软件系统 PVM 并行调试器 (14)

计算机科学 1996 Vol. 23 №. 4

65-69

PVM 上的并行调试器

TP31

黄 宁 金茂忠
(北京航空航天大学软件工程研究所 北京100083)

摘 要 This paper gives an introduction of portable message passing environment PVM and serveral popular softwares for PVM applications, and analizes PVM graghical monitoring environment XPVM, expounding the difference between XPVM environment and the practical PVM parallel debugging environment, then probes into the technological difficulties and design requirements in parallel debugging environment.

关键词 PVM(Parallel Virtual Machine), Message passing model, Parallel debugging, Visualization.

一、引 言

现代科学对计算机计算能力的需求，极大地推动了并行计算机的研究与发展，推出了许多高性能的并行计算机，如传统的向量巨型机 CRAY C-90，大规模并行处理系统和可扩展的并行处理系统 iNTEL PARAGON，可扩展工作站机群 IBM SPI 等。与并行计算机硬件的发展相比，并行软件的研究尤显滞后。

随着并行计算在各领域的应用，并行软件的研究近年来日益得到重视。基于并行软件设计所呈现的与传统串行软件不同的特点，人们提出了不少程序设计模型，其中显式消息传递模型由于克服了传统并行计算机兼容性差的缺点，能适应于异构型并行计算机并具有方便用户编程，易于移植和易于实现等特点，已逐渐成为并行软件研究中的热点。相对于其它消息传递软件系统，如 Express，Linda 和 MPI 而言，PVM 软件的功能并不强大，但由于其系统规模小，而且可在其核心之上扩充附属功能层，同时又是可自由获取的公开软件，因此 PVM 软件得到了广泛的推广和应用，这些领域包括分子动力学、模拟、超导研究和气象科学等。在我国刚完成鉴定的并行机曙光1000以及北航正在研制的可扩展并行机群之上都移植了 PVM 软件系统。

与串行程序相比较，并行程序的开发要困难得多，因此，除了并行程序设计环境，如 PVM 之外，人们迫切需要一些帮助用户进行并行程序开发的环境，如可视化的监视/调试器。PVM 提出的主要目标之一就是为用户提供一个异构机的并行调试器，但至今为止除了有一个监视器环境 XPVM 之外，尚未推出一个真正的并行调试环境。从本文以下的分析中可以看到，现有的一些软件如 Xmdb，CoCheck 和 XPVM 虽然可以帮助用户理解一些 PVM 并行应用程序的行为，但尚不能满足用户的实际需求。

并行调试是并行处理中的一个难点，其理论与技术尚不成熟，许多问题还有待解决。例如如何定义并行运行中多进程的“单步执行”与“断点”，如何在采集跟踪信息时，尽量减少“入侵影响”，在基于显式消息传递模型中，如何保证调试中进程间的同步等。本文将结合目前并行调试技术的现状，讨论基于 PVM 的并行调试环境的设计与实现中所关注的若干问题。

二、PVM 软件

PVM(Parallel Virtnal Machine)是由软件实现的基于消息传递模型的并行虚拟机。PVM 软件是由 Oak Ridge National Liboratory (ORNL) 的 AL Geist，Emory University 的 Vaidy Sunderamu 等人于1989年开始研制的，目前已发展到3.3.9版，而且还在推出新版本。PVM 软件支持异构的 unix 计算机用异构网络联接成一个“虚拟”的并行计算机，使它能象一台大型并行计算机一样进行工作。

2.1 组成

PVM 软件由两部分内容构成，一个是 Daemon，它驻留在用户的计算机中，形成虚拟机；另一部分是 PVM 接口例程，包括用户可调用的系统服务功能，

如消息传递、派出子进程、协调任务和修改虚拟机的配置等。

Daemon 又称 Pvmd3，有时简称为 Pvmd，它不做任何计算，而是提供消息发送器和控制器的服务，它是每一台计算机之间联接的桥梁，同时提供进程控制和错误识别等功能。

接口例程库是允许任务和 Pvmd 及其它任务打交道的函数集，它们能提供的函数包括消息打包、解包以及 PVM 的系统调用。PVM 的应用程序必须和此库连接方能应用 PVM。该库函数用 C 语言写成，C++所写的应用程序也可以连接到 PVM 库上，Fortran 应用程序可以通过一个 Fortran 77的接口来调用这些例程。例程库的高层包括了绝大部分程序设计的界面函数，它们以与机器和操作系统无关的形式写成，并与低层的函数分开。PVM 移植到新的 OS 或 MPP 之上时，仅需修改或用新机器的特殊文件来替换低层函数。

2.2 特点

毫无疑问，PVM 得以广泛使用的原因是因为它的公开性、通用性、易实现性及工作站网络上的适用性，它既适于 TCP/IP 网络，又适于 MPP 大规模并行系统。相对于其它可移植的消息传递环境而言，PVM 支持的界面原语很小，象其它环境所提供的收集性通讯（Collective Communication）、进程拓扑（Process Topologies）PVM 都省略了。但 PVM 支持基于其上的附加功能层的扩充，如 PICH（Portable Instrumentable Communication Library）。PVM 的这种省略及其可扩充性，使其系统规模仅约1.3Mb 大小，这也正是 PVM 之所以成功的原因之一。

和大部分消息传递环境不同，PVM 支持动态进程和虚拟机的管理而不是静态定义的进程结构。动态进程组属于以 PVM 基本功能为基础的附加功能层。一个进程可以归属于多个进程组，而进程组可以在计算的任何时候动态变化。PVM 为任务加入和离开进程组提供了相应的程序。

PVM 支持异构网络计算，使用户可以利用现已存在的环境配置成一个工作系统，其硬件平台可以包括不同体系结构的计算机，并且它们之间的相互联系可以使用多种网络类型。这种情况的特例就是工作站机群，即由一个单一的局网相联的相似或相同的工作站集合，PVM 今后的目标将致力于异构的应用开发模型以及相关的程序设计框架。

2.3 并行应用程序的常见错误

利用 PVM 软件所编写的应用程序中，最易出现的错误是通讯错误。例如多个进程向一个进程发送消息，如果接收消息的进程明确规定了消息到达的顺序，由于系统状态的不同使消息发送进程发送顺序无法确定，则必然导致接收消息的进程进入永久保持状态。具体编程中应使用一个循环语句来接收消息，无论是哪一个进程先发送到，接收进程均可接收。如果要区别不同进程发来的消息，则可在发送与接收时用消息类型（msgtype）来作判断。

在复杂编程中，还有可能出现由于资源竞争引发的错误甚至死锁。例如，本应只由单一进程操作的资源，由于先占有资源的进程加锁不够迅速，致使第二个进程也同时开始对同一资源的操作。因此人们迫切地希望有一个可行的监视调试环境来发现和排除上述错误和死锁，帮助开发出更多更好的 PVM 应用程序。PVM 现提供了一个监视器 XPVM 软件，它以图示的方式显示了并行程序的动态执行过程，可以帮助用户理解并行程序的行为，但下文的分析将指出，XPVM 并不能向用户提供完全的并行调试支持。

三、流行软件分析

并行程序的复杂性使人们在编程时迫切地需要一些辅助软件的帮助。客观地说，现今对并行调试技术的研究报告是比较多的，但真正实用的产品并不多。

3.1 Xmdb 软件

Xmdb 是 X Windows 下的消息调试器，是在单处理机上实现的 PVM 并行应用程序的仿真调试器，它设计了一个 mdblib 库取代了 PVM 函数库，使用户对 PVM 应用程序的调试完全在单处理机上进行，提供了竞争检测、重演等功能。

Xmdb 的主要优点有两个，其一是设计了一个替代 PVM 函数的库，使得 Xmdb 软件可以不受正在发展的 PVM 环境的影响，对于新版本 PVM 提供的新的函数只需向 Xmdb 库添加相应的库函数即可，其二是 Xmdb 软件提出了“受控进程”的概念，即由用户控制的进程，Xmdb 为该进程设立了“接收消息队列”，其它进程发送给它的消息放入该队列中，用户可决定何时发送及发送哪几个消息。“受控进程”的概念使用户从并行程序众多的进程中解脱出来，把注意力集中到该受控进程上。

Xmdb 仅能在单处理机上进行调试毕竟有其局限性，人们更希望在进行并行调试时能得到真正的并行支持，而且 Xmdb 的界面及仿真功能也并不尽

如人意。

3.2 CoCheck 和 TotalView

CoCheck 软件是一公用软件，它提供给用户设置检查点的功能，可以在 PVM 环境之上运行，但其主要目的是用于进程调度时的迁移和并行程序的容错性，即当一个程序崩溃时还可从上一个检查点继续执行。PVM 环境现在大多用于大型的科学计算，显然，CoCheck 的这种特征对调试这种用于科学计算的并行软件很有帮助。

TotalView 是一个商用软件，作为源代码级的多进程调试器，TotalView 可调试 PVM 应用程序，并能自动控制每一个 PVM 环境 spawn 出来的进程。

3.3 XPVM 软件

XPVM 软件是由 James Arthur Kohl 为 PVM 应用程序开发的监视器，它具有 PVM Console 命令的图形界面，并能监视 PVM 应用程序的执行，是当今与 PVM 一起用得最多的软件，许多 PVM 应用程序的开发人员都把它作为软件开发的辅助工具，遗憾的是 XPVM 并没有为用户提供真正并行调试的功能。

3.3.1 XPVM 简介　XPVM 软件由 C 语言写成，界面设计由 TCL/TK 工具库完成。欲用 XPVM 进行分析的应用软件需要使用 PVM3.3或以上的版本进行编译，只有这些版本才具有在运行时捕获跟踪信息的能力，而这点正是 XPVM 的基础。

XPVM 有两种显示进程状态的方式，一种是实时方式，即在某一个 PVM 应用程序由 XPVM 的 spawn 开始运行后，XPVM 一边对跟踪文件进行写入，一边就实时显示，另一种方式是事后显示(post-mortem)，此时用户可根据所记录的跟踪文件进行重演，并可单步显示应用程序的运行状态。XPVM 的跟踪文件以 SDDF 的格式产生，SDDF 即"自定义的数据格式"，这样，每一种事件类型在跟踪文件中都有对应的描述信息，以便于其它的分析工具也可以使用和解释跟踪文件中的跟踪事件数据。

3.3.2 XPVM 的图示　XPVM 提供了 Network，Space-Time，Utilization，Call-Trace 和 Task-Output 五种视图来帮助用户理解 PVM 应用软件的行为。

Network 显示了组成并行虚拟机的每一个主机的高层活动。主机用一图标表示，图标之上有该主机的结构和名字。PVM 应用软件运行时，图标上的不同颜色反应了运行于该主机上的任务的状态，如"活动色"表示该主机至少有一个任务正在做用户的工作。"系统色"表示该主机没有用户计算而是在做系统开销。

Space-Time 图显示了执行于所有主机之上的每一个任务的状态，每一任务用一平行于时间坐标的水平线显示，水平线的颜色代表了同一时间段上的该任务的状态。此图显示了任务间通讯的活动。若两条水平线之间有一红色线相联，则表示这两个任务间有消息传递。如果用鼠标键点此引线，则可在信息状态行显示出发送与接收该消息的进程号，以及消息的长度与其类型值。

Utilization 图是 Space-Time 图在每一个时间点的总结，显示在该时间点上进行计算、系统开销及等待消息的进程数目，同 Space-Time 图一样，不同的颜色代表了不同的进程状态。

Call-Trace 图提供了每一个任务的 PVM 函数的调用情况。该图中每一个任务占用一行，在该行上显示该任务的函数调用情况，包括调用参数和结果。

Task-Output 用以显示各进程的输出。如果用户想把这些输出记录在一个文件中，则可在此图示提供的输入中输入该文件名。

3.3.3 XPVM 分析　当选用 PVM Task Debug 由 XPVM 调用 PVM-Spawn 运行 Pvm 应用程序时，如 master.xpvm 为 master 产生了一个 dbx 调试窗，用户可以在此调试窗中给 master 程序设置断点，跟踪变量甚至单步执行。而且，Xpvm 还提供了"重演"功能，可在几个视图中重现上一次程序的执行情况。然而，尽管 XPVM 提供了上述并行程序运行监视和 dbx 调试，但它毕竟还不是真正的并行调试环境。

应用 XPVM 软件，用户可以为运行于主机之上的 master 开一个 dbx 调试窗，而对于 master 创建的 slaves，如果不更改用户程序，则无法用 dbx 调试。用户只能自己去猜测 slave 的运行情况，不能象对 master 一样，对 slave 设断点，跟踪及单步，也就是说，XPVM 不能满足用户对所有进程跟踪调试的需求。

其次，XPVM 即使是对 master 的跟踪调试也只有一次，在进行重演时，我们可以看到 XPVM 仅仅是读取 trace 文件并图示，而 dbx 并不能对重演中的进程跟踪调试，意即在 dbx 执行完 master 后，dbx 便失去了对 master 的控制。事实上，用户在重演时，也会需要用到 dbx。

我们在 space-time 图上可以看到，如果用户在 master 中设置了断点，则 master 的确是在断点处停

住了，而如果此时 slave 已产生，那么 slave 却不会停住，而是一直在运行。如果 master 的断点设在它的 slave 发消息以前，而 slave 接收消息用的是 pvm-trecv(由用户设置接口消息的保持时间，如时间到而没有消息到，则不接收)，那么，一旦 master 超过设置时间而不运行 slave，便要出错，这将使用户对 master/slave 的调试无法进行。

综上所述，XPVM 对 master 的跟踪与单步等操作实际上只能对单进程而言，它并不能真正实现对进程的并行调试。

四、PVM 上的并行调试器

并行程序之所以难于编写难于调试，是由并行程序的两大特点决定的。其一是并行程序所呈现的分布式计算的特征。它使我们无法对所发生的事件作一个完全的排序，全局状态也模糊不清，从而使串行调试技术中常用的单步和断点设置难于定义。虽然利用 Lamport 的理论，我们仍可定义一些协议，把分步式事件的偏序序列转换为一个一致(*Consistent*)的全序序列，但这种方法显然破坏了并行程序本身所固有的并行性，从而使用户无法在调试期间发现那些同步相关的错误。其二是并行程序本身所固有的不确定性。当给定相同的一组输入数据，而并行程序在不同的执行中给出不同的结果时，就显示出了并行程序的不确定性。这种不确定性有可能是程序员有意设计的，但一般来说，它都是并行程序的错误。进程通过消息相互通讯或是进程调度和消息等待中的变化都会引起并行程序中不确定性的产生。

并行程序的特点决定了普通的串行程序调试工具远不能够给予用户足够的帮助，需要为用户提供一个实用的并行调试器。

4.1 界面及可移植性

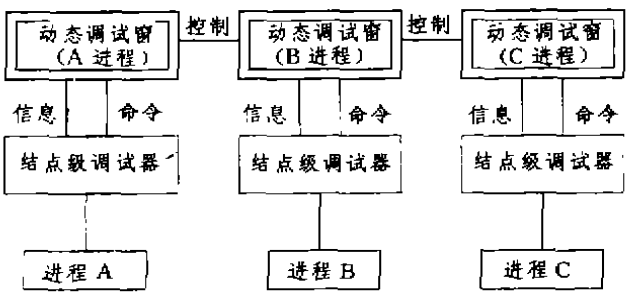

图1 进程控制图

并行调试器应提供对并行程序的单步跟踪、断点设置、变量及消息队列检查等功能，但如果该调试器直接提供这些功能，则必然导致其可移植性较差。为此，我们充分利用了各结点级的调试器，由它们负责单处理机上的进程调试，另为并行调试器提供一个高层界面，即动态调试窗。

动态调试窗提供对并行程序的全局控制，结点级调试器回送结点级的调试信息。调试时每个进程都受控于一个相应的动态调试窗，具体调试时用户可以仅选择其中的一个或几个进程进行观察，没选中的进程其活动不在屏幕上显示，但仍受相应动态调试窗的控制。

每个动态调试窗在其被用户操纵的瞬间成为主动控制窗，对该进程的操作由其翻译为相应的全局操作。图1显示了三个进程的控制图，其中进程 B 的动态调试窗正由用户操纵，可对进程 A 和进程 C 进行控制。

4.2 重演

并行程序往往需要调试由不确定性引发的错误，所采用的方法一般分为三类：一类是 Post-moterm 方式，又称重演，通过记录事件发生的顺序，并强行保证重复执行时的事件序列与记录的事件序列相同，从而达到再现上一次执行的目的；另一类是静态分析，采用语义分析等方式，在并行程序实际执行之前进行分析；第三类是 On-the-fly 方式，在程序执行时收集信息，在程序执行时检测不确定性。

具体实现中多采用第一类方法，为此有必要在程序执行时就记录下事件顺序作为重演的依据。具体到 PVM 上的调试器，由于 PVM 在其库中有任务跟踪系统，为每一次 PVM 函数调用记录其参数和结果。每一次函数调用时 PVM 库便产生一个跟踪事件的消息，这些消息以 TEV-SPNTASK，TEV-NEW-TASK 等标记作为不同事件的分类，因此我们可以在跟踪文件中方便地记录下程序执行时 PVM 函数调用事件的顺序。

重演的目的就是调试工具根据 trace 文件中记录的事件顺序再现程序的原来执行过程。为此，调试工具必须完全控制 PVM 应用程序中所有的进程通讯和 I/O。与前所介绍的 XPVM 不一样的是，此时程序的再次执行在调试工具控制之下，用户可以读取变量值，设断点，甚至可以改变程序的运行顺序。调试工具将在重演时提供“原执行”和“再执行”两种

状态，当用户对程序进行修改后，调试工具将比较现在程序的运行和记录顺序，如果确定是用户更改，则从"原执行"状态改为"再执行"状态，从当前点开始程序的运行，而不再以跟踪文件控制程序的执行。

4.3 设置检查点

PVM 环境在科学计算中应用广泛。科学计算的特点之一是每一次计算往往耗时很长，调试时由于增加了一些额外开销，使得运行时间成倍增长，调试工作难以进行，为解决这个问题，我们在调试时允许用户设置若干"检查点"(checkpoint)，在检查点处，调试程序将同时停住所有进程，记录必要的信息，以便所有进程能在任何时候都可以从检查点处重新开始执行。

程序在设置好检查点后开始运行，在此期间调试工具在 trace 文件中记录下必要的跟踪信息，程序运行结束后，用户可以对所设置的检查点进行检查，如果发现某个检查点的信息有问题，希望返回执行此检查点之前的代码，则可以从上一个检查点开始重演刚才的执行，而不必从头开始。这种做法对于耗时很长的科学计算很重要，它使用户在调试时可以象编辑文本文件时一样把某一可执行代码再重新执行一遍，而不用再等运行若干小时甚至几天之后才又执行到希望观察的代码。

可以看出，检查点的设置是以空间换取时间的策略，设置过多的检查点会占用太多的空间，而且在停下进程做记录时也会增加一些程序的执行时间，因此，实现时应限制用户对检查点的设置个数。

4.4 性能调试

对于串行调试工具来说，一般仅支持程序的逻辑调试，也就是针对程序正确与否的调试。但对并行程序来说，仅仅是逻辑上的正确还不够，因为人们采用并行的方法进行计算，其目的是为了提高程序的性能，因此，有时用户还希望在逻辑调试的基础上能提供性能调试的功能，帮助用户分析程序执行的瓶颈，比如如果能把并行程序在同步、负载平衡、通讯以及不完全并行方面的开销等情况提供给用户，就能有效地帮助用户理解和提高并行程序的性能。

4.5 可视化

调试信息能否以直观的图形方式显示给用户是调试环境能否成功的关键之一。并行调试中产生的信息量大而且高层信息与低层信息混杂，这使得并行程序难于理解，难于调试。

众所周知，一幅图所包含的信息远多于一页纸的描述，而且更加直观和易于理解，因此并行调试工具很重视可视化的工作。例如本文所描述的 XPVM 软件，其 Views 菜单所提供的几个图视就能为用户理解其并行程序的行为带来许多帮助；曙光1000之上的 ParaView 也为用户提供了良好的视图。公用软件包 ParaGragh 则可依据并行程序执行所产生的跟踪文件信息自动产生视图(当然，此跟踪文件的格式是由 PICL(Portable instrumentable communication library)产生的专门的格式，但其它格式的文件也可以转化为此格式)，在并行程序的监视，调试，可视化分析等软件中广为应用。

可视化的信息常常包括一些诸如结点状态，通讯情况，负载平衡等，有的软件所提供的视图很复杂，而且视图的形式各具特点，不利于用户的理解，因此，可视化的视图应该做到标准化，就如同今天的用户界面一样，应具有相同的视觉效果和理解意义。

参考文献

[1]Al Geist, Adam Beguelin, 《PVM3 User's Guide and Reference Manual》, UT/ORNL, 1994

[2]O. McBryan, An overview of message passing environments, Parallel Computing, 20(1994)

[3]V. S. Sunderam, The PVM Concurrent Computing System: Evolution, Experiences and Trends Same to [2]

[4]Proc. ACM/ORN Workshop on Parallel and Distributed Debugging, 1991

[5]Proc. ACM/ORN Workshop on Parallel and Distributed Debugging, 1993