DOI：10. 13471/j. cnki. acta. snus. 2022A064

面向边缘计算的神经网络MPS加速算法*

王卓霖[1]，江俊扬[1]，杨耿超[1]，姚清河[1]，蒋子超[1]，张仪[2]

1. 中山大学航空航天学院，广东 深圳 518107
2. 特文特大学机器人和机电一体化系，荷兰 恩斯赫德 7522NB

摘　要：在无网格法中，移动粒子半隐式方法（MPS）利用流体的不可压性构造泊松方程，在获得了压力求解准确性的同时也带来了巨大的计算量，导致其不适合实现大规模流体模拟. 针对上述问题提出新型算法NN-MPS，将泊松方程的求解转化为利用神经网络求解回归问题. NN-MPS算法通过构建每一步的流场特征与压力的预测模型，实现泊松方程的快速求解. 本文进一步将基于NN-MPS算法的泊松方程求解过程移植到Atlas 200 DK上，实现边缘侧加速求解泊松方程. 本文采用多种溃坝模型进行数值实验，结果表明，本文的MPS加速方法具有低成本、高速度且较少精度损失的特点，求解速度实现了一定的提升. 本文同时也验证了边缘计算设备在计算流体力学领域应用的可行性.

关键词：边缘计算；神经网络；NN-MPS；计算流体力学

中图分类号：O3　　**文献标志码：**A　　**文章编号：**2097－0137（2023）05－0067－11

An MPS algorithm accelerated by neural network on edge computing

WANG Zhuolin[1]，*JIANG Junyang*[1]，*YANG Gengchao*[1]，*YAO Qinghe*[1]，*JIANG Zichao*[1]，*ZHANG Yi*[2]

1. School of Aeronautics and Astronautics，Sun Yat-sen University，Shenzhen 518107，China
2. Department of Robotics and Mechatronics，University of Twente，Enschede，7522 NB，the Netherlands

Abstract：As a meshless method，the moving particle semi-implicit（MPS）method forms the pressure Poisson equation by using the incompressibility of fluid，which not only obtain the accuracy of pressure but also bring a high cost of calculation. Therefore，it is not appropriate for MPS to solve large-scale fluid simulations. IA new algorithm NN-MPS is proposed to solve the above problems，which transforms the solution of the Poisson equation into a regression problem by using neural network. The NN-MPS algorithm realizes the quick solution of the Poisson equation by constructing the prediction model of flow field features and pressure at each step. In this work，the procedure of solving the pressure Poisson equation is further transported to Atlas 200 DK device for a faster speed of solving procedure. Results show that the acceleration method of MPS mentioned in this work has the characteristics of low cost，high speed，and low accuracy loss，and the solution speed has been improved. We also verified the feasibility of applying the edge computing device in the field of CFD.

Key words：edge computing；neural network；NN-MPS；CFD

* **收稿日期：**2022－07－27　　**录用日期：**2023－01－03　　**网络首发日期：**2023－06－26
基金项目：国家自然科学基金（11972384）；广东省基础与应用基础研究基金（2021B1515310001）；
国家重点研发计划（2020YFA0712502）
作者简介：王卓霖（1998年生），男；**研究方向：**计算流体力学；E-mail：wangzhlin3@mail2.sysu.edu.cn
通信作者：姚清河（1980年生），男；**研究方向：**偏微分方程数值解；E-mail：yaoqhe@mail.sysu.edu.cn
杨耿超（1991年生），男；**研究方向：**山地灾害动力学；E-mail：yanggch8@mail.sysu.edu.cn
（姚清河、杨耿超为共同通信作者）

计算流体力学（CFD，computational fluid dynamics）是当前流体力学领域中十分重要的研究方法，通过数值求解流体运动的控制方程组，研究流体流动的物理现象. 目前主流计算流体方法主要分为两类：基于网格的欧拉方法和基于粒子的拉格朗日方法. 欧拉方法如有限体积法（FVM）、有限元法（FEM）、有限差分法（FDM）通过对计算区域划分网格，离散微分方程，获得网格点各时刻的物理量变化；拉格朗日方法则研究粒子的物理量随时间的变化，适用于比较复杂的边界情况（Chikazawa et al.，2001）或者大变形流体（Gotoh et al.，2006）模拟场景.

近年来拉格朗日方法得到发展，SPH（Monaghan，1992）、MPS（Koshizuka et al.，1996）等粒子方法逐渐受到CFD界重视. 其中MPS（moving particle semi-implicit ）最早由Koshizuka et al.于1996年提出，并由于其准确性和稳定性等特点受到了学术界的广泛关注. 随着MPS理论的进一步发展，Khayyer et al.（2008）提出了CMPS方法修正了梯度算子导致的动量不守恒，增加了自由表面粒子的稳定性，通过改进PPE方程和修正矩阵提出了MPS-HS方法、MPS-HL方法、ECS方法、GC方法等，降低了计算的不稳定性；Tsuruta et al.（2013）提出了MPS-DS方法修正了斥力模型，进一步增强了计算的稳定性；Tsuruta et al.（2015）提出了SPP方法通过引入空间势能粒子对非物理空腔问题做了优化；越来越多的修正方法强化了MPS的优越性，抑制了压力振荡，使其被大量应用于涉及复杂边界、大变形、多相流等复杂流体问题中（Shibata et al.，2007；张雨新等，2012；杨超，2018）.

MPS采用预估和修正（显式和隐式）两个步骤来求解流体控制方程（Koshizuka et al.，1996），预估步中通过体积力与黏性力显式更新速度和位移；在校正步中通过保持粒子数密度不变构造压力泊松方程，隐式求解下一时间步的压力，并利用压力梯度修正速度和位移. 然而，这种隐式计算方法需消耗大量计算资源. 与显式计算压力状态方程的SPH方法相比，传统MPS方法难以高效并行求解，使其不适合大规模仿真计算，在实际工程中无法广泛应用（张弛等，2011）. 前期研究中，我们提出了一种基于数据驱动求解MPS压力泊松方程的NN-MPS算法，该算法将原始MPS中压力泊松方程的求解重新描述为回归问题，提取对泊松求解起主导作用的特征参数作为回归问题的输入，并引入神经网络（NN，neural network）解决该回归问题.

边缘计算（何腾，2020）是相对于云计算（崔勇等，2017）而言的一种概念，指在靠近数据源头的网络边缘侧对数据进行直接计算，而无需将数据传输至云端数据处理中心的一种数据处理方式. 随着人工智能的发展，在数据源头的离线设备上直接进行实时神经网络推理的需求增加，昂贵、大体积、大功率的传统计算硬件如专业级GPU（Lee et al.，2010）、CPU集群已经无法满足目前的需求，廉价、高性能的面向神经网络的边缘计算硬件成为研究热点. 因此，多种专供神经网络的专用集成电路（ASIC）芯片被研发出来，如寒武纪思元系列（Liu et al.，2015）、海思科技的昇腾910、谷歌的TPU（Jouppi et al.，2017）及本文将使用的Atlas 200 DK搭载的Ascend 310等都属于这类ASIC芯片. 在工业生产领域，常有在边缘侧对监测数据进行离线分析的场景（如地质灾害监测（杜年春等，2022）、空气质量监测（滕云豪，2022）），而由于边缘端算力限制，目前在边缘侧利用流体仿真进行数据分析的应用仍较为少见. 我们在NN-MPS中对神经网络的引入，不仅提高了在传统硬件上的计算速度，也为MPS方法在这些前沿的边缘计算硬件上的计算提供了可能.

本文使用电功耗、成本、体积等各方面较有优势，且在张量计算等神经网络计算有出色计算效率的边缘计算设备Atlas 200 DK作为加速硬件，成功实现了以新型边缘计算硬件作为加速设备的计算流体仿真. 本文研究用较低的成本在小规模与大规模粒子法仿真中达到了与传统CPU计算效率相似甚至有所提高的计算效果. 该研究为将来特殊场景下的快速流体仿真提供了一种可行的参考，意味着我们可以在一些需要低成本或无法使用专业计算集群或是在实时采集实际物理边界信息的边缘侧场景中，提供一定的快速流体动力学分析能力.

1 移动粒子半隐式（MPS）方法

1.1 控制方程

不可压缩流体的控制方程组由连续性方程和Navier-Stokes方程组成：

$$\frac{\mathrm{D}\rho}{\mathrm{D}t}=\frac{\partial\rho}{\partial t}+\nabla\cdot(\rho\boldsymbol{v})=0, \tag{1}$$

$$\frac{\mathrm{D}\boldsymbol{v}}{\mathrm{D}t}=-\frac{1}{\rho}\nabla P+\nu\nabla^2\boldsymbol{v}+\boldsymbol{f}, \tag{2}$$

其中ρ为流体密度，P为压力，$\boldsymbol{v}$为速度向量，$\boldsymbol{f}$为流体单位质量力，ν是运动黏度系数.

1.2　粒子作用模型

1.2.1　核函数　MPS方法中，核函数与光滑半径内邻域粒子相互作用，计算控制方程中的微分算子. Koshizuka et al.(1996)提出的核函数由如下表示：

$$W(r)=\begin{cases}\dfrac{r_e}{r}-1, & 0\leqslant r<r_e,\\ 0, & r\geqslant r_e,\end{cases} \tag{3}$$

式中r_e为粒子的有效半径，即光滑长度.

核函数值在靠近中心处最大，并随着r的增加减小. 因此，核函数在计算中起到了类似于权重函数的作用，使较远粒子对于当前粒子的影响较小. 在本节后面的部分中将展示由权重函数表示的微分离散计算方法.

1.2.2　粒子密度模型　MPS中粒子数密度$\langle n\rangle_i$是第i粒子在核函数控制域内与其周围粒子核函数数值的累加：

$$\langle n\rangle_i=\sum_{j=1}^{N}W\left(\left|\boldsymbol{r}_i-\boldsymbol{r}_j\right|,h\right), \tag{4}$$

其中N为粒子i的相邻粒子，$W(\left|\boldsymbol{r}_i-\boldsymbol{r}_j\right|,\ h)$为核函数，$\boldsymbol{r}_i$、$\boldsymbol{r}_j$分别为$i$、$j$粒子的位置矢量，$h$为光滑长度. 粒子数密度与流体密度成正比.

1.2.3　Laplacian模型　Laplacian算子采用如下形式：

$$\left\langle\nabla^2\phi\right\rangle_i=\frac{2D}{n^0\lambda_i}\sum_{j=1}^{N}\left[\left(\phi_j-\phi_j\right)W\left(\left|\boldsymbol{r}_j-\boldsymbol{r}_i\right|,h\right)\right], \tag{5}$$

式中D为计算维数，

$$\lambda=\frac{\sum_{j=1}^{N}\left|\boldsymbol{r}_j-\boldsymbol{r}_i\right|^2W\left(\left|\boldsymbol{r}_j-\boldsymbol{r}_i\right|,h\right)}{\sum_{j=1}^{N}W\left(\left|\boldsymbol{r}_j-\boldsymbol{r}_i\right|,h\right)}. \tag{6}$$

1.3　预估-修正法

为求解由式(1)~(2)组成的方程组，MPS方法通过引入中间速度，将N-S方程分两步求解. 第一步通过外力信息和流体速度信息，显式求解中间速度$\boldsymbol{v}^*$，第二步通过压力梯度修正速度，即

$$\Delta\boldsymbol{v}=\Delta\boldsymbol{v}^*+\Delta\boldsymbol{v}', \tag{7}$$

其中

$$\begin{cases}\Delta\boldsymbol{v}^*=\Delta t*\left(\boldsymbol{f}^k+\nu\nabla^2\boldsymbol{v}^k\right),\\ \Delta\boldsymbol{v}'=-\dfrac{\Delta t}{\rho}\nabla P^{k+1}.\end{cases} \tag{8}$$

1.4　压力泊松方程(Pressure Poisson Equation)

传统MPS算法通过保持粒子数密度不变来实现流体的不可压缩性，利用中间密度ρ^*和中间速度$\boldsymbol{v}^*$，令下一时间步的速度散度为0、密度变化为0，可构造压力泊松方程，进而求解P^{k+1}. 参考Tanaka et al.(2010)、Lee et al.(2011)等的做法，本文采用基于散度的表达式：

$$\left\langle\nabla^2P^{k+1}\right\rangle_i=(1-\gamma)\frac{\rho}{\Delta t}\nabla\cdot\boldsymbol{v}_i^*-\gamma\frac{\rho}{(\Delta t)^2}\frac{\langle n\rangle_i-n^0}{n^0}, \tag{9}$$

其中右边第一项为基于散度为0构造泊松方程的源项、第二项为基于密度不变构造的泊松方程源项. γ为可变参数，本文中定义为0.2 .

1.5 时间积分

综上，MPS算法每个时间步的求解步骤如下：

（i）根据式(8)中黏性力和质量力获得中间速度$\boldsymbol{v}^*$，并计算中间位置$\boldsymbol{r}^*$；

（ii）根据式(4)计算中间粒子数密度n^*；

（iii）根据方程(9)求解压力泊松方程，得到P^{k+1}；

（iv）利用所求压力梯度获得修正速度$\boldsymbol{v}'$，并计算$\boldsymbol{v}^{k+1}$、$\boldsymbol{r}^{k+1}$.

2 基于神经网络的MPS算法（NN-MPS）

2.1 原始压力泊松方程

根据式(5)离散Laplacian模型，PPE左边项可以离散成如下形式：

$$\left\langle \nabla^2 P \right\rangle_i = \frac{2D}{n^0 \lambda}\left[\sum_{j=1}^{N} P_j \cdot W\left(\left|\boldsymbol{r}_j - \boldsymbol{r}_i\right|\right) - P_i \cdot \sum_{j=1}^{N} W\left(\left|\boldsymbol{r}_j - \boldsymbol{r}_i\right|\right)\right]. \tag{10}$$

代入式(9)，并改成矩阵形式可以表示为

$$\boldsymbol{A}\boldsymbol{P}^{k+1} = \boldsymbol{b}, \tag{11}$$

其中

$$\boldsymbol{A} = \frac{2D}{n^0 \lambda}\begin{bmatrix} -\sum_{j\neq 1} W\left(\left|\boldsymbol{r}_j - \boldsymbol{r}_1\right|\right) & \cdots & W\left(\left|\boldsymbol{r}_i - \boldsymbol{r}_1\right|\right) & \cdots & W\left(\left|\boldsymbol{r}_n - \boldsymbol{r}_1\right|\right) \\ \vdots & & \vdots & & \vdots \\ W\left(\left|\boldsymbol{r}_1 - \boldsymbol{r}_i\right|\right) & \cdots & -\sum_{j\neq i} W\left(\left|\boldsymbol{r}_j - \boldsymbol{r}_i\right|\right) & \cdots & W\left(\left|\boldsymbol{r}_n - \boldsymbol{r}_i\right|\right) \\ \vdots & & \vdots & & \vdots \\ W\left(\left|\boldsymbol{r}_1 - \boldsymbol{r}_n\right|\right) & \cdots & W\left(\left|\boldsymbol{r}_i - \boldsymbol{r}_n\right|\right) & \cdots & -\sum_{j\neq n} W\left(\left|\boldsymbol{r}_j - \boldsymbol{r}_n\right|\right) \end{bmatrix}, \tag{12}$$

$$\boldsymbol{b} = \frac{\rho}{\Delta t}\begin{bmatrix} (1-\gamma)\nabla\cdot\boldsymbol{v}_1^* - \gamma\dfrac{1}{\Delta t}\dfrac{\left\langle n^* \right\rangle_1 - n^0}{n^0} \\ \vdots \\ (1-\gamma)\nabla\cdot\boldsymbol{v}_i^* - \gamma\dfrac{1}{\Delta t}\dfrac{\left\langle n^* \right\rangle_i - n^0}{n^0} \\ \vdots \\ (1-\gamma)\nabla\cdot\boldsymbol{v}_n^* - \gamma\dfrac{1}{\Delta t}\dfrac{\left\langle n^* \right\rangle_n - n^0}{n^0} \end{bmatrix}, \tag{13}$$

其中下标n为参与计算的粒子数. 可知，PPE是一个包含所有粒子状态的矩阵方程，在大规模计算的情况下，求解该方程将需要非常大的存储空间和非常长的求解时间. 为了更高效地求解PPE、MPS中常用共轭梯度法（CG）、不完全Cholesky共轭梯度法（ICCG）进行求解大型矩阵方程，求解该方程的代价仍然极高.

2.2 数据驱动的压力泊松方程

为了解决PPE求解代价过高的问题，前期研究中，我们提出了一种数值驱动的NN-MPS方法. NN-MPS方法将求解PPE方程转化为机器学习领域的回归问题. 针对这一回归问题，算法采用神经网络建立求解模型. 对于高维问题，传统数值方法的计算成本过高，神经网络较强的提取特征的能力以及求解非线性问题的能力，使其成为了解决该回归问题一个很好的选择.

为了使回归求解器更加准确和稳定，我们需要为模型选择合适的输入. 参考原始MPS算法的PPE构造：

$$\frac{2D}{n^0 \lambda}\left[\sum_{j=1}^{N} P_j \cdot W\left(\left|\boldsymbol{r}_j - \boldsymbol{r}_i\right|\right) - P_i \cdot \sum_{j=1}^{N} W\left(\left|\boldsymbol{r}_j - \boldsymbol{r}_i\right|\right)\right] = (1-\gamma)\frac{\rho}{\Delta t}\nabla\cdot\boldsymbol{v}_i^* - \gamma\frac{\rho}{(\Delta t)^2}\frac{\left\langle n^* \right\rangle_i - n^0}{n^0}. \tag{14}$$

结合$\langle n \rangle$的表达式(4)，粒子i压力的可以表达成如下形式：

$$P_i^{k+1}=\frac{1}{n^*}\left\{\sum_{j=1}^{N}P_j\cdot W\left(\left|\boldsymbol{r}_j-\boldsymbol{r}_i\right|\right)-\frac{n^0\lambda}{2D}\left[(1-\gamma)\frac{\rho}{\Delta t}\nabla\cdot\boldsymbol{v}_i^*-\gamma\frac{\rho}{(\Delta t)^2}\frac{\langle n^*\rangle_i-n^0}{n^0}\right]\right\}. \tag{15}$$

观察式(15)，可以发现中心粒子的压力依赖于光滑半径内其他粒子的压力权重、自身的中间速度、中间粒子数密度. 据此，NN-MPS算法提出一种回归模型的构造形式：

$$P^{k+1}=H\left(\phi_i^{\text{pres}},\phi_1^{\text{pres}},\phi_2^{\text{pres}},\cdots,\phi_n^{\text{pres}},n_i^*,\nabla\cdot\boldsymbol{v}_i^*\right), \tag{16}$$

其中

$$\phi_i^{\text{pres}}=P_i^k n_i^*, \tag{17}$$

$$\phi_j^{\text{pres}}=P_j^k W\left(\left|\boldsymbol{r}_j-\boldsymbol{r}_i\right|\right), \tag{18}$$

式中P^k表示当前时间步该粒子的压力，ϕ_j^{pres}则体现了周围粒子与中心粒子的压力积分特征，ϕ_j^{pres}的数量由模型的邻域极限半径确定.

基于以上数据模型，NN-MPS构造了适合PPE回归求解器的输入.

2.3　DNN(Deep Neural Network)网络模型

多层感知机(Multi-Layer Perceptron)作为一种经典深度学习模型，是当前主流深度神经网络的基础(Gardner et al.,1998). 一个多层感知机包含一个输入层、一个输出层，以及多个隐藏层，每一层包括多个感知器，每个感知器具有一个或多个输入、偏置、激活函数，以及单个输出. 感知机每一层可以表示成：

$$f(\boldsymbol{x})=H(\boldsymbol{\omega}\boldsymbol{x}+\boldsymbol{b}), \tag{19}$$

其中$\boldsymbol{x}=(x_1,\ x_2,\ \cdots,\ x_n)$表示该层的输入向量，$\boldsymbol{\omega}$表示权重，$\boldsymbol{b}$表示偏置，$H$则称为激活函数，$f(\boldsymbol{x})$表示该层输出.

本文选择以多层感知机作为回归网络的基础结构. 隐藏层的层数对回归求解效果、计算速度有较大的影响，目前相关领域尚无法给出DNN模型深度(隐藏层层数)与计算误差之间的绝对关系. 因此，综合考虑了计算精度、计算效率之间的平衡，结合实际数值测试结果，我们选择将隐藏层数设置为5. 如图1所示，整个结构包含1个输入层、5个隐藏层和1个输出层，层与层之间采用ReLU(Glorot et al.,2011)函数作为激活函数. 输入层中ϕ_j^{pres}的排序由粒子与中心粒子距离决定，这种排序方式可以帮助神经网络学习系统的潜在特征.

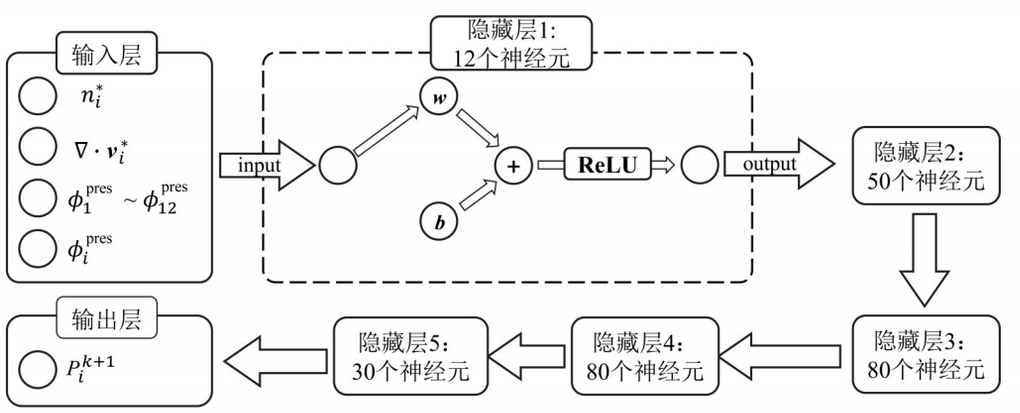

图1　神经网络结构示意图

Fig. 1 Structure of the neural network

改进后的MPS算法单时间步积分流程如下：

(i) 根据式(8)中黏性力和质量力获得中间速度$\boldsymbol{v}^*$，并计算中间位置$\boldsymbol{r}^*$；

(ii) 根据式(4)计算中间粒子数密度n^*；

(iii) 构造神经网络输入，并通过神经网络求解P^{k+1}；

(iv) 利用求得的压力，根据式(9)获得修正速度$\boldsymbol{v}'$，并计算$\boldsymbol{v}^{k+1}$，$\boldsymbol{r}^{k+1}$.

3 利用边缘计算设备加速 MPS 算法

3.1 Atlas 200 DK 开发板

华为公司的 Atlas 200 DK 开发者套件是以 Atlas 200 AI 加速模块为核心的开发板形态. 其中 Atlas 200 AI 加速模块集成了 Ascend 310 AI 处理器，Ascend 310 芯片内置 2 个 DaVinci AI core，可以实现最大 8TFLOPS/FP16 或 16TOPS/INT8 的乘加性能(华为技术有限公司,2022).

开发板可以实现图像、视频等多种数据分析与推理计算，目前广泛用于基于机器视觉的 AI 应用，如智能监控、机器人、无人机、视频服务器等场景. 机身包括 2 个 DaVinci AI core，1 个 8 核 A55CPU，支持 1000 M 以太网. 由于其对神经网络的高速计算的特性与高速以太网通信能力，我们选择 Atlas 200 DK 作为配置 NN-MPS 的边缘计算设备.

本文所有工作基于 Atlas 200 DK 开发者套件的 1.32.0.0 版本进行开发.

3.2 开发流程

本文采用 Python3 在 Mind Studio 平台上进行 NN-MPS 程序开发. 开发前开发平台为 Mind Studio，语言为 Python3、hiai 库。

华为公司为调用 AI Core 提供了一套如图 2 所示的昇腾软件栈，其相关概念的具体介绍可见华为官方文档. 在利用其中的模型转换工具将 Tensorflow 模型转换为其所需格式后，通过如下的流程实现调用 AI Core 完成神经网络推理：

① 初始化图(Graph)对象，图的作用是管理和编排计算引擎；

② 创建引擎(Engine)，引擎是执行功能的基本单元；

③ 设置模型文件(.om)路径；

④ 构造输出输入对象；

⑤ 调用引擎进行模型推理；

⑥ 结果后处理.

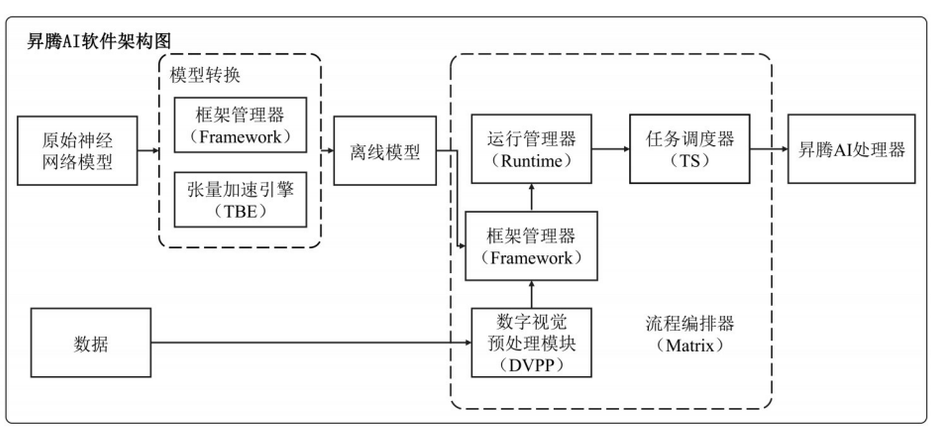

图 2 昇腾 AI 软件栈

Fig. 2 Software stack of Ascend AI

3.3 Socket 通信方案

Atlas 开发板板载 CPU 实际性能并不比一般计算机 CPU 高，因此将整个 NN-MPS 由开发板运行并不是个很合适的方案. 考虑到 Atlas 200 DK 支持 1000 M 以太网传输，将计算过程分成两个部分，推理部分由开发板完成，其余部分由电脑 CPU 完成具有相当高的可行性. 因此，为了结合计算机 CPU 算力与边缘计算设备对神经网络的加速能力，程序将求解压力泊松方程的神经网络推理部分在 Atlas 200 DK 开发板上实现，其余部分由计算机 CPU 完成，两者通过千兆以太网利用 Socket 通信协议实现快速的数据交换. 图 3 展示了整体程序结构.

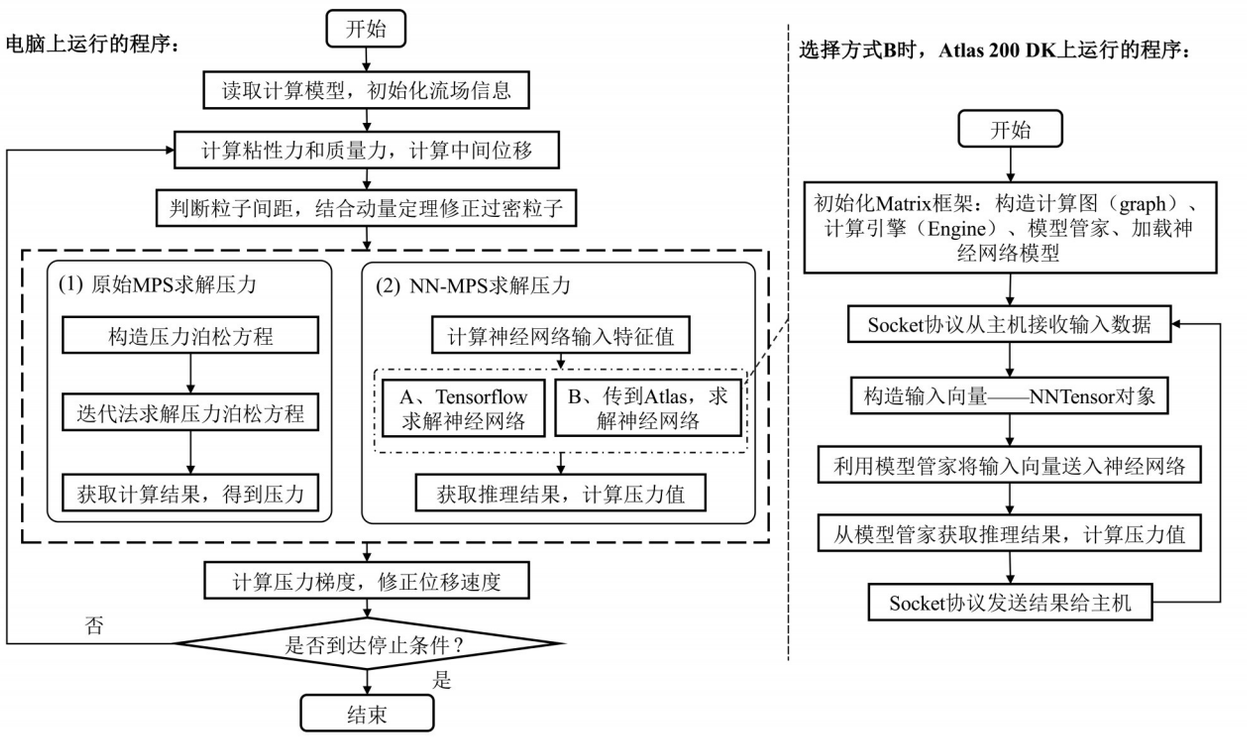

图3　整体计算流程

Fig. 3 Calculation flow chart

4　算例实验

4.1　模型描述

本文中采用如图4所示增加障碍物的溃坝模型作为验证算例，模型由传统溃坝模型加上圆柱形障碍物组成. 文中共有3种模型：溃坝加固定圆柱障碍物、溃坝加周期性移动圆柱障碍物，以及增大宽度的溃坝加固定圆柱障碍物. 其中，移动圆柱障碍物的模型则是在图4(a)的基础上为圆柱增加 $v_x = 2.5*2*\mathrm{sign}((\mathrm{step}+10)\%19-10)$ 的运动率，式中step为计算时间步，sign(a)代表取 a 的符号(+/−). 神经网络用的训练集为传统MPS计算经典无障碍物的溃坝模型过程中输出每时间步的压力积分特征、速度散度、粒子数密度与中心粒子下一时间步压力所得.

采用该模型的原因主要有以下几点：首先，溃坝模型为大部分MPS算例使用的经典模型，且模型结构简单易于实现；其次，增加了圆柱障碍物和移动障碍物可以更好表现MPS算法对于复杂边界条件的适应性和泛化能力；不同的粒子规模有助于比较不同规模流场场景下计算速度的差别.

4.2　误差分析

本文计算了在小模型场景下，固定障碍物与移动障碍物模型计算结果在经典MPS算法、使用CPU的NN-MPS算法、使用Atlas 200 DK加速的NN-MPS算法的计算精度. 使用如下3个参数进行数值比较：近壁面液面高度、液面前缘位置，以及如图5所示区域的平均压力.

图6~8是3个参数在两种模型下的计算结果，其中前缘位置与液面高度只取了流体与前壁碰撞前的数据. 比较两个NN-MPS与传统MPS的数据，使用NN-MPS算法在液面高度、前缘位置上有最大在0.1作有的相对误差，在压力结果上虽然与原始MPS算法相比产生了漂移与变形，相对误差较大，但是在总体变化趋势上与实际结果仍比较相似，这些差别是由神经网络模型的结构或训练参数导致，可以通过优化模型得到改善；在两种硬件运行的NN-MPS算法计算结果之间也有微小的差别，这是因为Atlas上的AI芯片只支持最大FP16浮点数，而CPU上则支持64位浮点运算，整体来看，由硬件区别导致的误差与由神经网络模型导致的误差相比，前者几乎可以忽略.

单独分析在不同硬件上的NN-MPS计算结果，发现计算前期阶段在CPU与Atlas上的NN-MPS计算结

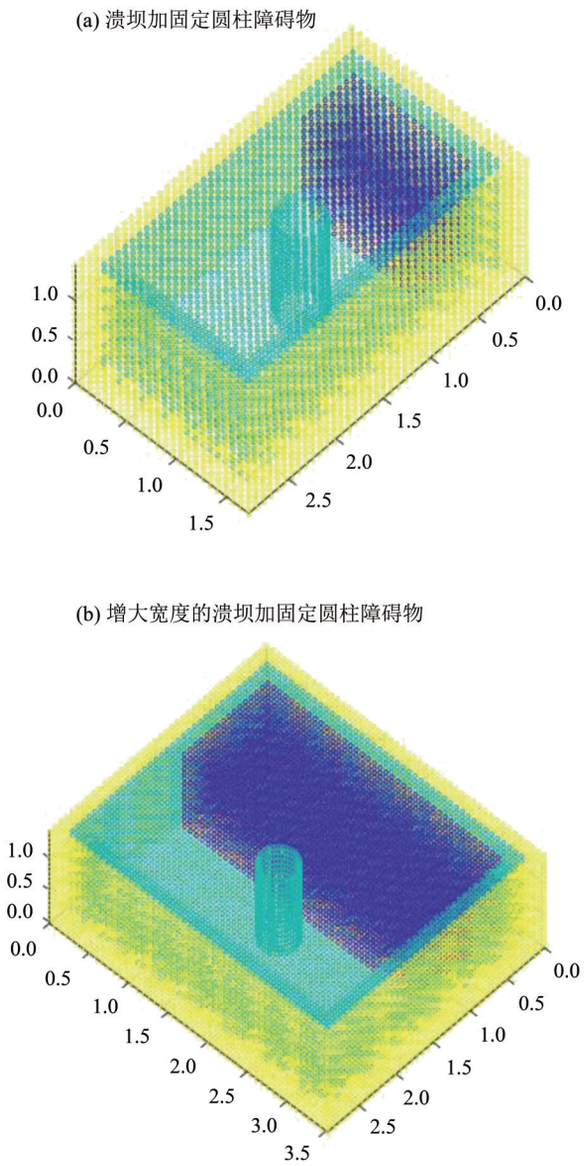

图 4　MPS 模型初始状态

Fig. 4 MPS model initial state

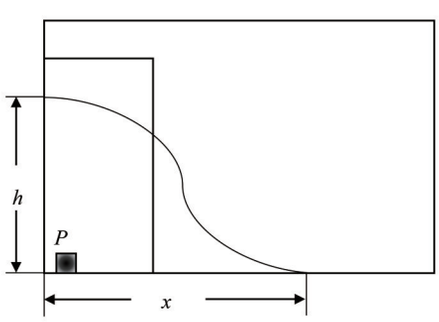

图 5　比较参数示意图

Fig. 5 Parameters used for comparison

果差别较小，而在第 75 时间步左右移动圆柱下的前缘位置参数 Atlas 与 CPU 计算结果开始出现差别；在 125 时间步附近，两者在静止圆柱和移动圆柱的液面高度也开始出现差别. 这可能是由于经过碰撞之后产生的流场变化加剧，导致由 FP64 移植到 FP16 设备上产生微小误差的积累增大，从而使结果出现较大差距. 这点可以通过静止圆柱与移动圆柱计算结果的区别加以证明：静止圆柱场景下，流场变化与移动圆柱场景相比较为缓和，因而移动圆柱场景下，两种 NN-MPS 算法的计算结果更早出现较大差别，同理在移动圆柱场景下，液面高度也比静止圆柱场景下出现了更大的差距.

4.3　速度对比

本文比较了加宽模型与小模型分别采用传统 MPS、神经网络改进后的 MPS、移植到 Atlas 200 DK 的 MPS 在求解 PPE 时的用时情况. 为了时间比较的合理性，移植到 Atlas 的 MPS 计算 PPE 求解时包括了 Socket 通信消耗的时间.

实例计算使用的 CPU：Intel（R）Core（TM）i7-10510U CPU@1.80 GHz .

表 1 展示了神经网络改进的 MPS 算法与移植到 Atlas 200 DK 上的改进的 MPS 算法在求解 PPE 的时间上都相较传统 MPS 算法有 10 倍以上的提升，数据规模较小的模型中（如算例 1、算例 2 中），Socket 通信耗时占比较大，通信时间与 AI 芯片神经网络求解时间处于同一量级，因而在 CPU 与在 Atlas 上面的 MPS-NN 算法相比，反而 CPU 的计算速度较快；而在数据规模较大的模型中（算例 3），Socket 通信时间占比减小，移植到 Atlas 200DK 上的 MPS 相较在 CPU 上的改进 MPS 算法求解 PPE 用时又有了 16% 的提升.

5　总　结

本文采用神经网络结合边缘计算硬件的方式，将 MPS 方法中的计算瓶颈——压力泊松方程的计算时间缩减，提出了一种面向边缘计算场景的流体动力学仿真新方案. 研究利用神经网络抽象压力泊松方程的求解过程，将其转化为光滑半径内粒子特征与下一时间步压力值之间的回归问题，成功减少了泊松方程求解时间. 同时，本文利用边缘计算硬件求解神经网络推理过程，探究在边缘计算场景下进一步加速计算的可能性. 在实验结果中，当粒子规模在 28 800 以上时，使用

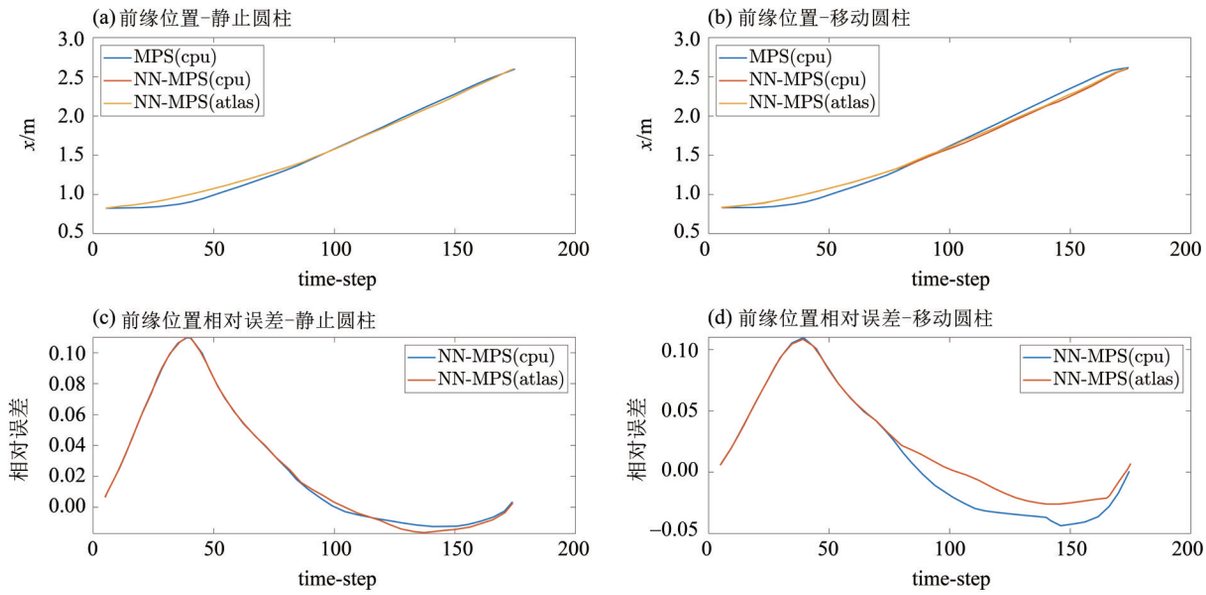

图6　前缘位置变化

Fig. 6 Movement of the leading edge

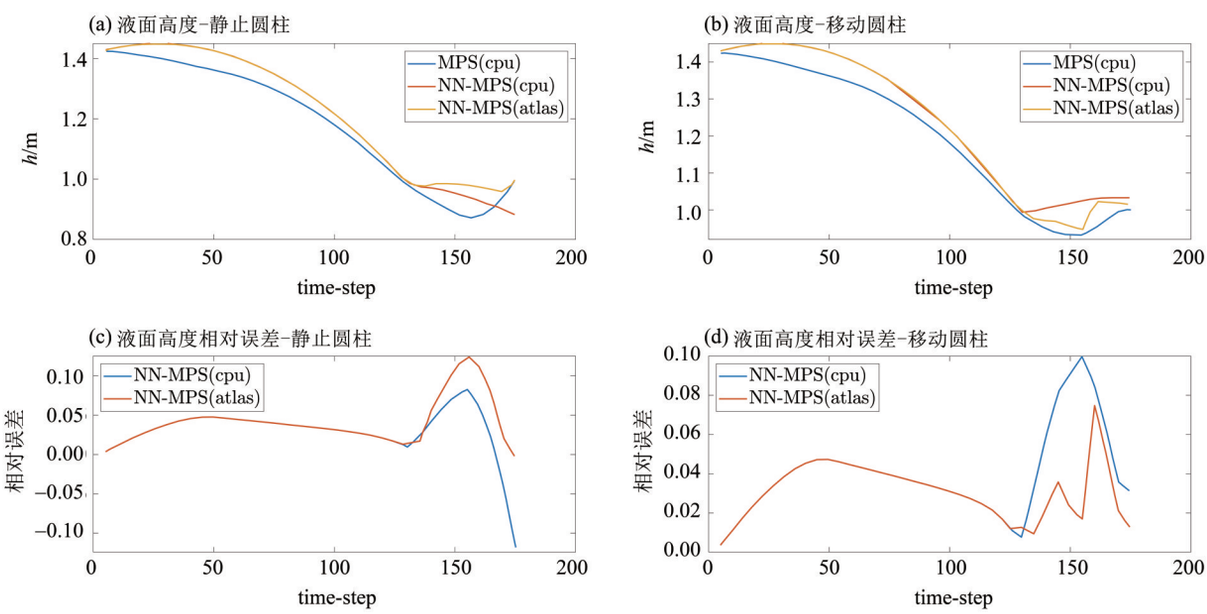

图7　近壁面液面高度变化

Fig. 7 The liquid level height near the left wall

表1　PPE求解时间比较

Table 1　Computational time for solving PPE

算例	描述	初始大小(粒子数)	总粒子数	泊松方程求解总用时/s		
				MPS(CPU)	NN-MPS(CPU)	NN-MPS(Atlas)
1	固定圆柱	8*16*16	13 456	5 103. 8	379. 4	392. 7
2	移动圆柱	8*16*16	13 456	5 169. 1	395. 1	401. 3
3	固定圆柱	8*40*16	28 800	41 756. 3	3 750. 9	3 151. 0

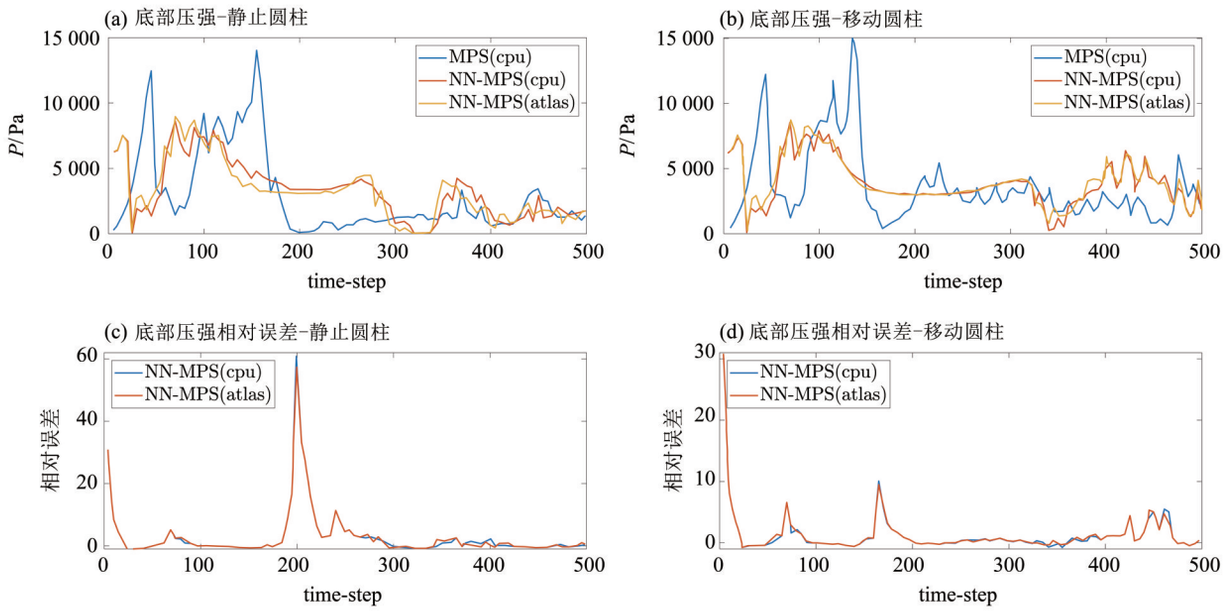

图 8　压力变化

Fig. 8 Pressure developments

边缘计算硬件的计算速度相对于使用CPU的计算速度有所提升，而在小规模场景下，使用边缘计算硬件与CPU计算速度相当，因此在计算规模较大时，可以使用本文所述方法作为加速手段. 本文采用的方案，相较于使用GPU等常见加速硬件，具有更低的功率需求和更低的成本，为一些边缘侧有流体动力学仿真需求的场景提供了一种有效的解决方案，同时也为边缘计算与基础学科进行学科交融提供新的切入方向.

参考文献：

崔勇，宋健，缪葱葱，等，2017. 移动云计算研究进展与趋势[J]. 计算机学报，40(2)：273-295.

杜年春，沈向前，谢翔，等，2022. Online SAR地质灾害监测和预警系统关键技术及应用研究[J]. 现代雷达，44(10)：28-32.

何腾，2020. 浅析边缘计算技术应用的现状及挑战[C]// 第三十四届中国(天津)2020'IT、网络、信息技术、电子、仪器仪表创新学术会议论文集：117-120.

华为技术有限公司，2022. Atlas 200 DK开发者套件-概述[EB/OL]. (2022-12-29)[2023-02-23]. https://www.hiascend.com/document/detail/zh/Atlas200DKDeveloperKit/1013/productdesc/atlas200_DK_pdes_19_0007.html

滕云豪，2022. 基于边缘计算的室内空气质量监测技术研究[D]. 上海：上海第二工业大学.

杨超，2018. 基于大涡模拟的移动粒子半隐式法研究流固耦合及冲击破坏问题[D]. 上海：上海交通大学.

张驰，张雨新，万德成，2011. SPH方法和MPS方法模拟溃坝问题的比较分析[J]. 水动力学研究与进展A辑，26(6)：11.

张雨新，万德成，2012. 用MPS方法数值模拟低充水液舱的晃荡[J]. 水动力学研究与进展A辑，27(1)：100-107.

CHIKAZAWA Y，KOSHIZUKA S，OKA Y，2001. A particle method for elastic and visco-plastic structures and fluid-structure interactions[J]. Comput Mech，27(2)：97-106.

GARDNER M W，DORLING S R，1998. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences[J]. Atmos Environ，32(14/15)：2627-2636.

GLOROT X，BORDES A，BENGIO Y，2011. Deep sparse rectifier neural networks[J]. J Mach Learn Res，15：315-323.

GOTOH H，SAKAI T，2006. Key issues in the particle method for computation of wave breaking[J]. Coast Eng，53(2/3)：171-179.

JOUPPI N P，YOUNG C，PATIL N，et al，2017. In-datacenter performance analysis of a tensor processing unit[C]//Proceedings of 44th Annual International Symposium on Computer Architecture：1-12.

KHAYYER A，GOTOH H，2008. Development of CMPS method for accurate water-surface tracking in breaking waves[J]. Coast Eng J，50(2)：179-207.

KOSHIZUKA S，OKA Y，1996. Moving-particle semi-implicit method for fragmentation of incompressible fluid[J]. Nucl Sci Eng，123(3)：421−434.

LEE B H，PARK J C，KIM M H，et al，2011. Step-by-step improvement of MPS method in simulating violent free-surface motions and impact-loads[J]. Comput Methods Appl Mech Eng，200(9/10/11/12)：1113−1125.

LEE V W，KIM C，CHHUGANI J，et al，2010. Debunking the 100X GPU vs. CPU myth：An evaluation of throughput computing on CPU and GPU[J]. Acm Sigarch Comput Archit News，38(3)：451−460.

LIU D，CHEN T，LIU S，et al，2015. PuDianNao：A polyvalent machine learning accelerator[J]. Acm Sigplan Notices，50(4)：369−381.

MONAGHAN J J，1992. Smoothed particle hydrodynamics[J]. Annu Rev Astron Astrophys，30：543−574.

SHIBATA K，KOSHIZUKA S，2007. Numerical analysis of shipping water impact on a deck using a particle method[J]. Ocean Eng，34(3/4)：585−593.

TANAKA M，MASUNAGA T，2010. Stabilization and smoothing of pressure in MPS method by quasi-compressibility[J]. J Comput Phys，229(11)：4279−4290.

TSURUTA N，KHAYYER A，GOTOH H，2013. A short note on dynamic stabilization of moving particle semi-implicit method[J]. Comput Fluids，82：158−164.

TSURUTA N，KHAYYER A，GOTOH H，2015. Space potential particles to enhance the stability of projection-based particle methods[J]. Int J Comput Fluid Dyn，29(1)：100−119.

（责任编辑　冯兆永）