Journal of System Simulation

Volume 34 | Issue 4 Article 2

4-19-2022

Design and Implementation of A Hybrid Solver on CPU and GPU Multi-target Machines

Lin Ma
College of Computer Science and Technology, Jilin University, Changchun 130012, China;, malin20@mails.jlu.edu.cn

Xuesong Zhang
College of Computer Science and Technology, Jilin University, Changchun 130012, China;, wetting@jlu.edu.cn

Xinlin Lei
College of Computer Science and Technology, Jilin University, Changchun 130012, China;

Tie Bao
College of Computer Science and Technology, Jilin University, Changchun 130012, China;

Design and Implementation of A Hybrid Solver on CPU and GPU Multi-target Machines

Abstract

Abstract: The traditional parallel solving methods for the ordinary differential equations mainly include the task-oriented parallelism and the method-oriented parallelism. However, these two solving algorithms have serious shortcomings, which can only use CPU resource or just design for the homogeneous form of ODE(ordinary differential equations) clusters. *By using RIDC(revisionist integral deferred correction) algorithm, a hybrid solver based on CPU and GPU multi-target machine is designed, which solves the differential equation system based on the pipeline form. Meanwhile, the parallel calculation within a single equation group and between the different equation groups is realized, which can give full play to the multi-core advantage of GPU, and also help to balance the load inside the computing node.* The simulation experiments verify the efficiency, accuracy and precision of the framework.

Keywords

ordinary differential equation, hybrid solver, multi-target machine, CPU, GPU

Recommended Citation

Lin Ma, Xuesong Zhang, Xinlin Lei, Tie Bao. Design and Implementation of A Hybrid Solver on CPU and GPU Multi-target Machines[J]. Journal of System Simulation, 2022, 34(4): 670-678.

第 34 卷第 4 期 系统仿真学报© Vol. 34 No. 4
2022 年 4 月 Journal of System Simulation Apr. 2022

面向CPU、GPU多目标机的混合求解器设计与实现

马琳，张雪松*，雷新丽，包铁
(吉林大学 计算机科学与技术学院，吉林 长春 130012)

摘要：传统常微分方程的并行求解方法主要包括面向任务的并行和面向方法的并行，但是这两种求解算法，只能利用CPU，或者只能面向同质形式的ODE(ordinary differential equations)簇，存在严重不足。*以RIDC(revisionist integral deferred correction)算法为基础，设计了一种面向CPU、GPU多目标机的混合求解器，基于流水线形式求解微分方程组，实现了单个方程组的内部和不同方程组之间的并行计算，进而能够充分发挥GPU的多核优势，有利于计算节点内部的负载均衡。*仿真实验验证了框架的效率、准确率和精准度。

关键词：常微分方程；混合求解；多目标机；CPU；GPU
中图分类号：TP391.9 文献标志码：A 文章编号：1004-731X(2022)04-0670-09
DOI: 10.16182/j.issn1004731x.joss.21-1317

Design and Implementation of A Hybrid Solver on CPU and GPU Multi-target Machines

Ma Lin, *Zhang Xuesong**, *Lei Xinlin*, *Bao Tie*
(College of Computer Science and Technology, Jilin University, Changchun 130012, China)

Abstract: The traditional parallel solving methods for the ordinary differential equations mainly include the task-oriented parallelism and the method-oriented parallelism. However, these two solving algorithms have serious shortcomings, which can only use CPU resource or just design for the homogeneous form of ODE(ordinary differential equations) clusters. *By using RIDC(revisionist integral deferred correction) algorithm, a hybrid solver based on CPU and GPU multi-target machine is designed, which solves the differential equation system based on the pipeline form. Meanwhile, the parallel calculation within a single equation group and between the different equation groups is realized, which can give full play to the multi-core advantage of GPU, and also help to balance the load inside the computing node.* The simulation experiments verify the efficiency, accuracy and precision of the framework.

Keywords: ordinary differential equation; hybrid solver; multi-target machine; CPU; GPU

引言

传统的常微分方程的并行求解方法主要包括2种形式：①面向任务的并行[1-6]。将同一ODE(ordinary differential equations)方程组的多个实例派发到GPU的流式多处理器上，除了初值不同外，各ODE的求解逻辑过程和其他参数配置都是完全相同的，这种并行处理依赖于GPU的单指令多数据集特征，要求所有的运算完全是同质的，才能并行执行；②面向方法的并行。采用多分辨率的求解方法，在低分辨率上串行完成ODE方程组的初值估算，再在高分辨率上利用估算的初值做并行求解，然后再次进行串行估算，最后进行结果误差修正，通过上述过程的多次迭代，直至满足求解精度要求。这种并行处理

收稿日期：2021-12-20 修回日期：2022-02-16
基金项目：国家重点研发计划(2018YFB1701600)
第一作者：马琳(1995-)，男，硕士生，研究方向为并行计算与复杂系统仿真。E-mail：malin20@mails.jlu.edu.cn
通讯作者：张雪松(1974-)，男，博士，副教授，研究方向为并行计算与复杂系统仿真。E-mail：wetting@jlu.edu.cn

受限于单个ODE方程组，计算规模较小，且需要串行、并行结合，难以发挥GPU的优势，目前主要以多线程的方式在CPU上实现。

上述求解算法中，只能利用CPU，或者只能面向同质形式的ODE簇，存在严重不足，主要体现在2个方面：

(1) 求解问题领域受限。传统方法面对的是具有同质特征的ODE，只能应用于大规模同质模型实例，如群体行为演化、分布式训练仿真等。但对于绝大多数仿真模型来说，其内部的数学模型描述形式是各不相同的，尤其是面向工业领域的复杂产品，如汽车，其内部包含发动机、液压、传动、刹车等多种不同子模型，无法构造空间离散的并行求解算法；

(2) GPU利用率不高或无法有效利用。在基于任务的并行划分方法中，如果包含的同质ODE方程组的数量较少，则模型中可并行执行的部分只占很小比例，绝大多数时间下，GPU的负载很少；而在多分辨率的并行方法中，计算任务完全依赖于CPU，GPU处于空闲状态，无法提供算力支撑。这2种方法都会导致计算节点内部计算资源的负载不均衡。尤其是面对复杂ODE的隐式求解时，其计算过程内部涉及循环迭代，如果无法有效利用GPU的计算资源，每个全局时间步的计算都会耗费大量CPU时间，延迟仿真推进速度，降低仿真效率。

因此，本文设计了一种面向CPU、GPU多目标机的混合求解器，具有如下特征：

(1) 利用基于时域的延迟校正方法。以CPU多线程或GPU多线程的形式对单个ODE方程组并行求解，当求解精度与并行线程的数量对应时，求解时间呈线性变化，即如果采用p个线程并行求解p阶精度问题，求解时间与单线程求解一阶精度时间近似；

(2) 支持异构ODE方程组的并行求解。通过CPU线程池或GPU Stream形式的任务封装，能够同时求解多个复杂ODE方程组(受限于硬件设备的并行能力)，进而支持复杂仿真模型间的实时耦合与交互；

(3) 同时实现了RIDC(revisionist integral deferred correction)形式的显式欧拉和隐式欧拉求解算法，能够对刚性问题模型求解提供有效支持；

(4) 面向大规模异构方程组求解时，支持面向CPU、GPU求解任务的任意比例划分与负载均衡。

1 背景介绍

随着硬件制造工艺的限制，处理器的时钟频率接近瓶颈，多核已经成为当今计算芯片的“横向扩展”标准，面向多核的任务划分与求解自然成为目前实时计算领域的热点问题。基于此，时域并行积分方法也再次受到关注。在传统分偏微分方程求解过程中，时域通常不用于并行化，但当空间上的并行化达到饱和时，时域提供了进一步并行化的方向。然而，由于时域方向前后间存在因果关系限制，后期求解要受到前期解的约束或影响。因此，时域并行算法与空间并行算法相比存在较大不同，通常需要在时间维度上进行迭代。

关于时域并行求解常微分方程的初值问题(ODE-IVP)，可以大致分为3种[7]：①问题域并行。将要求解的问题划分为一系列可被并行执行的子问题，通过迭代过程实现子问题间的耦合，如波形松弛法[8]；②步骤并行。时域被分解为多个子时间区域，在这些子时域上并行求解，如Parareal方法[9-10]，通过交替应用粗粒度顺序求解器和细粒度并行求解器达到加速的目的；③方法并行。在每个积分步内并行积分(多个函数同时求值)，由于其受限于具体的求解函数本身固有的特征，一般只适用于小规模并行，如多阶龙格库塔法的每个积分步内并行[11-13]。也可以使用“预测–修正”框架来生成方法并行的时间积分器，如平行外推方法[14]和RIDC积分器[15-16]。

Dutt等[17]提出的谱延迟校正(spectral deferred correction, SDC)通过求解误差方程的积分公式，对

DOI: 10.16182/j.issn1004731x.joss.21-1317

第 34 卷第 4 期 系统仿真学报 Vol. 34 No. 4
2022 年 4 月 Journal of System Simulation Apr. 2022

近似解进行迭代校正。这种积分公式形式克服了传统差分延迟校正方法的稳定性问题。SDC本身是一种串行方法，而RIDC是SDC的一种形式化变形。通过将连续计算流水线化，达到并行校正的目的。与使用Gauss-Lobatto节点的谱延迟校正不同，RIDC使用均匀间隔的节点来最小化内存占用，并允许嵌入高阶积分器。SDC和RIDC方法的基本思想是将原始初值问题(initial value problem, IVP)转换为求解相关的误差初值问题，进而逐步校正原始IVP解中的数值误差。并行性体现在同时求解原始IVP和相关误差IVPs上，即各阶误差校正计算是并行的。

1.1 误差初值

常微分方程的初值问题(ODE-IVP)可表示为

$$\begin{cases} y'(t)=f(t,y), \ \ t\in[0,T] \\ y(0)=y_0 \end{cases} \tag{1}$$

式中：t 为时间变量；y_0 为该方程在 t 为0时候的初值。

将式(1)的精确解表示为 $y(t)$，近似解表示为 $u(t)$，其中 $u(0)=y(0)$。则近似解的误差为 $e(t)=y(t)-u(t)$。将残差(有时也叫做缺陷)定义为 $r(t)=u'(t)-f(t,u)$，则误差的时间导数满足

$$e'(t)=y'(t)-u'(t)=f(t,u+e)-f(t,u)-r(t) \tag{2}$$

由于 $e(0)=u(0)-y(0)=0$，式(2)代表了相关误差的IVP。出于稳定性的考虑，将误差IVP写为积分形式[11]：

$$\left(e+\int_0^t r(\tau)\,\mathrm{d}\tau\right)'=f(t,u+e)-f(t,u) \tag{3}$$

如果式(3)通过数值方法求解，校正后的近似值 $u+e$ 仍然是一个近似值，本文采用了更一般的表示法，这种方法将允许迭代校正解，直到达到所需的精度。如果将初始近似值表示为 $u^{[0]}$、第 p 次近似表示为 $u^{[p]}$，$u^{[p]}$ 的误差表示为 $e^{[p]}$。那么，误差方程可以改写为

$$\left(e^{[p]}+\int_0^t r^{[p]}(\tau)\mathrm{d}\tau\right)'=f(t,u^{[p]}+e^{[p]})-f(t,u^{[p]}) \tag{4}$$

式中：$r^{[p]}=u^{[p]}(t)'-f(t,u^{[p]})$。

1.2 离散化

用代数方法，可以将式(4)进行基于显式方法的一阶离散化：

$$u_{n+1}^{[p+1]}=u_n^{[p+1]}+\Delta tf\left(t_n,u_n^{[p+1]}\right)-\Delta tf\left(t_n,u_n^{[p]}\right)+\int_{t_n}^{t_{n+1}} f\left(\tau,u^{[p]}\right)\mathrm{d}\tau \tag{5}$$

类似地，式(4)的隐式方法的一阶离散化解为

$$u_{n+1}^{[p+1]}=u_n^{[p+1]}+\Delta tf\left(t_{n+1},u_{n+1}^{[p+1]}\right)-\Delta tf\left(t_{n+1},u_{n+1}^{[p]}\right)+\int_{t_n}^{t_{n+1}} f\left(\tau,u^{[p]}\right)\mathrm{d}\tau \tag{6}$$

在式(5)~(6)中，需要一个足够精确的正交矩阵来近似当前的积分[11]。如果使用一阶预测器来计算式(1)的近似解，并且使用式(5)~(6)形式的一阶校正器，则可以用式(7)~(8)来逼近求解积分：

$$\int_{t_n}^{t_{n+1}} f\left(\tau,u^{[p]}\right)d\tau\approx\begin{cases}\sum_{v=0}^{p+1}\alpha_{pv}f\left(t_{n+1-v},u_{n+1-v}^{[p]}\right), n\geqslant p\\ \sum_{v=0}^{p+1}\alpha_{pv}f\left(t_{v},u_{v}^{[p]}\right), n<p\end{cases} \tag{7}$$

式中：α_{pv} 为正交权重。

$$\alpha_{pv}=\begin{cases}\int_{t_n}^{t_{n+1}}\prod_{i=0,i\neq v}^{p+1}\dfrac{t-t_{n+1-i}}{t_{n+1-v}-t_{n+1-i}}\mathrm{d}t, n\geqslant p\\ \int_{t_n}^{t_{n+1}}\prod_{i=0,i\neq v}^{p+1}\dfrac{t-t_i}{t_v-t_i}\mathrm{d}t, n<p\end{cases} \tag{8}$$

α_{pv} 的第一部分对应 $n\geqslant p$ 情况下的外推计算，第二部分对应 $n<p$ 情况下的内插计算。一般 n 最大不超过 p 的4倍，否则会导致计算结果误差增大。

1.3 收敛性

关于RIDC算法的收敛性详细证明可参见文献[18]。

定理1：假定式(1)中的 $f(t,y)$ 和 $y(t)$ 足够光滑，RIDC算法在 $k>M+1$ 个均匀分布节点下，嵌入 r_0 阶龙格库塔法作为预测步，分别嵌入 $r_i\,(1\leqslant i\leqslant p)$ 阶龙格库塔法作为修正步，则局部截断误差为 $O(h^{SM+1})$，其中 h 为步长，$SM=\sum_{j=0}^{M}r_i$。

由此可知，当 p 阶RIDC算法的各阶全部嵌入

一阶欧拉算法时，最终截断误差为$O(h^p)$。

2 软件与算法设计

2.1 总体框架

本文设计并实现了基于流水线形式的面向CPU、GPU目标机的仿真模型求解任务分配框架，来并行求解常微分方程组。这种并行体现在2个方面：①采用RIDC算法，以多线程形式进行单个方程组的内部并行求解；②方程组之间的求解过程也是并行的，与传统同构多实例方程组形式的GPU求解不同，本框架还支持异构形式的ODE方程组并行求解，能够充分发挥GPU的计算能力。该框架同时还具有计算资源目标动态分配的能力。对于单个ODE方程组，求解形式包括CPU单线程、CPU多线程、GPU多线程3种模式。当采用CPU求解时，线程数量与求解精度间的对应关系既可以是一对一，也可以是一对多，即一个线程内部的求解任务可以对应RIDC算法内部的多轮修正；采用GPU求解时，为发挥warp内部的并行能力，求解精度与线程之间采用一对一的关系。此外，本框架的内部实现技术采用了“预测–修正”形式的迭代模式，在计算流程上可以更好地支持许多迭代法的并行，如RIDC、Parareal等，尤其是对于那些需要低精度串行，高精度并行的求解算法，CPU、GPU混合运算模式更具负载均衡优势，还能获得较好的加速比。

该框架的核心思想是动态生成ODE求解任务，并根据求解设置调度到对应的硬件。如图1所示，该框架包括3个主要组成部分：任务生成器、调度器和同步器。当新的ODE求解需求出现时，任务生成器根据求解参数构建数据结构和即时编译(目前只针对GPU代码)例程，并将该任务放入ODE任务池的任务队列中。调度程序根据不同的队列设置，将某个队列中的任务派发给CPU、GPU或同时分配给两者。对于同时需要CPU和GPU的求解任务，同步器负责CPU和GPU之间的数据交换。同时，GPU被进一步划分为若干子设备(在CUDA (compute unified device architecture)中对应为Stream)，每个子设备只负责一个ODE方程组的求解，或一个ODE方程组内部的并行修正计算过程。而在CPU方面，取决于具体的求解配置，任何一个ODE方程组的求解任务都可以被分配到一个或多个线程(核心)，具有更大的灵活性。

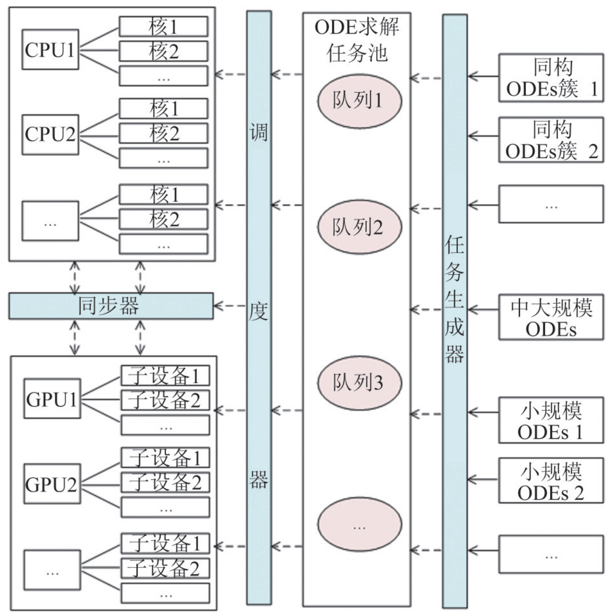

图1 流水线常微分方程求解框架
Fig. 1 Pipeline ODE solving framework

本框架支持的任务类型包括3种：①小规模的单个ODE方程组，各方程组对应了不同的模型，即求解任务各不相同；②同构类型的ODE方程组簇，即同一模型的多个不同实例，此种类型可以充分发挥GPU的并行优势；③是较大型的复杂ODE方程组(上万个状态变量)，此时的求解需要用到GPU的多个流处理器。

2.2 CPU实现

为了避免高精度求解时等距节点积分产生龙格现象，本文实现的求解方法还支持Gauss-Lobatt等非等距节点形式的积分，因而在初始化过程没有采用最小内存占用的空间优化形式。对于N阶精度需求，各阶校正节点间的计算依赖关系如图2所示。

https://dc-china-simulation.researchcommons.org/journal/vol34/iss4/2
DOI: 10.16182/j.issn1004731x.joss.21-1317

第 34 卷第 4 期　　系统仿真学报　　Vol. 34 No. 4
2022 年 4 月　　Journal of System Simulation　　Apr. 2022

高阶校正过程中的每个节点求解都要依赖于较低一阶内部特定数目的节点值，具体依赖数目随着求解精度的不同而不同。图3展示了采用等距节点形式，四阶精度求解的初始化过程，从步骤12开始，4个线程开始以流水线形式并行执行，每一时间步的最终求解结果由$I=3$所对应的校正线程输出。

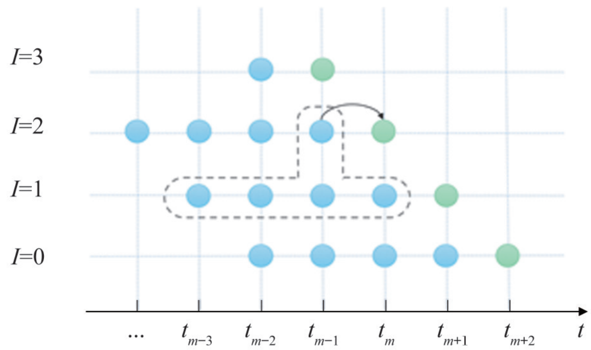

图2　各阶节点依赖关系
Fig. 2　Node dependencies of each order

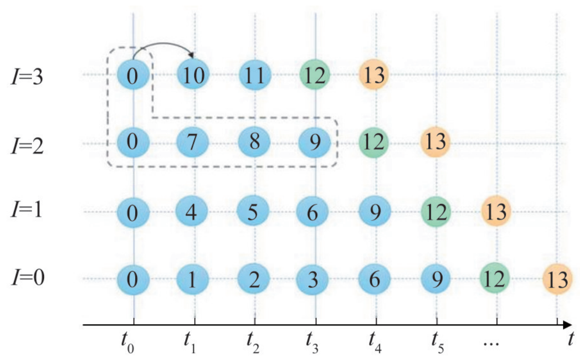

图3　节点计算顺序
Fig. 3　Node calculation sequence

通过图3的基本计算流程可知：假设求解节点有k个，求解精度为p阶，并采用p个核并行求解，则RIDC算法与一阶算法的计算量比值为$r=1+M^2/k$。其中，$M=p-1$，即迭代校正的次数。由此可以计算出四阶RIDC嵌入欧拉与一阶欧拉之间的理论求解时间比为$1+9/k$，随着k的增大，RIDC时间趋近于一阶欧拉时间。但随着k的增大，外推导致的求解误差也会增大。一般可以取k为20~30倍的M值，既提高了效率，又保证了一定的精度。

CPU并行求解具体算法如下：

算法 1　CPU并行求解

输入：

求解形式(显式或隐式)，ODE方程组函数指针，雅可比矩阵函数指针(隐式求解需要)，求解精度(阶数)ORDER，并行线程个数NT，状态变量初值X0，起止时间及步长step

输出：

当前step对应的状态变量值X

初始化：

```
initialize static thread pool with NT threads
for thread 1 do
    for i = 1 to ORDER do
        if i == 1 then
            predict state variables X from step 1 to step (ORDER -1) * (ORDER -1)
        else
            correct state variables X from step 1 to step (i -1) * (ORDER -1)
        end if
    end for
end
```

CPU线程并行求解

```
assign thread 1 with prediction task(order 1)
assign each thread in the pool with some specific correction task(order >1) in a interleaved fashion
for each thread in the pool
    if order == 1 then
        predict state variables X from current step to step current + ORDER -1
    else
        correct state variables X from current step to step current + ORDER -1
    end if
end for
barrier.wait()
```

21: for each thread in the pool

22: left shift state variables X in the memory with ORDER -1 position

23: end for

24: barrier.wait()

25: output state variables X of current time step

CPU求解算法支持灵活的线程分配模式，既可以采用单线程串行形式，也可以采用多线程并行形式，求解线程的最大并行总数等于求解精度的阶数P。采用最大并行模式能够减少计算时间，单步递进的墙上时间接近其他同阶串行算法的$1/P$。

2.3 GPU实现

算法2 GPU并行求解

输入：

求解形式(显式或隐式)，ODE方程组global函数指针，雅可比矩阵global函数指针(隐式求解需要)，求解精度(阶数)ORDER，状态变量初值X0，起止时间及步长step

输出：

当前step对应的状态变量值X

初始化：

1: dispatch one CPU thread in the dynamic thread pool as a worker

2: initialize host and device memory needed (include global and local device memory)

3: copy state variables X0 to device memory

4: create GPU streams(one stream for explicit solver, or ORDER streams for implicit solver)

并行求解：

5: for each GPU thread in the wrap(wrap number = ORDER)

6: if thread id == 1 then

7: predict initial value of state variables X by explicit solver

8: if implicit solver then

9: call Newton-Raphson method to get a refined value in an iterative manner

10: end if

11: loop from current step to current + ORDER -1 step

12: else

13: predict initial value of state variables X by explicit solver and quadrature marix

14: if implicit solver then

15: call Newton-Raphson method to get a refined value in an iterative manner

16: end if

17: loop from current step to current + ORDER -1 step

18: end if

19: end for

20: __syncthreads()

21: for each thread in the wrap

22: left shift state variables X in the memory with ORDER -1 position

23: end for

24: __syncthreads()

25: output state variables X of current time step

在GPU求解算法中，根据显式、隐式算法选择的不同，流程内部包含了多个不同核函数的调用，为了简洁，算法2中并没有体现出来。GPU内部的并行模式又细分为2种情况：①对于显式算法和隐式算法中的非牛顿迭代部分，GPU线程的并行数量与求解精度阶数P一致，计算过程由单个CUDA流完成；②对于隐式求解中的牛顿迭代部分，由于要同时求解P个不同的线性方程组，需要将每个方程组的求解分配到不同的Stream中，待所有Stream执行完毕，再重新转入第1种并行模式。

2.4 异构方程组并行求解

当前求解器的设计以ODE方程组为单位，无论是异构方程组，还是同构方程组的多个不同实例，都对应一个求解器实例。因此，与传统GPU

DOI: 10.16182/j.issn1004731x.joss.21-1317

第 34 卷第 4 期 系统仿真学报 Vol. 34 No. 4
2022 年 4 月 Journal of System Simulation Apr. 2022

求解方式不同，本求解器可以支持大规模异构ODE方程组的CPU、GPU并行求解或混合求解，具体并行求解数量仅受限于CPU、GPU本身的硬件资源容限。算法3展示了面向多个异构方程组的GPU并行调度伪码，CPU线程池中的每个线程负责特定ODE方程组的GPU求解流程调度，并在全局时间步进行回调，完成状态变量输出或其他形式的模型间数据交互。

算法3 异构ODE或多实例ODE GPU并行求解

```
1: initialize static thread pool with NT threads
2: for each thread in the thread pool
3:       assign each thread a GPU ODE solver
4:       initialize GPU ODE solver
5: end for
6: parallel executing GPU solver
7: callback on each global step
8: readpool.wait()
```

3 实验结果

本文设计了一个基于流水线形式的面向CPU、GPU目标机的仿真模型求解任务分配框架，为了验证模型性能，分别对基于显式欧拉算法、隐式欧拉算法、龙格库塔法[19]的CPU单线程程序和GPU多线程程序进行常微分方程组求解测试，并对比求解时间。

为验证程序的准确性，在实验1中，比较了CPU和GPU的显式欧拉算法、隐式欧拉算法、龙格库塔法误差率，如表1所示。从表1可见本文算法准确度在误差范围内。

表1 CPU和GPU算法误差比较
Table 1 CPU and GPU algorithm error comparison %

处理器	显式欧拉	龙格库塔	隐式欧拉
CPU	0.03	0	0.02
GPU	0	0	0.01

在实验2中，将方程数量设置为2，分别设立100、1 000、10 000、100 000个采样点，在CPU单线程实验中使用四阶显式欧拉算法和四阶龙格库塔法，其中龙格库塔法的使用通过调用boost odeInt开源库实现，GPU多线程实验中使用显式欧拉算法，结果如表2所示。

表2 两方程CPU和GPU显式算法时间比较
Table 2 2 equations on CPU and GPU explicit algorithmtime comparison s

采样点数(方程数2)	CPU单线程显式欧拉4阶	CPU单线程RK4阶	GPU显式欧拉算法
100	0.018 8	0.006 4	0.026 6
1 000	0.151 4	0.052 0	0.221 8
10 000	1.854 2	0.628 4	2.063 9
100 000	18.395 6	6.258 4	20.161 0

在实验3中，将方程数量设置为10，重复实验2中的实验，所得结果如表3所示。

表3 十方程CPU和GPU显式算法时间比较
Table 3 10 equations on CPU and GPU explicit algorithmtime comparison s

采样点数(方程数10)	CPU单线程显式欧拉4阶	CPU单线程RK4阶	GPU显式欧拉算法
100	0.031 0	0.012 8	0.049 8
1 000	0.304 4	0.130 4	0.451 6
10 000	3.038 3	1.298 8	4.312 1
100 000	30.305 6	12.986 4	43.526 8

在实验4和实验5中，同样分别将方程数量设置为2和10个，测试CPU和GPU在采用隐式欧拉算法时的性能差异。具体结果如表4~5所示。

表4 两方程CPU和GPU隐式算法时间比较
Table 4 2 equations on CPU and GPU implicit algorithm time comparison s

采样点数(方程数2)	CPU单线程隐式欧拉4阶	GPU隐式欧拉4阶	CPU单线程隐式欧拉8阶	GPU隐式欧拉8阶
100	0.814 6	0.286 2	4.334 6	0.933 6
1 000	5.588 4	2.644 8	31.725 0	8.930 8

表5 十方程CPU和GPU隐式算法时间比较
Table 5 10 equations on CPU and GPU implicit algorithm time comparison s

采样点数(方程数10)	CPU单线程隐式欧拉4阶	GPU隐式欧拉4阶	CPU单线程隐式欧拉8阶	GPU隐式欧拉8阶
100	13.498 4	0.671 8	69.640 8	2.132 2
1 000	85.591 6	6.331 6	483.671 8	20.578 2

实验2~3的结果显示，显式欧拉GPU并行迭代求解要比单线程迭代或RK4阶算法慢。这主要是是由于GPU的计算优势在大数据量、高并发情况下才能体现出来。对于单个方程组来说，并行线程的最大数量受限于求解精度，即只有4个线程。由于并发数量过小，严重制约了GPU的计算效率。但GPU具有更多的流处理器，当并行求解的方程组数量达到一定程度时，尽管每个流处理器内部的并发程度不高，多个流处理器并行仍然要比CPU更具优势。

从实验4~5的结果可以看出，隐式求解中，随着方程组中方程个数的增加，GPU的优势逐渐体现出来。这主要是由于算法中用到了拟牛顿法求解非线性方程组，无论是在不同精度阶之间，还是非线性方程组内部，这些计算过程都是并行的，即拟牛顿法的并行度要大于阶数，且随着方程组内部方程数量的增加，并行度会进一步提升。由此也可看出，非线性方程组求解耗费了数值求解的绝大部分时间。因此，在求解刚性方程组时，GPU的能力更易被有效利用。

通过上述5个实验，可以发现本文设计的模型框架对常微分方程组求解效率有显著的提高。对于显式欧拉算法，GPU求解时间接近CPU单线程求解速度；对于隐式欧拉算法，GPU求解时间小于CPU单线程求解速度，且随着方程数量和求解阶数的增加，本文模型效果提升更加明显。此外，CPU核心数目一般只有16，而GPU核心数目高达上万，是CPU核心数目的上百倍。因此，认为本文提出的基于流水线形式的面向CPU、GPU目标机的仿真模型求解任务分配框架能大幅度提高常微分方程的求解效率。

4　结论

本文设计并实现了一种基于流水线形式的面向CPU、GPU目标机的仿真模型求解任务分配框架，在并发求解异构常微分方程方程组时，能够有效减少求解时间，提高求解效率。尤其是面对复杂仿真系统时，可以充分发挥GPU的多核优势，有利于计算节点内部的负载均衡。通过5个实验，验证了框架的效率、准确率和精准度。未来我们将通过引入内存池、计算流预分配、方程组间设备内存数据交换方法等，进一步提高框架的运算效率和并发能力。

参考文献：

[1] Kindratenko V. Numerical Computations with GPUs[M]. Springer International Publishing, 2014: 159-182.

[2] Ahnert K, Demidov D, Mulansky M. Solving Ordinary Differential Equations on GPUs[M]. Springer International Publishing, 2014: 125-157.

[3] Stone C, Davis R. Techniques for Solving Stiff Chemical Kinetics on GPUs[C]//51st AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition. Texas: AIAA, 2013: 369.

[4] Sewerin F, Rigopoulos S. A Methodology for the Integration of Stiff Chemical Kinetics on GPUs[J]. Combustion and Flame(S0010-2180), 2015, 162(4): 1375-1394.

[5] Curtis N J, Niemeyer K E, Sung C J. An Investigation of GPU-Based Stiff Chemical Kinetics Integration Methods [J]. Combustion and Flame(S0010-2180), 2017, 179: 312-324.

[6] Stone C P, Alferman A T, Niemeyer K E. Accelerating Finite-Rate Chemical Kinetics with Coprocessors: Comparing Vectorization Methods on GPUs, MICs, and CPUs[J]. Computer Physics Communications(S0010-4655), 2018, 226: 18-29.

[7] Burrage K. Parallel Methods for ODEs[J]. Advances in Computational Mathematics (S1019-7168), 1997, 7(1/2): 1-31.

[8] Vandewalle S, Roose D. The Parallel Waveform Relaxation Multigrid Method[C]//Third SIAM Conference on Parallel Processing for Scientific Computing. Los Angeles: SIAM, 1987: 152-156.

[9] Lions J L, Maday Y, Turinici G. A "parareal" in Time Discretization of PDE's[J]. Comptes Rendus De l Académie des Sciences-Series I-Mathematics(S0764-4442), 2001, 332 (7): 661-668.

[10] Gander M J, Vandewalle S. On the Superlinear and Linear Convergence of the Parareal Algorithm[M]// Domain Decomposition Methods in Science and Engineering XVI. Springer, Berlin: Heidelberg, 2007:

DOI: 10.16182/j.issn1004731x.joss.21-1317

第 34 卷第 4 期 系统仿真学报 Vol. 34 No. 4
2022 年 4 月 Journal of System Simulation Apr. 2022

291-298.

[11] Miranker W L, Liniger W. Parallel Methods for the Numerical Integration of Ordinary Differential Equations [J]. Mathematics of Computation(S0025-5718), 1967, 21 (99): 303-320.

[12] Enenkel R F. DIMSEMS: Diagonally Implicit Single-Eigenvalue Methods for the Numerical Solution of Stiff Ordinary Differential Equations on Parallel Computers [D]. Toronto: University of Toronto, 1997.

[13] Ketcheson D, Waheed U B. A Comparison of High Order Explicit Runge-Kutta, Extrapolation, and Deferred Correction Methods in Serial and Parallel[J]. Mathematics (S2227-7390), 2013, 9(2): 175-200.

[14] Kappeller M, Kiehl M, Perzl M, et al. Optimized Extrapolation Methods for Parallel Solution of IVPs on Different Computer Architectures[J]. Applied Mathematics and Computation(S0096-3003), 1996, 77(23): 301-315.

[15] Christlieb A, Ong B, Qiu J M. Integral Deferred Correction Methods Con-Structed with High Order Runge-Kutta Integrators[J]. Mathematics of Computation (S0096-3003), 2010, 79(270): 761-783.

[16] Christlieb A, Ong B. Implicit Parallel Time Integrators [J]. Journal of Scientific Computing(S0885-7474), 2011, 49(2): 167-179.

[17] Dutt A, Greengard L, Rokhlin V. Spectral Deferred Correction Methods for Ordinary Differential Equations [J]. BIT Numerical Mathematics(S0006-3835), 2000, 40 (2): 241-266.

[18] Christlieb A J, Macdonald C B, Ong B W. Parallel High-Order Integrators[J]. SIAM Journal on Scientific Computing(S1064-8275), 2010, 32(2): 818-835.

[19] Simos T E, Tsitouras C. On High Order Runge-Kutta-Nyström Pairs[J]. Journal of Computational and Applied Mathematics, 2022, 400: 113753.