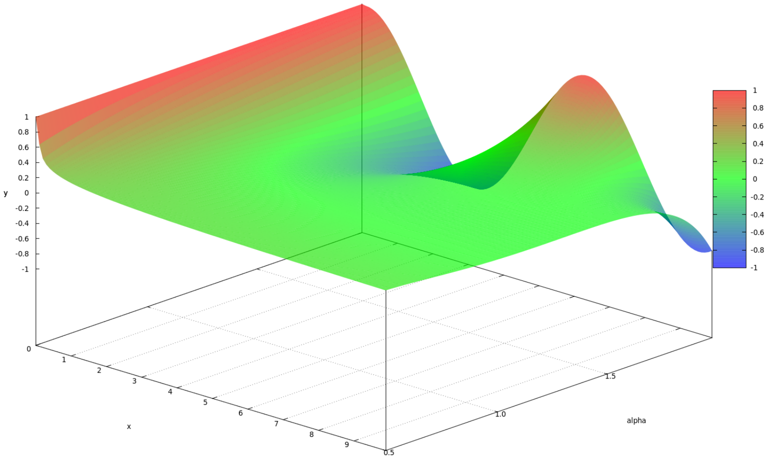


Figure: Runtime for the scheme for various values of opsPerThread and N.

Lets play...

Solution to the relaxation oscillation equation for $\alpha \in [0.5, 2]$.



Key references

1 K. Diethelm. The Analysis of Fractional Differential Equations. Springer, 2010.

2 K. Diethelm. An efficient parallel algorithm for the numerical solution of fractional differential equations. Fractional Calculus and Applied Analysis, 14:475490, 2011.

3 K. Diethelm, N.J. Ford, and A.D. Freed. Detailed error analysis for a fractional Adams method. Numerical Algorithms, pages 3152, 2004.