班　级　13-0911
学　号　13091104

西安电子科技大学

本科毕业设计论文

题　　目　基于 OpenSHMEM 的并行程序的设计与研究

学　　院　软件学院

专　　业　软件工程

学生姓名　葛诗瑶

导师姓名　张亮

Contributers:

HaoChen write the statement page

Olorin183795 bug report

yzg3307 bug report

Author by Xue-Jilong (xuejilong@gmail.com) and Justin-Wong (bigeagle@xdlinux.info)

Typeset by LaTeX 2ε and CTEX and provide for Bachelor Thesis of Xidian University.

The first page will not appear in your final thesis paper. Take it easy for this copyright footnote. :-)

目　录

第一章　绪论

1.1　题目背景介绍

1.1.1　并行计算

Gordon Moore 在 1965 年所做的预言在很长的一段时间内都被证明是准确的。在这个趋势下，CPU 的性能基本每两年可以翻一番[15] 。这种快速的增长一度对于用户以及软件开发者而言意味着，要获得更快速的计算更好的性能，他们只需要等待下一代的微处理器带来的翻倍性能即可。但是自从 2002 年开始，单核处理器的性能增长就已经以每年 20% 的速度下降[18]。并且，在 2005 年之后，主要的微处理器制造商都一致认为未来处理器性能增长的出路在于并行化。与其尝试着突破散热和晶体管密度等的物理限制继续研究制造快速的单核处理器，制造商们纷纷尝试将多个可用的通用核集成到一个片上系统中。

这个重大改变对于一直享受“免费午餐”的软件开发者而言意义重大：从单核向多核的转变意味着不可能在对已有的串行软件毫无修改的情况下移植到多核平台还能立即享受多核带来的性能提升。以前的软件无法自己意识到多个核心的存在，在这样的平台上运行的程序的性能最多只能和运行在单核上一样，甚至运行性能更差。

至此，并行计算不仅仅存在于学术研究领域，更快速爆发地进入了工业界的每个角落。并行计算不仅仅调动着硬件开发者的生产力，也让众多的软件开发者参与其中，围绕并行计算、高性能计算等主题提出了众多的并行算法、编程模型以及开发平台，并且以此为基础，进一步提出了例如云计算、大数据等新的概念与模型。

可以说，并行计算已经从数十年前针对特殊应用的解决方案，变成了当今计算机领域的另一块基石，也因此，对于并行计算软硬件的研究也变得意义非常。

1.1.2　并行软硬件的发展

一般而言，我们可以从结构模型、访存方式以及互联网络三个方面讨论并行计算机硬件的特征[17]。

并行计算领域常常引用弗林分类对计算机根据其数据流和指令流的数量划分为 4 类[17]。分别是单指令流单数据流（Single Instruction Single Data,SISD), 单指令流多数据流 (Single Instruction Multiple Data, SIMD), 多指令流单数据流 (Multiple Instruction Single Data,MISD) 以及多指令流多数据流（Multiple Instruction Multiple Data,MIMD)。其中 MISD 结构目前还没有成熟的产品，MIMD 和 SIMD 是当今比较常见的多核平台的架构。

另外，根据 CPU 访问内存的方式也可以对并行计算机进行分类。在规模较小的，一般为 2 ～ 100 个核之间的多核架构，通常采用共享内存的架构，即每个计

算单元或者 CPU 都可以访问同一块内存，并且较小的核数可以保证每个核都可以以相同的速度访问内存的任意地址，这种方式称为均匀存储访问模型 (Uniform Memory Access,UMA)。但是当核数增加，内存面积扩大之后，访问距离计算单元（核）较近的内存和访问较远内存的延迟将出现细微的差距，或者物理上内存是分布在各个处理单元内部，每个单元节点通过网络相连然后所有局部内存构成一个全局内存，这种情况下的内存访问模型就是非均匀的，称为非均匀存储访问模型（NonUniform Memory Access, NUMA），在非均匀存储访问模型中，如果保证缓存一致性，则被称为缓存一致的非均匀存储访问模型 (cache coherent-NUMA, cc-NUMA)。

最后，可以根据连接多个处理器或者连接处理器与内存之间的网络对并行计算机进行分类。根据今年来对多处理器平台上的片上网络（Network on Chip, NoC) 的研究和总结[1], 现在的片上网络实现存在多种方案，主要的有点到点结构、Fat-tree 结构、结构总线、星形结构、环状网络以及 Mesh 网络为主，其中 Mesh 网络在 05 年以后的架构中越来越常见。

并行硬件已经进入了我们的生活，现在几乎所有的台式电脑和服务器都使用的是多核处理器。然而，并行硬件似乎并没有跟上硬件的发展速度。除了一些操作系统、Web 服务器以及数据库系统外，很少有多核应用见诸市面。正如上一小节所言，现有的串行软件不能自主地利用多核的优势，软件开发者想要继续提升性能，就必须适应并且熟悉现在的分布式内粗怒或者共享内存架构。

在共享内存架构，一般采用如下方式：程序以一个进程的方式启动，在需要并行处理的地方开启多个线程。这种以线程为基础的并行编程方式的一个代表是 OpenMP 和 POSIX Thread，我们将在 二中进一步研究和讨论。另外，如果我们运行的是分布式内存的应用程序，一般都采用多个进程协作的方式，基于进程的并行编程方式的代表则为 MPI 和本文的核心 OpenSHMEM。

随着众核处理器架构的出现在高性能计算领域，相关的并行编程模型，工具以及库的开发比以往任何时候都备受重视。

以往的并行计算领域基本只关注以 MPI 为代表的消息传递编程模型，或者是以 OpenMP 为代表的共享内存模型，但是由于 PGAS 的抽象模型以及它能够构建的一个直接清晰的内存访问和通信模型，最近对于部分全局地址空间（Partitioned Global Address Space,PGAS) 的研究越来越多。现在比较有名的基于 PGAS 模型的语言或者库包括 Unified Parallel C(UPC), X10，Chapel，Co-Array Fortran(CAF), Titanium, 以及 SHMEM[3]。

1.1.3 并行技术的应用场景

数十年来，我们的生活在技术的推动下有了天翻地覆的变化，我们在享受这些变化的同时又成为了技术发展的直接推动力。我们现在正在进行的或者正在享受成果的一些项目，比如人类基因组项目，医学成像技术，Web 的发展甚至是越来越精细的电影特效都是建立在快速发展的技术带来的计算性能飞跃的基础上的。

但是，我们要解决的问题似乎并没有随着性能翻番而减少，一些以前认为不可能的问题都亟待解决，要解决这些问题，就需要并行计算。以下是一些例子：

- 天气预报。要更精确的预测天气和气候的变化，就需要更为精确的模型，由于预测需要考虑众多的气象、大气、海洋等的因素，更为精确的模型意味着更多的计算量，这样的计算量即使在最先进的单核上也是无法想象的。
- 模型模拟。科学计算一直是并行计算的最大“雇主”。在科学计算的领域，很多模型的验证需要大量的资金投入，有的甚至无法在现实中完成实验，这时就需要在计算机上进行模拟实验。而要在计算机中建立接近真实的模型，就需要大量的参数，随之而来的是大量的计算任务。最早的并行计算机就是在这样的驱动力下研究出来，目前为止，美国 NASA 依然是世界上最为著名的并行计算的主要支持者。
- 数据分析。从计算机发展以来，尤其是互联网时代的到来，整个社会产生了巨大的数据量，然而如果不分析这巨大数据量背后的信息，这些巨量的数据都是无用的。比如人类的 DNA，比如将要到来的物联网时代的传感信息，甚至是当今的云计算的海量数据存储和检索等等，都需要以并行计算为基础进行。

1.2　国内外研究现状

SIMD 的编程方法对于大型并行系统而言比较适合，目前有许多不同的遵循 SIMD 的编程模型，比如 Active Messagepassing（AM），分布式共享内存，以及 PGAS[14]。在 1.1.2部分中，我们提到了本文的主题，OpenSHMEM 项目。OpenSHMEM 项目虽然是在 2011 年被提出并成立，但实际上 SHMEM 编程模型在之前就存在了[2][3][20]。但是由于多个厂商提供了几个不同版本的 SHMEM 实现，虽然遵循了一样的通信模型和思想，但是在一些调用细节上有所差异，导致 SHMEM 的各个实现之间的不可移植性，限制了 SHMEM 的发展。在众核平台开始兴起之后，由于 PGAS 模型被认为在大型多核平台上会有良好的表现，对于 SHMEM 的研究和热情重新被燃起，于是 SHMEM 的标准化工作被提出，即 OpenSHMEM 项目。以下对 OpSHMEM 和 SHMEM 项目的几个方面的研究现状逐一叙述。

1.2.1　PGAS

PGAS 是一种同时适合于共享内存和分布式内存并行计算机的编程模型，这种计算机一般由几百甚至上千的 CPU 构成。一般 PGAS 的语言或者实现都具有以下特点：

1. 包含一组处理器，每个处理器都有自己的本地存储。每个本地存储的一部分被声明成是私有的，也就是无法被外部处理器所访问。

2. 一种共享机制，可以让每个处理器的存储的至少一部分与其他处理器共享。共享机制可以通过网络连接和软件实现，也可以通过保证缓存一致性的硬件共享内存实现。

3. 每个共享内存的地址有一个 affinity 处理器，指的是内存位于该处理器的本地内存区域，所以这个处理器访问该内存的延迟最小。

目前支持 PGAS 的语言有很多[21]，比较广泛的有 UPC，X10 以及 Titanium 等。此外，对于 PGAS 模型的支持软硬件研究，PGAS 模型的优势研究等也在进行中[8]。

1.2.2 SHMEM 与 GSHMEM,TSHMEM

SHMEM 通信库最早是由 Cray 公司为他们的 T3D 系统以及 T3E 模型系统所做的应用程序接口。这些系统通常包括了一个逻辑上共享内存地址空间，而物理上却是分布式存储的内存子系统，并且包含一个内存互联网络，一组处理单元（Processing Elements，PEs），一组 I/O 网关以及一个 host 子系统。这两个系列的系统可以支持一些延迟隐藏机制，比如预取机制，远程存以及块转换引擎 (Block Transfer Engine，BLT)。

SHMEM 后来被 SGI 采用，移植到 SGI 基于 Numa-Link 架构的产品上去，并且包含在了消息传递工具箱（Message Passing Toolkit) 中。以后的 SHMEM 实现基本都是从 SGI 的版本演化而来。现有的 SHMEM API 支持 C,C++ 以及 Fortran 语言[2]。

在 OpenSHMEM 之前，关于 SHMEM 的工作大多与平台相关，并且稍有区别。在 OpenSHMEM 标准化说明文档发布之后，除了主页放出的实现样例，还有佛罗里达大学所做的 GSHMEM 实现[11]。GSHMEM 是 GatorSHMEM 的简称，是一个面向大型服务器的 OpenSHMEM 实现。GSHMEM 采用 GASNet 作为通用的网络通信层，将大量的管理和通信任务交给成熟稳定的 GASNet 来完成。并且在 GSHMEM 的实现中，还参考了 MPI 的接口的使用情况，建议增加并且实现了两个新的接口，gather 和 scatter。

随后，在 2012 年，佛罗里达大学实现了 TSHMEM[14]。TSHMEM 是 TileraSHMEM 的简称。TSHMEM 是在 Tilera 众核平台上的 OpenSHMEM 实现。TSHMEM 的实现借鉴了 GSHMEM 的工作，但是这次没有采用 GASNet，而是直接调用了 Tilera 提供的网络通信接口。

除了这些在新平台上重新实现 OpenSHMEM 的工作，还有一部分工作思路是采用新的映射方法，在原有的平台或者操作系统上提供对 OpenSHMEM 的支持[12]。

1.2.3 GASNet

在 OpenSHMEM 主页放出的样例以及 GSHMEM 的实现中，都采用了 GASNet 作为较底层的网络通信层, 如图 1.1。GASNet 是 Global-Address Space Networking

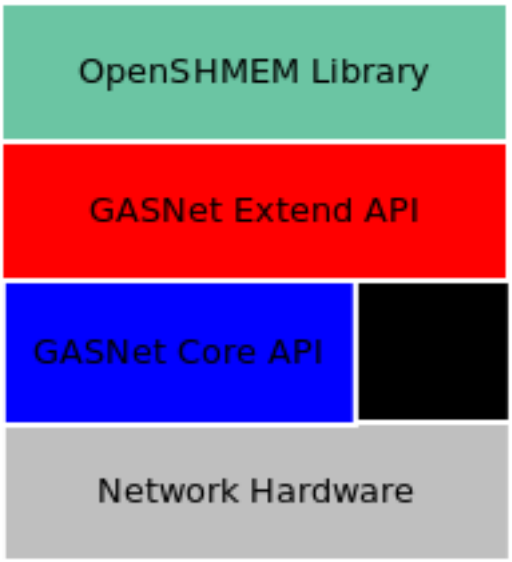

图 1.1 GASNet 和 OpenSHMEM 的层次结构

的简称，本身是语言无关的底层的网络层，是加州大学伯克利分校和劳伦斯伯克利国家实验室（LBNL）于 1993 年联合实现。GASNet 提供了网络无关的高性能通信原语，支持 SPMD 并行编程模型，尤其对 PGAS 类型的语言和库有直接的支持。

总的看来，GASNet 分为两层，分别是 GASNet 核心 API 以及 GASNet 扩展 API。其中核心 API 实现在不同的网络接口上，而扩展 API 则是完全网络无关的，提供中级或者高级抽象的网络通信操作接口。并且有的扩展 API 是直接实现在互联层 API 上的，因此在可能的地方，GASNet 可以支持 RMDA 来提高性能。目前，GASNet 可以支持在一系列常见的网络上运行，比如 UDP，MPI，Cray XT Portals 等等。

1.2.4　Tilera 众核平台

Tilera 公司，坐落于加州 San Jose，是一家以设计以及开发商用众核处理器的公司，Tilera 的产品着重强调了高性能和低功耗，因此在云计算、通用服务器以及嵌入式设备市场上十分流行。

每个 Tilera 的众核处理器都是一个可扩展的二维 Mesh 网络的连接结构，每个核被称为 tile，每个 tile 中包含了一个处理器核新，以及私有的 cache 系统，cache 系统可以通过片上网络和内存互联，这种技术被称为 iMesh。Tilera 的可扩展二维 Mesh 网络由提供数据路由的动态网络构成，数据可以在内存控制器、缓存以及外部 I/O 之间传递交换，开发者可以通过调用 Tilera 提供的通信原语控制数据的传输。

1.3　研究方法与设计思路

本文的安排如下。第一部分绪论，大致叙述了目前并行计算的发展情况，对并行计算的软硬件发展以及目前流行的几种编程模型做一简述，并且总结

有关 OpenSHMEM 和 SHMEM 相关的工作。第二部分，作为对一个较不熟悉的编程模型的研究，首先先将其和其他几种常见的编程方法做一比较，然后根据 OpenSHMEM 的特性说明 OpenSHMEM 为什么会重新受到关注。第三部分，简述并行程序设计的基础准则，之后用 OpenSHMEM 实现一个矩阵乘法的例子说明 OpenSHMEM 编程的原则，随后对于并行计算中最重要的议题，性能优化做一研究。在最后一个部分中，简述了 Tilera-Gx8036 平台的平台特性，并且尝试在该平台上实现 OpenSHMEM API 的一个功能子集。

第二章　并行计算模型的比较

2.1　并行计算模型综述

在多年的多核发展中，存在过多种不同的并行计算编程模型，这些编程模型或多或少体现了并行计算以及多核架构开发中软硬件协同的特性，不同的编程模型代表了一类多核架构硬件体系在软件开发领域的抽象，这种抽象的程度以及角度大大影响了使用该模型的开发人员的开发方法和思路，也最终对并行算法的设计产生影响。

在现今使用的多核平台，从普通消费级多核架构，如 SMP，到大规模的集群设备，使用最为广泛的几种并行编程模型分别为:

- 基于分布式内存和消息传递机制的模型，以 MPI 为代表
- 基于共享内存的模型，以 OpenMP 为代表
- 基于 PGAS 的模型，以 OpenSHMEM 为代表

在本章中，将对现有的常用的基于 CPU 的几种并行计算模型从平台模型、内存模型、执行模型以及编程模型 4 个方面叙述差异。并且最终着眼于 OpenSHMEM 的特性以及适用范围。

2.2　MPI

在并行计算发展的早期并不存在统一的编程模型，开发过程和资源基本上是以厂商提供的 ABI/API 为准的，也就是说软件开发直接受限于硬件平台，这种限制严重制约了软件开发的效率和可移植性，这种情况类似于计算机行业发展的早期，开发人员直接操作卡片的方式，对于软硬件开发的分离和统一标准的呼声越来越高。

MPI 的标准化工作始于 1992 年，在威吉尼亚的威廉姆斯堡召开的分布存储环境中消息传递标准的讨论会，由由 Dongarra,Hempel,Hey 和 Walker 建议的初始草案于 1992 年 11 月推出，并且在 1993 年 2 月完成了修订版，即为 MPI1.0。由 Dongarra,Hempel,Hey 和 Walker 建议的初始草案于 1992 年 11 月推出并在 1993 年 2 月完成了修订版，这就是 MPI 1.0。为了促进 MPI 的发展，一个称为 MPI 论坛的非官方组织应运而生，该论坛对 MPI 的发展起了重要的作用。

1995 年六月推出了 MPI 的新版本 MPI1.1，对原来的 MPI 作了进一步的修改完善和扩充，但是当初推出 MPI 标准时，为了能够使它尽快实现并迅速被接受，许多其它方面的重要但实现起来比较复杂的功能都没有定义，比如并行 I/O，当 MPI 被广为接受之后，扩充并提高 MPI 功能的要求就越来越迫切了。于是在 1997

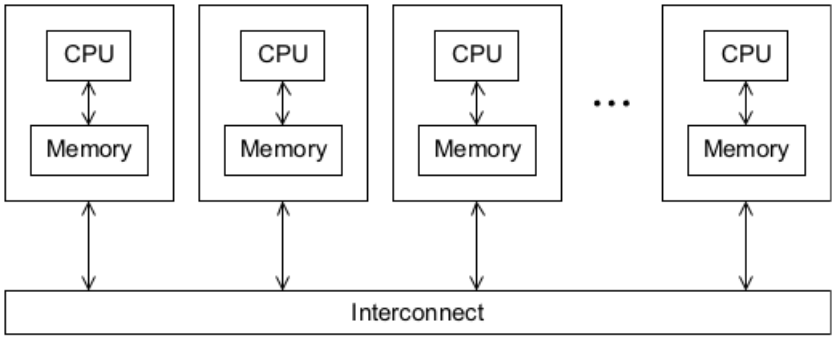

图 2.1 分布式多核系统示意图

年 7 月，在原来 MPI 做了重大扩充的基础上，又增加了 MPI 的扩充部分 MPI-2，MPI-2 中并没有对 MPI-1 做功能上的扩展，而是增加了 3 个重要的方面：并行 I/O，远程存储访问和动态进程管理。至此，MPI-1 和 MPI-2 已经是一个巨大的多达 200 余个调用的编程库，并且与 Fortran77、C、Fortran99 和 C++ 四种语言绑定，涵盖了科学计算领域以及较底层的系统和应用程序对并行编程的需求。

2.2.1 内存模型

严格的说，并行编程模型并不直接定义硬件的结构，也就是说，并不存在某个并行编程模型只能执行在某种类型的平台上的限制，但是实际上，硬件发展的影响对于软件开发的影响直接体现在编程模型对于平台的抽象上，即在使用某个模型时，开发人员将假定程序是运行在这种类型的平台上，而不去考虑真正的底层实现如何。

MPI 是消息传递接口（Message Passing Interface）的缩写，并且 MPI 已经成为这种编程模型的代表和事实标准。消息传递模型在分布式内存并行机上十分容易理解并且高效。

如图 2.1所示为分布存储环境下的消息传递的抽象平台，每个计算单元带有自己私有的内存系统，并且通过互联网络和其他计算单元相连接，两个 CPU 之间通过传递消息的方式进行通信，这是 MPI 的核心，即对提供进程之间通信的方式。

假设进程 1 需要访问进程 0 的内存内容，则它必须显式的创建一个消息，并且等待进程 0 接收并且处理该消息，将它需要的内容作为消息内容返回。

2.2.2 编程模型

MPI 在程序中通过普通库调用的方式调用，开发者显式的分布任务。MPI-1 以及 MPI-2 库的大小虽然已经很庞大，为开发者提供超过 200 条调用接口。但是仅仅需要 6 个调用，就可以写出最简单的 MPI 程序，这 6 条调用也是 MPI 的核心。以一个最简单的 Hello world 调用这 6 个函数，如图 2.2所示说明 MPI 的编程顺序。源码在附录中。如上图所示，开发者需要自己显式的发送和接受数据，并且可以自己定义通信空间 (Communication Space), 从而组建进程子集进行通信。在

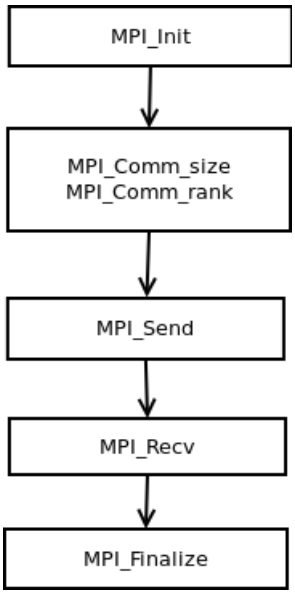

图 2.2 MPI 的编程模型简图

MPI_Send 和 MPI_Receive 中将目标地址或者源地址作为参数传递，另外需要传递 tag 以作为数据配对的依据。

2.2.3　执行模型

在图 2.2中显示的基本的 MPI 编程模型，编程过程是一个指定串行的过程，然而在 MPI 程序运行起来之后，具体的说是在 MPI_Init() 执行之后，MPI 通信空间建立，多个进程将同时运行。如图 2.3所示。

其中参数 n 为启动的进程数目，n 在程序调用时指定。在 MPI-1 中，启动的进程数目在进程启动后就无法更改，而在 MPI-2 中增加了动态管理进程的特性。

2.3　OpenMP

OpenMP 是共享内存应用的编程接口，它的特性是建立在以往对共享内存并行编程的探索之上。OpenMP 被设计为可以适应在一个大范围的 SMP 架构上实现。随着多核机器以及多线程处理器在市场上的流行，OpenMP 可能也会被用于创建运行在单核集群的应用。

就像以前的一些实验性的实现，OpenMP 也没有被实现成一种新编程语言，而是被设计成可以通过向串行程序中添加标注的方式达到并行的目的。现有的可以绑定的语言有 Fortran, C 以及 C++，OpenMP 描述了如何在多个运行在不同处理器或者核上的线程之间分布任务，并且说明了它们如何访问共享的数据。OpenMP 这种插入标注的方式可以让一些现有的串行程序可以在仅需要极少修改的情况下很快的运行在 SMP 架构上并且享受这种多核架构带来的加速。在实际中，有许多应用中仍存在可观的可以挖掘的并行潜力。

OpenMP 的成功归功于一系列的因素，其一是 OpenMP 对于结构化并行编程

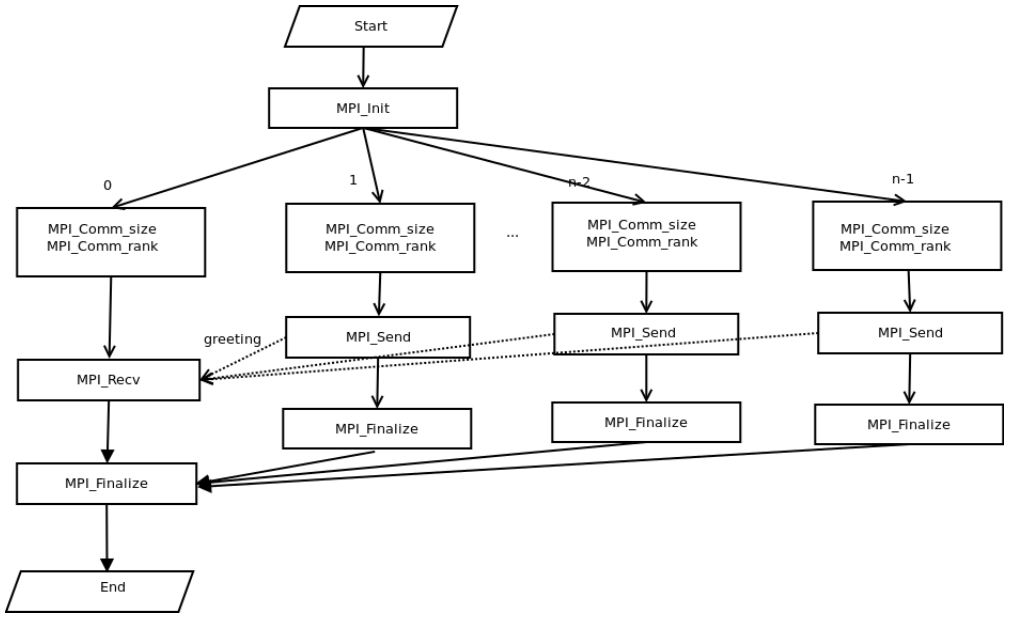

图 2.3 MPI 程序执行模型

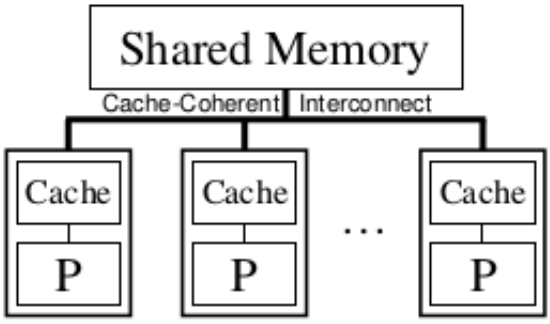

图 2.4 cache 一致的共享内存结构

的强调，其二是 OpenMP 易于上手使用，因为 OpenMP 将大部分的并行任务都交给编译器来完成，并且 OpenMP 的程序也有比较好的移植性，所以 OpenMP 程序可以运行在多个不同的平台上。

2.3.1 内存模型

OpenMP 仅强调平台模型中所有的内存都是可以共享的，但是并没有要求实际的硬件平台必须是共享内存。一般的 OpenMP 应用程序中的平台模型如图 2.4所示。

每个处理器（图中的 P 表示 Processor) 带有自己的私有缓存，缓存中的数据是不可共享的，需要某种方法保证缓存和内存中内容的一致性，不能出现两个处理器访问同一个内存地址时，因为缓存和内存内容没有同步而出现同一地址两个副本的情况。

内存是全局统一编址的，所有参与的处理器都可以访问，并且多个处理器根据一个地址会访问到同样的内容。很明显，由此引发的最重要的问题就是互斥。

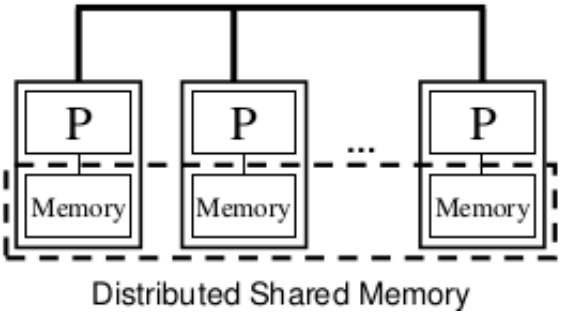

图 2.5 分布式环境下的共享内存

这部分将在2.3.3部分予以讨论。

另外，在分布式内存环境中，如果要实现 OpenMP 程序，则需要模拟出共享内存的环境，如图 2.5所示。实际上，这种平台内存结构和 PGAS 模型十分相似。

2.3.2　编程模型

与另一个共享内存的编程模型 Pthreads 相比，OpenMP 的编程方法有一些本质的不同[16][18]。Pthreads 的方法要求开发者指定每个线程启动之后具体的行动细节，并且要求开发者保证线程动作的可靠性，比如对锁的使用。而 OpenMP 则仅仅要求开发者在需要并行的地方“说明”需要并行执行的区块，然后将真正决定哪个线程执行哪个或者哪些任务的工作则要交给编译器以及运行时系统来完成。

OpenMP 的开发人员也提出了增量式并行过程，即要是用 OpenMP 进行并行开发，则首先需要创建串行的版本，接着分析串行版本中哪些部分可以并行，就将该部分声明为并行执行区域。这种开发方式很明显对于那些已经存在的串行代码有好处，要将它们重新用 MPI 或者 Pthreads 改写，付出的代价要比 OpenMP 大的多。

OpenMP 的并行方式是” 基于指令”的。在 C/C++ 中，只需要添加一些预处理命令，就可以让可以辨认这些指令的编译器创建并行程序，而对于不兼容的串行编译器可以直接忽略，并不影响程序的正确性。

在 C/C++ 中，使用的命令是 *#pragma omp*。其余的形式和串行程序一致，可以参考附录中的示例代码。

2.3.3　执行模型

OpenMP 的执行模型是比较典型的 fork-join 形式。在进入并行区域之前，程序中只有一个线程执行，这个线程可以称为主线程。进入并行区域后主线程立即 fork 出其他执行线程，可以称为从线程，并且和主线程一起执行计算操作。在计算操作结束后，即到达并行区域的结尾，如图 2.6, 此处会有一个隐式的同步操作，随后所有的线程 join 到一处，并且结束执行，仅留下主线程继续执行。

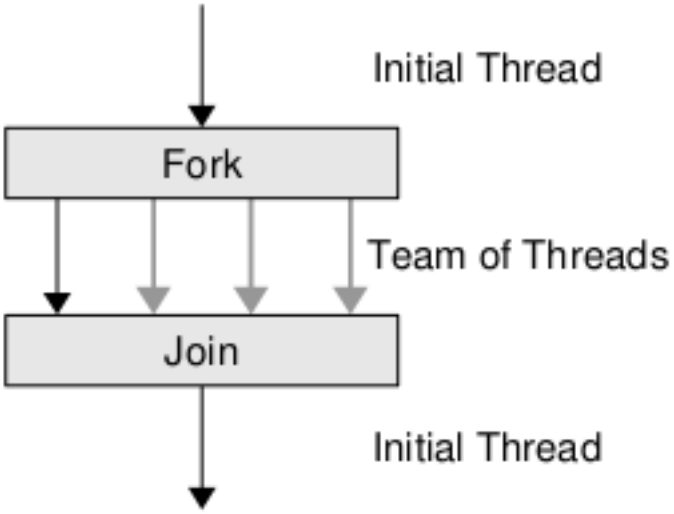

图 2.6 典型的 OpenMP 的执行模型

2.4 OpenSHMEM

SHMEM 是 Symmetric Hierarchical MEMory access 的简称[21]。

如在绪论中介绍的，OpenSHMEM 是一个基于 PGAS 模型的编程库的标准化工作，它提供了单端通信 (One-sided Communications) 的能力。在 OpenSHMEM 之前，曾经有过多种不同平台和版本的 SHMEM 实现，但是由于细微的差异，这些 SHMEM 库都不具有移植性。并且相对于其他同样具有远程内存访问 (Remote Memory Access, RMA) 语法的支持库，OpenSHMEM 也同样具有优势。

现在在科学计算领域使用最多的编程模型和编程方法是 MPI，然而 MPI 使用的是双端通信模型 (Two-sided Communications)，具体来说，每次通信都必须双方共同的主动参与，发送方需要调用 MPI_Send, 接收方则需要调用 MPI_Recv 接收，这样才能完成一次通信。这样做的好处是可以在数据通信的同时带来隐式的同步，但在某些场合，这种隐式同步带来的等待开销也是需要被优化掉的。OpenSHMEM 提供的单端通信模型就降低了数据传输和和同步之间的耦合度，减少了数据通信的等待时间，并且为 RMA 提供了可能。

2.4.1 内存模型

OpenSHMEM 说明文档中说明了数据是如何在内存中存储以及在处理单元 (Process Element, PE) 之间传递。

在 OpenSHMEM 中，数据对象可以存储在 PE 的本地私有内存空间中，也可以存储在 PE 可以远程访问的内存空间中，存储在本地私有空间中的数据对象只能被拥有该空间的 PE 所访问，PE 可以通过 OpenSHMEM 调用访问存储在共享区

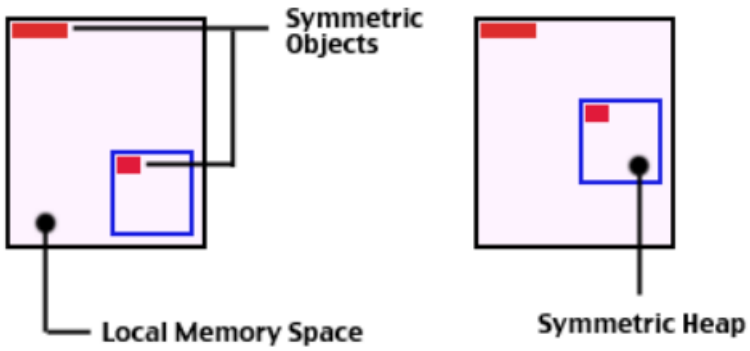

图 2.7 OpenSHMEM 中对称数据对象示意图

域的数据对象。可以远程访问的数据对象在 OpenSHMEM 术语中又称为对称对象 (Symmetric Objects)。当数据对象在每个 PE 中都有相同类型和内容的副本时，就被称为对称对象。在 C 和 C++ 中，可以作为对称对象的可以是静态变量和全局变量，因为静态变量和全局变量在编译时就已经确定，所以逻辑上它们应该在所有 PE 的相同偏移处，并且具有相同的数据类型。如图 **??**所示。另外，OpenSHMEM 还允许创建动态分配的对称数据对象，和静态的对称数据对象, 如全局变量等不同，动态对称数据对象被分配在称为对称堆 (Symmetric Heap) 的一段特殊的内存区域中，对称堆的分配是在运行时动态决定的，也就是说，对称堆的地址在每个 PE 上不一定都是一样的。

2.4.2　编程模型

OpenSHMEM 采用单指令多数据 (Single Process Multiple Data, SPMD) 的方式表达并行。

一个 OpenSHMEM 程序由多个处理器执行，在 OpenSHMEM 术语中称为 PE。和 MPI 一样，在 OpenSHMEM 程序开始执行并行任务之前需要先调用 start_pes() 进行初始化，如图2.8所示。在 start_pes() 中执行必要的初始化步骤，比如建立对称堆, 组织 PE 并且将所有 PE 映射到一个连续数组中，数组下标称为该 PE 的序号 (rank)。rank 从 0 开始，到 n-1，n 为 PE 的数目。

接着可以执行数据传输，或者比如广播、归约等操作。可以执行的操作在说明文档[2] 中一一说明。

和 MPI 不同的是，OpenSHMEM 中并没有规定显式的结束调用，至于如何处理并行程序的结束则是根据实现不同。

2.4.3　执行模型

因为 OpenSHMEM 和 MPI 的调用形式类似，编程方式类似，实际上 OpenSHMEM 程序的执行顺序和 MPI 也十分类似。不同之处在于以下两点：

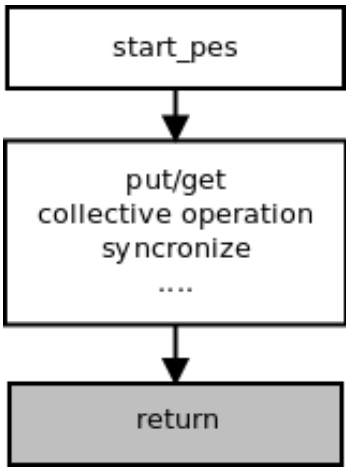

图 2.8 OpenSHMEM 编程模型

1. MPI 中的数据传输例程需要双端通信，需要收发两方参与，所以在并行部分，Send 和 Recv 总是成对出现，而 OpenSHMEM 使用的单端读写方式，仅需要主动的一方调用 put 或者 get, 无需等待对方的相应;

2. MPI 程序的最后需要显式的调用 Finalize，作为对并行环境的后续处理，而 OpenSHMEM 则没有这样的调用。

2.5 OpenSHMEM 的内容和特性

在 OpenSHMEM 说明文档[2] 中，说明了现有版本的 OpenSHMEM 提供的所有 API。总的说来，可以分为以下 6 类:

1. 数据传输操作

2. 原子内存访问操作

3. 集体操作

4. 同步操作

5. 锁操作

6. 环境参数 API

从以上可以看出，OpenSHMEM 的操作基本是围绕数据进行的，这是因为 OpenSHMEM 库基本上是针对科学计算等这样以数据为基础的应用设计的，因此 OpenSHMEM 规定的调用功能范围相对 MPI 更为精确，可用的调用仅有不到 50 条，相对于 MPI 这样庞大的体积，OpenSHMEM 则更为灵活小巧。

另外，和 MPI、OpenMP 以及 Pthreads 等其他现在主流的编程方法比较，OpenSHMEM 强调了以下这样两个特性。

2.5.1　PGAS 模型

从前文中对于 MPI 和 OpenMP 的介绍中可以看出，MPI 基于的硬件模型是分布式存储体系，而 OpenMP 采用的是共享内存体系，在实际应用中，实际的硬件模型往往是两者的结合，因此在并行计算发展过程中，出现了将 MPI 和 OpenMP 混合使用的架构，并且取得了不错的效果[22]。从 PGAS 模型看来，PGAS 模型正是这两种模型的折中，在保留每个核可以唯一访问的私有内存的同时允许共享内存的存在，可以通过调用接口访问远端的内存，也可以以这样的方式通信。实际上，这种模型可能是最接近实际应用的硬件体系抽象，基于该模型的内存模型和通信方式都更直观。

具体到 OpenSHMEM 中，在2.4.1小节中可以看到，每个 PE 都有自己可以访问的私有内存空间，需要通过调用 API 获取远端共享内存的数据内容，这点和 MPI 十分类似；而每个 PE 都可以在任意时刻访问共享内存，无需等待其他 PE 的应答或者允许，也就是说 PE 可以通过共享内存快速的通信，这点和共享内存模型也十分契合。

2.5.2　单端通信模型

单端通信模型并非在 OpenSHMEM 中首次提出。在 MPI 成为最广泛应用的编程方法的同时，MPI 的双端通信的一些弊端也被讨论和改进。因为 MPI 的每次通信都必须通过类似握手的机制才能完成，在通信未建立时，发送方只能等待，这种同步带来的时间消耗在数据量十分大的情况下可以看作比较小的消耗，但是对于数据量较小但是发送较频繁的情况，这种同步消耗称为比较明显的需要优化的点，即如何将数据传输和同步解耦合。

在 OpenSHMEM 的介绍中，可以看到 OpenSHMEM 的数据通信方式都是单端的，一个 PE 的操作无需等待另一个 PE 的应答就可以单方面的完成。OpenSHMEM 的基本操作是远程内存访问（RMA），提供的基本方法 remote put/get 也是每个支持 RMA 的编程方法都具有的。RMA 可以算作单端通信的一个例子。要完成 RMA，一定是单端的，因为无法预期在何时会有远程访问，因此除了不断轮询这种耗时的方法外，很难将 RMA 实现为双端的。

在并行计算发展的很长一段时间里，网络硬件的发展十分缓慢，并行体系在设计时需要考虑对网络的兼容和在有限带宽下提高性能。然而，在过去的几年中，网络通信硬件的飞速发展为底层通信带来了几个新的概念，比如零拷贝(Zero-Copy),OS bypass[7] 等等。一个著名的新一代网络硬件的例子是 Infiniband，在 Infiniband 中，硬件集成了对 RDMA 的支持，这代表了新一代网络发展的方向，然而在 MPI 和 OpenMP 中，并没有表现出对这种新特性的支持。因此，可以认为，对于单端通信模型的支持是 OpenSHMEM 的一大亮点，对于 RDMA 以及尤其对小数据量友好的单端通信模型是近年来 PGAS 模型语言重新获得关注的主要原因。

第三章　并行程序设计

3.1　并行程序设计基础

并行算法是适合在并行机上实现，遵从某一种编程模型，实现将任务或数据分块执行处理的算法，一个好的并行算法应该具备发挥问题解本身和并行机并行处理潜能的能力。

并行计算机的出现是来源于问题域本身存在的内在并行度这一事实，因此，在设计并行程序的时候，充分的发掘问题域本身的可并行部分是并行计算机提高性能的关键，而并行算法作为程序开发的基础，自然也有举足轻重的地位[24]。

而根据 Amdahl 定律[18], 在并行算法提出和实现的早期，如何识别和并行化可并行化部分是程序实现的主要工作，而这样的性能提升实际上十分有限，进一步的优化需要仔细分析问题域增大可并行部分比例，有时甚至需要完全修改算法结构。

设计并行程序时，需要考虑以下四个方面的问题。首先，必须分析和识别出计算过程中可以并行的部分；其次，需要选择一种分解策略，将步骤或者数据进行分解从而可以跑在多个进程上；然后，需要具体的实现设计好的程序，实现时需要确定编程方法和实现接口；最后，根据以上的分析和设计的程序选择实现风格和语言。

在本节，仅围绕适合 OpenSHMEM 实现的方式，研究探讨相关的并行技术和原则。

如上讨论，设计实现并行程序首先要确定，哪些部分是需要并行的。通过回答一个基本问题：在这个程序中，有哪些部分我们是可以分别同时执行的, 即可以找到最明显的可并行部分，另外，我们还需要保证经过并行的程序在每次运行都能产生和串行程序同样的结果。一般，我们所指” 并行运行” 往往并不严格要求“同时”，而是以一种异步的形式运行，所以在并行运行的结束处需要一次同步约束。所以，总的说来，并行版本的程序会在运行时产生一批可以并行执行的进程或者线程，分别执行不同的计算，在运行的结尾处同步结果。

第二个需要考虑的问题是，如果已经找到了需要并行的部分，又该如何划分程序。一般来说，划分策略可以分为两类。

第一类，称为任务并行，该名称在不同的写作和翻译过程中有多种叫法，但说的都是同样的事情：哪些指令可以同时互不影响的执行。比如说，一个进程可以负责处理输入数据，另外一个进程则可以执行进程 0 之前处理过的数据，这种执行模型十分常见，广泛应用在如服务器等大型多任务并发的情况中。

第二类，称为数据并行。即将问题的数据域分割成几块分别由不同的处理器进行处理。比如一个二维的 1000×1000 的方阵可以切割成若干个 100×100 的子

阵，分布在 10×10 的处理器上分别进行运算。如果该处理是绝对并行的，理论上可以加速 100 倍。在科学计算中，数据并行的形式更为常见。

任务并行和数据并行通常是结合出现的，更为通用的方式是采用流水线化的方式进行处理，让计算机的各个部分同时保持运转状态，在理想情况下，流水线化的方式会让吞吐量提高到流水线的深度倍数[23]。

3.2 并行算法实现示例

根据以上对于并行程序设计的一些介绍，以下通过一个比较简单的矩阵乘法程序介绍并行程序的开发，使用 OpenSHMEM 实现。

在科学与工程计算中，矩阵运算占据着相当重要的地位，其中，矩阵相乘、求解线性方程组和矩阵特征值问题是矩阵运算的核心。在许多用于高性能科学计算的计算机上，都配备着高效的运算库。以下，我们考虑的问题是

$$C = A \times B \tag{3-1}$$

假设在式 3-1中，A 是$m \times q$ 的矩阵，B 是$q \times n$ 的矩阵，结果 C 是$m \times n$ 的矩阵。

3.2.1 矩阵乘法串行版本

矩阵乘法的串行是数学意义上的矩阵相乘过程的直接实现。其伪代码形式如下所示。

串行算法

```
function multiple_matrics_serial(A, B)          ▷ 矩阵 A，B 相乘的串行版本
    INIT(C)
    for i ← 1, m do
        for j ← 1, n do
5:          for k ← 1, q do
                C[i][j]+ = A[i][k] × B[k][j]        ▷ 基本的乘加操作
            end for
        end for
    end for
10:  return C
end function
```

可以看出，矩阵乘法的串行版本的时间复杂度为 $O(n^3)$。同时，我们也可以从中看出串行的矩阵乘法存在着明显的可以优化的过程/数据并行部分。

3.2.2 简单的矩阵乘法并行算法

subsection 在展示并行版本的矩阵乘法算法之前，首先对之前的串行版本做一分析。在3.2.1中，可以看出，算法的 3 ～ 9 步存在着 for 语句，并且每一层 for 循环都不存在数据依赖关系，很明显这是可以并行的部分。

表 3.1 $A_{M,Q}$ 按行分割为两部分

$$\left(\begin{array}{cccc} a_{1,1} & a_{1,2} & \cdots & a_{1,q} \\ \vdots & \cdots & \cdots & \vdots \\ a_{\frac{m}{2},1} & a_{\frac{m}{2},2} & \cdots & a_{\frac{m}{2},q} \\ \hline a_{\frac{m}{2}+1,1} & a_{\frac{m}{2}+1,2} & \cdots & a_{\frac{m}{2}+1,q} \\ \vdots & \cdots & \cdots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,4} \end{array}\right)$$

表 3.2 $A_{M,Q}$ 按列分割为两部分

$$\left(\begin{array}{ccc|ccc} a_{1,1} & \cdots & a_{1,\frac{q}{2}} & a_{1,\frac{q}{2}+1} & \cdots & a_{1,q} \\ a_{2,1} & \cdots & a_{2,\frac{q}{2}} & a_{2,\frac{q}{2}+1} & \cdots & a_{2,q} \\ & \vdots & & & \vdots & \\ a_{m,1} & \cdots & a_{m,\frac{q}{2}} & a_{m,\frac{q}{2}+1} & \cdots & a_{m,q} \end{array}\right)$$

在3.2.1中，最核心的基本执行语句是乘 -加操作，并且不存在其他可以和乘 -加操作的同时执行的操作，因此，矩阵乘法是一个典型的可以数据并行的计算过程。

下一步，需要确定如何将数据，即输入矩阵进行划分。

对于矩阵的划分存在几种方式[24]。最直观的是对输入矩阵进行行列划分，然后分配给每个 PE。如下两个矩阵分别表示将矩阵行分割和列分割。在本节中，先实现一个较为简单的并行方式，将矩阵 A 按照行分割，同时矩阵 B 在多个进程中共享，最后由第一个进程负责输出结果。伪代码显示如下。

行分割并行版本

function multiple_matrics_parallel(A, B)　　▷ 矩阵 A，B 相乘的简单并行版本
  $INIT(C)$
  for each PE **in** PE_Set **do**　　▷ 每个 PE 仅需要处理 Block 中部分 A 矩阵
    $Block \leftarrow \frac{m}{q} \times rank$ **to** $\frac{m}{q} \times rank + 1$
    for each row **in** Block **do**
      $C[indexof(row)][*] \leftarrow multiple\_vector\_matrix(row, B)$
    end for
  end for
  return C
end function

伪代码忽略了进程间数据通信的部分。在 MPI 的实现中，需要在计算开始前显式的分发数据，一般地为了减少通信开销，在程序运行的时候，矩阵数据并不存在同一个处理器的内存中，而是分布在各个处理器中，处理器只对本地的数据进行计算，最后将结果发给某个处理器进行统一的显示。

表 3.3 $A_{M,Q}$ 棋盘分割

$$\left(\begin{array}{ccc|ccc} a_{1,1} & \cdots & a_{1,\frac{q}{2}} & a_{1,\frac{q}{2}+1} & \cdots & a_{1,q} \\ & \vdots & & & \vdots & \\ a_{\frac{m}{2},1} & \cdots & a_{\frac{m}{2},\frac{q}{2}} & a_{\frac{m}{2},\frac{q}{2}+1} & \cdots & a_{\frac{m}{2},q} \\ \hline a_{\frac{m}{2}+1,1} & \cdots & a_{\frac{m}{2}+1,\frac{q}{2}} & a_{\frac{m}{2}+1,\frac{q}{2}+1} & \cdots & a_{\frac{m}{2}+1,q} \\ & \vdots & & & \vdots & \\ a_{m,1} & \cdots & a_{m,\frac{q}{2}} & a_{m,\frac{q}{2}+1} & \cdots & a_{m,q} \end{array}\right)$$

然而在 OpenSHMEM 中，由于 Symmetric Memory 的概念，这部分数据如果是全局的，则对所有 PE 都可见，省略了分发数据的通信开销。至于数据具体存在多份备份还是一份备份，则与具体实现有关。关于此处实现，使用虚拟内存映射以及写时拷贝技术 (Copy-On-Write,COW) 可以有效的节约内存，减少通信开销[12]。

3.2.3 矩阵乘法的 Connon 算法

另外，还存在一种行列分割结合的棋盘式分割的方式，如下所示。在3.2.2中，讨论并且实现了对矩阵 A 行分割的计算并行化，并且给出了具体的实现，Connon 算法采用的是如表3.3的棋盘分割方式，并且在计算过程中需要不断的在处理单元之间传递数据，因此，展示和实现 Connon 算法对于区分 MPI 和 OpenSHMEM 通信模型和优缺点，效果较3.2.2更为明显。

基于表3.3,Connon 算法的伪代码描述如下。

矩阵乘法的 Cannon 算法

function Cannon(A, B) ▷ 计算矩阵 A，B 相乘的 Cannon 算法

divide_and_distribute(A,B,PE_set) ▷ 将 A，B 矩阵分割分发到 PE_set 中的 PE 上

end function

3.3 并行程序优化度量

并行编程的难点或许在于并行化似乎并不能直接等于性能提升，因此，为了充分利用机器的潜能，就需要在设计时充分考虑多个可能影响程序运行性能的因素。

首先，就像使用时空复杂度度量算法性能，并行程序的性能也需要量化方法度量性能提升。在比较全面的性能理论中[25], 性能可以是以下几种度量：

- 减少运行一次计算所需要的时间，即减少延迟；

- 增加可以同时计算出结果的比例，即增大吞吐量；
- 减少一次计算的耗能。

出于简化的目的，本节仅讨论减少时间消耗带来的性能提升。

两个用于评价性能和并行化的度量分别为加速比 (Speedup) 和有效性 (Efficiency)。加速比用于评价同样的计算运行在单核平台上和对核平台上的延迟消耗：

$$speedup = S_P = \frac{T_1}{T_P} \tag{3-2}$$

在式3-2中，T_1 表示运行在单核平台上所需的完成时间，T_P 表示使用 P 个进程的情况下所需的完成时间。

有效性则是加速比除以参与运算的进程数量：

$$efficiency = \frac{S_P}{P} = \frac{T_1}{PT_P} \tag{3-3}$$

第四章　OpenSHMEM 在 Tilera 平台上的实现框架

4.1　Tilera Gx8036 平台简介

Tilera 公司是位于硅谷的新创无晶圆半导体公司，因为在多核技术方面拥有独家的先进技术，该公司曾被美国知名媒体 EETIMES 评为全球最有希望的 60 家新兴企业之一。2009 年 11 月，Tilera 公司发布了令其公司名声大震的众核平台 GX 系列，其中包括拥有 100 个核心的 TILE-Gx100，同时也有本文用于搭建 OpenSHMEM 简单框架的 Gx8036 平台。Tile-Gx 系列采用 40nm 处理器制造工艺，CPU 时钟频率为 1.2GHz。Tile-Gx 系列尤其注重对功耗的把握，在运行一个典型的网络应用的情景下，该平台的运行功耗约为 25 ～ 30W，考虑到该平台极高的集成度，这个功耗值约为同等高性能处理器的 1/2 到 1/4。在这个平台上，36 个通用核中的每一个都是相同的 64 位处理器，每个处理器都带有私有本地缓存，并且实现了一个较为复杂的虚拟内存系统。每个核上都可以单独的跑一个自己的操作系统，也可以多个核成组运行一个 SMP 操作系统，本文的实现环境就是 Tielra 提供的 SMP Linux。

TILE-Gx36 的硬件架构如图 4.1 所示。由图中可以看出，TILE-Gx36 平台提供了相当丰富的外设接口资源，尤其针对对于网络流的处理做了硬件优化。但是在本次实现中，暂时不考虑通过外部网络级联两个或以上板卡扩展性能的情况，仅在一个平台内进行实验。TILE-Gx36 平台中每个核是通过片上高速网络 iMesh 相连接。根据[1] 关于高速片上网络拓扑结构的比较，Mesh 或者 Mesh-like 网络结构是现有可靠的、高速的、低功耗的片上网络结构的主要实现，并且呈现越来越广泛的使用趋势。在 TILE-Gx36 平台上，使用的是 Tilera 公司的 iMesh 技术。iMesh 是基于 Mesh 网络结构的高速片上互联网络的高带宽、低延迟的实现。iMesh 网络互联的 tile 可以通过网络快速的访问其他 tile 的缓存、外部主存以及 I/O。并且，在 TILE-Gx36 上实现的分布式共享一致性缓存架构，解决了网络访问的瓶颈问题，并且最小化了功耗。在 TILE-Gx8036 中，提供了如图 4.2的 5 个动态网络 (Dynamic Network, DN)，分别连接相邻的两个 tile，提供快速的数据、指令交换策略。这 5 个网络连接中，对于用户层可用的为 UDN(User Dynamic Network) 以及 IDN(Instruct Dynamic Network)，由于 UDN 的表现以及性能对于 OpenSHMEM 的实现比较重要，将在下文以及平台测试部分再次详细讨论。

另一方面，在前面的讨论中可知，影响多核平台编程以及性能表现的另外一个因素即为内存子系统。在 TILE-Gx36 平台中，每个处理单元拥有私有的片上 cache 系统以及 memory controller，但是需要和其他 tile 共享一个内存系统。这样的内存系统结构可以同时提供对共享内存模型以及消息传递模型的较好的硬件支持。

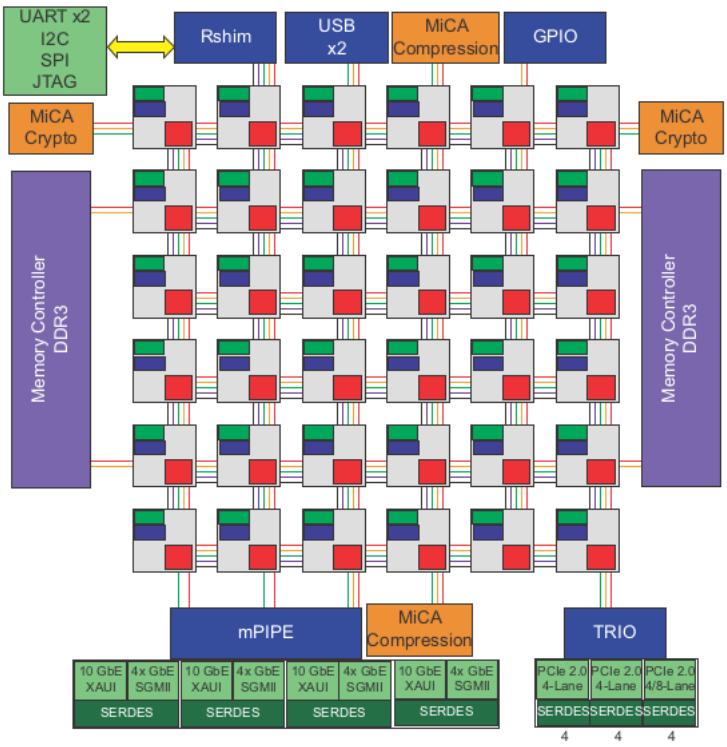

图 4.1 TILE-Gx36 平台示意图

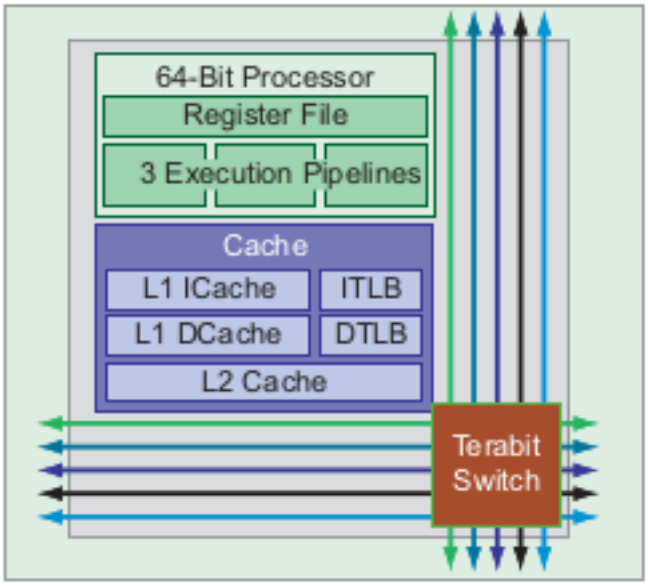

图 4.2 TILE-Gx36 中 Tile 结构以及片上网络示意图

4.1.1　Tile-Gx36 平台的片上网络

4.1.2　Tile-Gx36 平台的内存体系

4.2　平台性能测试设计

4.2.1　Common Memory 的性能测试

4.2.2　Spin 与 Barrier 同步原语的延迟性能测试

4.2.3　UDN 用户空间中断性能测试

4.2.4　UDN 性能测试

4.3　OpenSHMEM 实现设计

4.3.1　环境建立

4.3.2　put 与 get

4.3.3　集体操作

参考文献

[1] Eduard Fernandez-Alonso, David Castells-Rufas, Jaume Joven and Jordi Carrabina *Survey of NoC and Programming Models Proposals for MPSOC* IJCSI International Journal of Computer Science Issues, 2012, March, Vol.9. Issue 2, No.3

[2] OpenSHMEM Specification v1.0 Final http://openshmem.org/.

[3] SHMEM API for parallel programming http://www.shmem.org/.

[4] Introduction to MPI http://www.citutor.org

[5] Introduction to Multi-core Performance http://www.citutor.org

[6] Open MPI document v1.6.4 http://www.open-mpi.org/doc/

[7] Scott Atchley and David Dillow and Galen Shipman and Patrick Geoffray and Jeffrey M. Squyres and George Bosilca and Ronald Minnich The Common Communication Interface (CCI) 19th Annual IEEE Symposium on High-Performance Interconnects (2011.08)

[8] Keith D.Underwood, Michael J.Levenhagen, Ron Brightwell Evaluating NIC Hardware Requirements to Achieve High Message Rate PGAS Support on Multi-Core Processors ACM, SC07, 2007, November. pp. 10-16

[9] Henry Kasim, Verdi March, Rita Zhang, and Simon See Survey on Parallel Programming Model IFIP International Federation for Information Processing,(2008) . pp. 266-175

[10] Christian Bell, Dan Donachea, Rajesh Nishtala, Ketherine Yelick Optimizing Bandwidth Limited Problems Using One-Sided Communication and Overlap IPDPS 2006. 20th International , (2006, April). pp.10 pp. 25-29

[11] Changil Yoon, Vikas Aggarval, Vrishali Hajare, Alan D.George, Max Billingsley III GSHMEM: A Portable Library for Lightweight Shared-Memory Parallel Programming *PGAS'11*, (2011)

[12] Ron Brightwell, Kevin Pedretti An Itra-Node Implementation of OpenSHMEM Using Virtual Address Space Mapping *PGAS '11*, (2011)

[13] Olivier Serres, Ahman Anbar, Saumil Merchant and Tarek El-Ghazawi Experiences with UPC on TILE-64 Processor IEEEAero paper1609, Version 2, (2010, December)

[14] Bryant C. Lam, Alan D. George, Herman Lam TSHMEM: Shared-Memory Parallel Computing on Tilera Many-Core Processors *PGAS' 12*, (2012)

[15] John L.Hennessy, David A.Patterson Computer Architecture Fifth Edition USA: Morgan Kaufmann Publishers, Elsevier (2012). pp. 344-426

[16] Barbara Chapman, Gabriele Jost, Ruud van der Pas Using OpenMP, Portable Shared Memory Parallel Programming USA: Massachusetts Institute of Technology Publish (2007). pp. 1-124

[17] 陈国良 并行算法的设计与分析 北京：高等教育出版社 2002.11

[18] Peter Pacheco An introduction to Parallel computing USA: Morgan Kaufmann Publishers, Elsevier (2011)

[19] David Culler,Jaswinder Pal Singh, Anoop Gupta USA: Morgan Kaufmann Publishers, Elsevier (1997)

[20] Stephen W. Poole, Oscar Hernandez, Jeffery A. Kuehn, Galen M. Shipman, Anthony Curtis, and Karl Feind. Encyclopedia of Parallel Computing, OpenSHMEM - Toward a Unified RMA Model US: Springer, (2011).pp. 1379-1391

[21] David Padua, etc. Encyclopedia of Parallel Computing US: Springer (2011). pp. 1539-1544, pp.1812

[22] Rabenseifner R, Hager G, Jost G. Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes[C] Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on. IEEE, (2009). pp. 427-436.

[23] Dongarra J J, Foster I, Fox G, et al. Sourcebook of parallel computing[M]. San Francisco: Morgan Kaufmann Publishers, (2003)

[24] 张林波, 并行计算导论 [M], 清华大学出版社有限公司, 2006. pp.278-285

[25] McCool M, Reinders J, Robison A. Structured Parallel Programming: Patterns for Efficient Computation[M]. Morgan Kaufmann, 2012. pp.54-67