

容错的并行多重网格算法

田 凯, 卢宇彤, 所 光

(国防科技大学 计算机学院, 湖南 长沙 410073)

摘 要: 高性能计算机上并行程序用到的结点越来越多,而在程序运行期间中发生结点失效的概率也随之增大.对于计算时间很长的程序,容忍结点失效的容错能力显得尤为重要.并行多重网格算法(MG)被广泛用于求解大型工程和物理问题中的偏微分方程组的数值解.为了实现 MG 算法的容错能力,提出了一种基于容错 MPI 的容错并行多重网格算法 FT-MG.实验结果表明:FT-MG 算法在引入少许开销的条件下实现了 MG 算法的容错能力.

关键词: 容错;MG;FT-MG;容错 MPI

中图分类号: TP391

文献标识码: A

文章编号: 1000-7180(2012)09-0001-04

Fault Tolerant Parallel Multiple Grid Algorithm

TIAN Kai, LU Yu-tong, SUO Guang

(School of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract: As the number of nodes used in high performance applications increases, the probability that failures occur in the computing processes of applications increases. For long running applications it is essential that fault tolerance be used to survive fail-stop failures. The parallel multiple grid algorithm (MG) is widely used to solve the PDEs in large-scale project and physical problems. To implement the fault tolerant ability of MG algorithm, we design a fault tolerance algorithm (named FT-MG) based on fault tolerance MPI. Experimental results prove the fault tolerance ability of FT-VMG and demonstrate that the fault tolerant overhead of FT-MG is pretty small.

Key words: fault tolerance; MG; FT-MG; fault tolerance MPI

1 引言

高性能计算发展的速度总是落后于对其性能的需求,高性能计算程序的执行时间可能长达数十天^[1].在高性能计算程序运行期间发生结点失效的概率越来越大,如果程序在发生结点失效后重新运行,将会带来巨大的时间和计算资源的浪费,甚至导致程序无法完成计算.因此,容错对于高性能计算程序越来越重要.

MG 算法可以有效的迭代求解偏微分方程的数值解,被广泛应用于求解超大型数值计算问题^[2-3].为了使得采用 MG 算法的高性能计算程序具有容错能力,文中提出了一种容错的 MG 算法:FT-MG. FT-MG 算法不仅适用于所有类型的 MG 算法,而

且与 MG 算法相比改动较少,易于实现.实验结果表明,FT-MG 算法在实现容错的同时引入了较小的性能开销.

2 相关工作

与本文研究工作相关的主要技术有 Diskless Checkpoint 和 FT-MPI.检查点技术是经常采用的一种容错方法.与传统的检查点技术相比,Diskless Checkpoint 技术把检查点数据保存在内存中而不是写入文件系统,这样就避免了多个结点同时写入文件系统带来的性能瓶颈^[1,4].FT-MPI^[5-7]有两个含义,一是对 MPI 标准的容错扩展定义,二是一个建立在 HARNESS 之上,具有用户级、系统级容错能力的 MPI 系统.后者是前者的实现.在本文中

收稿日期: 2012-02-10; **修回日期:** 2012-03-16

基金项目: 国家自然科学基金项目(61120106005,61025009);国家“八六三”计划项目(2012AA01A301)

把后者称为 FT-MPI 系统以做区别. HARNESS 为 FT-MPI 系统提供进程管理和失效通知的功能. FT-MPI 没有在应用级定义如何处理进程失效,而是为 MPI 对象在失效发生前后定义了各种状态,提供了很多用于处理 MPI 进程失效的功能,把容错方法的选择权留给程序开发者. 典型的 MPI 容错包括以下三步:失效检测,失效通知和恢复. 对于失效检测和通知,FT-MPI 只做了一个规定:运行环境发现了失效进程,随后,并程序的其他进程都被告知失效事件. 当一个进程收到失效事件的通知后,它的状态从 no failures 变成 failure recognized,之后它就进入了失效恢复过程. 对于失效恢复,分为两个部分:MPI 库和运行环境的恢复、应用程序的恢复. 其中,后者应由程序开发者负责,前者则是 FT-MPI 要解决的问题.

3 FT-MG 算法

FT-MG 算法是以 MG 算法为基础而设计的. MG 算法有 V 循环、W 循环、FMV 循环等多种类型. 所有类型的 MG 算法都有两个共同点. 一是均为循环迭代结构,主循环是算法的核心部分. 二是算法求解的线性方程组系数矩阵均为稀疏矩阵. 我们基于 MG 算法的这两个特性设计了 FT-MG 算法. 程序要获得容错能力离不开运行环境的支持,FT-MG 算法使用容错 MPI 作为容错支持环境. 容错 MPI 是我们对 MPI 标准进行容错扩展而得到的,与 FT-MPI 类似,但又比 FT-MPI 更简单、易于使用. 我们将会在其他文章中对其进行详细介绍. 下面从主要结构和执行流程两方面对 FT-MG 算法进行深入分析.

FT-MG 算法主要包括两个部分:在 MG 算法循环体开始处的保存变量部分和在循环体上方的数据恢复部分. 检查点的位置被选择在主循环的开始处. 这么做有两个原因,一是检查点代码在这个位置会被周期的执行,实现了周期的检查点操作,二是,在该处程序的状态简单,因此需要保存的变量较少. 容错算法主要结构如图 1 所示:

图 1 中的 MPI_Send(), MPI_Receive(), setjmp() 函数均为容错 MPI 提供的库函数. setjmp() 的作用是使正常进程在失效发生后跳转至此处. 算法需要的失效检测以及进程状态管理功能均由容错 MPI 提供. X 在这里指的是关键变量集合,而不是单个变量. X 被保存在本地内存和伙伴进程的内存中. 保存在本地内存的冗余数据是为本地进程在

其他进程发生失效时恢复关键变量数据而准备的,而保存在伙伴进程的冗余数据是为本地进程失效这种情况准备的,代替失效进程的重启进程从伙伴进程处接收关键变量数据从而完成恢复. 由于进程失效的局部性,伙伴进程的距离应该尽量远. 从图中不难看出 FT-MG 不论是在本地内存还是伙伴进程的内存中均维持两组关键变量状态数据:当次循环和上次循环的数据. 这是考虑到在没有完成所有的检查点操作时发生进程失效的情景. 在这种情况下,部分伙伴进程保存的数据可能还是上次检查点的状态,导致了检查点数据的不一致,此时程序只能根据上一次的检查点数据,恢复到上一次循环的开始处. 在算法的数据恢复部分包括分别为发生失效后的正常进程和重启进程设计两个代码块. 在选择了一致状态后,正常进程恢复关键变量数据,同时失效进程的伙伴进程向重启进程发送状态数据. 重启进程接收到状态数据后完成数据恢复.

```

MG 初始化
//数据恢复
If(失效后正常进程)
    选择一致的状态 s
    x=xstate[s]
    if(是失效重启进程的伙伴进程)
        MPI_Send(xpstate[s],partner)
    endif
Endif
If(失效后重启进程)
    MPI_Receive(x,partner)
Endif
//主循环
Doiteration=1,...
    setjmp//设定失效跳转点
    if(没有进程失效)
        //保存关键变量
        s 被周期的赋值 1,2
        xstate[s]=x
        MPI_Send(x,partner)
        MPI_Receive(xpstate[s],partner)
    else
        跳转到数据恢复
    endif
MG 循环体
Enddo
MG 计算结果的处理与输出
  
```

图 1 FT-MG 算法结构图

FT-MG 算法的执行流程有三种,如图 2 所示.

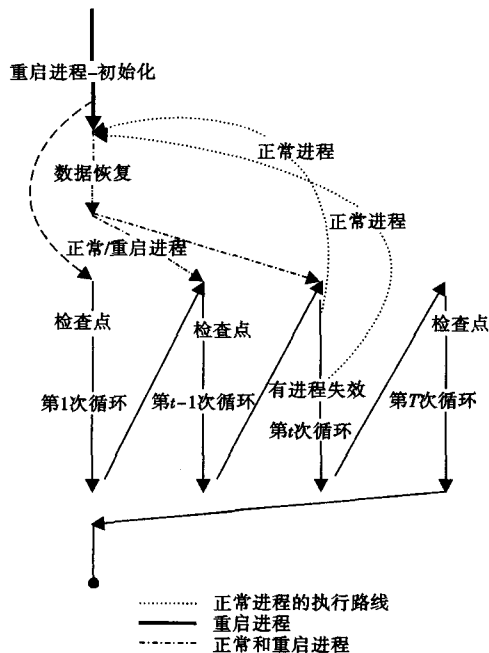


图 2 FT-MG 算法执行流程图

当程序正常执行时,数据恢复部分的代码不会被执行.所有进程在每次循环的开始处保存关键变量,将其保存在本地内存和伙伴结点的内存中.此时所有进程的执行流程都是一样的.

假设有进程在第 t 次循环发生失效,且所有进程都已经完成检查点操作.容错 MPI 在检测到失效事件后,立即把事件通知到与失效进程在同一通信器内的其他正常进程.正常进程在收到失效事件通知后立即跳转至 `setjmp()` 函数设定的位置,然后跳转至数据恢复代码开始处.此时,正常进程的关键变量数据与检查点相比可能已经改变,因此需要从保存在本地内存中的检查点数据恢复关键变量.如果是失效进程的伙伴进程,还应向重启进程发送对应的状态数据.代替失效进程的重启进程则从程序起始处执行.为了使重启进程不阻塞在 MG 初始化部分的通信调用,应该为重启进程添加相关跳转语句.在数据恢复部分,重启进程从伙伴进程处接收状态数据完成对关键变量的恢复.所有进程完成数据恢复后跳转至第 t 次循环的开始处,至此,程序恢复正常的执行.

假设有进程在第 t 次循环发生失效,且有进程没有完成检查点操作.此时,第 t 次循环的状态数据不是完整的.因此,不能恢复到第 t 次循环.正常进程与重启进程都用第 $t-1$ 次循环的状态数据恢复

关键变量.完成恢复后,所有进程跳转至第 $t-1$ 次循环的开始处.程序在第 $t-1$ 次循环恢复正常的执行.

4 FT-MG 算法实现

要实现 FT-MG 算法,核心问题是需要在检查点保存哪些变量.这与 MG 算法的不同类型和具体实现相关.可以通过分析变量的 Define-Use 链信息找到这些变量. Define-Use 链(简称 DU 链)是一个包含对变量的定义与使用(即写与读)信息的数据结构,可由程序代码静态分析得到^[8].

定义 1 关键变量集合是指满足以下条件的变量集合:在检查点保存集合中全体变量,在有进程失效后,恢复集合全体变量可以使程序在检查点继续正常执行.

定义 2 对于变量 X ,如果存在一条 Define-Use 链穿过了检查点 C ,则 X 为活性变量,否则为非活性变量.

定理 1 全体活性变量组成的集合是一个关键变量集合.

证明:只需证明不恢复非活性变量也能使程序在检查点恢复正常执行.对于任意一个非活性变量 X ,不存在一条 DU 链穿过了检查点 C .假设在检查点不保存变量 X .有进程失效后,正常进程要跳转至数据恢复代码开始处进行数据恢复,此时 X 的值与上一次检查点相比可能不同,但因为没有保存 X ,所以不会恢复 X .恢复完成后再跳转至循环体,进程遇到的第一个涉及 X 的语句是对 X 进行定义,即赋值,此时 X 的数据恢复正常.同理,对于重启进程, X 的数据在执行到对 X 定义的语句时自动恢复正常.因此,不保存 X 程序也可以恢复正常执行.得证.

5 实验与分析

根据上述分析,针对求解三位泊松方程 $\Delta^2 u = v$ 的 V 循环 MG 算法实现了容错程序 FT-VMG.程序 VMG 是相应的非容错实现.以 NPROCS 表示进程数,CLASS 表示问题规模,分别在 (CLASS = C, NPROCS = 32)、(CLASS = C, NPROCS = 64)、(CLASS = D, NPROCS = 64)、(CLASS = D, NPROCS = 128) 四种情形下测试这两个程序.其中,CLASS = C 时,程序要处理的网格大小为 512×512 ,做 20 次 V 循环,CLASS = D 时,网格大小为 $1024 \times 1024 \times 1024$,做 50 次 V 循环.在实验的集群系统中,每个计算结点都是采用采用两个主频为

2. 93GHz Intel Xeon X5670 处理器的 12 核 SMP 系统, 结点主存为 24GB, 结点之间使用单向带宽 160Gbps 的 GLEX 互联连接. 结点上使用的操作系统为 Red Hat Enterprise Linux Server release 5. 5. 对于 VMG 测其执行时间, 而对 FT-VMG 记录正常执行的执行时间、在执行过程中发生进程失效的执行时间. 测试结果如图 3、图 4 所示.

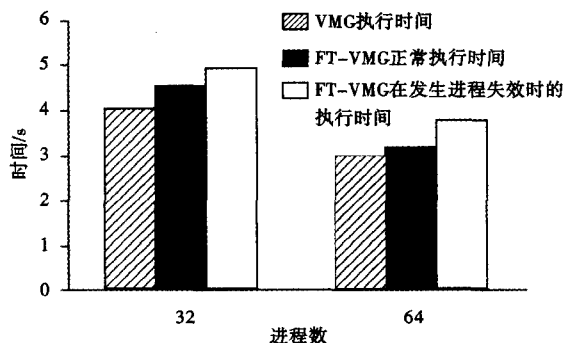


图 3 CLASS=C 时程序的执行时间

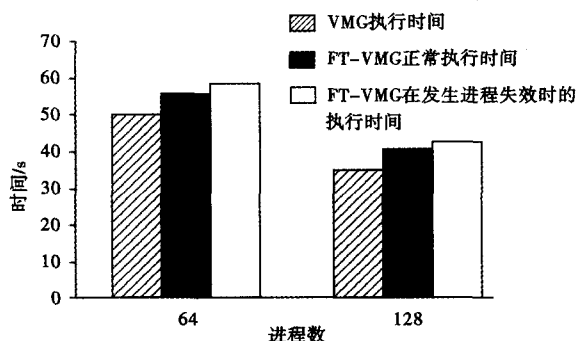


图 4 CLASS=D 时程序的执行时间

从图 3、图 4 中不难看出, VMG 的执行时间、FT-MG 没有遇到进程失效的执行时间、FT-MG 在执行过程中发生进程失效的执行时间总是保持递增顺序且增幅不大. 由于 FT-MG 算法中的检查点操作会在正常执行时周期的进行, 这会用掉额外的执行时间. 因此, FT-VMG 在正常执行时其执行时间会比 VMG 的时间长, 这就是容错带来的时间性能开销. 另一方面, 增加的幅度并不大, 说明了 FT-MG 算法为容错引入的时间性能开销小. 失效恢复带来的时间开销主要与失效次数相关. 从图中可以看出这个时间开销相对于程序正常执行时间是很小的.

6 结束语

本文提出了一种基于容错 MPI 的容错 MG 算法 FT-MG. 实验表明, FT-MG 算法在引入少量性

能开销的情况下实现了容错, 而且具有较好的可扩展性. FT-MG 算法的容错方法可推广到其他具有类似循环迭代结构的并行 SPMD 算法中, 而且实际应用中有很多高性能程序都符合这种要求. 本文证明了可以根据程序中变量的 Define-Use 链的特点可以找出关键变量, 这意味着可以通过编译技术在程序编译时自动的找出关键变量, 进一步, 还可以在编译过程中直接生成容错的程序代码, 这些对程序开发者都是透明的. 这将是下一步的工作.

参考文献:

- [1] Gropp W, Lusk E. Fault tolerance in message passing interface programs[J]. International Journal of High Performance Computing Applications, 2004, 18(3): 363-372.
- [2] 李晓梅, 莫则尧. 多重网格算法综述[J]. 中国科学基金, 1996, 1(1): 8-15.
- [3] 刘超群. 多重网格法及其在计算流体力学中的应用[M]. 北京: 清华大学出版社, 1995.
- [4] Plank J S, Li K, Puening M A. Diskless checkpointing[J]. IEEE Transactions on Parallel and Distributed Systems, 1998, 9(10): 972-986.
- [5] Fagg G E, Bukovsky A, Dongarra J J. HARNESS and fault tolerant MPI[J]. Parallel Computing, 2001, 27(11): 1479-1495.
- [6] Fagg G, Gabriel E, Bosilca G, et al. Extending the mpi specification for process fault tolerance on high performance computing systems[C]// Proceedings of the 2004 International Supercomputing Conference. Heidelberg, Germany: Primeur, 2004.
- [7] Fagg G, Gabriel E, Chen Z, et al. Process fault-tolerance: semantics, design and applications for high performance computing [J]. International Journal for High Performance Computing Applications, 2005, 19(4): 465-477.
- [8] 裘巍. 编译器设计之路[M]. 北京: 机械工业出版社, 2011.

作者简介:



田 凯 男, 硕士研究. 研究方向为高性能计算.