若干发展的消息传递界面:PVMPI,IMPI 与 FT-MPI

魏兵海

(华中科技大学计算机科学与技术学院　武汉430074)　(国家外存储专业实验室　武汉430074)

摘　要　本文介绍了诸如 PVMPI、IMPI 的改进型消息传递界面(在异构环境中不同的 MPI 实现能够彼此互操作),也介绍了具有容错能力的 FI-MPI。分析了 MPI 的特征和体系结构及其性能。

关键词　消息传递界面,异构环境,容错能力

Some Improved Message Passing Interfaces: PVMPI, IMPI and FT-MPI

WEI Bing-Hai

(School of Computer Sci. & Tech., Huazhong Univ. of Sci. & Tech., Wuhan 430074)

(National Storage System Laboratory, Wuhan 430074)

Abstract　Some improved message passing interfaces such as PVMPI, IMPI, in which different MPI implementations in a heterogeneous environment can interoperate with each other, and FT-MPI, which have fault tolerant capability, are introduced, then the feathers and architectures of these MPI are analyzed, the performance are also discussed. All these might provide researchers with reference.

Keywords　Message passing interface, Heterogeneous environment, Interoperability, Fault tolerant capability

1　引言

在分布式并行计算系统中,处理器或节点主机之间通过消息传递界面实现消息通信的功能,消息传递界面的特性决定了系统中特定功能实现的可能性。改进消息传递界面的架构和机制对实现不同的系统功能有着重要意义。其中,消息传递界面的可移植性、互协性、异构性、容错功能等是相关研究中令人感兴趣的重点内容。

PVM[1,2]并行虚拟机是一种较常采用的消息传递实现库。由于早期的 PVM 开发面向工作站网络系统,考虑到工作站网络的节点机在主机硬件体系结构、子网络结构和操作系统等方面的异构可能性,PVM 因此提供了灵活的进程动态控制机制,其主要动态特征包括:在任务运行过程中增加或删除节点机;主机检测、消息通告与进程动态迁移;相关任务调度及资源管理、故障恢复与容错。PVM 在机器层、网络层和应用层三层支持异构动态连接的并行分布式计算。由于 PVM 较 MPI 复杂的消息数据结构,节点机间通信借助相对低速的网络而非 MPI 型的高效本机附带消息传递机制,因此,PVM 中消息传递速度小于 MPI 中消息传递速度。

MPI[3,4]消息传递界面则是一种主流的消息传递工业标准。早期的 MPI 设计面向 MPPs 厂商要求的高效优化应用,对于速度等性能的考虑高于对进程动态控制等功能的考虑,消息传递使用优化的本机固有消息传递机制,因此,MPI 虽然具有高效通信能力,然而 MPI 应用程序只具有单机可移植性,异构的 MPI 实例之间无法实现互协操作,即使后来的 MPI-2增加了进程动态管理功能,MPI 也不具备类似 PVM 那样的异构机群上不同节点机间的互协操作能力。

基于 PVM 和 MPI 的不同特点,一些研究机构推出了若干改进的消息传递库和工业标准,1997年 Tennessee 大学开发了一种集成 PVM 和 MPI 功能的 PVMPI[5,6];2000年国际标准组织公开发布了能够实现异构 MPI 间互协操作的 IMPI 标准[7,8];2001年 Tennessee 大学实现了一种具有较强容错功能的 FT-MPI[9,10]等。本文拟分别对上述典型的改进消息传递界面加以介绍和分析,探讨其各自不同的架构和功能特点。

2　可移植消息传递界面 PVMPI[5,6]

PVMPI(Portable Virtual Machine MPI)意即(异构节点机环境下)可移植虚拟机消息传递界面。PVMPI 大致可分为三层:PVM 层、附加的中间层、MPI 层,其中,PVM 层是作为不同 MPI 实现之间的一个通信和控制管理层,即 PVM 提供对异构 MPI 应用之间互协操作的支持,中间软件层用于连接 PVM 和 MPI,实现 PVM 对异构 MPI 程序的调度管理,MPI 层用于同构节点机之间进行本机优化的高速通信。通过集中 PVM 和 MPI 两者的功能优点,PVMPI 在尽可能的位置如 MPP 机内、同构的 MPP 实现之内使用高速的 MPI 型本机固有消息传递机制,而在不同的 MPI 实现之间则采用 PVM 型网络协议通信机制。在 PVM 环境中,PVM 提供对进程实现灵活控制(如进程的派生、节点机的删增等)的功能函数,而在 PVMPI 环境下,PVM 功能函数可灵活控制和调度管理不同的 MPI 实现(MPI 实现的启动、装有不同 MPI 实现的节点机的删增等)。并且,PVMPI 对进程控制的灵活性超过 MPI-2。

3　互协操作消息传递界面 IMPI[7,8]

IMPI(Interoperable MPI)意即(在异构环境下不同 MPI 实现之间)能够互相高效协同操作的消息传递界面。2000年国际标准组织正式公开发布了相关标准。IMPI 允许一个 MPI 应用可以包含多个异构的 MPI 实现,多个不同的 MPI 实现可以在具有不同体系结构和操作系统的异构机群系统上协同

*)华中科技大学博士后基金项目(AA183107)。**魏兵海**　博士后,主要研究方向为并行与分布式系统、高性能机群与网格计算等。

运行。IMPI 保持原 MPI 的界面，对用户级界面例程不作任何改变，IMPI 只在用户层的底层对 MPI 功能进行扩展，向下兼容目前所有的 MPI 语义、MPI 实现和 MPI 应用程序，兼顾不同实现内各种组通信算法的独立可扩展性。

在 IMPI 环境中，一个单独同构 MPI 实现内的通信可以在各节点机上使用高效的本地通信机制，其通信性能（消息传递速度等）基本不低于甚至高于非 IMPI 环境中的 MPI 实现（如 MPI-2）内的通信性能，而相比之下，MPICH、LAM 等可移植 MPI 实现则在各同构（即相同的 MPI 环境）节点机上使用统一的 MPI 库进行通信，不支持厂家随机优化的各种本地高速通信机制，因而牺牲了通信速度等性能。

在 IMPI 环境中，不同的异构 MPI 实现间通信时，消息传递速度因使用公共网络通信协议（如 TCP/UDP）而低于一个 MPI 实现内的通信速度，然而，IMPI 并不在异构 MPI 实现间使用密集的连接拓扑，仅只在异构 MPI 实现间使用尽可能少的小部分通信信道连接，将消息传递速度的损失降到最小。

IMPI 支持不同 MPI 实现间的组通信操作，例如：在不同 MPI 实现之间由一个进程给所有其它进程进行广播（MPI_Bcast），在不同 MPI 实现之间进行所有进程的同步（MPI_Barrier）。PVMPI 则不支持不同 MPI 实现间的组通信操作。

在 IMPI 环境中，运行于不同系统上的同一个任务的异构 MPI 实现通过一个名为 IMPI 服务者的系统集合进程进行启动，独立于各具体实现的 IMPI 服务者进程使用无结构的消息以便各系统移植和分享。IMPI 服务者进程首先给出一字符串，字符串包括服务者进程主机的英特网地址、服务者进程的 TCP/IP 监听端口等以便于各系统同服务者连接，通过服务者中转各系统对话确定各实现进行公共通信的符号如发送和接收一个单一信息的最大字节数等，此后，各 MPI 实现之间可进行直接通信。而在 PVMPI 环境下，各 MPI 实现之间始终通过 PVM 中转进行通信。

IMPI 启动时使用64位关键字进行身份验证，将来可扩展成为更复杂的身份验证系统，因此 IMPI 比其他的消息传递库更便于在将来的广域 GRID 系统中使用。

4 容错型消息传递界面 FT-MPI[9,10]

FT-MPI（Fault Tolerant MPI）意即具有容错能力的消息传递界面。在普通 MPI 语义中，一个 MPI 进程或消息传递的故障将引发相关的所有通信体失效，并且 MPI 标准中不提供任何恢复"腐坏"通信体或决定是否释放"腐坏"通信体的机制，最后唯一的结果只能是 MPI_COMM_WORLD 失效而导致整个 MPI 应用程序瘫痪退出。

随着实际应用中节点机的增加，如下一代 Peta-flop 系统、网格系统，单节点机的可靠性降低，对 MPI 支持容错功能的需求更为迫切。FT-MPI 则提供一系列的机制如多种通信体状态鉴别、失效通信体重构等进行失效恢复和容错。

最初的 FT-MPI 利用检查点技术、回卷技术和复制技术实现容错的消息传递库，例如 Co-Check MPI[11]使用 Condor 库实现了 MPI 之上的同步检查点机制，当出现进程故障或要求任务迁移时，整个 MPI 应用程序回卷到上一个同步检查点后重新启动，因此，这种同步检查的时间开销较大。StarFish MPI[12]则基于分布式系统自身的函数实现检查点机制，StarFish MPI 使用严格的原子型组通信协议管理消息传递和进程状态变化，避免 Co-Check MPI 中的消息同步，节省了同步检查上的时间耗费。此外还可见一种 IFT-MPI（Implicit Fault Tolerance MPI）容错实现，IFT-MPI 使用包含数个备用进程的通信体模型实现 SPMD 中的 MPI 容错，通过在观察员进程中存储所有进程的消息拷贝，实现观察员进程对故障进程中丢失消息的再生，失效的进程则由备用进程替代。

Tennessee 大学最近在 HANESS 系统上实现的 FT-MPI 支持对静态 MPI 的动态容错应用。其 FT-MPI 的容错机制如下所述。首先，FT-MPI 对通信体状态声明和进程状态声明进行了扩展，通信体状态定义由{VALID，INVALID}两种扩展到{FT_OK，FT_DETECTED，FT_RE-C OVER，FT_RECOVERED，FT_FAILED}五种，进程状态定义由{OK，FAILED}两种扩展到{OK，UNKNOWN，UNREACHABLE，JOINING，FAILED}五种。当 MPI 进程或消息传递中出现故障时，在普通 MPI 环境中的进程将从 OK 态直接跳转到 FAILED 态，通信体则从 VALID 态直接跳转到 INVALID 态，而在 FT-MPI 环境中则将通过一定的规则和策略决定进程和通信体将跳转的状态，在不同的状态下根据确定的处理算法进行容错处理，例如使用 MPI 通信体重构函数 MPI_Comm_{create，split or dup}重新构建 FT_RECOVER 态通信体以恢复错误。通信体的处理操作可有四种模式：SHRINK：通信体已经被收缩，不再含有"洞隙"（即故障进程），强制 MPI 应用调用 MPI_COMM_RANK 以改变各进程的标识号，通信体数据结构连续无缝。BLANK：与 SHRINK 基本相同，但通信体目前仍包含此后将填充的"洞隙"。与"洞隙"的通信将返回一个无效的进程标识号错误，调用 MPI_COMM_ SIZE 将返回通信体内进程的总数，而非其内部的合法进程总数。REBUILD：最为复杂，它强制创建新的进程以填充任何的"洞隙"而不改变通信体的大小，新创建的进程要么使用空缺的进程标识号，要么在对通信体进行收缩后加到进程队列的末尾。ABORT：当侦测到错误时，强制 MPI 应用退出。控制通信体内进程间通信的消息控制模式分为两种：NOP（NO OPERATION）：不对消息错误进行任何操作，即不在用户级别进行错误消息的处理，对所有通信错误简单地返回一个错误码。NOP 模式用于 MPI 应用从任意运行点返回到一个状态，以便于根据上述五种确定的状态处理方式加以恢复操作。CONT（CONTIGUOUS）：所有还没有到达故障（以及与故障进程相关）节点的消息传递能够正常继续进行。在通信体状态被重设之前，与失效节点的通信将返回一个错误码。在 FT-MPI 环境下，含有故障的组操作的结果可能不受故障的影响，也可能发生组操作的异常中断，例如 BROADCAST 组操作，即使在某一接受节点位置存在故障，其它的接受节点也可以不受影响地接收到正确的消息。

HANESS 系统上的 FT-MPI 目前实现了全部的 MPI-1 功能，局部实现了 MPI-2，支持 C 和 FORTRAN，但还不支持 C++。HANESS 的 FT-MPI 在底层借用了 PVM3.4，实现了具有动态特征的容错型消息传递。

结论 并行虚拟机 PVM 提供动态的资源管理功能、动态的进程控制功能、异构环境下的容错能力，适合于长周期大规模并行计算应用中加强过程控制和管理。消息传递界面 MPI 则具有丰富的消息通信函数、虚拟进程拓扑任务分派功能，益于实现高效的消息通信。PVMPI 集成两者的功能优势，实现了异构环境中不同 MPI 实现间的通信互操作。

IMPI 不仅支持异构环境中不同 MPI 实现间的通信互操作，而且通信尽可能采用本机优化的通信机制，支持不同

（下转第169页）

```
where
   nesproc=map(f nextpid)(map(untag nextid)future)
   future=join(man(Spawner(nextid t1))
          (join(map(merge ms)newproc)))
system=join(map(Spawner1)
        (join(map(merge boot)system)))
```

3 通道系统要求

3.1 读通道

ReadChan name

功能：打开通道 name 以便输入。若成功，则返回通道的内容，若通道不存在，则返回 Failure(SearchError String)，所有其它类型的错误信息为 Failure(ReadError String)。

一旦 ReadChan 已打开了一个特定的通道，同样的通道就不再为其他请求打开，所以在要求某个通道时，必须检测该通道是否存在，以及该通道是否已被打开，这个工作由 TestChan 来完成：

```
TestChan1::TagReqList->IO TagRespList
TestChan1[(n,ReadChan name)]
=\(fs,(ics,ocs))->
   let ack=(n,if snd(ics name)
               then Failure(OtherError"Channel already open\n")
               else(fst(ics name))(fs,ocs)
   in([ack],(fs,(ics,ocs)))
```

输入通道的类型为 Ics=String->(Agent,Open)，那么 ics name 的类型为(Agent,Open)，snd(ics name)得到的类型为 Open，它判断输入通道是否打开，fst(ics name)的类型为 Agent，它判断该通道是否已为某个进程打开，(fst(ics name))(fs,ocs)的类型为 Response。

测试完通道以后，若输入通道没有被打开，则打开此通道，修改系统的状态，若已经打开，则不修改系统状态，这个工作由 newics 完成：

```
newics::String->TagRespList->IO TagRespList
newics name resp
=\(fs,(ics,ocs))->(resp,(fs,(update ics name(fcs(ics name),
                    update open n true),ocs))
```

那么，ReadChan name 的语义为：

```
os[]=unitSt[]
os(n,ReadChan name);es
=TestChan1[(n,ReadChan name)]          'bindST' \a->
  newics name a                        'bindST' \b->
  os es                                'bindST' \c->
unitST(a++c)
```

3.2 写通道

AppendChan name String

功能：把 String 写入通道 name 中，若此通道系统不存在，返回 Failue(SearchError String)。与 ReadChan 类似，首先要检查通道是否存在，但是不要检测通道是否已被打开，这个功能由 TestChan2来完成：

```
TestChan2::TagRespList->IO TagRespList
TestChan2[(n,AppendChan name String)]
=\(fs,(ics,ocs))->
  let ack=
        (n,case (ocs name)of
              Failue msg -> Failue (SearchError "Nonexist
                    channel")
              Str ochan->Success)
        In([ack],(fs,(ics,ocs))
```

然后修改通道系统：

```
newocs::String->TagRespList->IO TagRespList
newocs name resp
=\(fs,(ics,ocs))->
  (resp,(fs,(ics,case(ocs name)of
              Failure msg->ocs
              Str ochan -> update ocs name(str(ochan++
                  contents))))
```

那么，AppendChan 的语义为：

```
os[]=unitST[]
os(n,AppendChan name contents);es
=TestChan2[(n,AppendChan name contents)]  'bindST' \a->
  newocs name a                           'bindST' \b->
  os es                                   'bindST' \c->
unitST(b++c)
```

结论 非确定性 Monad 在操作系统、交互式系统以及并行算法中有广泛的应用，但是纯函数式语言缺乏处理非确定性的能力。本文通过使用非确定性 Monad 来描述操作系统的进程网，很好地解决纯函数式语言通道系统的问题。

参考文献

1 袁华强，肖倩，孙永强. 基于非确定 Monad 的纯函数式树搜索算法. 软件学报，1997(8. 增刊)：189～193
2 袁华强，孙永强. 纯函数式 I/O 的操作语义. 计算机学报，1998，21(11)：1009～1014
3 石跃祥，袁华强，孙永强，陈静. 函数式语言中的赋值语句. 软件学报，1999，10(3)
4 石跃祥，袁华强. 纯函数式语言中的状态转换器与调用. 湘潭大学学报，2000，22(3)：25～29
5 袁华强. 函数式 I/O 系统的实现与研究：一种 Monad 方法：[博士论文]. 上海交通大学，1996
6 Moggi E. Computational Lamda-calculus and monads. In：Proc Symposium on Logic in Computer Science，CA，1989. 14～24
7 Moggi E. Notions of computation and monads. Information and Computation，1991，93：55～97
8 Wadler P. The essence of functional programming. In：Proc ACM Symposium on Principles of Programming Languages，NM，1992. 1～14

(上接第162页)

MPI 实现间的组通信操作，具有较强的可扩展性。消息传递性能优于 PVMPI，IMPI 的体系结构和功能正朝着融入 GRID 系统的方向发展。

FT-MPI 通过检查点和回卷技术以及通信体重构技术实现了静态 MPI 的动态容错应用，其高可靠性以及其使用本机优化组通信机制的高效通信功能为下一代 Peta-flop 系统、大规模广域网格系统提供了技术基础。

在 MPPS 系统、NOW 系统和 GRID 系统分布式计算应用中，可根据系统和应用的不同特点和要求选用不同的消息传递界面。

参考文献

1 Geist A，Beguelin C，et al，PVM：Parallel Virtual Machine. Boston：MIT Press，1994
2 PVM Documentation. Available at：http://www.epm.ornl.gov/pvm
3 Gropp W，Lusk E，et al. Using MPI-2：Advanced Features of the Message Passing Interface，firsted.，MIT Press，Cambrdge，MA，2000
4 Message Passing Interface Forum. MPI：A message -passing interface standard. The International Journal of Supercomputer Applications and High Performance Computing，1994，8(3/4)
5 Faag G，Dongarra J，Geist A. PVMPI provides interoperability between MPI implementations. In：Proc. 8th SIAM Conf. on Parallel Processing，SIAM 1997
6 Graham E F，Dongarra J. PVMPI：An Integration of the PVM and MPI Systems. Calculactures Paralleles，1996，8(2)：151～166
7 William L，George J，et al. IMPI：Making MPI Interoperable. Journal of Research of the National Institute of Standards and Technology，2000，105(3)：343～348
8 IMPI Steering Committees. IMPI：Interoperable Message -Passing Interface. Journal of Research of the National Institute of Standards and Technology，2000，105(3)：349～428
9 Graham E F，Antonin B，et al. HARNESS and fault tolerant MPI. Parallel Computing，2001，6：1～17
10 Beck D，Fagg G，et al. HARNESS：a next generation distributed virtual machine. Journal of Future Generation Computer Systems，Elsevier Science B. V.，1999，15
11 Stellner G. CoCheck：Checkpointing and Process Migration for MPI. In：Proceedings of the International Parallel Processing Symposium，Honolulu，April 1996. 526～531
12 Agbaria A，Roy F. Starfish：Fault-Tolerant Dynamic MPI Programs on Clusters of Work -stations. In：the 8th IEEE Intl. Symposium on High Performance Distributed Computing，1999