

基于 MPI 的进程拓扑感知映射研究

李东洋, 王云岚

(西北工业大学 计算机学院, 陕西 西安 710072)

摘 要: 本文首先介绍了获取应用程序通信拓扑和物理拓扑的方法, 结合进程间通信量和通信频次定义了进程间亲和度, 并针对具有分层网络、NUMA 效应的硬件系统, 设计了相应的进程映射算法. 该算法根据进程间的亲和度对进程集合进行划分, 在划分过程中构造出和物理拓扑树同构的树状结构, 随后在两棵同构树的对应节点上做映射来解决应用程序进程到物理节点、处理核心的映射问题. 在一个多核集群系统上的实验表明, 使用本文提出的进程映射方法可以有效提高并行应用程序的通信效率, 显著减少程序的总体执行时间.

关键词: 通信拓扑; 物理拓扑; 进程映射; NUMA; MPI

中图分类号: TP393.1

文献标识码: A

文章编号: 1000-7180(2013)05-0067-05

Research on MPI Topology—Aware Process Mapping

LI Dong-yang, WANG Yun-lan

(School of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract: This paper firstly introduce the method used to obtain application communication topology and hardware topology. After that process proximity is defined considering both communication volume and communication frequency among processes. And then based on this work a process mapping method is proposed for computing systems with hierarchical networks and NUMA effect. This algorithm partitions the sets of processes base on process proximity. During the processes partition, a tree-like structure is built which is isomorphic to the physical topology tree. Then a mapping between any two correspondent nodes on the two isomorphic trees is performed. Experimental evaluations on a multicore cluster show that the process mapping method presented in this paper can effectively improve the communication efficiency of MPI application and cut down the whole execution time significantly in most cases.

Key words: communication topology; physical topology; process mapping; NUMA; MPI

1 引言

随着多核架构的出现, 集群的组成方式也发生了显著变化, 由用简单网络互连起来的单处理器节点组成变为以片上多核对称多处理器 (SMP—CMP) 节点和复杂网络层次结构为特征的架构. 另一方面, 大型计算系统上的节点个数稳步递增, 也因此导致互连网络的规模迅速扩大. 目前大部分的集群计算系统采用 Fat Tree 网络拓扑. Fat Tree 网络是一个树状结构, 计算节点位于树的叶子节点上, 交

换机位于中间节点上. Infiniband 网络就是一种典型的胖树网络. 在 2012 年 6 月发布的 Top500 中, 有 38.4% 的系统采用了 Infiniband 网络. 在这种架构上, 通信性能将会随着通信进程在系统中位置的不同而发生剧烈的变化, 因为通信进程所在处理核心间的物理距离越近, 通信效率就会更高. 所以, 如 MPI 等编程标准及其实现在调节整体应用性能上扮演关键角色. 但 MPI 规范作为一个跨平台的编程标准, 其在设计时就是独立于架构的, 本身没有提供任何接口来帮助编程者检索硬件信息并将此信息传

递给应用程序. 由于 MPI 不能感知硬件架构, 所以没有提供有效的拓扑映射机制.

众多学者在拓扑感知映射上做了很多的研究. 其中文献[1]研究了在 MPI 中将虚拟拓扑映射到物理拓扑上的算法. 文献[2]使用 Scotch^[3]图划分软件实现了将赋权通信图映射到赋权节点架构图, 通信图的权重是依据总通信量而设定的, 但是其并没有考虑网络的分层. 文献[4]通过修改 MPI 拓扑创建函数的实现, 使用嵌入图法和节点网络发现模块来匹配应用程序通信拓扑和多核多层网络集群的物理拓扑. 文献[5]提出了 TreeMatch 算法来计算在 NUMA 集群上进程到资源近似最优的映射, 和本文工作不同, 其主要关注节点内架构而并未充分考虑网络层次.

2 模型和方法

解决 MPI 应用程序进程到计算节点及处理核心的映射问题, 首先要做两个基础准备工作: 1, 获取应用程序进程间的通信信息, 并据此信息为应用程序建立通信拓扑图以描述其通信特征; 2, 获取硬件信息, 包括节点间互连网络及节点内的存储器层次信息, 并据此信息建立物理拓扑描述模型. 随后, 才能提出相应的进程映射解决方案.

2.1 应用程序通信拓扑

应用程序通信拓扑描述的是给定 MPI 应用程序进程间的通信信息, 如进程间的通信量, 通信的频繁程度等, 并以此反映进程间的亲和度. 亲和度高的进程放在相近的物理位置上可以获取较高的通信效率. 过去的研究一般用进程间的通信总量来描述通信拓扑, 这种方式用一个 $P \times P$ 的矩阵来存储每对通信进程在整个程序执行期间的通信总量, 其中 P 代表进程的个数. 文中同样采用通信量作为描述通信拓扑的重要信息, 还考虑了进程间的通信频率. 因为虽然一些进程间通信总量较小, 但频繁的发送小数据量信息, 短信息对延迟较为敏感. 而信息发送和接收的准备时间是一定的, 和消息大小无关. 此外不同时刻发送消息时网络拥塞状况不同, 这样就会使发送多个小数据量信息的总开销要大于发送某条长信息的时间开销, 即使多个短消息的总大小和长信息接近. 在考虑了通信次数后, 定义了进程 i 和 j 间的亲和度 (proximity):

$$\text{proxim}_{i,j} = \sqrt{\text{scom}_{i,j} \times n_{i,j}}$$

其中 $\text{scom}_{i,j}$ 为进程 i 和进程 j 间的通信总量, $n_{i,j}$ 为进程 i 和进程 j 的通信次数. 并用一个大小为

$p \times p$ 的亲和度矩阵来存储每对进程间的亲和度, 以此亲和度矩阵代替通信量矩阵来描述应用程序通信拓扑. 为获取应用程序的通信拓扑信息, 需要知道进程间的通信量和通信次数, 并以矩阵形式来表示. 进程间点到点的通信信息可以通过 MPE (MPICH2 自带的程序剖析工具) 来获取, 但速度相当慢. 所以我们实现了一个简单的 MPI 程序剖析工具, 该工具可以对进程间点到点的通信信息进行追踪, 记录通信次数和通信量. 此外, 我们还对 MPICH2 各种集合通信函数的源码进行了分析, 以了解各集合通信函数的实现算法. 在此基础上, 设计并实现了集合通信的剖析方法, 以追踪执行集合通信时所涉及到实际的各进程间的点到点通信量和通信次数. 该剖析工具针对集合通信的剖析算法是基于具体的 MPI 源码实现的.

2.2 物理拓扑

节点内物理拓扑的获取已经有较为成熟的工具, 其中最为常用的工具就是 Hardware Locality (HWLOC)^[3]. 使用 HWLOC 可以很容易的获取关于节点内硬件架构的详细信息, 如 NUMA 节点的 Socket、核心的个数, 以及存储器分层信息, 并以树状拓扑表示. 例如对于本文实验所用到的计算节点可以获取如下信息: 节点内有两个 Socket, 每个 Socket 上有四个核心, 每个核心 (core) 有一个处理单元 (PU), 每个核心独享 64KB 的一级缓存, 512KB 的二级缓存, 同一个 Socket 上的核心共享 6MB 的三级缓存及 4GB 的主存. 由于同一个 Socket 上的核心共享缓存, 所以位于同一个 Socket 不同核心上的进程间的通信速度理应快于位于不同 Socket 上的进程间的通信. 而同一个节点上不同处理器间可以通过 HYPERTRANSPORT 或 QPI 高速数据传输总线通信, 所以位于同一节点不同处理器上的进程间的通信应快于不同节点上通过网络进行通信的进程. 我们用两个连接在同一个交换机上的计算节点测量了不同大小 MPI 消息的相对发送带宽, 测量时两个通信进程所在处理核心的物理位置关系分以下三种情况: ①共享缓存 (case 1); ②在同一主板但不在同一 socket 上 (case 2); ③位于一跳路由距离的不同节点上 (case 3). 通过将 case 3 的带宽视为 1, 计算出 case 1, case 2 对 case 3 的相对带宽, 测试结果如图 1 所示.

我们可以看到, 三种情形下的带宽在不同消息大小时保持稳定的比例关系, 在消息大小在 1B—1KB 之间时, case 1, case 2, case 3 通信带宽的比例

关系约为 13:10:1,位于 2KB—64KB 时,此比例关系为 16:11:1;128KB—4MB,比例关系为 10:12:1. 整体平均比例关系为 13:11:1.

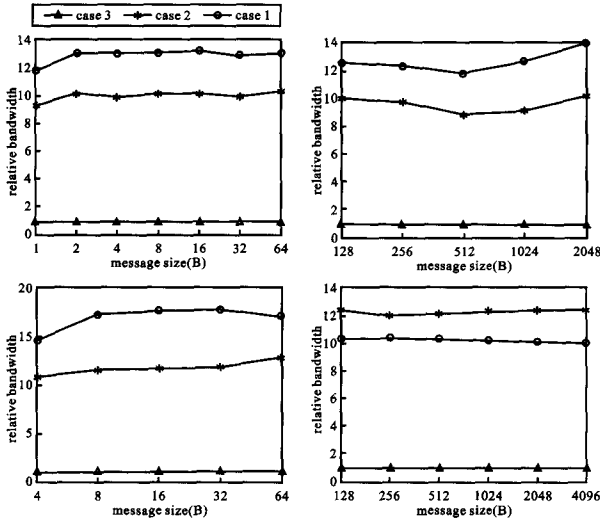


图1 case 1, case 2 相对于 case 3 的带宽

为了获取集群系统的完整拓扑视图,还需要获取整个网络的拓扑图. 此拓扑图可以反映胖树网络路由器层次结构,计算节点间的通信距离(路由跳数). 目前网络拓扑的获取还没有统一的工具,可以根据计算节点间的物理连接手工编写网络拓扑. 同时很多设备厂商提供了其专用的拓扑查询工具,如表 1 所示.

表 1 专用网络拓扑查询工具

互连网络	拓扑查询工具
Infiniband(OFED)	fm_db2wirelist
Myrinet(MX)	ibdiagnet&ibnet-discover
SeaStar(Cray XT)	xtprocadmin & xtdb2proc
BlueGene/P(DCMF)	DCMF API

根据目前组成集群的对称多处理器节点的常用架构,我们按照如下标准定义了两个通信核心间的距离模型:(1)位于同一个处理器内;(2)位于同一个 Socket 内;(3)位于同一主板上;(4)位于同一节点不同主板上;(5)不在同一节点上. 满足(1),距离定义为 d_1 . 若不满足(1),我们将检查是否满足(2). 满足(2)距离定义为 d_2 ;只满足(3)不满足(2)距离定义为 d_3 ;满足(4)不满足(3)距离定义为 d_4 ;不满足(4)只满足(5),则核间距离也即为网络上节点间的距离,路由跳数为 1 的距离定义为 d_5 ,两跳和两跳以上的距离的简化建模为 $n \times d_5$, n 为不同节点上核间通信的网络路由跳数. 其中, $d_1 < d_2 < d_3 < d_4 < d_5$. $d_i (i=1, 2, \dots, 5)$ 的具体取值可以根据硬件设备的实测结果或经验而定. 例如,针对本文实验所用到

的硬件设备,核间位置关系只简单涉及到标准(1), (2), (5);根据图 1 测得的带宽比例关系,定义 $d_1 = 1$, $d_2 = 2$, $d_5 = 10$. 节点的拓扑层次结构可以简化描述为 N—M—S—P—C,其中 N(Nodes)表节点层, M(motherboard)表示主板层, S(Socket) socket 层, P(processor)处理器层, C(core)处理核心层. 根据处理核心所在位置共享的物理层次结构,可以设定核间的距离关系为 $d_P < d_S < d_M < d_N$.

3 映射算法设计

拓扑映射问题是一个 NP 完全问题^[6],有效的解决途径就是通过一些启发式算法寻找接近最优的解. 这一部分将介绍求解这一问题的同构树映射算法框架. 为了便于处理,我们对此映射问题进行简化,假设物理拓扑树是平衡的,所有的叶子节点(即计算节点)都在相同的层次上,同一个层次上的节点的孩子节点数相同,每一个计算核心只运行一个进程. 这种假设在所使用的处理器,插槽个数,节点都完全相同的并行计算系统上是非常符合实际的. 同构树算法的目标就是为每一个进程指定一个执行节点或核心,也即是网络拓扑树和节点拓扑树的叶子节点. 为了使通信代价最小,同构树映射算法依据硬件系统的层次结构、进程间的亲和度来进行映射.

由于我们对硬件进行了简化假设,结合算法的需要,网络拓扑树的表示可以简化为 $Hnet = (S, N, H, cnbr, nlevel)$,其中 S 表示物理拓扑树所有节点的集合,包括交换机节点和计算节点, N 表示所有计算节点的集合, H 表示物理拓扑树的高度, $cnbr$ 表示每个节点的计算核心个数都为 $cnbr$, $nlevel(i)$ 表示物理拓扑树第 i 层的节点个数. 节点内拓扑可以表示为 $Hnode = (Cores, nlevel)$,其中 $Cores$ 是该节点上可用处理核心的集合, $nlevel(i) (i \in \{M, S, P, C\})$. 应用程序拓扑图可以表示为 $G = (P, Proxim)$,其中 P 为应用程序进程集合, $Proxim$ 为进程间亲和度矩阵, $Proxim(i, j)$ 为进程 P_i 和 P_j 间的亲和度.

算法框架分两个阶段进行. 第一阶段,进行进程到计算节点的映射,映射的准则就是尽可能的将亲和度高的进程放置到同一个物理节点上,最小化网络通信代价. 该算法的描述如下:

(1) 输入网络拓扑树 $Hnet = (S, N, H, cnbr, nlevel)$ 和应用程序通信拓扑 $G = (P, Proxim)$;

(2) 检查系统可用核心数 $N \times cnbr$ 是否大于应用程序进程个数 $|P|$,若大于则转入(3),否则转到

(4);

(3)对应用程序通信拓扑添加 $|N| \times cnbr - |P|$ 个虚拟进程,这些虚拟进程相互间以及和其他进程间的亲和度为零,并对亲和度矩阵 $Proxim$ 进行扩展,以加入这些虚拟进程的信息。

(4)记当前层次为 $l(l=1,2,\dots,H)$,当前进程集合为 $P_l, P_1=P$,当前进程亲和度矩阵为 $Proxim_l, Proxim_1=Proxim$. 将进程集合 P_l 中进程按照相互间的亲和度进行划分,聚合成 $nlevel(l)$ 个进程组 $(P_{l,1}, P_{l,2}, \dots, P_{l,nlevel(l)})$,每一个进程组中进程个数为 $cnbr$,使组内亲和度尽可能的大,组间亲和度尽可能的小。之后创建进程组集合 $P_{l+1} = \{P_{l,1}, P_{l,2}, \dots, P_{l,nlevel(l)}\}$,并如下公式为此进程组集合生成亲和度矩阵 $Proxim_{l+1}$:

$$Proxim_{l+1}(i,j) = \sum_{a \in P_{l,i}, b \in P_{l,j}} Proxim_l(a,b)$$

(5) $l=l+1$,若 $l < H$,转到(4);

(6)至此建立了集合 $P_1, P_2, \dots, P_H$ 成员间的树状包含关系,此树状包含关系和网络扑树的层次包含关系相匹配,遍历这两棵树就可以找到两棵树叶子节点间的对应关系 $P_{l,i} \rightarrow N_j$,将进程子集 $P_{l,i}$ 中进程放置到节点 N_j 上,完成进程到节点的映射。

为详细说明此算法,图 2 显示了第一阶段算法的一个简单的算例。图 2 左半图显示了网络拓扑。在第一层上有四个计算节点: N_1, N_2, N_3, N_4 。其它节点都是路由器或交换机。右半图显示了在进程分组过程中创建的树状结构。从这两棵同构树上可以直观的看到进程组 $P_{1,1}, P_{1,3}, P_{1,2}, P_{1,4}$ 中的进程可以映射到节点 N_1, N_2, N_3, N_4 上。

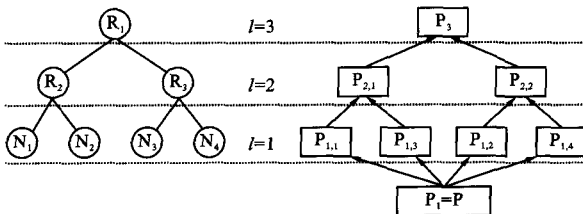


图2 第一阶段算法算例

在第一阶段算法只完成进程到计算节点的映射,还需要执行第二阶段算法,根据节点内 NUMA 分层结构来完成计算进程到节点内具体处理核心的映射。第二阶段算法内容如下:

(1)输入单个节点的拓扑树 $Hnode=(C, nlevel)$,亲和度矩阵 $Proxim$,以及映射到该节点的进程子集 $P_{1,j}$ 。

(2)从 $Proxim$ 中抽取进程子集 $P_{1,j}$ 中的进程间的亲和度信息,并生成针对该子集的亲和度矩阵

$Proxim_{1,j}$ 。为方便处理这里用 1,2,3,4,分别表示节点拓扑树中的 P, S, M , 层,并记 $Proxim = Proxim_{1,j}, P=P_{1,j}$ 。

(3)记当前层次为 $l(l=1,2,3,4)$,当前集合为 $P_l, P_1=P$,当前进程亲和度矩阵为 $Proxim_l, Proxim_1=Proxim$. 将进程集合 P_l 中进程按照相互间的亲和度均等划分成 $nlevel(l)$ 个进程组 $(P_{l,1}, P_{l,2}, \dots, P_{l,nlevel(l)})$,使组内亲和度尽可能的大,组间亲和度尽可能的小。之后创建进程组集合 $P_{l+1} = \{P_{l,1}, P_{l,2}, \dots, P_{l,nlevel(l)}\}$,然后用和第一阶段第四步相同的公式为此进程组集合生成亲和度矩阵 $Proxim_{l+1}$ 。

(4) $l=l+1$,若 $l < 4$,则到(3);若 $l=4$,则转到(5)。

(5)同第一阶段算法第 6 步类似,我们建立了集合 P_1, P_2, P_3, P_4 成员间的树状包含关系,此树状包含关系和节点内部的树状层次包含关系相匹配,遍历这两棵树我们就可以找到两棵树叶子节点间的对应关系 $p_{1,i,k} \rightarrow C_j$,将进程子集 $P_{1,i}$ 中进程 k 放置到核心 C_j ,完成进程到节点具体核心的映射。

图 3 显示了第二阶段算法的一个简单算例,其中左半图显示了单个节点内的拓扑结构,该节点只有 1 个主板,两个双核处理器分别插在两个不同的 Socket 上。右半图显示了在进程或进程组划分过程中建立的树状结构。可以看到,进程子集 $P_{1,1}$ 中两个进程应映射到处理器 P_1 的两个处理核心上, $P_{1,2}$ 中的两个进程应映射到处理器 P_2 的两个核心上。在第一阶段算法第 4 以及第二阶段的第 3 步中涉及到进程的分组,也即是带权图的划分问题。解决图划分问题已有一些成熟的工具,如 SCOTCH^[3]。这里我们使用了 SCOTCH 提供图划分功能,当然也可以根据具体情况选用其它的图划分方法。

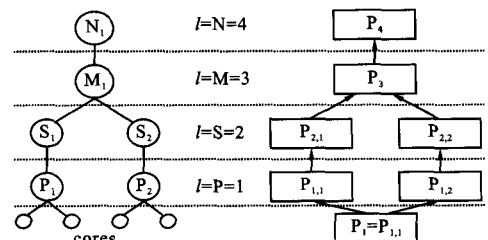


图3 第二阶段算法算例

4 实验

我们进行了一系列的实验以评估采用同构树进程映射算法对 MPI 应用程序进程映射绑定后所带来的性能提升。所有的测试都在曙光 TC2600 集群

系统上进行.实验使用了该集群中的八个计算节点,节点间使用千兆以太网互连,每个节点拥有8GB内存,两个四核AMD Opteron2380处理器,主频为2.5GHz,每个处理器的四个核心共享6MB三级缓存,每个核心独享512KB二级缓存和6KB一级缓存.集群所用操作系统为Ubuntu server 11.04.为了测试本文算法对实际应用程序进行映射绑定的效果,我们使用了NPB基准测试工具在MPICH2环境下进行了测试.

为了验证在实际应用中本文提出的映射绑定方法的有效性,使用NPB基准测试程序进行了绑定测试.考虑到实验所用集群系统的处理能力,选用的数据规模大小为NPB所规定的C级.实验结果表明使用本文方法对测试程序进程映射绑定后,大部分程序的性能都有明显的提升,并且随着进程数的增多,性能提升成递增趋势.

5 结束语

本文介绍了获取硬件拓扑结构和应用程序拓扑结构的方法,并结合进程间的通信量和通信次数定义了进程间的通信亲和度.针对具有NUMA效应,分层网络特性的计算设备提出了同构树映射算法,该算法根据进程间的亲和度对进程集合进行划分,在划分过程中构造出和物理拓扑树同构的树状结构,随后在两棵同构树的对应节点上做映射完成进程到物理核心的映射.在具有NUMA特性的多核集群系统上的实验表明,不论是在MPICH2还是OpenMPI环境下,针对通信密集且通信模式具有显著局部性的应用程序,使用本文映射算法对进程进行映射绑定后,程序的通信效率提升明显,程序的执行时间显著缩短.

未来的工作,继续完善MPI程序剖析工具,使之可以对MPICH2之外的其它MPI环境下的集合通信进行剖析.我们也会寻求通过对源码静态分析以获取应用程序通信信息的方法.接下来还将深入研究通信次数和通信量对通信开销的影响,以建立更为精确的进程间通信亲和度模型.

参考文献:

- [1] Hatazaki T. Rank reordering strategy for MPI topology creation functions[J]. Recent Advances in Parallel Virtual Machine and Message Passing Interface, 1998 (1497):188-195.
- [2] Mercier G, Je ro me Clet-Ortega. Towards an efficient process placement policy for mpi applications in multicore enviroments[C]// In Euro PVM/MPI 2009, volume 5759 of LNCS. Espoo, Finland, Springer, 2009:104-115.
- [3] Pellegrini F Scotch, LibScotch 5.1 User's Guide. ScalApplix project, INRIA Bordeaux - Sud - Ouest, ENSEIRB & LaBRI, UMR CNRS 5800 [EB/OL]. [2008-10-19]. <http://www.labri.fr/perso/pelegrin/scotch/>.
- [4] Rashti M. Multi-core and network aware MPI topology functions[C]//The Euro MPI users' group conference (EuroMPI). Greece, 2011.
- [5] Jeannot E, Mercier G. Near-Optimal Placement of MPI Processes on Hierarchical NUMA Architectures [J]. Euro-Par 2010 - Parallel Processing, 2010 (6272):199-210.
- [6] David V, Fraboul C, Rousselot J, et al. Partitioning and mapping communication graphs on a modular reconfigurable parallel architecture[C]// Parallel Processing: CONPAR92-VAPPV, 2nd Joint International Conference on Vector and Parallel Processing. Lyon, France, 1992.

作者简介:



李东洋 男,硕士研究生.研究方向为高性能计算、MPI/OpenMP并行程序设计及性能优化.