Scalable Computing: Practice and Experience, ISSN 1895-1767, http://www.scpe.org
© 2020 SCPE. Volume 21, Issues 4, pp. 701–707, DOI 10.12694:/scpe.v21i4.1837

PARALLEL ALGORITHM FOR NUMERICAL METHODS APPLIED TO FRACTIONAL-ORDER SYSTEM

FLORIN ROŞU *

Abstract. A parallel algorithm is presented that approximates a solution for fractional-order systems. The algorithm is implemented in CUDA, using the specific GPU capabilities. The numerical methods used are Adams-Bashforth-Moulton (ABM) predictor-corrector scheme and Diethelm's numerical method. A comparison is done between these numerical methods that adapts the same algorithm for the approximation of the solution.

Key words: GPU processing, HPC processing, parallel algorithm, numerical methods, fractional-order systems

AMS subject classifications. 65Y05

1. Introduction.

A few applications of the fractional-order derivative system that models real world phenomena are presented in [1, 14, 15, 17]. Unfortunately, the analytical method are not yet discovered for finding the solution of fraction-order derivatives system. The numerical methods provides an approximation of the solution.

The Adams-Bashforth-Moulton predictor-corrector method is widely study [2, 3] and still improved to get more accurate solution [18]. There are several implementation of this method using different parallel computing technologies: MPI and OpenMP [4, 16]; Matlab [5].

The parallel numerical algorithm that was implemented for BlueGene/P supercomputer [7] and adapted to run on GPU [8] is using the Adams-Bashforth-Moulton predictor-corrector method that estimates the solution for Caputo-type fractional-order system. In this paper we adapt the same algorithm to estimate the solution using Diethelm's method [13] and compare them from different point of view.

2. Preliminaries.

The fractional order equations has several definitions and to mention a few there is the Riemann-Louville definition, the Caputo type, Grünwald-Letnikov. In some conditions, some of them are equivalent [11].

The Caputo-type definition is also known as the initial value problem. We will considered the simplified written form:

$$\begin{cases} \mathrm{D}_*^{\alpha}\, y(t) = f(t, y(t)), t \in [0, T] \\ y(0) = y_0 \end{cases} \tag{2.1}$$

where $0 < \alpha < 1$.

The interval $[0, T]$ which represents the elapsed time T is split into steps of h. The accuracy of the estimation is given by h with a smallest value possible. The number of points where the estimation is computed is $N = \frac{T}{h}$. A nth value in our solution it is caracterized for a point in time $t_n = n\dot{h}$. So, the elapsed time from 0 to T will be simulated by t_n values.

2.1. ABM predictor-corrector method.

The numerical method Adams-Bashforth-Moulton predictor corrector is described [2] by the following steps.

Having the initial condition as y_0 and at the time t_n having computed the $y_n = y(t_n)$ and $f_n = f(t_n, y_n)$, in order to compute the next value of y_{n+1} first we have to compute the **predictor**, which will give a first

* Dept. of Mathematics and Computer Science, West University of Timişoara, Timişoara, Romania, (florin.rosu@e-uvt.ro)

approximation y_{n+1}^P of our solution:

$$y_{n+1}^P = \sum_{k=0}^{\lceil\alpha\rceil-1} \frac{t_{n+1}^k}{k!} y_0^{(k)} + h^\alpha \sum_{k=0}^{n} b_{n-k} f_k, \tag{2.2}$$

where

$$b_n = \frac{(n+1)^\alpha + n^\alpha}{\Gamma(\alpha+1)}.$$

When computing the **corrector**, we will have an approximation of the solution for the time t_{n+1} with:

$$y_{n+1} = \sum_{k=0}^{\lceil\alpha\rceil-1} \frac{t_{n+1}^k}{k!} y_0^{(k)} + h^\alpha \left(c_n f_0 + \sum_{k=1}^{n} a_{n-k} f_k + \frac{f(t_{n+1}, y_{n+1}^P)}{\Gamma(\alpha+2)} \right), \tag{2.3}$$

where the weights a_n and c_n are defined as:

$$a_n = \frac{(n+2)^{\alpha+1} - 2(n+1)^{\alpha+1} + n^{\alpha+1}}{\Gamma(\alpha+2)}$$

and

$$c_n = \frac{n^{\alpha+1} - (n-\alpha)(n+1)^\alpha}{\Gamma(\alpha+2)}$$

2.2. Diethelm's method. The Caputo-type definition of fractional ordered equation can be written also in the form [13]:

$$D^\alpha[y - y_0] = \beta y(t) + f(t), \text{ where } 0 \le t \le 1 \tag{2.4}$$

As it can be seen in the equation 2.4 the time interval is considered to be $[0, 1]$, while the initial condition y_0 is incorporated in the equation itself.

According to Diethelm [13], the approximation of the solution is given by:

$$y_k = \frac{1}{\Theta_{0k} - \left(\frac{k}{n}\right)^\alpha \Gamma(-\alpha)\beta} \left(\left(\frac{k}{n}\right)^\alpha \Gamma(-\alpha) f_k - \sum_{j=0}^{k} \Theta_{jk} y_{k-j} - \frac{y(0)}{\alpha} \right) \tag{2.5}$$

where the weight Θ_{jk} are obtained as a solution of the equation:

$$\alpha(1-\alpha)k^{-\alpha}\Theta_{jk} = \begin{cases} -1 \text{ when } j = 0 \\ 2j^{1-\alpha} - (j-1)^{1-\alpha} - (j+1)^{1-\alpha} \text{ when } 0 < j < k \\ (\alpha-1)j^{-\alpha} - (j-1)^{1-\alpha} + j^{1-\alpha} \text{ when } j = k \end{cases} \tag{2.6}$$

3. Parallel numerical simulation in CUDA.

3.1. Numerical algorithm for ABM method. The Adams-Bashforth-Moulton predictor corrector method was implemented in CUDA using a parallel algorithm [8]. The core of the algorithm can be described in the Algorithm 1.

The main challenge in the algorithm 1 is the computation of predictor and corrector in a parallel environment. More precise, for the predictor (2.2) the part $\sum_{k=0}^{n} b_{n-k} f_k$ and for the corrector (2.3) the part $\sum_{k=1}^{n} a_{n-k} f_k$.

In this implementation, at start, the weights a_n, b_n and c_n are computed in parallel and stored in the global memory.

Algorithm 1: Parallel Algorithm in CUDA

```
Data: $T$ end of the time interval.
Data: $N$ global number of points.
Data: $B$ number of Blocks.
Data: $Threads$ number of threads.
Data: $SOL$ numerical solution.
$N_P \leftarrow N/P$;
$y_0 \leftarrow$ initial condition;
$Threads \leftarrow 1024$ ;
WEIGTHS<<<$B$, $Threads$>>>($N, a, b, c$);
for $n \in [1, N]$ do
    $B = \sqrt{n/Threads} + 1$;
    cudaDeviceSynchronize();
    /* Compute the partial predictor/corector in each block        */
    SUM<<<$B$, $Threads$>>>($n, PP\_B, PC\_B$);
    cudaDeviceSynchronize();
    /* Reduce predictor/corector and compute new $SOL_n$           */
    Reduce<<<1,B>>>($n, PP\_B, PC\_B, SOL$);
end
cudaMemCopy($SOL$, DeviceToHost);
```

Algorithm 2: Partial SUM computation

```
SUM($n, PP\_B, PC\_B$)
begin
    for (each blockId) do
        Data: $shmP[1024]$ shared memory for predictor
        Data: $shmC[1024]$ shared memory for corector
        for (each threadId) do
            compute $shmP[threadId]$;
            compute $shmC[threadId]$;
        end
        $PP\_B[blockId] \leftarrow reduce\ shmP$;
        $PC\_B[blockId] \leftarrow reduce\ shmC$;
    end
end
```

In CUDA the sum reduction can be done using two kernels [9]. The kernel *SUM* computes partial sums for each block. The partial sum in blocks are computed in parallel using 1024 threads. Each thread compute it's own partial sum. The algorithm for *SUM* kernel is presented in the Algorithm 2.

The reduction in the *SUM* kernel is done at block level, using shared memory in each block [10]. As a detailed implementation in CUDA, the reduction is done in two steps. In the first step, all 1024 threads are divided in 2 groups, each thread from the lower id's will add the result from the thread in the upper part. In this case, the threads from $0 \dots 511$ will add the sum from the threads $512 - 1023$. Then, the $0 \dots 511$ is split again in 2 groups, so the threads $0 \dots 255$ will add the sum from the threads $256 \dots 511$. This process of splitting and adding it is repeated until it remains only 32 threads.

The second step is the reduction from the last 32 threads, that can be computed directly. This is possible with the NVidia hardware architecture [9]that runs in parallel instructions from warps of 32 thread.

The second kernel runs in one block, with the number of threads as the number of blocks from the previous kernel. The partial sums for predictor and corrector were computed by the *SUM* kernel and stored the results in *PP_B* and *PC_B*.In the *Reduce* kernel the final reduction from each block is done, so the critical part in computing the predictor and corrector is solved. In the same kernel, after the reduction is done, in one of the thread, the ABM method will be apply and with the predictor and corrector, the approximated solution will be computed. The algorithm 3 describes how the final value for the solution at step n is obtained.

Algorithm 3: Reduce and approximate SOL

```
Reduce($n, PP\_B, PC\_B, SOL$)
begin
    Data: $sumP$ sum for predictor
    Data: $sumC$ sum for corrector
    Data: $predictor$ the value of the predictor
    Data: $corrector$ the value of the corrector
    $sumP \leftarrow reduce\ PP\_B$;
    $sumC \leftarrow reduce\ PC\_B$;
    if (threadId == 0) then
        $predictor \leftarrow compute\ from\ sumP$;
        $corrector \leftarrow compute\ from\ predictor\ and\ sumC$;
        $SOL[n] \leftarrow compute\ from\ corrector$;
    end
end
```

All the computations and the results are stored in global GPU's memory. Only at the end of the algorithm the results are transfer to RAM and saved the numeric solution on HDD.

The most efficient way to compute in a parallel environment is to have the work balanced between threads/blocks. That's why, for a step of n, having 1024 threads for each block, the balanced is establish by having $\sqrt{n/Threads} + 1$ blocks.

3.2. Numerical algorithm for Diethelm method. For the approximation of the solution using Diethelm's method [13], the Algorithm 1 is adapted. The core of the algorithm remains the same.

The adaption is done only in the kernels. In the *SUM* kernel, there is only one sum to be computed. The downside is that the weights needs to be generated each time when the partial sum needs them for computation.

The *SUM* kernel can be presented in the Algorithm 4.

Algorithm 4: Partial SUM computation

```
SUM($n, P\_B$)
begin
    Data: $\Theta[n]$ weights for step $n$
    for ($i \in [0 \dots n]$) do
        compute $\Theta_{i,n}$;
    end
    for (each blockId) do
        Data: $shmP[1024]$ shared memory for sum
        for (each threadId) do
            compute $shmP[threadId]$;
        end
        $P\_B[blockId] \leftarrow reduce\ shmP$;
    end
end
```

The *Reduce* kernel is easily adapted from the Adams-Bashforth-Moulton implementation. In Algorithm 5 it is presented the core execution of the kernel, and it can be seen that it's actually a simplified version.

Algorithm 5: Reduce and approximate SOL

```
Reduce($n, P\_B, SOL$)
begin
    Data: $sumP$ sum that needs to be reduced
    $sumP \leftarrow reduce\ P\_B$;
    if (threadId == 0) then
        $SOL[n] \leftarrow compute\ from\ sumP$;
    end
end
```

4. Numerical experiment and simulation results. For our simulations we use the fractional-order equation described in [13] and have the following Caputo fractional-order operator:

$$D^{\alpha}[y - y_0] = \beta y(t) + f(t) \tag{4.1}$$

with $y_0 = 0$, $t \in [0, 1]$, $\beta = -1$, $\alpha = 0.75$ and the function

$$f(t) = t^2 + \frac{2t^{2-\alpha}}{\Gamma(3-\alpha)}$$

The Table 4.1 presents the execution time for our simulations depending on the value of h that splits the $[0, 1]$ interval.

The Diethelm method takes almost 4 times longer than ABM method. It can be seen that having more points to compute, the differences in the computation times increases. These big differences are caused by the fact that computation for the weights are much more complicated in Diethelm method. The weights are computed every time, at each step, for all the coefficients. In ABM method the weights are computed upfront, as they are fixed for each step, so the weights can be stored in global memory and just reused at each step.

The equation has the exact solution of $y(t) = t^2$. At first glance, in the Fig 4.2 the approximation of the solution with both methods show good results.

On a further investigation, performing a zoom and having the step size h with a smaller value, we can observe the Adams-Bashforth-Moulton predictor corrector is more accurate. It is much closer to the exact solution than the Diethelm's method as it is visible in Fig 4.3.

5. Conclusions and future work. The Algorithm 1 for numerical simulations proved to be a generic algorithm. It was implemented and easy to adapt to run for any input functions, even functions in multiple dimensions [7, 8].

Although at first it was design to use the Adams-Bashforth-Moulton predictor corrector, in this paper it was demonstrated that the Algorithm 1 can be applied to use other numerical methods.

Having at the core the same algorithm, it is the perfect framework to make the comparison between different numerical method. We had proven that ABM method is better than Diethelm's method, both by the criteria of execution time and accuracy of the estimated solution.

The algorithm can be adapt to other numerical methods. As the core of the algorithm is to compute efficiently large number of sums with large number of terms, it can be incorporated in running simulations based on other numerical methods. For example, Lubich's fractional linear multi-step method can be implemented with this algorithm and more comparison can be done against ABM method and Diethelm's method.

TABLE 4.1
Simulation results in seconds for different numbers of time steps

#steps	ABM	Diethelm	Ratio
100000	18.406	27.311	1 : 1.483
500000	131.988	357.034	1 : 2.705
1000000	372.181	1254.215	1 : 3.369
1500000	691.348	2667.044	1 : 3.857

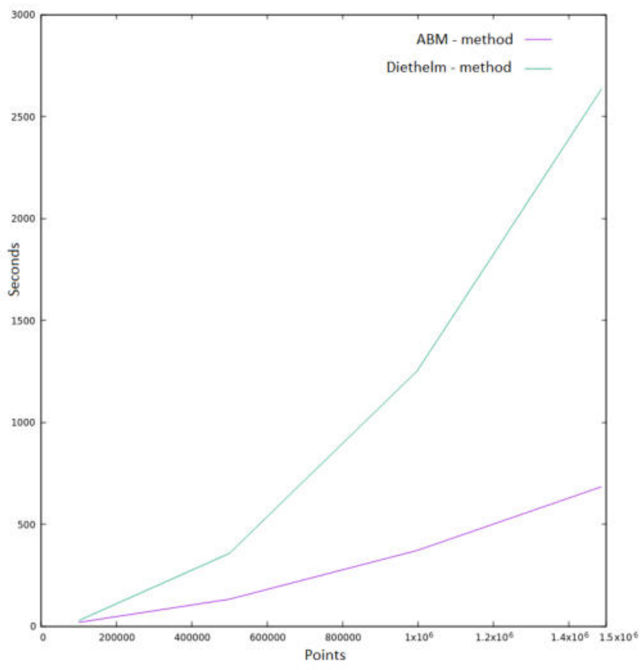

FIG. 4.1. *Execution times*

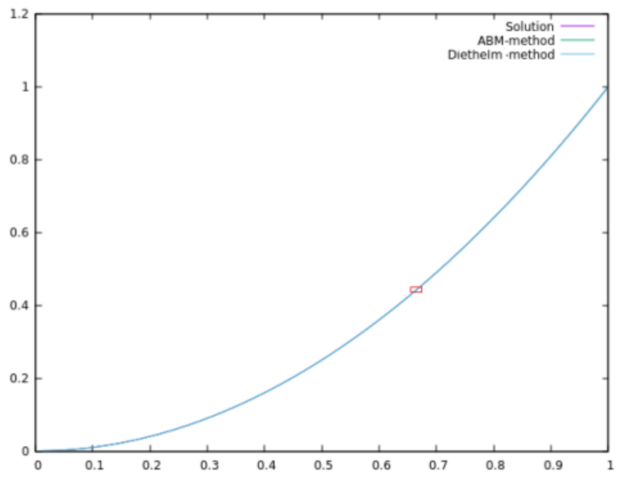

FIG. 4.2. *Aproximated solution with mark for zoom*

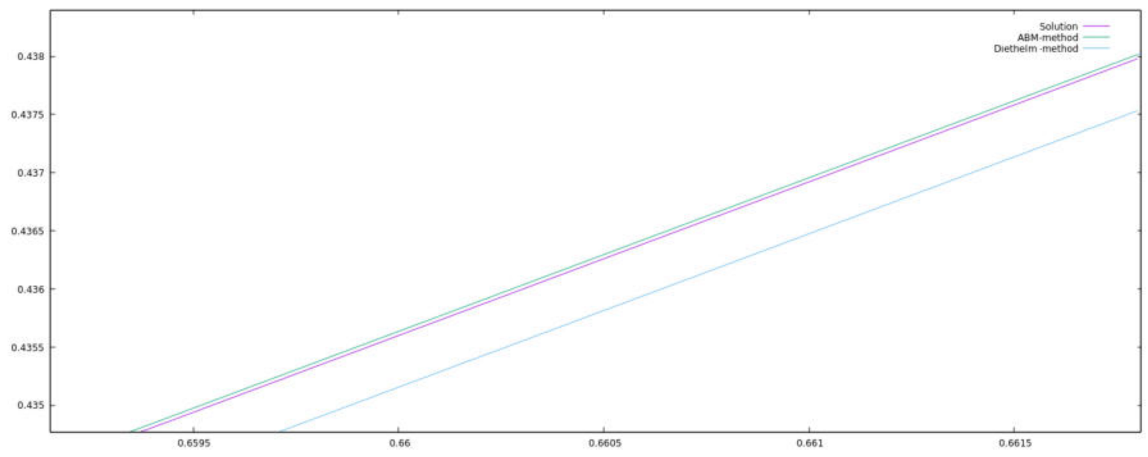

Fig. 4.3. *Aproximated solution zoom region*

REFERENCES

[1] G. Cottone, M. Di Paola, R. Santoro, *A novel exact representation of stationary colored Gaussian processes (fractional differential approach)*,Journal of Physics A: Mathematical and Theoretical, Volume 43, Page 085002, 2010

[2] K. Diethelm, N.J. Ford, and A.D. Freed, *A predictor-corrector approach for the numerical solution of fractional differential equations*. Nonlinear Dynamics, volume 29(1-4), pages 3-22, 2002.

[3] R. Garrappa, *On linear stability of predictor–corrector algorithms for fractional differential equations*, International Journal of Computer Mathematics, volume 87, pages 2281-2290, 2010

[4] W. Zhang, X. Cai, *Efficient implementations of the Adams-Bashforth-Moulton method for solving fractional differential equations* 2012

[5] N.E. Banks, *Insights from the parallel implementation of efficient algorithms for the fractional calculus*, PhD Thesis University of Chester, United Kingdom 2015

[6] L. Galeone, R. Garrappa, *Explicit methods for fractional differential equations and their stability properties, Journal of Computa- tional and Applied Mathematics*, volume 228(2), pages 548-560, 2009.

[7] C. Bonchiş, E. Kaslik, F. Roşu, *HPC optimal parallel communication algorithm for the simulation of fractional-order systems*, Journal of Supercomputing, volume 75, pages 1014–1025, 2019

[8] F. Roşu, C. Bonchiş, E. Kaslik, *Numerical simulation algorithm for fractional-order systems implemented in CUDA*, SYNASC, 2020, to be published

[9] NVIDIA Corporation, CUDA Programming Guide Version 3.1, 2010.

[10] M. Harris, *Optimizing parallel reduction in cuda* URL: http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf, 2007

[11] C. Li, W. Deng, *Remarks on fractional derivatives*, Applied Mathematics and Computation, volume 187(2),777– 784, 2007

[12] D. Cafagna, G. Grassi, *On the simplest fractional-order memristor-based chaotic system*, Nonlinear Dynam. 70 (2012) 1185–1197.

[13] K. Diethelm, *An algorithm for the numerical solution of differential equations of fractional order*, Electronic Transactions on Numerical Analysis, volume 5, pages 1-6, year 1997

[14] N. Heymans, J.-C. Bauwens, *Fractal rheological models and fractional differential equations for viscoelastic behavior*, heologica Acta 33 (1994), pages 210-219

[15] K. Diethelm, *The Analysis of Fractional Differential Equations: An application-Orientated Exposition Using Differential Operators of Caputo Type* Springer, 2010

[16] K. Diethelm, *An efficient parallel algorithm for the numerical solution of fractional differential equations*,Fractional Calculus and Applied Analysis 14 (2011), 475-490

[17] L. Song, S. Xu and J. Yang, *Dynamical models of happiness with fractional order, Communications in Nonlinear Science and Numerical Simulation*, volume 15, pages 616-628, 2010

[18] V. Gejji, Y. Sukale, S. Bhalekar, *A new predictor–corrector method for fractional differential equations*, Applied Mathematics and Computation, volume 244, 2014

Edited by: Viorel Negru
Received: Nov 18, 2020
Accepted: Dec 18, 2020