文章编号:1007-4708(2010)01-0021-07

基于 OpenMP 的三维显式物质点法并行化研究

黄 鹏[1,2], 张 雄*[1,3], 马 上[1], 王汉奎[1]

(1. 清华大学 航天航空学院,北京 100084;2. 中国工程物理研究院 总体工程研究所,绵阳 621900;
3. 大连理工大学 工业装备结构分析国家重点实验室,大连 116024)

摘 要:基于 OpenMP 技术开发了三维显式物质点并行程序 MPM3DMP。为了避免节点更新阶段的数据竞争,采用区域分解法将背景网格分解为均匀的子域,每个线程负责一个子域的节点变量更新,然后将更新后的节点变量装配到整体。在质点更新阶段采用了循环分解方法进行并行。针对 Taylor 杆碰撞的三种计算模型,在双 Intel Woodcrest 4 核 CPU 服务器下进行了测试:粗模型在 4 核下加速比为 3.82,在 8 核下为 6.23,中模型在 4 核下加速比为 3.79,在 8 核下加速比为 6.23;细模型在 4 核下加速比为 3.75,8 核下加速比为 6.26。因此,本文的并行程序具有较好的并行效率和可扩展性。

关键词:OpenMP;显式物质点法;区域分解法;循环分解方法;加速比
中图分类号:TB115;O344.3 **文献标识码**:A

1 引 言

Sulsky 等将用于流体动力学的质点网格法[1,2] PIC(Particle in Cell)扩展到固体力学问题中,提出了物质点法[3,4] MPM(Material Point Method)。它将连续体离散成一组带有质量的质点,质点携带了所有物质信息,其运动代表了物质的变形。物质点法在计算中采用未变形的背景网格,避免了拉格朗日法因网格畸变而产生的数值困难,非常适合于分析超高速碰撞[5,6],动态断裂[7],以及金属加工大变形问题[8]。研究并行化的 MPM 程序具有一定的工程意义,通过 MPM 的并行计算,可以缩短计算时间,提高计算效率,扩大计算规模。

Parker 等基于背景网格区域分解法和 MPI 技术研究了并行的 MPM 程序[9,10]。MPI 为分布式内存并行技术,适合于在大规模并行机和机群上运行。基于 MPI 的 MPM 并行方法涉及到背景网格的区域分解,每一个时间步均需要显式地进行节点和质点间的通信,而且还要建立复杂的并行 I/O,对串行程序的前后处理均需要做较大改动。另外一种选择是基于 OpenMP 技术对已有的串行 MPM 程序进行并行化。OpenMP 为共享内存并行技术,适合于在多核计算机或者 SMP 机器上运行,由于 OpenMP 支持增量并行,而且不会涉及到复杂的子区域之间的通信,因此已经逐步在显式有限元计算[11]、分子动力学计算[12]和流体动力学计算的并行化中得到应用[13],而基于 OpenMP 技术的 MPM 并行化至今未见报道。

文中首先简要介绍物质点法的原理,阐述显式物质点法的计算过程和流程图,简单阐述 OpenMP 的并行机制。为了避免节点更新阶段的数据竞争问题,基于区域分解方法将背景网格划分为若干均匀的子域,每个子域由一个线程进行计算。在质点更新阶段,则采用质点循环分解方法对主要循环模块进行并行化。最后通过 Taylor 杆碰撞的算例分析,给出并行程序 MPM3DMP 的加速比和并行效率。

收稿日期:2008-03-28;**修改稿收到日期**:2008-11-17.
基金项目:国家自然科学基金(10872107);国家重点基础研究发展计划(2010CB32101)资助项目.
作者简介:黄 鹏(1977-),男,博士生,工程师;
张 雄*(1966-),男,博士,教授
(E-mail:xzhang@tsinghua.edu.cn).

2 显式物质点法及其实现

显式物质点法中采用质点对连续体进行离散,如图 1 所示,质点运动代表了物体运动。背景网格可以固定或自由布置,用于动量方程的求解和空间导数的计算。显式物质点法按照显式时间积分方

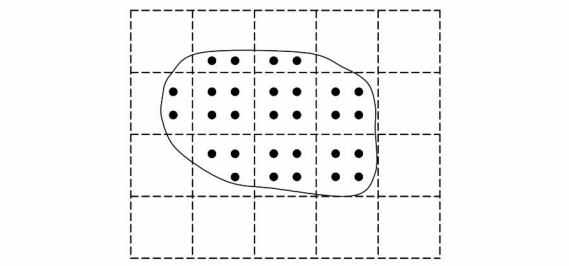


图 1 物质点法中的物质离散
Fig. 1 Material discretization in MPM

法求解,每个时间步的求解过程可以分为两阶段:节点更新阶段和质点更新阶段。节点更新阶段如同传统的有限元计算,质点变量通过形函数映射到节点上获得节点变量,进而进行节点动量方程求解。在质点更新阶段,将更新后的节点变量映射到质点上,更新质点的应变和应力。

以下对物质点法计算过程进行推导,若已知质点上第 k 个时间步的物理量,求第 $k+1$ 个时间步的质点物理量。用带有下标 i 的量来代表网格节点上的变量,用带有下标 p 的量来代表质点物理量,并且令质点 p 的形函数值为 $S_{ip}=N_i(\mathrm{X}_p)$,形函数的导数值为 $\mathrm{G}_{ip}=\nabla N_i|_{\mathrm{x}_p}$,$\mathrm{X}_p$ 为质点 p 的空间位置矢量。

具体实现可以按照以下步骤进行[6]:

(1) 首先计算网格节点数据

网格节点质量:

$$m_i^k=\sum_p m_p S_{ip}^k \tag{1}$$

网格节点动量:

$$\mathrm{p}_i^k=\sum_p m_p \mathrm{v}_p^k S_{ip}^k \tag{2}$$

网格节点力:

$$\mathrm{f}_i^k=(\mathrm{f}_i^{int})^k+(\mathrm{f}_i^{ext})^k \tag{3}$$

节点内力:

$$(\mathrm{f}_i^{int})^k=-\sum_p \boldsymbol{\sigma}_p^k\cdot \mathrm{G}_{ip}^k\frac{m_p}{\boldsymbol{\rho}_p^k} \tag{4}$$

节点外力:

$$(\mathrm{f}_i^{ext})^k=\sum_p m_p S_{ip}\mathrm{b}_p+\int_{\Gamma_t} S_{ip}\mathrm{t}\,\mathrm{d}\boldsymbol{\Gamma} \tag{5}$$

式中 b 为作用在物体上的体力,t 为作用在物体边界上的面力。

(2) 在背景网格节点上积分动量方程,并施加固定边界条件:

$$\mathrm{p}_i^{k+1}=\mathrm{p}_i^k+\mathrm{f}_i^k\Delta t \tag{6}$$

在固定边界:$\mathrm{p}_i^{k+1}=0$,$\mathrm{f}_i^k=0$

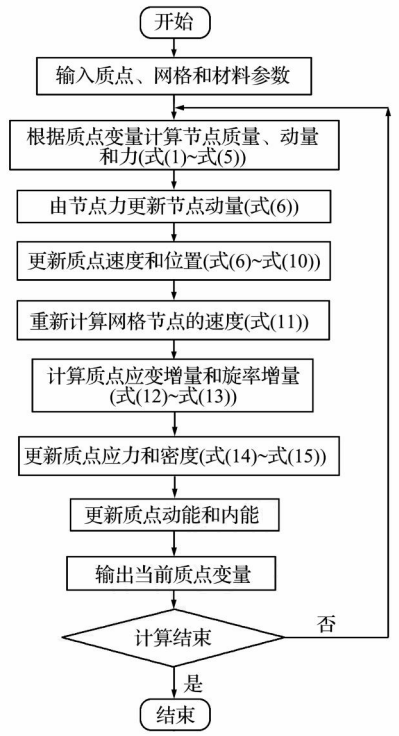


图 2 计算流程图
Fig. 2 Flowchart of MPM3D code

(3) 更新质点位置和速度

$$\overline{\mathrm{v}}_p^{k+1}=\sum_{i=1}^{8}\frac{\mathrm{p}_i^{k+1}}{m_i^k}S_{ip}^k \tag{7}$$

$$\mathrm{a}_p^k=\sum_{i=1}^{8}\frac{\mathrm{f}_i^k}{m_i^k}S_{ip}^k \tag{8}$$

质点位置:

$$\mathrm{x}_p^{k+1}=\mathrm{x}_p^k+\overline{\mathrm{v}}_p^{k+1}\Delta t \tag{9}$$

质点速度:

$$\mathrm{v}_p^{k+1}=\mathrm{v}_p^k+\mathrm{a}_p^k\Delta t \tag{10}$$

(4) 将质点速度映射回背景网格节点

$$\mathrm{v}_i^{k+1}=\frac{1}{m_i^k}\sum_p m_p\mathrm{v}_p^{k+1}S_{ip}^k \tag{11}$$

(5) 计算质点应变增量和旋率增量

$$\Delta\boldsymbol{\varepsilon}_p^k=\frac{\Delta t}{2}\sum_{i=1}^{8}[\mathrm{v}_i^{k+1}(\mathrm{G}_{ip}^k)^{\mathrm{T}}+\mathrm{G}_{ip}^k(\mathrm{v}_i^{k+1})^{\mathrm{T}}] \tag{12}$$

$$\Delta\boldsymbol{\omega}_p^k=\frac{\Delta t}{2}\sum_{i=1}^{8}[\mathrm{v}_i^{k+1}(\mathrm{G}_{ip}^k)^{\mathrm{T}}-\mathrm{G}_{ip}^k(\mathrm{v}_i^{k+1})^{\mathrm{T}}] \tag{13}$$

(6) 更新密度,应力:

$$\boldsymbol{\rho}_p^{k+1}=\boldsymbol{\rho}_p^k/(1+\mathrm{tr}(\Delta\boldsymbol{\varepsilon}_p^k)) \tag{14}$$

$$\boldsymbol{\sigma}_p^{k+1}=\boldsymbol{\sigma}_p^k+\Delta\mathrm{r}_p^k+\Delta\boldsymbol{\sigma}_p^k \tag{15}$$

式中 $\Delta\mathrm{r}_p^k=\Delta\boldsymbol{\omega}_p^k\cdot\boldsymbol{\sigma}_p^k-\boldsymbol{\sigma}_p^k\cdot\Delta\boldsymbol{\omega}_p^k$,$\Delta\boldsymbol{\sigma}_p^k$ 为 Jaumann 应力率的应力增量形式,可根据弹塑性材料模型更新,此处考虑了材料的大变形和转动。对于超高速撞击问题,应力球量可由状态方程给出。

基于以上格式,采用面向对象编程技术和 Fortran95 编制串行的 MPM3D 物质点程序。程序中大量采用了 module 和自定义数据类型,以增强程序的模块化。图 2 给出串行 MPM3D 的计算流程图,由图 2 可见,MPM 计算主要包含两个部分,由式(1) ～ 式(6) 的节点更新阶段,由式(7) ～ 式(15) 的质点更新阶段。在节点更新阶段,质点变量保持不变。而在质点更新阶段,节点变量保持不变。

3　OpenMP 的并行机制

OpenMP 的并行模式为 fork/join 式并行[14]。当遇到并行指令语句 ! $omp parallel 时,主线程派生分叉,产生多线程,在并行区域内多线程协同执行,并行代码结束时,多线程汇合到主线程,其计算原理如图 3 所示。

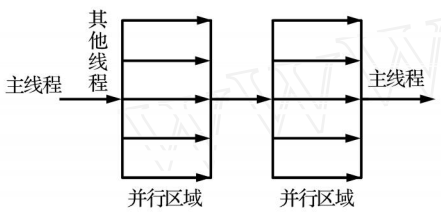


图 3　Fork/join 式并行

Fig. 3　Fork/join parallelism

OpenMP 支持增量化并行和共享内存,OpenMP 的并行基本方法是循环分解方法[15],循环分解方法要求各循环之间必须无相关性。由于 OpenMP 编程基于共享内存,所以需要采用各种措施避免内存数据读写的数据竞争机制,这也是编写 OpenMP 并行程序的难点所在[16]。

4　MPM 的并行化

4.1　节点变量更新的并行

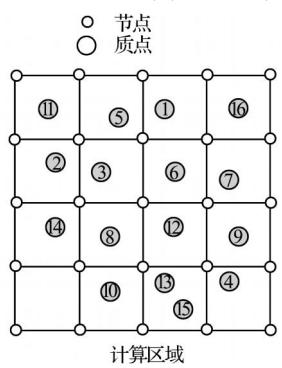
节点变量的更新由式(1) ～ 式(6) 完成,在节点变量更新过程中采用了质点循环的方法。如果直接采用质点循环分解进行并行,不同的质点可能会在同一时刻更新某个节点变量,这样便会产生数据竞争问题。为了消除节点更新阶段的循环依赖,此处采用了区域分解方法。

区域分解方法将背景网格区域分解为大小相同的子域,一个子域的网格节点更新由一个线程来完成,各个子域节点更新完成以后,将其进行装配便得到整体节点值。区域分解的过程如图 4 所示,背景网格区域被分解为 4 个均匀的子域,在图 4(a) 中给出了质点的整体编号,在图 4(b) 中给出了质点在各个子域内的子编号。由于计算过程中采用相同的背景网格,因此各个子域内的节点编号保持不变。

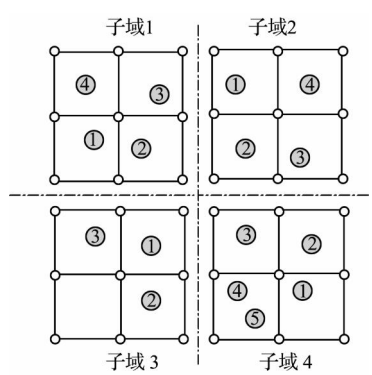
在显式 MPM 计算中,质点运动导致质点在不同时间步位于不同的背景网格,因此,在每一个时间步要对质点进行判断,给出质点所在的子域及其子编号。质点的子编号通过数组 Pindex 给出,Pindex(*i*,*j*) 给出了子编号 *i* 质点的整体编号,*j* 为质点所在子域,数组 Pindex 可通过并行计算完成。

区域分解完成以后,采用子域的循环分解方法进行并行,在各个子域内对所属的质点进行循环,进而更新各个子域内的节点变量。因此,节点变量更新并行时需要加如下 OpenMP 指令:

```
! $omp parallel do  &
! $omp private (与质点相关的变量) &
! $omp default(shared)
do 子域循环
do 子域内质点循环
更新子域内节点变量
```



(a) 计算区域及质点整体编号



(b) 子域及质点子编号

图 4　计算域的区域分解

Fig. 4　Decomposition of computational domain

end do

end do

进行完各个子域内的节点变量更新,将各个子域内的节点变量装配到整体节点变量,得到更新后的整体节点变量,该过程也可通过并行计算完成。

4.2 质点变量更新的并行

对每一个质点进行位置和速度更新,其程序实现以式(7) ~ 式(10) 为准。进行质点循环分解,各质点变量均作为私有变量,而节点变量作为公有变量仅供读取,不会造成数据竞争,因此,并行时需要加如下 OpenMP 指令:

```
!$omp parallel do &
!$omp private (与质点相关的变量) &
!$omp default(shared)
do 质点循环
更新质点位置和速度
end do
```

这样便完成了质点位置和速度更新程序的并行。

为了得到更为精确的计算结果,将质点速度映射回背景网格,重新得到背景网格节点的速度,其程序实现以式(11) 为准。同样此处涉及到节点变量的数据竞争,可采用上节讨论的区域分解方法避免数据竞争。

根据式(12) 和式(13) 计算每一个质点的应变增量和旋率增量,根据式(14) 和式(15) 更新每一个质点的应力和密度,此过程仅仅对质点进行计算,各质点变量均作为私有变量,而节点变量作为公有变量仅供读取,不会造成数据竞争。因此,采用质点循环分解方法便可完成质点密度和应力的并行计算。

在质点应力计算时,涉及到对本构计算程序 Constitution() 的线程私有调用,可将与本构关系相关的材料变量作为局部变量。另外一种方法是在材料本构的模块中,将一些和材料相关的全局变量进行线程私有化,这样也可以实现本构计算程序 Constitution() 的线程私有调用。

4.3 质点能量计算的并行

基于质点循环分解方法进行质点总动能和总内能的并行计算,此处采用了归约操作来避免能量计算的数据竞争。动能并行计算时需要加如下 OpenMP 指令:

```
!$omp parallel do &
!$omp private (与质点相关的变量) &
!$omp reduction(+:EngKinetic)
do 质点循环
更新 EngKinetic
end do
```

以上语句中 EngKinetic 为动能,内能的并行计算与动能的并行计算类似。

4.4 负载平衡

在区域分解法中,每一个线程或者 CPU 处理一个子域,但是每个子域内的质点数是不同的,而且在一些计算中,由于质点的大范围运动,可能导致在某个时间步的某些子域内没有质点,这样加剧了负载不平衡。

此处,采用了一个简单的负载平衡算法,即采用自适应背景网格方法,在一个时间段以后,背景网格被重新定义,根据质点的位置重新定义背景网格的范围,这样便避免了空子域,弱化了负载的不平衡。

5 并行算例分析

5.1 算例简介

最终开发的并行程序被命名为 MPM3DMP,MPM3DMP 并行程序可在 Linux 和 Windows 系统上运行。并行测试的算例为 Taylor 杆碰撞问题,本文的部分计算参数取自文献[6],并以其中的实验结果作为参照。

Taylor 杆材料为铜,偏应力按照各向同性强化模型计算,应力球量则按照 Grüneisen 状态方程计算,材料参数见表 1。杆初始长度 L_0 = 25.4 mm,直径 D_0 = 7.6 mm,初速度 v_0 = 190 m/s。离散质点时,采用了三种计算模型,分别为粗模型、中模型和细模型,模型的质点数、初始的背景网格数和节点数见表 2。计算过程中采用了自适应背景网格,背景网格重构 20 次。三种模型的 MPM 计算结果和实验结果比较见表 2,其中 L 为撞击结束后的杆长,D 为撞击端的杆直径。粗模型碰撞前后的杆外

表 1　Taylor 杆材料参数

Tab. 1 Material parameters of Taylor bar

材料模型	材料参数			强化模型			状态方程		
参数	ρ (Kg·m^3)	E/ GPa	υ	A/ MPa	B/ MPa	n	C_0/ (m·s)	s	γ
值	8930	117. 0	0. 35	157. 0	425. 0	1. 0	3940	1. 49	1. 96

表 2　模型参数和计算结果

Tab. 2　Model parameters and computational results

计算模型	模型参数			结果变量		
	质点数	初始网格数	初始节点数	L/ mm	D/ mm	最大 ε_{pef}
粗模型	56056	49152	53361	16. 5	13. 6	1. 6
中模型	244524	165888	175273	16. 3	13. 4	1. 6
细模型	1155192	1022208	1053493	16. 3	13. 6	1. 9
实 验	-	-	-	16. 2	13. 5	-

形对比如图 5(a,b) 所示,图中 ε_{pef} 为等效塑性应变。

5. 2　并行效率分析

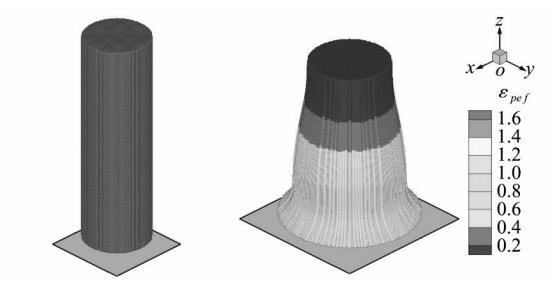
冲击过程的模拟时间为 80 μs,此时动能接近于 0,杆的外形基本不再变化。运行的平台为双 Intel Woodcrest 4 核 CPU 服务器,主频为 2. 66 GHz,内存为 8 GB,操作系统为 Linux。并行程序调试完成后,并行计算结果必须和串行计算结果相同,才能表明并行程序是可靠的。在分析并行 MPM 的效率之前,首先定义加速比和并行效率如下:

$$s_p = \frac{T_s}{T_m}, \quad e_f = \frac{s_p}{n} \tag{16}$$

式中 T_s 为单线程的计算时间, T_m 为多线程(多核) 的计算时间, n 为计算所用的核个数。

三种模型的加速比随核个数变化如图 6 所示,三种模型的并行效率随核个数变化如图 7 所示。图 6 计算结果表明,粗模型在 4 核下加速比为 3. 82,在8核下为6. 23;中模型在4核下加速比为3. 79,在 8 核下加速比为 6. 23;细模型在 4 核下加速比为 3. 75,8 核下加速比为 6. 26。以上数值计算结果表明,本文给出的并行程序可以缩短计算时间,有效提高计算效率。

由图 6 和图 7 可知,并行加速比和并行效率与模型规模无关,表明本文的并行计算程序具有较好的扩展性。图 7 表明并行效率随着核个数增加而减



(a) 撞击前　　(b) 撞击后

图 5　Taylor 杆的变形和塑性应变

Fig. 5　Deformation and plastic strain of Taylor bar

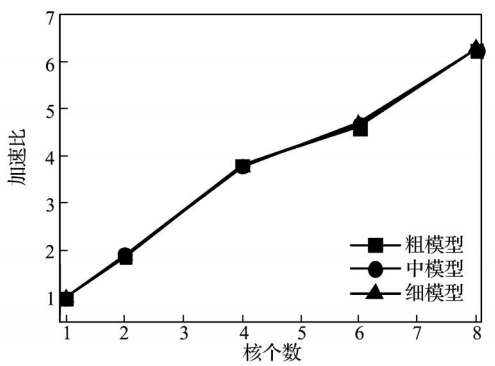


图 6　加速比 S_p 随核个数的变化

Fig. 6　Speedup curve S_p-thread

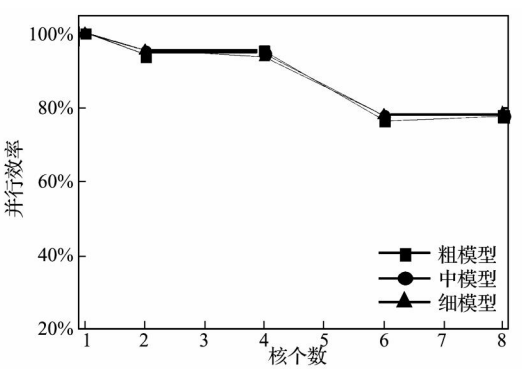


图 7　并行效率 e_f 随核个数变化

Fig. 7　Parallel efficiency curve e_f-thread

小,其原因在于多核计算机的内存访问带宽是共享的,这样线程过多便会影响内存的访问效率。

6 结 论

本文阐述了显式物质点法的计算过程及程序流程图,阐述了 OpenMP 的并行机制,基于区域分解方法避免了节点变量更新过程中的数据竞争问题。对于无数据竞争的质点变量计算,直接采用了质点循环分解方法进行并行。为了有效提高 CPU 的利用率,给出了基于自适应背景网格的负载平衡算法。

对开发的并行物质点计算程序 MPM3DMP 进行了测试,三种模型的计算表明了本文给出的并行程序可以缩短计算时间,有效提高计算效率;结果表明并行加速比和并行效率不受模型规模影响,本文的并行计算程序具有较好的扩展性。

致谢:感谢蔡杰博士生,薛魏博士,张志谦博士,黄清南研究员等,作者接受了他们的诸多建议。

参考文献(References) :

[1] HARLOW F H. The particle-in-cell computing method for fluid dynamics[J]. *Methods in Computational Physics*,1963,**3**:319-343.

[2] BRACKBILL J U, KOTHE D B, RUPPEL H M. FLIP: a low-dissipation, particle-in-cell method for fluid flow [J]. *Computer Physics Communications*, 1988,**48**:25-38.

[3] SULSKY D, CHEN Z, SCHREYER H. A particle method for history-dependent materials[J]. *Computer Methods in Applied Mechanics Engineering*, 1994, **118**:179-196.

[4] SULSKY D, ZHOU S, SCHREYER H. Application of a particle-in-cell method to solid mechanics [J]. *Computer Physics Communications*, 1995, **87**: 236-252.

[5] ZHANG X, SZE K Y, MA S. An explicit material point finite element method for hyper-velocity impact [J]. *International Journal for Numerical Methods in Engineering*,2006,**66**:689-706.

[6] 马 上,张 雄,邱信明. 超高速碰撞问题的三维物质点法[J]. 爆炸与冲击,2006,**26**:272-278. (MA Shang, ZHANG Xiong, QIU Xin-ming. Three dimensional material point method for hypervelocity impact [J]. *Explosion and Shock Waves*, 2006, **26**: 272-278. (in Chinese)]

[7] GUO Y, NAIRN J A. Three-dimensional dynamic fracture analysis using the material point method[J]. *Computer Modeling in Engineering and Sciences*, 2006,**16**:141-156.

[8] SULSKY D, SCHREYER H L. Axisymmetric form of the material point method with applications to upsetting and Taylor impact problems [J]. *Computer Methods in Applied Mechanics Engineering*, 1996, **139**:409-429.

[9] PARKER S G. A component-based architecture for parallel multi-physics PDE simulation [J]. *Future Generation Computer Systems*,2006,**22**:204-216.

[10] PARKER S G, GUILKEY J, HARMAN T. A component-based parallel infrastructure for the simulation of fluid-structure interaction [J]. *Engineering with Computers*,2006,**22**:277-292.

[11] PANTALE O. Parallelization of an object-oriented FEM dynamics code: influence of the strategies on the Speedup [J]. *Advances Engineering Software*, 2005,**36**:361-373.

[12] COUTURIER R, CHIPOT C. Parallel molecular dynamics using OPENMP on a shared memory machine [J]. *Computer Physics Communications*, 2000, **124**: 49-59.

[13] AYGUADEA E, GONZALEZA M, MARTORELLA X, et al. Employing nested OpenMP for the parallelization of multi-zone computational fluid dynamics applications[J]. *Journal of Parallel and Distributed Computing*,2006,**66**:686-697.

[14] QUINN M J. *Parallel Programming in c with MPI and OpenMP* [M]. McGraw-Hill Companies, New York,2004.

[15] CHANDRA R, DAGUM L, MAYDAN D, et al. *Parallel Programming in OpenMP* [M]. Morgan Kaufmann Publishers,2001.

[16] CHAPMAN B, JOST G, VANDERPAS R. *Using OpenMP: Portable Shared Memory Parallel Programming*[M]. The MIT Press,2007.

Parallelization of 3D explicit material point method using OpenMP

HUANG Peng[1,2], ZHANG Xiong[*1,3], MA Shang[1], WANG Han-kui[1]

(1. School of Aerospace, Tsinghua University, Beijing 100084, China;
2. System Engineering Institute, China Academy of Engineering Physics, Mianyang 621900, China;
3. State Key Laboratory of Structural Analysis of Industrial Equipment,
Dalian University of Technology, Dalian 116024, China)

Abstract: Based on the OpenMP technique, a parallel 3D explicit material point method (MPM) code, MPM3DMP, is developed in this paper. The domain decomposition method is presented for avoiding data races in updating nodal variables. The background grid is decomposed into some uniform patches, and each thread deal with a patch in domain decomposition method. After updating nodes in all patches, their nodal variables are assembled into the global one. The code for updating particle variables can be parallelized using the loop splitting method directly. To test the performance of the developed code, the Taylor bar impact is simulated by using three models with different number of particles on a computer with dual quad-core Intel Woodcrest processors. A speedup of 3.82 is achieved for the coarse model in 4 cores, and 6.23 in 8 cores, while a speedup of 3.79 is achieved for the medium-sized model in 4 cores, and 6.23 in 8 cores. A speedup of 3.75 is achieved for the fine model in 4 cores, and 6.26 in 8 cores. Thus, MPM3DMP has good parallel efficiency and extension ability.

Key words: OpenMP; Explicit MPM; domain decomposition method; loop splitting method; speedup