Calculix三级并行优化及其在天河二号超级计算机中的应用

姜文超[1] 林 穗[1] 王多强[2] 李东明[3] 金 海[2]
(广东工业大学计算机学院 广州 510006)[1] (华中科技大学计算机学院 武汉 430074)[2]
(广州船舶及海洋工程设计研究院 广州 510250)[3]

摘 要 针对开源有限元软件Calculix传统计算模式在大规模数值计算中的低效问题,提出了Calculix三级并行优化策略,即预处理并行优化、节点间并行调度以及节点内多核多线程并行改造。预处理并行优化在方程组分解过程中与分解过程后,分别对其参数矩阵进行有条件的动态舍弃,据此构造了部分列选主元多行双门槛不完全LU分解预处理算法,并对算法的可行性、有效性以及收敛性给出了证明。为充分发挥TH-2超级计算机强大的资源优势,相继给出了基于QoS的节点间任务动态调度算法,以及节点内多核多线程并行任务调度算法,进一步实现计算任务与资源之间的优化匹配和QoS需求。在实验环节中搭建了针对天河二号(TH-2)超级计算环境的有限元并行计算与分析平台,并完成了针对船舶疲劳强度分析问题的实际工程应用测试。理论分析与工程算例测试结果充分证明:Calculix三级并行优化方案能够有效提高Calculix求解线性方程组的速度,在可获取足够计算资源的前提下,与传统计算模式相比,实际工程算例的计算速度平均提高了2～4倍。

关键词 并行计算,超级计算,Calculix,并行优化,疲劳强度分析
中图法分类号 TP311.13 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.03.008

Three-level Parallel Optimization and Application of Calculix in TH-2 Super-computing Environments

JIANG Wen-chao[1] LIN Sui[1] WANG Duo-qiang[2] LI Dong-ming[3] JIN Hai[2]
(School of Computer Science and Technology, Guangdong University of Technology, Guangzhou 510006, China)[1]
(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)[2]
(Guangzhou Ship and Marine Engineering Design and Research Institute, Guangzhou 510250, China)[3]

Abstract To increase the computing efficiency of Calculix in large scale numerical calculation, this paper researched Calculix and proposed a three-level parallel optimization scheme including preprocessing-level, task level and thread level. A multiple row double threshold incomplete LU matrix decomposition strategy based on column pivoting and dynamic abandon was presented. The corresponding algorithm was given and proved to be efficient, available and convergent. Furthermore, to utilize the powerful computing resources of TH-2 supercomputer, a task level parallel scheduling algorithm using between computing nodes and a thread level parallel algorithm using between multiple computing cores were also developed and deployed in the experimental platform which focused on the ship fatigue analysis. Both the theory analysis and the actual engineering cases testing were provided to show that the three level parallel optimization scheme based on Calculix can increase the solving speed of liner equations and the analysis efficiency in engineering design areas, and the average speed-up ratio can reach about 2～4 when enough resources can be obtained.

Keywords Parallel computing, Super-computing, Calculix, Parallel optimization, Fatigue strength analysis

1 Calculix求解器

Calculix是一款开源有限元软件系统,可以完成线性和非线性问题的计算求解,在连续体力学领域,对船体结构疲劳强度的静态与动态特性分析以及解热传导、电磁场、流体力学等连续性问题的求解具有重要的应用价值[1]。Calculix包括两部分:前端处理器CGX和后端处理器CCX。CGX是有限元求解的前处理与后处理器,它负责产生有限元几何模型,接收有限元计算后的结果并对其进行可视化;CCX是Calculix的计算处理器,是进行有限元计算的执行模块。实践证明,求

到稿日期:2015-10-16 返修日期:2016-03-22 本文受广东省科技计划项目(2015B010109001),广东省产学研合作项目(2015B090901051),教育部留学回国人员启动基金项目(14ZK0152),NSFC-广东联合基金(第二期)超级计算科学应用研究专项资助,国家超级计算广州中心,广州市科技计划项目(2012Y2-00040)资助。

姜文超(1977－),男,博士,讲师,主要研究方向为高性能计算、分布式系统、网格计算、云计算,E-mail:jiangwenchao@gdut.edu.cn;林 穗(1970－),女,硕士,副教授,主要研究方向为高性能计算、云计算;王多强(1969－),男,博士,副教授,主要研究方向为高性能计算、分布式系统;李东明(1968－),女,高级工程师;金 海(1966－),男,教授,主要研究方向为高性能计算、分布式系统、网格计算、云计算等。

解线性方程组花费 Calculix 大部分的求解时间[2-3]。所谓预处理技术[13]是指对线性方程组

$$Ax=b \tag{1}$$

进行转换，使其与原方程组同解。要求转换后的方程组更易于迭代求解，产生的转换矩阵 M 称为预条件子。如果 M 为一个在某种程度上和 A 近似的非奇异矩阵，则预处理后的线性方程组为：

$$M^{-1}Ax=M^{-1}b \tag{2}$$

与线性方程组 $Ax=b$ 相同。线性方程组的预处理技术[14]主要有填充元缩减策略、匹配技术、因子分解、近似逆条件子构造等，其中不完全 LU 分解较为常用[4-7]。假设矩阵 A 的 $n-1$ 个顺序主子式不为 0，那么矩阵 A 可以唯一分解为一个下三角矩阵 L 和一个上三角矩阵 U 的乘积，即：

$$A=LU \tag{3}$$

若令误差矩阵 $Q=\overline{L}\ \overline{U}-A$ 满足一定的条件限制，例如某些特定位置的元素值为零，则称 $\overline{A}=\overline{L}\ \overline{U}$ 为不完全 LU 分解。在不完全 LU 分解中，选择合适的填充元缩减策略十分重要。

2　基于动态舍弃策略的部分列选主元多行双门槛不完全 LU 分解算法

2.1　动态舍弃策略

线性方程组进行迭代求解前，首先对矩阵进行划分，再对各个部分进行列选主元，然后对矩阵元素进行第一次舍弃，留下满足舍弃策略条件的矩阵元素来进行不完全 LU 分解；另外，需要对已完成分解的 L 和 U 中的某几行元素进行第二次舍弃。这里对两次舍弃策略进行优化，第一次舍弃采用 τ 与矩阵的谱范数乘积作为舍弃界限，舍弃小于此界限的矩阵元素。第二次舍弃则使用动态多行舍弃策略，根据矩阵某几行非零元素的个数与参数的 τ 乘积作为舍弃界限，舍弃超过此界限的矩阵元素。经过两次舍弃后，把各个部分的计算结果进行归约，得到的矩阵为最终的预处理矩阵，供线性方程组进行迭代求解使用。

在矩阵分解过程中，可以对分解因子中的元素值进行判断，舍去幅度较小的元素，但是这种舍弃方法开销较大，并且当门槛越来越小时，趋近于直接法，无法达到控制存储空间和减少计算量的目的。针对该情况，引入矩阵范数作为舍弃参数。矩阵范数是描述矩阵扰动灵敏度的特征数，虽然预处理的迭代舍弃可以提高计算效率，但也引入了舍弃误差，这里通过矩阵范数来控制预处理的舍入误差。谱范数满足定义 1，谱范数是刻画元素间距离的量，谱半径一般小于谱范数，所以使用矩阵谱范数作为参数，可以较好地控制不完全 LU 分解的舍弃过程造成的误差。在矩阵进行不完全分解时，选取 $\tau(0<\tau<1)$，当矩阵元素小于 τ 与矩阵的谱范数乘积时，则舍弃；反之，则留下。分解完成后，分解因子 L 和 U 分别保留 p 个绝对值较大的元素。

2.2　部分列选主元多行双门槛不完全 LU 分解算法

对 Calculix 有限元求解器的 ccx 模块代码进行并行化修改。首先初始化 MPI，使用 FrontMtx_MPI_split 对矩阵进行划分，实现节点间的任务分配，然后在各节点实施基于动态舍弃策略的部分列选主元多行双门槛不完全 LU 分解，算法中涉及的主要函数包括：1）*column pivoting*()，对矩阵进行列选主元；2）*abandon strategy*()，对多个行向量进行谱范数条件下的舍弃策略；3）*dynamic abandon strategy*()，对上三角矩阵和下三角矩阵进行动态舍弃策略；4）FrontMtx_MPI_Reduce，对结果进行归约，得到最终结果。算法可以通过调用 Arpack 接口得以实现。对 Arpack 中的 *preconditioning* 函数进行修改来完成矩阵的预处理，最后再调用并行程序进行计算求解。核心算法伪代码描述如下。

```
Function preconditioning(n,b,A)
  for i=1:n
    if z==b then z=0;
    else z++;
    a_ii=column pivoting(A);
    α=a_i;
    for k=1;i-1 and when α_k=!0
      α_k=α_k/a_kk;
      abandon strategy(α);
      if α_k=! 0 then α=α-α_k * u_k;
    end for k
    abandon strategy(α);
      if z==b;
      then dynamic abandon strategy(α);
    for j=1:i-1
        l_ij=u_ij;
    for j=i:n
        u_ij=α;
end for i
```

3　任务并行调度策略

3.1　节点间任务调度

鉴于 TH-2 作业提交模式和 Calculix 的运行机制，为实现各结点间的负载平衡，设计基于 QoS 的动态任务调度方案，节点间的任务调度算法描述如下：

```
MPI_Init(&argc,&argv);//定义变量,MPI 初始化
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
MPI_Comm_size(MPI_COMM_WORLD,&nproc);//利用 MPI 消息传递机制读取数据,创建对象 A→InpMtx,b→DenseMtx,并分配数据。
InpMtx_init(mtxA,INPMTX_BY_ROWS,type,nent,0);
DenseMtx_init(mtxY,type,0,0,nrow,nrhs,1,nrow);//查找最小填充排序,产生分配到结点的映射图,并将数据广播出去。
adjIVL=InpMtx_MPI_fullAdjacency(mtxA,stats,msglvl,msgFile,
        MPI_COMM_WORLD);
frontETree=orderViaMMD(graph,seed+myid,msglvl,msgFile);
frontETree=ETree_MPI_Bcast(frontETree,root,msglvl,msgFile,
        MPI_COMM_WORLD);//各处理器求解计算分量的新值,并通过扩展收集操作重新广播给所有处理器,进行下一次计算。
FrontMtx_MPI_split(frontmtx,solvemap,stats,msglvl,msgFile,
firsttag,MPI_COMM_WORLD);
FrontMtx_MPI_postProcess(frontmtx,ownersIV,stats,msglvl,msg-
```

File,firsttag,MPI_COMM_WORLD);//各结点机数据更新,利用MPI消息传递汇总数据到管理结点。
MPI_Finalize();//MPI进程结束。

3.2 节点内任务调度

节点间任务调度完成后,Calculix后端计算模块CCX可以调用SpoolesMT实现在单个计算结点上的多核多线程并行化计算。针对SpoolesMT的任务调度并行处理算法描述如下:

```
Begin
//确定可用的CPU核心数
if(nthread > (nfront=FrontMtx_nfront(frontmtx)))
    {nthread=nfront;}
//进行多核域分解映射
DV_init(cumopsDV,nthread,NULL);
//确定各CPU核心与front的匹配
ownersIV=ETree_ddMap(frontETree,type,symmetryflag,
    cumopsDV,1.0/(2.0*nthreads));
//对各小单元矩阵在映射到的线程上进行因式分解
SubMtxManager_init(mtxmanager,LOCK_IN_PROCESS,0);
FrontMtx_init(frontmtx,frontETree,symbfacIVL,
    type,symmetryflag,
    FRONTMTX_DENSE_FRONTS,
    pivotingflag,LOCK_IN_PROCESS,0,
    NULL,mtxmanager,msglvl,msgFile);
//多核多线程并行进行矩阵分解
Rootchv=FrontMtx_MT_factorInpMtx(
    frontmtx,mtxA,tau,droptol,chvmanager,
    ownersIV,lookahead,&error,cpus,stats,msglvl,msgFile);
//矩阵分解后处理
FrontMtx_postProcess(frontmtx,msglvl,msgFile);
//并行分析各对象参数初始化
solvemap=SolveMap_new();
SolveMap_ddMap(solvemap,symmetryflag,FrontMtx_upperBlockIVL(frontmtx),FrontMtx_lowerBlockIVL(frontmtx),nthread,ownersIV,FrontMtx_frontTree(frontmtx),seed,msglvl,msgFile);
mtxX=DenseMtx_new();
DenseMtx_init(mtxX,type,0,0,neqns,nrhs,1,neqns);
DenseMtx_zero(mtxX);
//多核多线程并行求解
FrontMtx_MT_solve(frontmtx,mtxX,mtxB,mtxmanager,solvemap,
cpus,msglvl,msgFile);
//结果回置
DenseMtx_permuteRows(mtxX,newToOldIV);
End
```

4 系统实现与实验分析

基于Maxdomainsize,Maxsize和Maxzeros的最佳搭配组合,使用船体相关独立组件模型作为实际算例分别进行测试,测试采用记录5次计算时间求平均值的方法,其中3个测试算例的模型图如图1—图3所示。

算例A:网格划分数是17524,单元数是8500。

算例B:网格划分数是7690,单元数是718。

算例C:网格划分数是85968,单元数是85832。

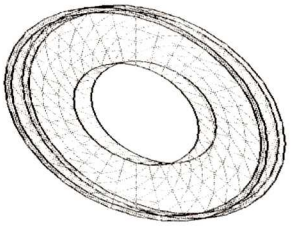

图1 算例A的模型图

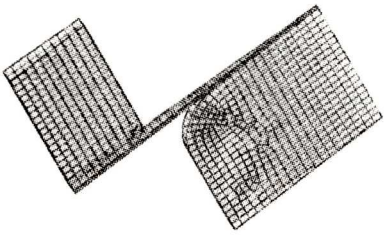

图2 算例B的模型图

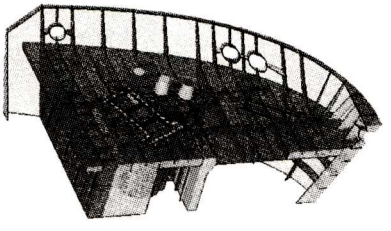

图3 算例C的模型图

并行计算中的加速比用并行前的执行速度和并行后的执行速度之比来表示,具体实验测试结果如表1和表2所列。

表1 各算例测试结果/s

算例	单节点改造前	单节点改造后	双节点改造前	双节点改造后
算例A	213	138	138	73
算例B	97	55	61	32
算例C	1071	787	602	376

表2 各算例测试加速比

算例	改进前加速比	改进后加速比
算例A	1.54	1.89
算例B	1.76	1.90
算例C	1.36	1.60

在对以上3个算例进行具体分析之后,为进一步证明该并行改造方案的有效性,实验取20个数据规模差距不大的船舶独立部件作为研究算例进行测试,分别对这些算例逐渐提交到1节点、2节点、3节点和4节点上的平均计算时间进行对比,统计结果如图4所示。

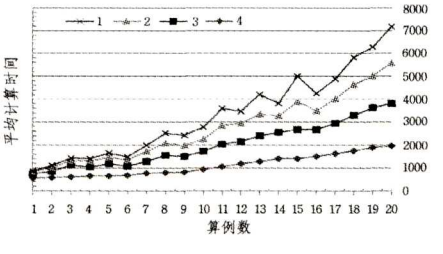

图4 平均计算时间对比图

由图4可知,虽然由于不同算例的计算量有一定差异,在

不同计算节点下，平均计算时间都没有随算例的增加呈现出严格的增长趋势，但无论是单节点还是多节点，该并行方案都有效地提高了计算效率，并且随着节点数的增加，效果更加明显。

结束语　针对开源有限元软件 Calculix 传统计算模式在大规模数值计算与分析过程中的低效问题，提出了包括预处理并行优化、节点间并行调度与节点内多核多线程并行改造在内的 Calculix 三级并行优化方案，理论分析与实际工程算例测试结果充分证明：Calculix 三级并行优化方案可以有效地提高工程计算与分析过程中求解线性方程组的速度。

参 考 文 献

[1] GUO L L, CHEN Z F, LUO J R, et al. A Review of the Extended Finite Element Method and its Applications [J]. Chinese Quarterly of Mechanics, 2011, 32(4): 612-625. (in Chinese)
郭历伦，陈忠富，罗景润，等. 扩展有限元方法及应用综述[J]. 力学季刊，2011，32(4)：612-625.

[2] GOURI D, EMMANUEL L. Finite Element Method [M]. John Wiley & Sons, 2012.

[3] YU Y T, DU P G, WANG Z W. Research on the current application status of finite element method[J]. Journal of Machine Design, 2005, 22(3): 6-9. (in Chinese)
于亚婷，杜平安，王振伟. 有限元法的应用现状研究[J]. 机械设计，2005，22(3)：6-9.

[4] CALCULIX. A Free Software Three-Dimensional Structural Finite Element Program [OL]. http://www.calculix.de.

[5] FUNG Y C. Foundations of Solid Mechanics [M]. Englewood Cliffs, N J: Prentice Hall, 1965.

[6] DHONDT G, WITTIG K. Calculix: A free software three-dimensional structural finite element program[OL]. http:/www.calculix.de.

[7] WANG H, DING J H. Constructing ANSYS distributed high performance platform using LINUX cluster technology[J]. Development and Application of High Performance Computing, 2011(1): 34-39. (in Chinese)
王惠，丁峻宏. LINUX 集群技术构建 ANSYS 分布式高性能计算平台[J]. 高性能计算发展与应用，2011(1)：34-39.

(上接第 15 页)

[3] LAPPAS T, VIEIRA M R, GUNOPULOS D, et al. On The Spatiotemporal Burstiness of Terms[J]. PVLDB, 2012, 5(9): 836-847.

[4] LAPPAS T, ARAI B, PLATAKIS M, et al. On burstiness-aware search for document sequences[C]// KDD. 2009: 477-486.

[5] ZAHARIA V, DAS T, LI H Y, et al. Discretized streams: fault-tolerant streaming computation at scale[C]// SOSP. 2013: 423-438.

[6] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters[J]. Commun. ACM (CACM), 2008, 51(1): 107-113.

[7] XIANG L H, CHEN Y W, ZHANG Y L. High Order Matrix Multiplication by Mapreduce Algorithm Based on Hadoop Platform[J]. Computer Science, 2013, 40(S1): 96-98. (in Chinese)
向林泓，陈芋文，张昱琳. 基于 Hadoop 平台的高阶矩阵相乘 MapReduce 算法研究[J]. 计算机科学，2013，40(S1)：96-98.

[8] YAN Y L, DONG Y H, HE X M, et al. FSMBUS: A Frequent Subgraph Mining Algorithm in Single Large-Scale Graph Using Spark[J]. Journal of Computer Research and Development, 2015, 52(8): 1768-1783. (in Chinese)
严玉良，董一鸿，何贤芒，等. FSMBUS：一种基于 Spark 的大规模频繁子图挖掘算法[J]. 计算机研究与发展，2015，52(8)：1768-1783.

[9] MANKU G S, MOTWANI R. Approximate Frequency Counts over Data Streams[J]. Proceedings of the VLDB Endowment, 2003, 5(1): 1699.

[10] DEMAINE E D, LÓPEZ-ORTIZ A, MUNRO J I. Frequency Estimation of Internet Packet Streams with Limited Space[C]// ESA. 2002: 348-360.

[11] KARP R M, SHENKER S, PAPADIMTRIOU C H. A simple algorithm for finding frequent elements in streams and bags[J]. ACM Trans. Database Syst. (TODS), 2003, 28: 51-55.

[12] MISRA J, GRIES D. Finding Repeated Elements[J]. Science of Computer Programming, 1982, 2(2): 143-152.

[13] METWALLY A, AGRAWAL D, ABBADI A E. Efficient Computation of Frequent and Top-k Elements in Data Streams[C]// ICDT. 2005: 398-412.

[14] CHARIKAR M, CHEN K, FARACH-COLTON M. Finding Frequent Items in Data Streams[J]. Theoretical Computer Science, 2002, 312(1): 3-15.

[15] CORMODE G, MUTHUKRISHNAN S. What's hot and what's not: tracking most frequent items dynamically[J]. ACM Trans. Database Syst. (TODS), 2005, 30(1): 249-278.

[16] CORMODE G, MUTHUKR I SHNAN S. An improved data stream summary: the count-min sketch and its applications[J]. J. Algorithms (JAL), 2005, 55(1): 58-75.

[17] KLEINBERG J. Bursty and hierarchical structure in streams [C] // Proceeding of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2002: 91-101.

[18] PUI G, FUNG C, YU J X, et al. Parameter Free Bursty Events Detection in Text Streams[C]// VLDB. 2005: 181-192.

[19] HE Q, CHANG K Y, LIM E P. Analyzing feature trajectories for event detection[C]// SIGIR. 2007: 207-214.

[20] VLACHOS M, MEEK C, VAGENA Z Z, et al. Identifying Similarities, Periodicities and Bursts for Online Search Queries[C]// SIGMOD. 2004: 131-142.

[21] LAPPAS T, ARAI B, PLATAKIS M, et al. On burstiness-aware search for document sequences[C]// ACM SIGKD International Conference on Knowledge Discovery and Data Mining. Paris, France, 2009: 477-486.

[22] DOBKIN D P, EPPSTEIN D. Computing the Discrepancy[C]// Symposium on Computational Geometry. 1993: 47-52.

[23] ARLITT M, JIN T. World Cup Web Site Access Logs[OL]. http://ita.ee.lbl.gov/html/contrib/WorldCup.html.