

面向嵌入式实时操作系统的 MPI 实现

郭德源, 何 虎, 杨 旭

(清华大学 微电子学研究所, 北京 100084)

摘 要: 为嵌入式 VxWorks 实时操作系统提出了一种并行编程解决方案, 首次将 MPI 并行编程平台引入到 VxWorks 系统当中, 以 Linux 系统中的开源 MPI 实现 mpich2 为基础, 对 mpich2 中的 MPI 函数库部分和 smpd 进程管理程序进行跨系统移植, 通过解决 Linux 与 VxWorks 系统之间的诸多差异, 如系统函数支持、进程创建方式等, 为 VxWorks 系统提供了一款基于 socket 通信接口的 MPI 实现——eMPI, 搭建了嵌入式 MPI 并行应用开发平台。

关键词: MPI; 嵌入式; VxWorks; 并行计算; RTP

中图分类号: TP391

文献标识码: A

文章编号: 1000-7180(2011)03-0035-04

An MPI Implementation for Embedded Real Time Operation System

GUO De-yuan, HE Hu, YANG Xu

(Institute of Microelectronics, Tsinghua University, Beijing 100084, China)

Abstract: This paper proposed a parallel programming solution for embedded VxWorks real time operation system, and introduced the MPI parallel programming platform into the VxWorks system for the first time. Based on the open source MPI implementation mpich2 under Linux system, we ported the MPI library and smpd process manager into the VxWorks system, through solving the various differences between Linux system and VxWorks RTOS, such as system functions supporting, process creation mechanism, etc., provided a new MPI implementation based on socket interface—eMPI for the VxWorks system, and built an MPI parallel application development platform.

Key words: MPI; embedded; VxWorks; parallel computing; RTP

1 引言

在嵌入式系统领域, 多核处理器和集群系统已成为主流的发展趋势. 单个处理器的主频不再是性能的关键, 制约性能提升的瓶颈在于, 如何尽可能地利用硬件资源的并行性, 开发出高效率的并行计算程序.

目前, 并行编程模型的构建主要分为三种方式: 一是为既定系统创造一种全新的并行编程语言和并行计算模型; 二是将现有的串行编程语言扩展成为并行编程语言, 如 OpenMP; 三是在传统的串行编程语言中引入并行库的支持, 如 MPI^[1].

MPI 并行编程模型是针对分布式存储结构而设计的, 它能够较好地适应多核、异构多核乃至集群

嵌入式系统的特性^[2].

本文为嵌入式 VxWorks 实时操作系统提出了一种并行编程解决方案.

2 移植难点

2.1 VxWorks 与 Linux 系统的差异

将 Linux 系统环境下的 mpich2 移植到嵌入式 VxWorks 系统中, 属于跨系统移植, 两个操作系统之间的主要差异包括如下各个方面, 在移植过程中需要解决这些问题:

(1) 进程特性. VxWorks 系统的 RTP 不支持 fork() 函数, 因此, 对于 Linux 系统中常用的 fork() + exec() 的进程创建方式, 在 VxWorks 中需要全面修改.

收稿日期: 2010-05-11; 修回日期: 2010-07-28

基金项目: 核高基重大专项(2009ZX01034-001-001-001)

(2) 开发环境. 在 Linux 系统中开发一般使用 GCC、Make 等工具进行本地编译, 而对于 Vx-Works6.x 来说, 它的 Workbench 开发环境运行于 Windows 操作系统, 是一个交叉编译的过程, 与 Linux 存在较大差异.

(3) 系统函数支持. 除了与进程相关的一系列函数, 还有一些 Linux 系统中的标准函数在 Vx-Works 中未能支持, 如 poll()、fcntl()、gethostname()、md5() 等, 另外部分系统头文件也存在差异. 对于这类差异, 其解决办法就是使用功能相似的函数来代替, 或者自己实现该函数的功能.

(4) 进程优先级. 在 Linux 操作系统中, 进程的优先级是由系统自动调度的, 而在 VxWorks 中, 采用带优先级的时间片轮转调度策略, RTP 的优先级要由用户显式指定. 必须为 RTP 指定合适的优先级, 才能确保系统对 RTP 进程的正确调度.

(5) 标准 I/O. 在嵌入式系统中, 往往不具备标准的输入输出设备, 仅使用串口进行调试. 如果使用标准输入函数 scanf() 则可能会造成进程阻塞, 无法正常接收用户的输入. 对于不同的系统环境, 要做相应的处理.

2.2 模块化的 mpich2

Mpich2 是一款功能强大的 MPI 实现, 它的源码模块化较好, 因而具有较好的可移植性^[3]. 其应用程序接口层支持对 C 和 Fortran 语言的绑定; 底层的设备层则支持 Socket、共享内存、SCTP、Nemesis、Gasnet 等多种通信协议; 对于 MPI 进程的管理, 它支持 Python 代码和 smpd 守护程序两种方式^[4]. 为了将 mpich2 移植到 VxWorks 系统中, 必须为其选择合适的模块配置方式.

由于 VxWorks 系统提供了标准的 Socket 通信接口, 但不支持 Python, 所以本次移植中, mpich2 的通信层选择基于 Socket 接口, 进程管理程序选择使用 smpd 守护程序.

MPI 应用的开发/运行流程如图 1 所示. 从图中可以看出, 整个 mpich2 的移植工作包含了三个部分: MPI 并行函数库的移植、smpd 进程管理程序的移植、mpiexec 进程启动程序的移植.

3 移植步骤

3.1 MPI 函数库部分的移植

MPI 函数库是 smpd 进程管理程序的基础, 所以第一步首先需要移植 MPI 函数库. 此部分的移植主要是进行基于 Socket 接口的通信层的移植.

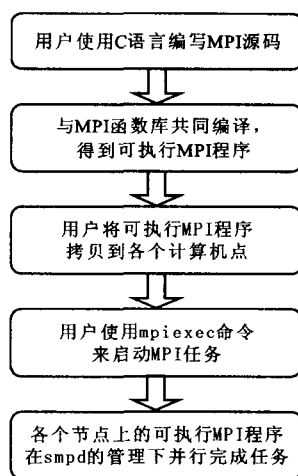


图 1 MPI 应用的开发/运行流程

3.1.1 工程创建及配置

由于 VxWorks 的交叉编译环境与 Linux 系统环境存在较大差异, 为了得到 VxWorks 中使用的 MPI 函数库, 可采用如下步骤:

(1) 编译配置头文件的生成. 为了减小移植工作量, 首先在 Linux 系统中运行 configure 脚本, 来对系统环境进行检测并生成一系列配置头文件. 然后根据 VxWorks 与 Linux 系统的差异, 对这些配置头文件进行手动修改, 主要是确定一些函数及头文件是否可用.

(2) 建立静态库工程. VxWorks 的交叉编译环境称为 Workbench, 它不像 Linux 系统使用 Makefile 来控制编译流程, 而是将所需的全部源码添加到工程中, 再进行编译^[5]. 此时需要确定保留哪些函数模块, 而不需要的模块代码则删除掉, 否则会出现函数名冲突. 比如对于通信层, 需要保留 socket 通信模块, 删除掉其它通信模块的源码.

(3) 修改系统差异的函数和头文件. 在 VxWorks 系统中, 许多头文件的名称与 Linux 系统不同, 系统函数的名称和行为也有差异. 对于此类情况, 需要在编译过程中逐个修改.

3.1.2 Socket 通信模块的移植

如图 2 所示, 整个 mpich2 的函数库分为 API 层、ADI 层和底层的 Channel 通信层. 这里的 API 和 ADI 层基本与操作系统无关, 而其 Channel 层将多种通信协议封装起来, 为上层提供了统一的接口. 由于本次移植环境为基于 Ethernet 互连的 VxWorks 系统, 故选择使用基于 Socket 接口的通信层.

在 VxWorks 中, 虽然提供了标准的 Socket 函数, 但是如 fcntl()、poll()、gethostname() 等函数并

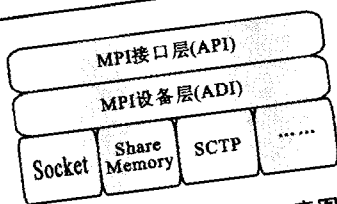


图2 MPI函数库结构示意图

未实现,使得 VxWorks 与 Linux 系统在阻塞/非阻塞控制上存在差异。对于这种情况,在 VxWorks 系统中,可通过使用功能近似的 `ioctl()`、`select()`、`hostGetByName()` 等函数^[6],对上述几个系统函数进行模拟,使得 socket 模块能够适应 VxWorks 系统环境,保证 MPI 进程能够正常通信。

3.1.3 其他差异的处理

当一个操作系统运行某些程序出现错误时,会返回错误代码,VxWorks 与 Linux 系统的错误代码也不完全相同。例如当一个 socket 连接关闭后再去读它,VxWorks 系统就会返回一个 `ESHUTDOWN` 的错误值,但 `mpich2` 中并未对这个错误值进行识别,就会导致进程异常。因此,在移植过程中,需要为 `mpich2` 添加对于这些存在差异的错误情况的处理。

3.2 Smpd 和 mpiexec 部分的移植

3.2.1 Smpd 进程管理程序

如图3所示,在集群环境下,当用户在根节点运行 `mpiexec` 来启动一个 MPI 并行应用时,各个计算机节点上的 `smpd` 进程管理程序会首先建立一个二叉树结构的连接,然后各自在节点上创建相应数目的 MPI 应用进程(`P0~P5`)。

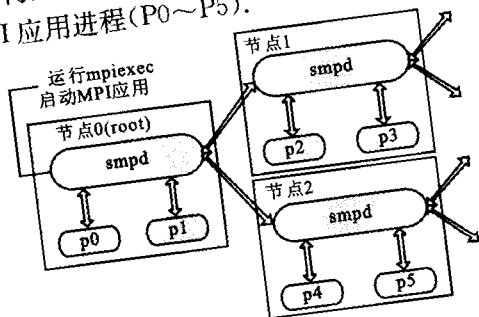


图3 集群环境下 smpd 工作原理图

在这里,不同节点上的 `smpd` 之间,以及 `smpd` 与 MPI 应用进程之间,都是通过 socket 连接进行通信的。所有 MPI 并行进程之间的通信,也需要通过 `smpd` 进行管理和中转^[7]。

3.2.2 MPI 进程创建方式的处理

为了对 MPI 应用进程进行管理,`smpd` 与 MPI 进程之间建立了 socket 连接。每个 MPI 进程的标准输入输出文件描述符(`stdin`、`stdout` 和 `stderr`)被重定向为 3 个与 `smpd` 通信的 socket 连接,当 MPI 进

程有任何输入输出行为时,都可被 `smpd` 监听到。同时,`smpd` 与 MPI 进程之间还有一个专用的 MPI 通信连接,用来传输 `smpd` 控制命令。在创建进程时,通过环境变量将此连接的文件描述符传递给 MPI 进程。

然而对于一个通用的 MPI 应用程序而言,显然不应为这些 socket 连接的建立而负责。因此,这些连接应该在 MPI 进程被创建的时候,由 `smpd` 来建立。在 Linux 系统中,通过使用 `socketpair()`、`fork()`、`exec()` 等一系列函数,可以令 `smpd` 实现上述功能。

如图4所示,Linux 系统中的 MPI 进程创建方式为:首先,`smpd` 使用 `socketpair()` 函数来创建本地的 socket 连接;其次,通过调用 `fork()` 函数,`smpd` 进程被复制为父子两个进程,同时 socket 文件描述符也被复制;接下来由子进程进行 I/O 重定向工作,即将 `stdin`、`stdout`、`stderr` 重定向为 socket 文件描述符,使得父子进程之间建立真正的 socket 连接;最后,使用 `exec()` 函数启动 MPI 应用程序,同时将已重定向的 socket 文件描述符继承给 MPI 进程。这样,在 `smpd` 与 MPI 进程之间,就建立了必要的 socket 连接,使 `smpd` 的进程管理功能得以实现。

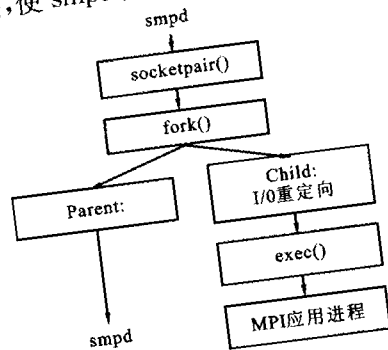


图4 Linux 系统中的 MPI 进程创建方式

然而,在 VxWorks 系统环境下,没有提供本地 socket 连接的 `socketpair()` 函数,也未提供于进程复制的 `fork()` 函数,仅提供了最基本的创建函数 `rtpSpawn()`。在这种情况下,`smpd` 与 MPI 进程之间的 socket 控制连接既无法由 `smpd` 也不能由 MPI 进程来建立。所幸的是,VxWorks 在创建 RTP 进程的时候,提供了对文件描述符继承的支持^[5],因此,通过创建临时进程的

以实现 `smpd` 对 MPI 进程的控制连接的建立。如图5所示,在这里引入一个临时进程,再由临时进程创建 MPI 应用进程。

时进程的工作主要有:与 smpd 建立 4 个 socket 控制连接;将标准输入输出文件描述符重定向到新建的 socket;在环境变量中设置 smpd 控制连接的文件描述符;关闭不使用的文件描述符.当完成上述工作后,临时进程调用 `rtpSpawn()` 函数来创建 MPI 应用进程,已重定向好的文件描述符就被继承给 MPI 进程,于是, smpd 进程管理程序与 MPI 应用进程之间的 socket 控制连接就建立起来了.通过这样的 MPI 进程创建方式,使得 smpd 在不使用 `fork()` 函数的情况下,能够实现预期功能,而不需要对 smpd 的进程管理方式进行大幅修改.

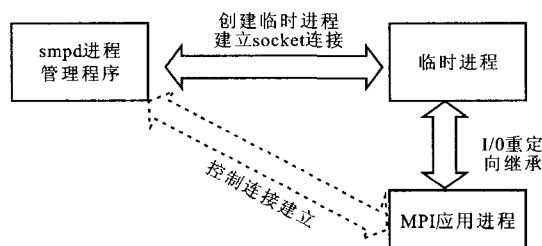


图 5 eMPI 中的 MPI 进程创建方式

3.2.3 Smpd 标准输入的重定向

在 MPI 进程创建的过程中,每个 MPI 进程的标准输入输出文件描述符都已重定向为与 smpd 相连的 socket 连接,它们的输入输出都是由 smpd 来统一管理的.在 Linux 系统中,标准输入输出文件描述符(`stdin`、`stdout`、`stderr`)也可看做是一个普通的 socket 连接,允许对其进行阻塞/非阻塞控制. Smpd 利用这一点,将自己的标准输入 `stdin` 设置为非阻塞模式,与各个 MPI 进程的 socket 控制连接同时在状态机中进行监听,从而达到统一管理的目的.

在 VxWorks 系统中,标准输入输出文件描述符不能够被看做是 socket 连接,也不能对其进行阻塞/非阻塞控制,为了便于处理,故仍采用临时进程的方式,在临时进程与 smpd 之间建立一个 socket 连接,然后将 smpd 的标准输入重定向为此 socket 连接.

3.2.4 创建 smpd 与 mpiexec 工程

作为两个独立运行的可执行程序, smpd 进程管理程序与 mpiexec 任务启动程序需要在 workbench 中单独建立各自的 RTP 工程,并在工程中指定 MPI 函数库的路径,使得这两个程序可以与 MPI 函数库共同编译,生成 VxWorks 系统中的可执行文件(`smpd.vxe` 和 `mpiexec.vxe`).

4 测试验证

4.1 实验环境

测试验证的系统环境如图 6 所示,硬件环境为

PowerPC 开发板(单核 CPU, 7447A),运行的操作系统为 VxWorks 6.5,通过 Ethernet 局域网与 PC 主机相连,开发板通过 FTP 协议从 PC 主机上加载 VxWorks 系统镜像和应用程序.同时,通过串口来对开发板上 VxWorks 系统的输入和输出进行监控.

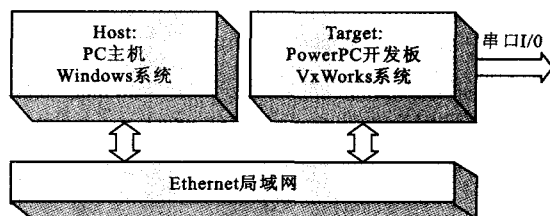


图 6 测试验证硬件环境

4.2 验证结果

在验证过程中,使用了 mpich2 源码包中自带的一系列 MPI 测试程序对 eMPI 进行了功能验证,还使用了俄亥俄州立大学 NBCL 实验室提供的 MPI 标准测试程序 OMB-3.1.1 对 eMPI 的性能进行了评估^[8].

经过验证,在功能方面,eMPI 能够在 VxWorks 系统环境下,正确实现 MPI 的功能.

在性能方面,在 VxWorks 系统中进行了两个 MPI 进程之间通信的延时测试(乒乓测试),并与普通 RTP 进程间的 socket 通信的延时进行了对比.如图 7 所示,横轴表示数据包的字节数,纵轴表示一个进程发送数据,另一个进程接收到数据的延时.

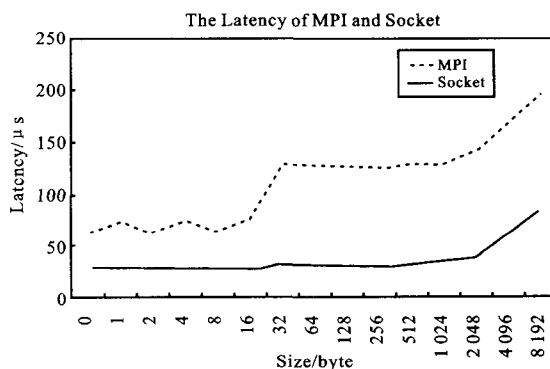


图 7 两进程间的延时测试结果

从图 7 可见,MPI 进程间的通信延时要比普通 socket 通信的延时高一些,考虑到 MPI 是在通信性能与开发效率之间的折衷,这些延时仍在可接受的范围内.

5 结束语

通过移植,首次为嵌入式 VxWorks 实时操作系统提供了一款基于 socket 连接的嵌入式 MPI 实

(下转第 42 页)

表 1 服务器平均负载量对比表 bit/s

服务器	DWSF	DUD
Server1	2 930	2 900
Server2	2 000	2 010
Server3	1 620	1 600

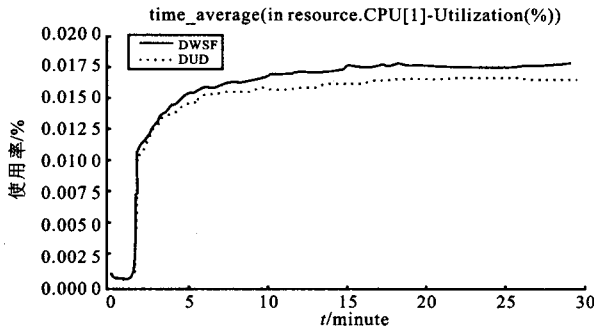


图 4 CPU 使用率对比

5 结束语

提出了基于离散型均匀分布的集群负载均衡机制。仿真结果表明 DUD 负载均衡机制在负载均衡效果上优于常用的静态算法。与传统的动态负载均衡算法相比,由于 DUD 负载均衡机制引入变速率请求的服务请求流来承载各负载节点的负载信息,因而使前端均衡器只需运行简单有效的负载分配算法。这样就降低了前端均衡器的算法复杂度,减小了前端均衡器

工作量,从而也降低了前端均衡器成为集群系统中瓶颈的可能性。实践证明 DUD 机制适用于有前端均衡器的集群负载均衡问题的负载分配。

参考文献:

- [1] Chen L C, Choi H A. Approximation algorithms for data distribution with load balance of Web servers[C]// Proceedings of 3rd IEEE International Conference on Cluster Computing. Mclean, VA: IEEE Computer Society Press, 2001: 274—281.
- [2] 程伟, 卢泽新, 王宏. 一种新的服务器集群系统负载均衡技术[J]. 计算机工程与科学, 2006, 28(2): 31—32, 34.
- [3] 嵇智辉, 倪宏, 刘磊. 动态消息队列负载均衡策略的研究与应用[J]. 计算机工程, 2009, 35(8): 31—33.
- [4] Mattner L, Roos B. Maximal probabilities of convolution powers of discrete uniform distributions[EB/OL]. [2009-03-12]. <http://arXiv.org/abs/math>.
- [5] 王和兴, 张建波, 才书训. 基于 OPNET 的网络仿真与性能测量[J]. 微电子学与计算机, 2005, 22(11): 25—27.
- [6] 陈敏. OPNET 网络仿真[M]. 北京: 清华大学出版社, 2004.

作者简介:

李海生 男, (1974—), 博士, 副教授. 研究方向为智能信息处理及可视化.

史鸿雁 男, (1972—), 硕士研究生. 研究方向为网络安全.

(上接第 38 页)

现——eMPI. 在单核/多核或集群的环境下, 通过使用 eMPI 平台, 开发者可以快速进行通用的 MPI 并行应用的开发, 在保证效率的同时, 可兼顾核间、系统间的并行度, 能够满足特定场合下高计算量、高并行度的应用需求。

参考文献:

- [1] 都志辉. 高性能计算之并行编程技术——MPI 并行程序设计[M]. 北京: 清华大学出版社, 2001.
- [2] 赵永华, 迟学斌. 基于 SMP 集群的 MPI+Open MP 混合编程模型及有效实现[J]. 微电子学与计算机, 2005, 22(10): 7—11.
- [3] VxWorks Application API Reference 6. 4[R]. Alameda, CA, USA: Wind River Systems, 2006.
- [4] David Ashton, William Gropp, Ewing Lusk, et al. MPICH2 Design Document[R]. Chicago: argonne national laboratory, 2003.
- [5] VxWorks Application Programmer's Guide 6. 4. Wind River Systems[R]. Alameda: Wind River Systems, 2006.

- [6] Rajeev Thakur, William Gropp. Open Issues in MPI Implementation[C]// Proceedings of the 12th Asia-Pacific Computer Systems Architecture Conference. Argonne, USA: MIT Press, 2007: 327—338.
- [7] David Ashton. SMPD PMI Wire Protocol Reference Manual Version 0. 1[R]. Chicago: Mathematics and Computer Science Division, Argonne National Laboratory, 2008.
- [8] OMB-3. 1. 1. NBCL, Ohio State University[EB/OL]. [2010-05-11]. <http://mvapich.cse.ohio-state.edu/benchmarks/>.

作者简介:

郭德源 男, (1984—), 硕士, 工程师. 研究方向为并行编译器、嵌入式 MPI 并行环境.

何虎 男, (1974—), 副教授. 研究方向为高性能数字信号处理器系统设计.

杨旭 男, (1981—), 博士后. 研究方向为低功耗解译器、导构多核并行应用开发.