软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(4):764−785 [doi: 10.13328/j.cnki.jos.005197]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向国产异构众核系统的 Parallel C 语言设计与实现*

何王全, 刘 勇, 方燕飞, 魏 迪, 漆锋滨

(江南计算技术研究所,江苏 无锡 214083)

通讯作者: 何王全, E-mail: wangquan_he@163.com

摘 要: 异构众核架构具有超高的性能功耗比,已成为超级计算机体系结构的重要发展方向.但众核系统更为复杂的并行层次和存储层次,给编程和优化带来了极大的挑战.因此,研究面向众核系统的并行编程技术,对于降低国产众核系统并行应用的编程难度、提升并行程序的性能都具有重要的意义.提出统一架构的多模式并行编程模型,包括异构融合的加速运算模型和按同构方式编程的自主运算模型,根据编程模型设计了 Parallel C 语言,能够有效地描述国产众核系统的异构并行性.与其他众核系统上 MPI+X 的使用模式相比,编程和系统优化都具有全局视角,在多级局部性描述、单边消息、兼容已有多核应用等方面具有特色;基于 Open64 构建了 Parallel C 编译系统,全面支持加速运算模型和自主运算模型,提出并实现了数据布局与自动 DMA、编译指导的线程代理和拓扑位置感知的集合通信等优化.Micro Benchmark 和实际应用在神威太湖之光计算机系统上的测试数据结果表明:Parallel C 语言和编译系统具有良好的性能和可扩展性,能够有效支撑大型应用.

关键词: 异构众核;编程模型;并行语言;Parallel C;编译器;消息传递

中图法分类号: TP314

中文引用格式: 何王全,刘勇,方燕飞,魏迪,漆锋滨.面向国产异构众核系统的 Parallel C 语言设计与实现.软件学报,2017,28(4):764−785. http://www.jos.org.cn/1000-9825/5197.htm

英文引用格式: He WQ, Liu Y, Fang YF, Wei D, Qi FB. Design and implementation of Parallel C programming language for domestic heterogeneous many-core systems. Ruan Jian Xue Bao/Journal of Software, 2017,28(4):764−785 (in Chinese). http://www.jos.org. cn/1000-9825/5197.htm

Design and Implementation of Parallel C Programming Language for Domestic Heterogeneous Many-Core Systems

HE Wang-Quan, LIU Yong, FANG Yan-Fei, WEI Di, QI Feng-Bin

(Jiangnan Institute of Computing Technology, Wuxi 214083, China)

Abstract: Heterogeneous many-core architecture, with ultra-high performance to power consumption ratio, has become an important trend of supercomputer architecture development. However, many-core systems always have more complex parallel hierarchy and memory hierarchy, hence posing a great challenge to programming and optimization. Therefore, the study of many-core-oriented parallel programming techniques is of great significance, since it can reduce the difficulty of parallel programming on domestic many-core systems and improve the performance of parallel programs. This work proposes a multi-model parallel programming model upon unified architecture, including heterogeneous-fused speedup programming model and isomorphic independent programming model. Based on this model, Parallel C programming language is designed to effectively describe heterogeneous parallelism of the domestic many-core system.

* 基金项目: 国家重点基础研究发展计划(973)(2016YFB0200502); 国家高技术研究发展计划(863)(2012AA010903, 2015AA01A301); 计算机体系结构国家重点实验室基金(CARCH201403)

Foundation item: National Basic Research Program of China (973) (2016YFB0200502); National High Technology Research and Development Program of China (863)(2012AA010903, 2015AA01A301); Fund Projects of State Key Laboratory of Computer Architecture (CARCH201403)

收稿时间: 2016-06-20; 修改时间: 2016-09-08; 采用时间: 2016-11-26; jos 在线出版时间: 2017-01-24

CNKI 网络优先出版: 2017-02-20 13:43:29, http://www.cnki.net/kcms/detail/11.2560.TP.20170220.1343.004.html

Compared to MPI+X programming pattern, programming with Parallel C has a global perspective, as well as advantages in the hierarchy locality description, one-side message passing and multi-core applications compatibility. The Parallel C compiler system constructed with Open64 fully supports the heterogeneous-fused speedup programming model and isomorphic independent programming model. In addition, the design and implementation of data layout and automatic DMA optimization, compiler-directed thread proxy optimization and topology-aware collective communications optimization are presented. The performance of the proposed method is evaluated with the Miro Benchmark and practical applications on Sunway Taihu Light computer system. Experimental results show that Parallel C language and the compile system have good performance and scalability to effectively support large-scale applications.

Key words: heterogeneous many-core; programming model; parallel language; Parallel C; compiler; message passing

众核处理器具有计算能力强、性能功耗比高等突出优点,异构众核架构已成为当前超级计算机体系结构的重要发展方向.2015 年 11 月,全球 Top 500[1]上榜系统中,众核系统已经接近 20%;而排名前 10 的系统中,有 4 台采用众核处理器构建.已公开发布的 E 级计算计划[2−4],无一例外地都将采用众核处理器.

然而,由于拥有更多的计算核心、更复杂的并行层次和存储层次,众核系统的存储墙、通信墙和编程墙等问题比多核系统更为突出,给编程、调试和优化带来了更大的困难.从目前来看,众核系统的可编程性问题仍然是高性能计算领域尚未很好解决的挑战性难题.因此,研究面向众核系统的并行编程技术、设计与系统架构相适应的编程模型和并行语言、开发配套的并行编译系统,对于降低众核系统应用的开发难度、提升并行程序的性能都具有重要的意义.

本文提出的面向国产众核系统的 Parallel C 语言,正是针对众核并行编程挑战的一次有益的尝试.第 1 节介绍领域相关研究进展.第 2 节介绍国产异构众核系统.第 3 节介绍面向国产异构众核系统的并行编程模型.第 4 节介绍 Parallel C 语言的设计思想和主要内容.第 5 节介绍 Parallel C 编译系统的设计与实现.第 6 节介绍测试结果.最后是结束语.

1 相关研究

现有众核系统上的并行编程环境,通常采用 MPI[5]+X 的模式,其中,X 包括 OpenMP[6]、PGAS 语言[7]、OpenACC[8]、HMPP[9]、CUDA[10]等,它们具有各自的优势,已应用于多种众核系统.

MPI 是当前分布存储系统上并行编程事实上的工业标准,它基于消息传递模型,能够很好地描述粗粒度并行,但不适合描述众核节点内细粒度并行,在众核系统上需要与其他语言配合使用.最新的 MPI 3.0 标准增加了更完备的单边消息通信模型、非阻塞的集合操作以及消息日志等支持.面对未来 E 级计算的挑战,MPI 将重点研究混合编程模型、可扩展性设计(特别是存储可扩展)、异构计算等技术.

PGAS 维护共享数据的全局视图,编程方便,一些研究机构和大学开始研究 PGAS 在异构系统上的实现,在大规模系统上利用 MPI+PGAS 嵌套进行并行程序设计.PGAS 语言 UPC[11]和 X10[12]扩展了对 Work-Stealing 的任务并行模式支持,弥补了 PGAS 模型在应对动态任务调度方面的不足.

异构编程模型大多是面向节点内加速器设计的,支持对众核内存储层次和并行层次的描述.作为传统的共享存储编程模型,OpenMP 4.0 增加了针对加速器(主要面向 MIC 处理器)的编译指示.CUDA,HMPP,OpenACC,ParaC[13]等异构模型支持将核心计算映射到加速器(主要面向 GPU)上执行.目前,CPU 和加速器的存储空间是独立的,异构模型需要描述 CPU 和加速器间的数据交互,不同异构平台存在可移植性问题,MIC 平台和 GPU 平台的应用就难以互相兼容.

除了进一步发展现有编程模型外,国际学术界和工业界针对未来 E 级系统的并行编程问题开展了许多研究工作.IESP 计划[2]将编程模型如何解决并行性问题列为 X-stack 的五大关键要求之一,并明确了编程模型和语言的关键研究点和路线图;BDEC 计划[14]在融合 E 级计算和大数据编程模型和语言方面开展了预先研究;Intel、微软与加州大学伯克利分校、UIUC 联合建立了 ParLab[15]和 UPCRC[16]等,这些研究工作取得了一系列进展.从总体上看,国际上对面向 E 级的并行编程模型和语言设计,认为有两种可能的途径:一种是走 MPI+X 的改进演化道路,另一种是设计全新的编程模型和语言.在语言设计方面,需要重点考虑:(1) 要能够描述足够多的并行性来

开发硬件数百万到 10 亿量级规模处理器核心的并发度,能表示多种类型、不同层次的并行性;(2) 要能有效支持系统复杂的存储结构、局部性描述和高效的数据移动.

2 国产异构众核系统

面向高性能计算的众核处理器包括 Intel 的 MIC[17]、Nvidia 和 AMD 的 GPU[18,19]、Godson-T[20]、申威众核处理器等.申威众核处理器的结构如图 1 所示[21],每颗处理器包含 4 个 core-groups(CGs),每个 CG 包含 1 个 MPE(management processing element,简称主核)、1 个 8×8 的 computing processing element(CPE,简称从核) cluster 和 1 个 memory controller(MC),4 个 CG 通过片上网络(NoC)互连,处理器通过 System interface(SI)连接外部设备.申威众核处理器的主核和从核共享 memory,从核采用轻量级的核心设计,配备由软件管理的高速存储器 SPM(scratchpad memory)[22],支持通过 DMA(direct memory access)方式在 memory 和 SPM 间批量传输数据.

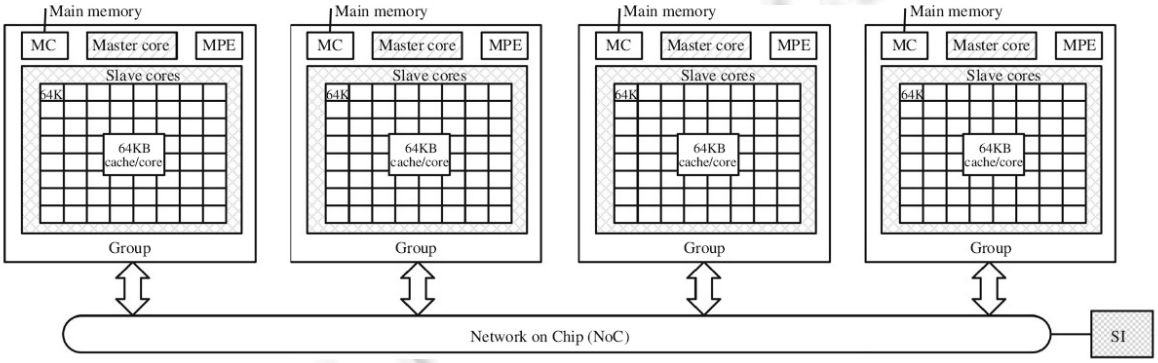


Fig.1 Architecture of the new Sunway processor

图 1 申威众核处理器结构

基于申威众核处理器的异构众核系统架构如图 2 所示,运算系统采用申威众核处理器构建,通过中心交换网络和管理网络与存储系统和管理系统连接,系统的登陆界面和存储空间采用单一映像组织,为用户提供统一的视图.

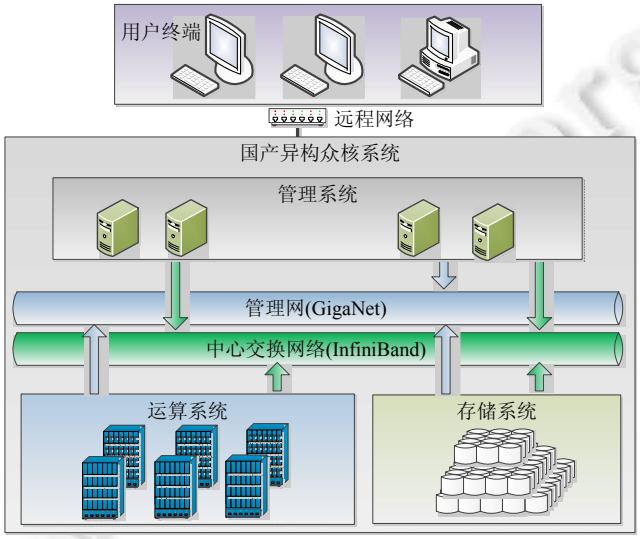


Fig.2 Architecture of domestic heterogenous many-core system

图 2 异构众核系统结构

3 面向异构众核系统的并行编程模型

众核系统的显著特征是计算资源众多、并行层次比多核系统更为复杂,编程和优化的难度也比多核系统大,需要在并行编程模型和语言设计上有所突破,满足不同层次人员的编程需要.针对申威众核处理器异构并行、共享主存的架构特征,我们提出了多模式并行编程模型,包括异构融合的加速运算模型和按同构方式编程的自主运算模型,并且在此基础上设计实现了统一架构的并行编程语言 Parallel C.

异构融合的加速运算模型如图 3 所示,该模型有如下特点.

1) Parallel C 的进程(类似 MPI 进程)运行于众核处理器的主核,负责管理控制、通信、I/O 等复杂的操作,大量的加速线程运行于众核处理器的从核,负责加速核心计算代码;
2) 进程和加速线程共享 memory,与 x86+GPU 或 x86+MIC 上的编程模型相比,无需在两种存储器之间通过 PCI-E 或其他接口进行数据传输,编程方便、效率也更高;
3) 在节点内,可以通过共享扩展描述,方便地实现共享编程,提高节点内数据交换的效率;
4) 在进程之间,通过消息进行数据交换,并支持多级局部性描述,使 Parallel C 可高效扩充到大规模环境.

该模型支持消息、共享和加速运算的任意组合,可有效描述异构系统多维度的并行,解决 MPI+X 模型要求用户掌握 2~3 种编程语言的问题.

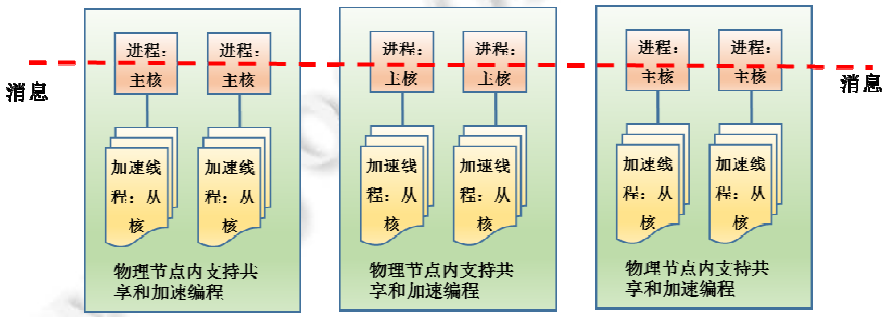


Fig.3 Accelerated computing model

图 3 异构融合的加速运算模型

设计按同构方式编程的自主运算模型有两个目的.

- 一是让程序员以熟悉的多核方式编程,最大限度地屏蔽异构众核系统的复杂性;
- 二是兼容 Parallel C 在多核环境下的遗产代码.

自主运算模型如图 4 所示,程序员只需按传统多核方式进行编程,由编译系统将同构、单维度并行描述映射到异构、多维度并行的体系结构,Parallel C 进程运行于从核上,通信、I/O、系统调用等复杂的功能则由主核在后台处理,对程序员屏蔽异构系统复杂的结构;原有多核环境下的 Parallel C 遗产代码可以不用修改,平滑地移植到众核环境下.

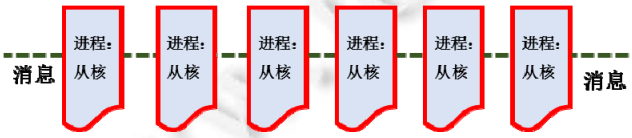


Fig.4 Autonomic computing model

图 4 自主运算模型示意图

4 面向异构众核系统的 Parallel C 语言

通过分析研究并行应用的特征,基于 ANSI C,采用扩展面向众核系统并行描述的方式设计了 Parallel C 语

言,该语言全面支持加速运算模型和自主运算模型.Parallel C 支持加速运算模型的功能包括两部分:一部分是进程相关扩展,运行于多核或众核处理器的主核上;另一部分加速线程相关扩展,运行于众核处理器的从核上.Parallel C 自主运算模型的功能是加速运算模型的子集,即进程相关部分的扩展.

Parallel C 加速运算模型的执行模型如图 5 所示,作业启动后,若干进程以 SPMD 的方式并行执行,进程执行过程中,可以根据需要创建加速线程对核心计算代码进行加速.Parallel C 自主运算模型用户可见的执行模型仅为图 5 的进程部分,但在编译系统实现时,会隐式地翻译为加速模型.

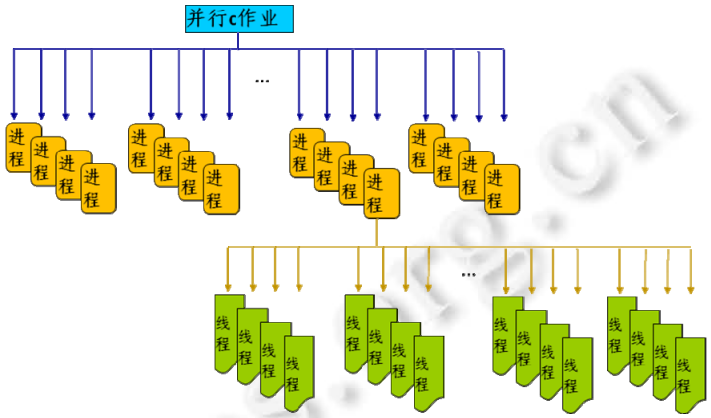


Fig.5 Execution model of Parallel C accelerated computing model

图 5 Parallel C 加速运算模型的执行模型

Parallel C 的存储层次如图 6 所示.在进程一级,进程有自己的私有空间,在物理节点内的进程,还可以在进程间共享内存空间(大节点共享空间和节点共享空间,具体描述见第 4.3 节);在线程一级,每个加速线程除了私有空间外(具体描述见第 4.5.2 节),与 pthread,OpenMP 等类似,线程还可以共享访问进程的空间(多线程共享).

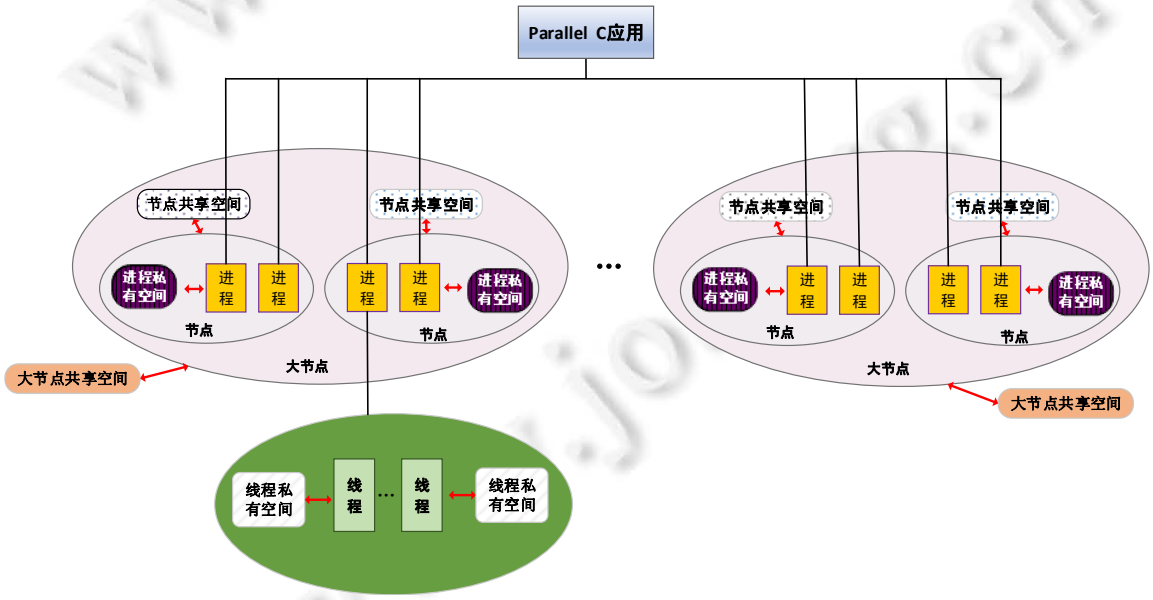


Fig.6 Memory hierarchy of Parallel C

图 6 Parallel C 的存储层次

Parallel C 语言支持局部共享、多层次的循环并行划分、动态任务调度、单边消息,支持消息、共享和众核加速的任意组合,可提高国产众核系统的可编程性和并行程序开发与运行的效率.Parallel C 语言的功能比较多,限于篇幅,本节仅对 Parallel C 有特色的内容进行介绍.

4.1 局部并行域

在大规模环境中,挖掘各级局部性对并行程序的性能至关重要.MPI 默认只有全局并行域,对应的通信子为 *MPI_COMM_WORLD*,在进行局部控制时,需要程序员调用相关的接口创建新的通信子来进行控制.分析典型应用的特征可以发现:除最为常用的全局并行域(标识为 *CCC_ALL*)外,很多应用还会在局部范围内进行并行控制.Parallel C 根据这些需求预置了多种范围的局部并行域,其标识既可用于 forall、动态任务调度框架等并行描述(见第 4.3 节),也可用于同步、锁和集合通信(在做这些操作时,相当于通信子).

- 超节点并行域:通常由若干个物理节点组成,大小可以根据应用需要设置.在大规模系统中,可以方便地利用网络局部通信速度强的优势开发通信局部性,标识为 *CCC_SNODE*;
- 大节点并行域:范围在一个物理计算节点内,标识为 *CCC_BNODE*;
- 节点并行域:范围在一个物理计算节点内,标识为 *CCC_NODE*,大小是 *CCC_BNODE* 的真因子.比如 *CCC_BNODE* 域的大小是 16 进程,*CCC_NODE* 可以根据局部性控制的需要设置为 2,4,8 进程;
- 加速线程域:范围为一个进程创建的所有加速线程构成的并行域,标识为 *CCC_ARRAY*.

通常,不同应用关注的局部性可能不同:有的应用重点关注节点间通信的局部性;有的应用则按节点编程,重点关注节点内的局部性;有的应用关注加速线程的局部性.在进行应用的算法设计和并行程序的开发过程中,应根据需要选择适合的局部并行域.在同一个程序中,为了开发多级局部性,可以嵌套使用局部并行域,通常使用不同层次的同步、锁和集合通信机制来保证程序的正确性.

4.2 局部共享扩展

为了在物理节点内的多个处理器核心间共享数据,传统的做法通常有两种.

- 一种是显式调用 mmap 申请空间或用 pthread 机制进行控制,使用上不是很方便;
- 另一种是用 OpenMP 进行控制,性能依赖编译系统,如果要实现多级共享,则需要使用高级的嵌套并行功能,对用户的要求比较高,很多 OpenMP 编译器也不支持.

Parallel C 以关键字 *node_shared* 和 *bnode_shared* 的形式扩展了节点共享和大节点共享描述机制,扩展的语法为

node_shared TYPE var; *bnode_shared TYPE var*.

这种显式的描述方式容易理解,使用上也非常方便,辅以配套的同步和锁机制(各自均采用统一的接口,同步或锁的控制范围由局部并行域 *CCC_BNODE* 和 *CCC_NODE* 作为实参给出),可以在物理节点内灵活实现多级共享.例如:由 2 个 CPU 构成的 *CC-NUMA* 节点,每个 CPU 有 4 个核心,编程时可以将进程间交换最为频繁的 *A* 数组定义为 *node_shared* 类型,指定 4 进程共享;而空间要求大、交换相对不频繁的 *B* 数组定义为 *bnode_shared*,指定 8 进程共享,这样可以更好地利用节点内 NUMA 结构的局部性,如图 7 所示.程序员可以根据应用的需要,使用一种局部共享,或者是两种局部共享的组合,*node_shared*,*bnode_shared* 的范围根据需要以运行参数的形式进行灵活设置.

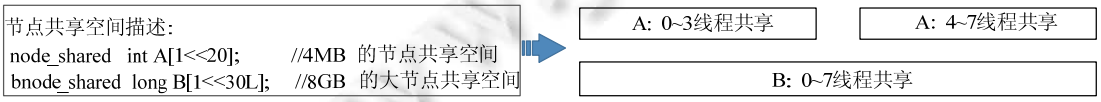


Fig.7 Description of shared node and big shared node, space partition

图 7 节点共享、大节点共享描述和空间分布示意图

4.3 并行任务描述

为方便描述并行任务,Parallel C 提供了两种并行任务描述机制:forall 并行循环和动态任务调度框架.

4.3.1 forall 并行循环

for 循环在 C 语言中使用频度非常高,通常,for 循环也是进行并行迭代划分的基本单位.OpenMP 扩展了 for 并行循环编译指示,UPC 扩展了 forall 语法.Parallel C 采用与 UPC forall 循环相似的风格,但比 UPC 更为灵活.

1) UPC 的 for 循环只能在整个作业的全局范围内进行并行划分,Parallel C 比 UPC 的 forall 语法多设置了并行域,根据需要,可以在全局范围内进行并行划分,也可以在局部范围内进行并行划分,能够灵活地描述不同层次的并行;
2) 亲缘性表达式与 UPC 相比增加了 ANY,编译器有更大的自由度进行迭代划分.

```
forall (expr; expr; expr; affinity; scope)
        statement;
affininty:
    BLOCK: 所有迭代平均分配给 scope 域内的所有进程执行
    BLOCK(n): n 次迭代为一个块,采用以块为单位循环分配给 scope 域内的所有进程执行
    整数表达式:令 r = affininty % scope 域内的进程总数,由 scope 域内相对进程号为 r 的进程执行相应的迭代
    ANY: 由编译器在 scope 域内的进程间根据优化需要自由划分
scope:
    scope:并行域,取值为 Parallel C 的并行域,如 CCC_ALL,CCC_NODE,CCC_ARRAY 等.
```

forall 在进程和加速线程中使用时风格一致,forall 循环在进程代码中使用,则在该并行循环内在进程间进行并行迭代划分,否则就在加速线程间进行并行迭代划分.

4.3.2 动态任务调度框架

任务并行类应用广泛存在于药物筛选、基因研究、信息安全、核模拟等领域,多数应用的子任务之间没有相关性,但子任务的计算量可能存在显著差异.在大规模环境下,如何保证这类应用良好的负载平衡,是应用性能的关键.目前,在很多应用中,并行任务的动态调度由程序员使用消息接口编写,程序员的负担比较重,多数使用简单的 Master-Slave 模型,在大规模环境下可扩展性差,要高效实现动态任务调度对普通程序员来说极具挑战性.为此,Paralell C 提出了一种动态任务调度并行编程框架,适合子任务间无相关性的应用.

```
/*并行任务调度编程框架*/
while ((task_id=get_task_id(任务总量,检查点文件,并行域))≥0)
{
        do_job(task_id);   /*用户代码,根据 task_id 完成相应的任务*/
}
```

该编程框架以 *get_task_id*(·)原语的形式提供给程序员,*get_task_id*(·)返回代表任务的编号或结束标志(小于 0 表示任务分配结束),检查点文件记录已完成的任务,并行域指出动态任务调度的范围.该框架可以在进程中调用,也可以在加速线程中调用,实现了接口的统一.

4.4 消息传递接口

Parallel C 设计了接口简洁、方便使用的消息传递接口,包括双边消息、单边消息和集合操作等.

- 双边消息和集合操作

Parallel C 双边消息和集合操作与 MPI 相关功能类似,接口参数比 MPI 略简单.双边消息包括阻塞的发送/接收、非阻塞的发送接收通信接口,支持跨步操作;集合操作包括同步、规约、全规约、广播、全交换等接口.

- 精简的单边通信机制

单边消息在处理不规则问题的通信方面具有优势,MPI 从 2.0 开始支持单边消息,但使用比较复杂,对程序员的要求较高.Infiniband 的 RDMA 可以直接发送数据,但主动方和被动方需要通过查询完成消息队列来确认

消息完成,与 Infiniband 底层硬件实现密切相关,程序员直接使用十分困难.针对这些问题,Parallel C 面向大规模系统高性能计算应用的需求,设计了精简、高效的单边 put 和 get 通信机制,为程序员提供容易理解和使用的简洁单边消息接口,同时能够获得良好的通信效率.Paralell C 的单边 put 和 get 如图 8 所示,其中,修改远程回答字不是必须的动作,根据程序员需要可以省去.

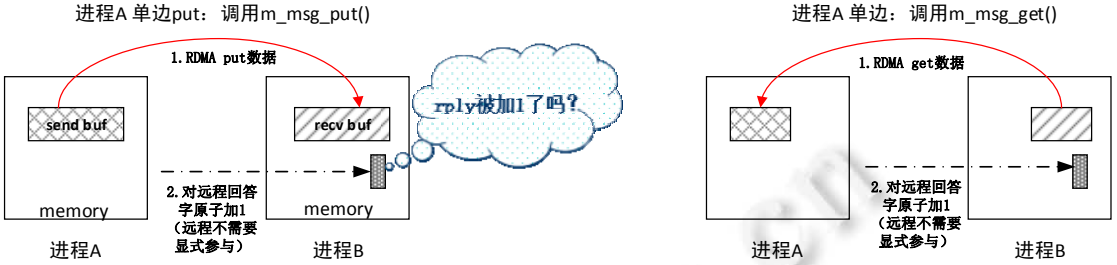


Fig.8 One-Side put and get operation in Parallel C

图 8 Parallel C 单边 put 和 get 示意图

Parallel C 支持阻塞、非阻塞的单边消息,分不带跨步和跨步操作两种,一共 4 种接口,下面以最常用的不带跨步、非阻塞单边 put 为例加以说明.

接口描述:

*handle_t m_msg_put(int dest_rank,void*source_addr,void*dest_addr,long size,int*dest_rply);*

参数说明:

int dest_rank; //put 操作的被动方 rank(远程将收到数据的进程)

*void*source_addr*; //单边消息操作数据的源地址,对 get 来说,是远程数据源的首地址;对 put 来说,是本地将要发送数据的首地址

*void*dest_addr*; //单边消息操作数据的目的地址,对 get 来说,是本地接收数据的首地址;对 put 来说,是远程接收数据的首地址

long size; //消息长度(字节数)

*int*dest_rply*; //put 的远程回答字地址,put 完成后,将被动方 dest_rply 的内容加 1,被动方以此来判断 put 数据是否到达,*n* 个消息可以使用同一个回答字,*dest_prly* 的内容等于 *n* 时,表明 *n* 个消息都已完成

返回值:

消息句柄,主动方以此调用 *m_wait_msg*(·),*m_test_msg*(·)来判断消息是否出错、检测或等待消息完成.

为支持单边消息,Parallel C 还提供了相关的辅助函数,包括根据消息句柄等待主动方消息完成、测试消息完成等.

Parallel C 代码中,通常将某个进程计算得到的结果调用 *m_msg_put*(·)直接发送给对方进程,不要求对方进程显式参与.而被动方缓冲区是否准备好,一般由上一次同步保证,被动方检查数据是否到达只需要看 rply 指定地址的内容是否等于期望收取的消息个数.而需要使用 *m_msg_get*(·)取远程数据时,被动方通常在上一次同步前就将数据准备好,主动方直接调用 *m_msg_get*(·)取远程数据,被动方不需要参与,也不需要判断回答字,使用非常方便,效率也更高.

4.5 加速线程相关描述

4.5.1 加速线程创建和回收

Parallel C 进程执行中可以创建加速线程,将 kernel(核心计算代码)加载到从核阵列中并行执行.Parallel C 支持两种加速线程创建和回收方式,如图 9 所示.

- 一种是同步方式,进程创建加速线程后,同步等待加速线程执行完毕,再继续执行后续代码,这种方式编程简单,但不利于开发进程与加速线程的并行性;
- 另一种是异步方式,进程创建加速线程后,继续执行与加速线程无关的代码,比如消息、I/O 操作等,或者

为下一次 kernel 的执行准备数据,这样可以开发进程与线程的异步并行性.

```
//同步方式
#pragma ccc sync call
kernel_1();  //加速kernel函数


//异步方式
#pragma ccc async call
kernel_2();  //加速kernel函数
…  //进程代码，与加速线程并行执行
#pragma ccc wait  //等待线程结束
```

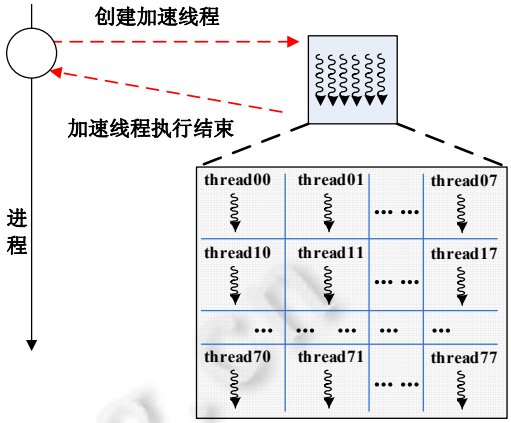


Fig.9 Spawn and join of accelerate threads

图 9 加速线程创建和回收示意图

4.5.2 加速线程私有空间描述和批量数据传输

每个线程可以描述主存中的私有空间,Parallel C 采用兼容 GNU 标准的*__thread* 关键字描述.

SPM 与 Cache 结构相比具有硬件开销小、效率高的特点,已在众核处理器中使用广泛,GPU、Tilera 和申威众核处理器均采用了片上 SPM 结构.Paralell C 扩展了关键字*__thread_local*,显式地描述 SPM.另外,Parallel C 还支持动态申请和释放 SPM 空间,编译系统也会在对程序进行分析的基础上,自动地使用剩余的 SPM 空间进行程序优化.

```
语法格式:__thread_local  TYPE  var;
示例:     __thread_local  int  A[1024];
```

Memory 的访问特性决定了批量访问的效率最高,在 cache 结构的处理器中,cache 脱靶或淘汰时,总是以 cache 行为粒度进行操作,众核处理器也需要以内存行为单位进行批量操作才能充分发挥访存带宽.为了提高应用效率,Parallel C 设计了 SPM 和 memory 间数据批量传输的 DMA 接口,包括单线程 DMA 和广播 DMA,支持同步和异步操作,异步 DMA 可以开发计算和访存的并行性.

5 Parallel C 编译系统的设计与实现

5.1 编译系统设计思想和架构

5.1.1 设计思想

根据 Parallel C 语言文本,我们以优秀的开源编译器 Open64[23]为基础,设计了面向国产众核系统的 Parallel C 并行编译系统.设计思想为继承和利用 Open64 编译器成熟的分析和优化能力,在此基础上扩展针对众核系统结构特征的优化,并自主开发针对国产众核系统的高效运行时库,达到编译生成高质量 Parallel C 可执行目标代码、充分挖掘众核系统的性能的目的.

5.1.2 编译系统架构

与传统的并行编译系统类似,Parallel C 编译系统由编译器和运行时库两大部分组成,如图 10 所示.

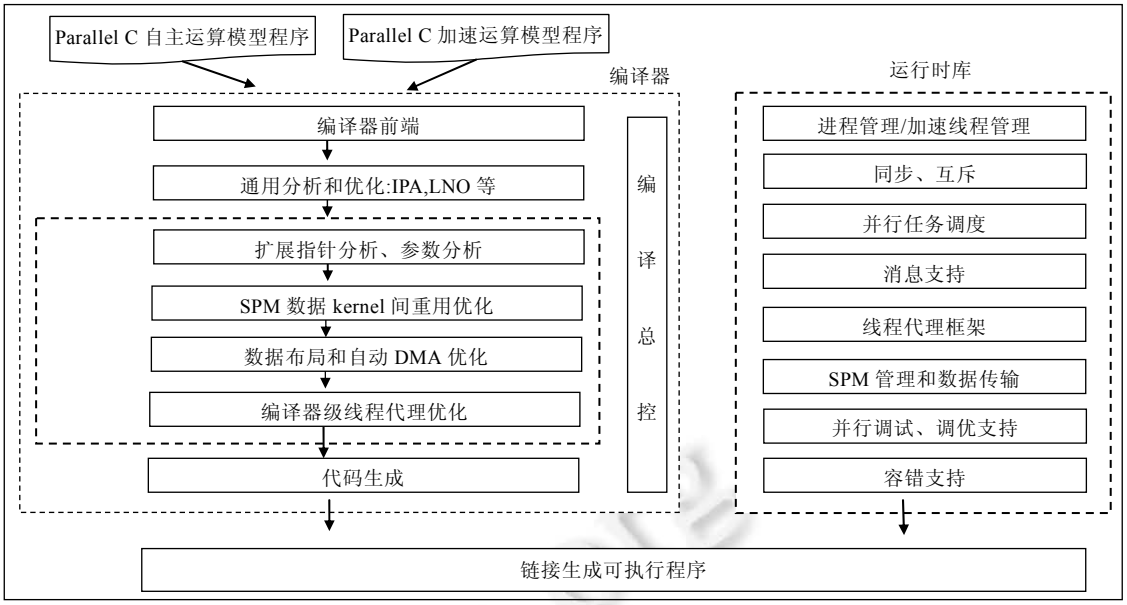


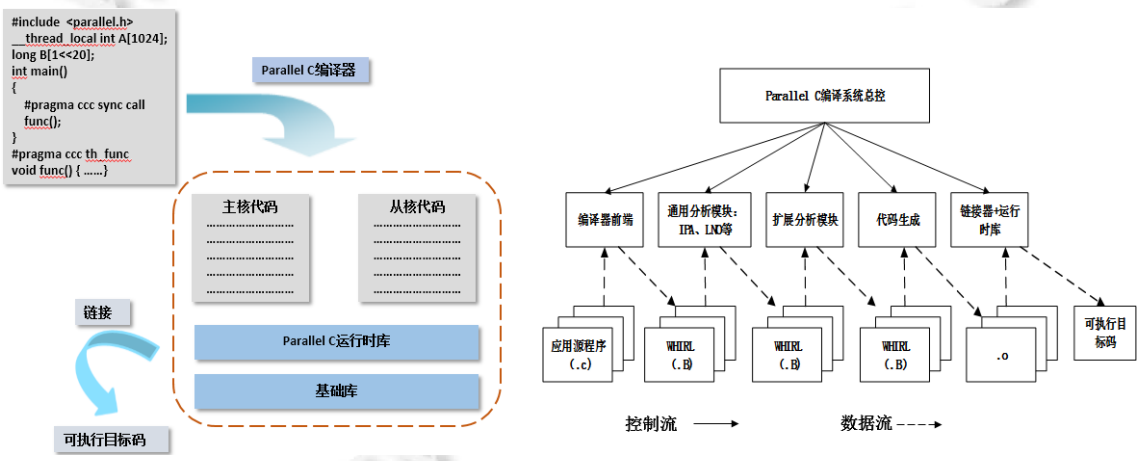
Fig.10 Architectrue of Parallel C compile system

图 10 Parallel C 编译系统架构

- 编译器

Parallel C 编译器采用模块化设计,由编译总控、编译器前端、通用分析和优化、扩展分析和优化、代码生成等几部分构成.

编译总控是编译器的控制程序,它接收 Parallel C 程序,根据不同的编译选项调用编译器前端、通用分析和优化模块、扩展分析和优化模块、代码生成模块,生成主核代码和从核代码,并调用链接器链接 Parallel C 运行时库和底层基础函数库生成可以在众核系统上运行的目标码,Parallel C 程序的编译过程如图 11(a)所示.编译器的不同模块通过 Open64 的公共中间表示 WHIRL 进行交互,编译过程中的控制流和数据流如图 11(b)所示.



(a) 编译过程 (b) 编译中的控制流和数据流

Fig.11 Compile process of Parallel C

图 11 Parallel C 编译示意图

编译器前端采用 GCC 源码实现,处理 Parallel C 关键字和编译指示扩展,通过词法、语法语义分析,翻译到 WHIRL 中间表示.

通用分析优化模块来自 Open64,基于中间表示,进行与目标平台无关的通用分析和优化,主要包括 IPA 和 LNO,完成依赖关系分析、过程间信息收集和 SSA 优化、循环嵌套优化等,分析优化的结果可以为后续的扩展优化提供支持.

扩展分析和优化是 Parallel C 编译器最重要的部分.该模块针对众核系统的特征,在 Open64 的基础上增加了相应的分析和优化,其中,扩展指针和参数分析基于 IPA 的结果和 Def-Use 信息,确定 kernel 中的指针和函数形参可能的来源,为 kernel 重用和数据布局提供支撑;SPM 数据 kernel 间的重用优化可以将 kernel 间不同时使用的全局 SPM 数据映射到相同空间,以提高 SPM 的利用率;数据布局和自动 DMA 优化模块通过建立收益评估模型,将程序访问最频繁且小于局存容量的变量布局在 SPM 中,同时为大数组阶段性设置 SPM 缓存,并通过自动 DMA 分批传送需要的数据,提高数据访问效率.编译器级线程代理优化模块分析加速线程需要由主核代理执行的语句,通过合并、将部分线程代码迁移主核的方法,实现多个操作的聚合,并减少线程代码复杂的解析,提高代理的效率.

- 运行时库

Parallel C 运行时库提供进程/加速线程的管理、同步和互斥机制、并行任务调度、消息支持、线程代理框架、SPM 管理与传输等功能,对并行程序的调试、调优和协同容错提供支撑.运行时库是 Parallel C 在众核系统上高效运行的基础,为了挖掘系统性能,我们全部采用自主设计,在保证模块化的基础上,尽量减少复杂的调用层次,以获得更高的执行效率.

Parallel C 编译系统涉及的内容比较多,第 5.2 节~第 5.6 节将选择有 Parallel C 特色和众核实现与优化相关的部分内容进行介绍.

5.2 数据布局与自动DMA

众核处理器的存储墙问题更为突出,充分利用片上 SPM 存储空间和 DMA 批量传输支持,是应用程序挖掘众核处理器性能潜力的关键.即使是由程序员显式地进行 SPM 描述和控制,也难免存在遗漏,因此,非常有必要在 Parallel C 编译器中实现数据布局到 SPM 和自动 DMA 优化,以提高应用效益.另外,Parallel C 自主运算模型没有 SPM 描述和批量传输支持,必须依靠编译器实现的数据布局和自动 DMA 优化来提升应用性能.

5.2.1 基于访存收益评估的数据布局模型

数据布局期望将利用率最高,也就是最有价值的数据静态或动态布局到 SPM 中,达到降低访存开销的目的.设计数据布局模型的思想是尽量减少访问 memory 次数、提高数据布局收益,即“变量直接访问 memory 开销-利用 SPM 和 DMA 后的访问开销”最大化.显然,选择数据布局的对象非常关键.假设变量 V_i 的大小(bytes)为 $S(V_i)$,设定 SPM 的大小为 C_m,将变量分为两类情况讨论.

- $S(V_i)<C_m$

这类变量能够整体布局到 SPM 中,但是未必是最优选择.根据该变量在程序中的访问特点,分下面两种情况分别加以讨论.

a) 程序对 V_i 的访问不具有空间局部性,只能整体布局到 SPM,定义为 I 型变量,需要的 SPM 空间为 $S(V_i)$;

b) 程序对 V_i 的访问具有空间局部性,既能将其整个数组布局到 SPM,也可以通过分块缓冲+DMA 进行访存优化,定义为 II 型变量.V_i 整体布局需要的 SPM 空间为 $S(V_i)$;通过缓冲方式,V_i 需要 SPM 缓冲空间的大小用 $S_k(V_i)$表示,k 表示若干种缓冲 size 选择中可能的一种.

- $S(V_i)>C_m$

这类变量无法整体布局在 SPM 中,若对 V_i 的访问不存在空间局部性,则放弃对该变量的 SPM 布局考虑;否则,可以考虑在 SPM 中为其申请一块缓冲空间,然后利用 DMA 将数据分阶段导入导出 SPM 缓冲进行优化.定义这类 V_i 为 III 型变量.V_i 需要 SPM 缓冲空间的大小用 $S_k(V_i)$表示,k 表示若干种缓冲 size 选择中可能的一种.

通过上述分析,建立基于访存收益评估的数据布局模型,目标收益函数.

$$\max\left\{\sum_i x_i \times F(V_i) + \sum_j \sum_{k=1}^{K_j} y_{j,k} \times F_k(V_j)\right\}.$$

约束条件为

$$\sum_{i\in I} x_i \times S(V_i) + \sum_{j\in J} \sum_{k}^{K_j} y_{j,k} \times S_k(V_j) \leqslant C_m,$$

其中,

- I 是 I 型变量的标号集合,J 是 II 型、III 型变量的标号集合,显然有 $I\cap J=\varnothing$;
- $F(V)$是变量 V 布局在 SPM 中(整体布局或分配局部缓冲)带来的访存收益;
- x_i 和 $y_{j,k}$ 都是 0-1 变量,用来指示变量利用 SPM 优化的情况.变量 V_i 布局到 SPM,则 x_i=1;否则,x_i=0.变量 V_j 的缓冲布局到 SPM 中,则 $\sum_{k=1}^{K_j} y_{j,k}=1$; 否则,$\sum_{k=1}^{K_j} y_{j,k}=0$; K_j 表示变量 V_j 缓冲大小可以选择的个数.

数据布局优化的目标就是,求解使得最大收益目标函数的解$\{x_i,y_{j,k}\}$.

5.2.2 数据布局与自动 DMA 在编译器中的实现

数组布局与自动 DMA 在编译器中的实现如图 12 所示,依次进行全局指针分析和参数分析、提取数组布局优化评估对象、收集整理对象数组访问信息、数组访问模式分析/关联分析、访存收益评估、参数化分块方法的模型求解、程序变换等步骤.

Fig.12 Data layout and automatic DMA

图 12 数据布局与自动 DMA 实现流程

变量在程序中的访问方式可以分为以下几种:离散读、离散写、连续读、连续写、区间读、区间写、单个读、单个写.Parallel C 编译器对不同的访问可以进行不同方式的优化.为了保证程序的正确性,充分重用 DMA 缓冲空间,需要对数组访问进行依赖关系分析和关联分析,包括冲突分析、构建同类项、各种访问模式的相关性分析等.在依赖和关联分析后,针对程序的控制结构、循环的参数以及存储结构特点,给出几个效果较好的缓冲大小参数,传递给后面的数据布局优化模型进行选择.

编译器根据求解复杂度情况采用穷尽法、first-fit 和启发式快求解法求解优化模型.根据数据布局优化模型的解算结果生成程序的优化代码,主要包括:(1) 指示哪些变量整体布局到 SPM,并更改这些变量的存储属性;(2) 为 II 型、III 型变量申请 SPM 缓冲区;(3) 对循环依据数据布局的参数进行分块;(4) 在程序中合适的位置插入主存与 SPM 间的数据批量传输代码,并对数据访问点进行相应变换.

5.3 线程代理框架和编译指导的代理优化

5.3.1 线程代理框架

Parallel C 支持在轻量级的核心上使用动态任务调度框架、I/O 操作、系统调用等,使用自主运算模型还需

要在轻量级核心上支持 CPU 间的消息.由于从核上不运行完整功能的 OS,为了支持这些功能,我们设计了线程代理框架,轻量级从核上难以直接实现的功能由主核与从核协同完成.线程代理框架如图 13 所示,运行于从核上的加速线程执行到 I/O、跨 CPU 的消息、动态任务请求、系统调用时,进入代理请求引擎,对需要代理的内容进行解析,发送基础请求到主核,并等待主核的响应,有的复杂操作如 printf 根据实际参数的个数,可能会发送多次基础请求;主核的主进程或轮询线程收到代理请求时,启动代理处理引擎,解析请求内容,完成线程期望执行的动作,并将结果返回给从核的加速线程.

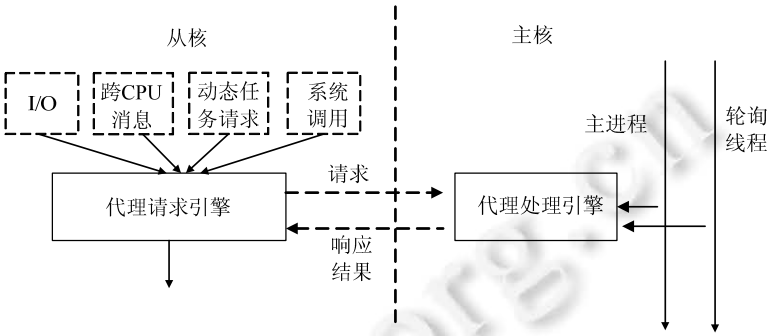


Fig.13 Framework of thread proxy in Parallel C

图 13 Parallel C 线程代理框架

该框架的思想能够用于其他类似的功能扩展,比如 socket 操作甚至一个完整的 Web 服务器,也可以用于在其他平台上的实现.我们基于同样的框架在 GPU 平台上扩充实现了 CUDA 在设备端的 I/O 操作[24],取得了很好的效果.

5.3.2 编译指导的 I/O 代理优化方法

I/O 是加速线程中经常使用的操作,线程中的代理请求引擎直接解析和处理相关操作的代价比较大,例如 printf 函数,由于 C 库的代码非常庞大,往往会引起指令 cache 的颠簸,多个线程同时指令脱靶对性能的影响很大;当 printf 有多个参数时,会发送多次代理请求,影响处理的效率.为此,我们提出了编译指导的 I/O 代理优化方法,如图 14 所示.该方法利用编译器的分析能力,完成两个方面的优化:一是将复杂的线程处理代码迁移到主核,减少线程指令 cache 颠簸的压力;二是对多个 I/O 操作进行聚合优化,对代理优化所需的数据进行打包,提高交互效率.

```
//运行于从核上加速线程代码
printf("Thread_id=%d,a[0]=%d\n",tid,a[0]);
printf("Thread_id=%d,Test OK!\n",tid);
```

```
// I/O 迁移后的线程轻量级代码
wn_addr = 0x2af76cc5b350;   //识别本次代理的标识
cccr_io_pack(&wn_addr,8);
cccr_io_pack(&tid,4);
cccr_io_pack(&a[0],4)
cccr_io_pack(&tid,4);
cccr_mail(·);
cccr_wait_io_buf_available(·);
```

```
switch(type_value)   //由编译优化迁移到进程(主核)代码
{
    case 0x2af76cc5b350:
        recv_buf=(int*)&cccr_io_buf;
        printf("Thread_id=%d,a[0]=%d\n",recv_buf[0],recv_buf[1]);
        printf("Thread_id=%d,Test OK!\n",recv_buf[2]);
        cccr_send_io_buf_available(·);
        break;
    default:
        break;
}
```

Fig.14 Compiler-Directed I/O agent optimization

图 14 编译指导的 I/O 代理优化

5.4 消息的实现和优化

Parallel C 针对国产众核系统的计算网络,实现了语言的所有消息机制.其中,双边消息与 MVAPICH[25]实现机制类似,根据消息的粒度,采用 Eager,Rendezvous 协议;单边消息实现和针对大规模众核系统的集合通信优化具有一定的特色,下面对其思想和实现机制进行简要阐述.

5.4.1 单边消息实现

- 基于 RDMA 和 send/recv 的单边消息协议

如何利用底层网络的特征降低单边消息的延迟、充分发挥网络的带宽,并且能够适应多种主流网络,是设计 Parallel C 单边消息协议遵循的重要原则.

Infiniband 在高性能计算系统中占主流位置,既支持 send/recv 队列消息,也支持 RDMA 消息.队列消息需要通信双方都参与,而 RDMA 具有 CPU 占用率低、使用灵活的优点,常用于上层通信协议的优化,以太网也开始支持 RDMA.RDMA 与 Parallel C 的单边消息比较接近,可以用于实现单边消息数据的传输,但 RDMA 无法满足 Parallel C 回答字原子加 1 的需要,需要有 Active Message 的机制.考虑 Infiniband 底层通信原语的功能及性能特点,Parallel C 设计了基于 RDMA 和 send/recv 队列消息相结合的单边消息协议.

以 Parallel C 单边 put 为例,处理流程如图 15 所示.当消息长度小于消息阈值 L_m 时(≤send/recv 包长),通过 IB 底层提供的 send/recv 请求,直接将控制包头和数据打包并一次性发送到被动方接收缓冲;被动方在 poll 到主动方的消息请求后,根据消息类型解析协议各字段,将数据复制到用户目的地址,并将回答字内容原子加 1.短消息协议下,主动方和被动方只有一次异步通信过程,减少了额外的握手操作,能够有效保证消息的延迟性能.当消息长度大于消息阈值 L_m 时,将控制包头和数据内容分开,首先通过 RDMA 写操作将数据内容直接发送至被动方目的空间,再通过 send/recv 发送控制包头,被动方解析控制包头后将回答字内容原子加 1.RDMA 写操作实现了数据内容传输过程零拷贝,保证了长消息的高带宽.

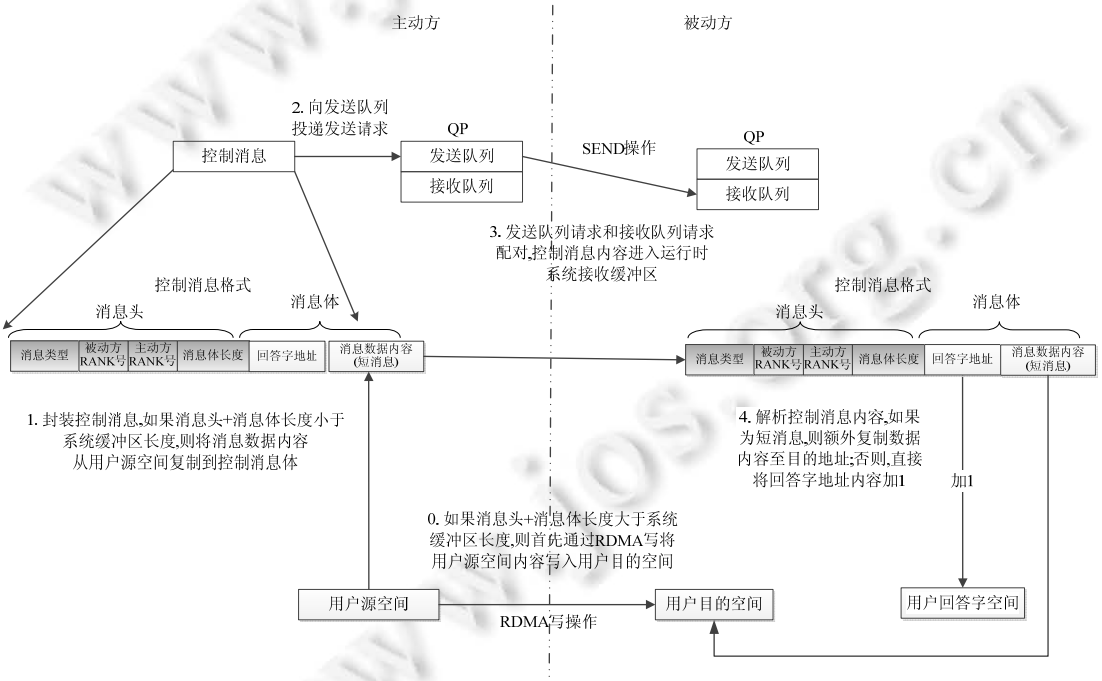


Fig.15 Process of one-side put operation of Parallel C

图 15 Parallel C 单边 put 处理流程

- 远程消息的响应机制

Parallel C 单边消息的被动方完全可以不显式参与,若消息的被动方一直在进行计算,如果没有相应的远程消息请求处理机制,则单边消息就会饿死,因此需要 Active Message 实现远程请求的隐式处理.通常有中断和轮询两种方式可实现需要的功能,中断方式处理简单,但中断的开销比较大,导致消息延迟比较长,因此,Parallel C 采用轮询方式.

Parallel C 运行时库在初始化时,创建一个轮询线程(用于处理单边消息、加速线程代理、动态任务分配等请求),定期查询是否有未处理的远程请求,消息轮询机制如图 16 所示.为了提高消息性能,Parallel C 的进程在处理消息时,也会顺带探查(poll)消息队列中是否有远程请求达到且还未被响应.消息的处理必须保证原子性,因此,轮询线程和进程探查消息队列时需要互斥机制.通过进程主动探查和轮询线程的结合,高效地实现了 Parallel C 的单边消息机制.

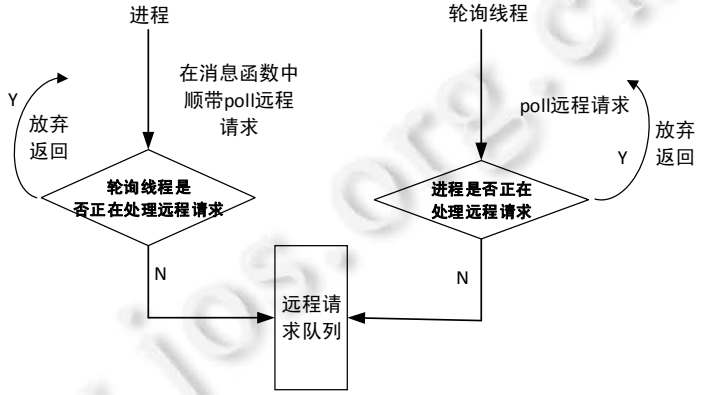


Fig.16 Remote messages polling mechanism of Parallel C

图 16 Parallel C 远程消息轮询处理机制

5.4.2 拓扑位置感知的集合通信优化

大规模系统的顶层网络一般都有裁剪,并行程序会映射到不同的物理节点,这些节点拓扑互连不同,通信带宽存在差异,发生路由冲突的情况也不一样.因此,有必要感知并行程序映射物理节点的拓扑位置,并据此采用合理的通信策略,避免路由冲突,提高通信效率.尤其是对通信带宽敏感的课题,这一点显得尤为重要.以 alltoall 通信为例,针对大规模环境建立了传统 ring、2 层 ring、simple spread、pair-wise、聚合行列交换等通信策略算法,根据物理节点的拓扑形状和网络路由算法,识别当前作业使用的物理节点资源之间通信耦合的松紧强度,选择合适的通信算法.

而以节点为基本并行单位的应用,集合通信可以感知并利用节点内的高速数据交换支持,获得比通过网络传输更好的通信性能.在多核系统上,以共享内存进行通信优化是成熟的方法.在申威众核处理器上,除了主核可以使用共享内存进行消息优化外,还可以利用空闲的从核资源进行节点内的交换,获得更好的效率.图 17 所示为物理节点内 alltoall 优化的示意图.

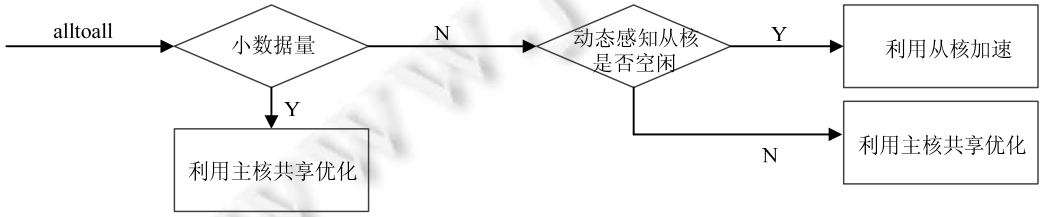


Fig.17 Alltoall optimization intra node

图 17 物理节点内 alltoall 优化

5.5 面向大规模众核系统的并行任务调度策略

Parallel C 的动态任务调度框架是支持任务并行类应用编程和高效运行的有效手段,在大规模环境下,需要采用分层策略来提高效率.众核系统的核心数量非常大,比多核系统至少多了一级并行,高效的动态任务调度实现面临着挑战,对于子任务间执行时间很不平衡的应用,采用 MPI+X 两级并行的方式,kernel 的结束是一次同步过程,因此,如果采用一次循环多次调用 kernel 且每次给 kernel 分配的任务量少的方法,kernel 内慢的线程会拖后腿,类似组团旅游总要先集合再到下一个景点,总会有等待;如果采用一次给 kernel 分配大量的任务,则可能将大量慢的任务分配给少数节点,会在进程之间造成很大的不平衡,慢的进程会大幅拖后腿,大量进程最后会长时间等待少数慢的进程,在多核系统上,可以尝试用 work-stealing 来缓解,但在轻量级的从核上,实现 work-stealing 难以取得很好的效果.

Parallel C 的动态任务调度框架支持在加速线程内直接向全局主进程申请任务,程序员调用一条语句就可以实现,不需要显式地采用两级并行编程,避免了 kernel 内的线程 join 时的等待,所有加速线程都相当于在旅游中采用个人游的方式,游完当前景点就能马上自主动身去下一个景点.Parallel C 的动态任务调度框架的实现如图 18 所示,运行时系统按照动态任务调度框架中的角色,从上到下分为全局主进程、区域主进程(可选)、普通进程和加速线程,下层向上层提交任务申请、上报任务完成,上层向下层提供任务分配服务.运行时库采用了局部任务池预取策略,当出现本地任务池任务不够时,提前向上层提出任务申请进行适当的预取,以降低任务分配的延迟.

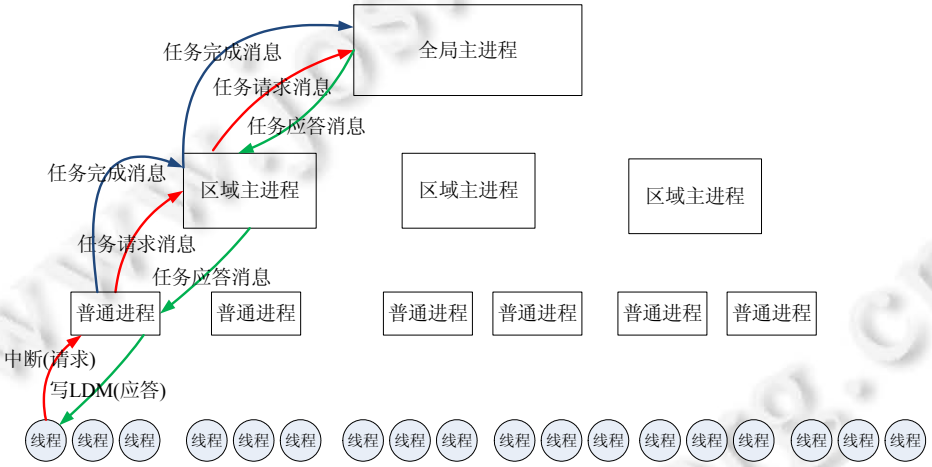


Fig.18 Task scheduling strategy

图 18 任务调度策略

很多 MPI 应用固定使用某些进程来做任务分配,不进行计算,浪费了计算资源.Parallel C 运行时库有消息轮询线程的支持,动态任务调度框架中的每个进程和线程均参与运算,最大限度地利用了计算资源.

5.6 自主运算模型的实现

设计自主运算模型的目的是简化编程和兼容已有的多核 Parallel C 程序,在程序员的视角中只有进程的概念,编译系统需要隐式地将该类程序自动地转换为加速运算模型的程序.

如下面的例子所示,编译器将程序员的 main 函数变换为 user_main,并作为唯一的 kernel 函数进行加载;全局的私有数据首先调整为线程私有类型,由于自主运算模型没有描述 SPM 和 DMA 的接口,因此数据布局和自动 DMA 优化对自主运算模型的性能非常重要,编译器将根据数据布局和自动 DMA 优化分析的结果将一部分空间较小、使用频率较高的数据静态布局在 SPM 中,并为大数组在 SPM 中预留 DMA 缓冲区;消息、I/O 函数和其他系统调用,将被编译器替换成线程级的接口,利用编译器级的线程代理优化进行处理.

```
int A,B[1024],C[1<<20];
main()
{
   if (myrank == 0)
        m_msg_put(…);
   printf("Hello world!\n");
}
```

⇨

```
__thread_local int A,B[1024], dma_buf[4096];
__thread int C[1<<20];
user_main(·) {
     if (myrank==0) m_msg_put_thread(…);
     printf_thread("Hello world!\n");
}
main(·){
     #pragma ccc sync call
     user_main(·);
}
```

6 实验结果

为了验证 Parallel C 语言和编译系统的正确性和有效性,我们利用国家超算无锡中心的神威太湖之光计算机系统进行了大规模测试.该系统的每个节点由一颗申威众核处理器构成(260 核,1.5GHz),配备 32GB 内存,运行 Linux 操作系统,节点间采用 56Gbps 的 Infiniband FDR 网络连接.测试中,最大程度地使用了神威太湖之光计算机系统 4096CPU 共 1 064 960 核的计算资源.

6.1 Micro Benchmark测试

为了验证 Parallel C 编译系统的主要功能,我们编写了一组 Micro Benchmark,在神威太湖之光环境下进行了测试.

- 加速线程创建和回收的性能

spawn-join-test 测试加速线程“创建-回收”的性能,测试了两种实现方式,未优化的实现采用单个线程依次“创建-回收”的方式,优化方式采用批量“创建-回收”方式.从表 1 的测试数据来看,优化实现的线程的“创建-回收”非常高效.

Table 1 Performance of threads spawn and join

表 1 Parallel C 加速线程创建和回收性能

	1 次“创建-回收”开销(拍)	每秒可执行“创建-回收”的次数
单个线程依次创建-回收	26 643	56 299
优化实现(批量创建-回收)	859	1 746 216

- 数据布局与自动 DMA 优化

dpdma-test 测试了一组典型数据访问方式的数据布局和自动 DMA 优化性能,包含连续访问、连续和离散访问混合、离散访问等.dpdma-test 的优化加速效果如图 19 所示.

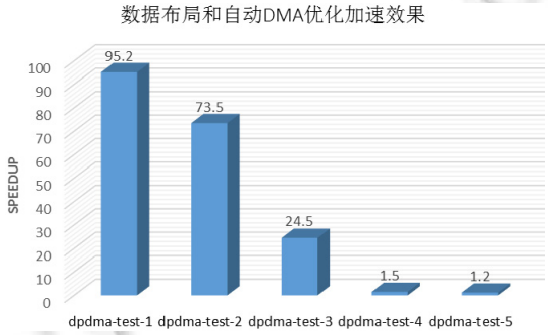


Fig.19 Result of data layout and automatic DMA optimization

图 19 数据布局与自动 DMA 优化效果

dpdma-test-1 的数据量小,所有数据可以布局到 SPM 中,优化后加速效果达到 95 倍;dpdma-test-2 的访问最频繁的数据可以布局到 SPM,其余大数组可以通过 DMA 优化,可加速 73 倍;dpdma-test-3 主要依靠 DMA 优化,可加速 24 倍;dpdma-test-4 只有小部分数据访问可优化,其余的是离散访问,dpdma-test-5 的程序结构过于复杂,主要数据访问未能优化,优化效果不明显.

- 单边通信性能

msg-test 测试 Parallel C 单边消息性能,分别测试了延迟和带宽.

- msg-test-1 采用 pingpong 方式,测试了 Parallel C 单边带回答字 put 消息的延迟;
- msg-test-2 测试了 MPI 的两种单边消息模式:一种是主动目标模式,采用 pingpong 方式,主动方采用 start-put-compelte 的方式,被动方采用 post-wait 方式;另一种是被动目标模式,主动方采用 lock-put-unlock 方式,被动方无任何操作.

MPI 单边消息的两种模式都需要多次握手,而 Parallel C 单边消息只需要 1 次 RDMA+1 次 Active Message 或 1 次 Active Message.从表 2 的测试结果可见,Parallel C 的单边消息在延迟方面具有非常明显的优势.

Table 2 One-Side put latency of Parallel C and MPI

表 2 Parallel C 与 MPI 单边 put 延迟对比

消息长度	Parallel C 单边 put 延迟(μs)	MPI 单边 put 消息延迟(主动目标)(μs)	MPI 单边 put 消息延迟(被动目标)(μs)
4B	2.74	11.63	8.56
1KB	3.62	13.06	11.07
64KB	9.83	23.83	37.91
4MB	368.65	703.61	717.43

msg-test-3 测试了 Parallel C 单边 put 的带宽,采用了两种方式:一种是带回答字的 put 消息,连续发送 32 个消息;另一种是不带回答字的 put 消息,连续发送 32 个消息.msg-test-4 测试了 MPI 单边 put 的带宽,采用了两种方式:一种是主动目标方式,start 后,连续发送 32 个消息,最后 compelte;另一种是被动目标方式,lock 后,连续发送 32 个消息,最后进行 unlock 操作.测试结果见表 3.Parallel C 不带回答字的单边 put 带宽最高;MPI 主动目标方式好于 Parallel C 带回答字的消息,原因是该测试方式只有 1 个 start、1 个 compelte,中间的 32 次消息全部都是底层 RDMA 操作,而 Parallel C 带回答字的 put 每次都有回答字原子加 1 操作产生的额外异步消息通信,因此带宽略低.但实际应用中,少有同一进程在单次通信中向同一目标发送 32 个 put 的用法,大多是发送 1~2 个消息,在这种意义下,从表 2 单次 put 的数据可以看出,Parallel C 单次单边 put 的带宽要明显高于 MPI.

Table 3 One-Side put bandwith of Parallel C and MPI

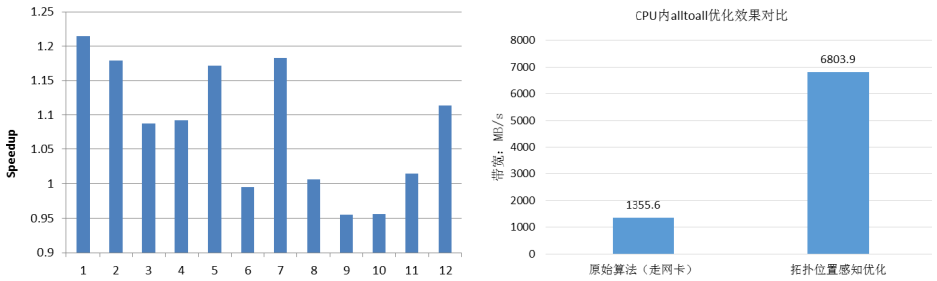
表 3 Parallel C 与 MPI 单边 put 的带宽对比

消息长度	Parallel C 单边 put 带宽(MB/s)	Parallel C 不带回答字单边 put 带宽(MB/s)	MPI 单边 put 消息带宽(主动目标)(MB/s)	MPI 单边 put 消息带宽(被动目标)(MB/s)
4B	1.17	1.20	0.73	1.23
1KB	202.94	339.49	178.53	253.41
64KB	5 520.04	5 867.44	5 818.35	4 307.37
4MB	5 783.63	5 813.34	5 782.97	5 757.12

- alltoall 通信优化

alltoall-test-1 测试大规模环境下 alltoall 并行通信的性能,运行 4096 进程,每个进程 alltoall 交换的数据量为 2GB,alltoall 内单进程每轮通信的数据量为 512KB.为验证拓扑位置感知的通信优化效果,将 alltoall 程序分别在 12 种不同拓扑结构位置的资源上进行了测试,性能与采用的传统算法 ring 算法(MVAPICH 大数据量的 alltoall 采用该算法)相比,性能平均提升了 8.1%,优化的效果如图 20(a)所示.

alltoall-test-2 测试 CPU 内的 alltoall 通信性能,每个进程 alltoall 交换的数据为 2MB,alltoall 内单进程每轮通信的数据量为 512KB,测试结果如图 20(b)所示,直接通过网卡传输数据的性能较低,通过利用从核空闲的计算能力来加速 CPU 内的 alltoall,则取得了非常明显的加速效果.



(a) 不同拓扑位置 1024CPU alltoall 优化效果 (b) 单 CPU 内 alltoall 性能优化前后对比

Fig.20 Result of topology-aware alltoall optimization

图 20 拓扑位置感知的 alltoall 通信优化效果

- 编译指导的代理优化

io-agent 测试使用第 5.3 节中的例子,循环 100 次,通过对比优化前和优化后的运行时间可以看出,优化前打印耗时 5 673ms,优化后耗时 1 131ms,可以使线程打印性能提升 4.01 倍.

- 动态任务调度

dyn-task-test 测试动态任务调度框架的性能,我们采用了一种最为苛刻的测试方式,将单个任务的执行时间设定为恒定,这样,集中申请任务冲突的概率比任务不平衡的应用要大,能够更好地衡量动态任务调度框架的开销.测试使用了 4096CPU,100 万个加速线程,每个线程执行 20 个时间固定的任务,对比了静态划分和动态调度的时间.表 4 的测试数据表明:在 100 万线程的大规模环境下,动态任务调度框架的开销控制得很好.在实际应用中,如果平均每个核心需要执行任务数量很大、单个任务的执行时间很短,则可对任务进行一定粒度的合并,再使用动态任务调度框架.

Table 4 Cost of Parallel C dynamic task scheduling

表 4 Parallel C 动态任务调度性能开销

	静态划分执行时间(s)	动态任务调度框架执行时间(s)	相对于静态划分执行时间增加的比例(%)
单个任务执行时间 16s	320	324.95	1.55
单个任务执行时间 64s	1 280	1 283.88	0.30
单个任务执行时间 256s	5 120	5 122.95	0.06

6.2 实际应用

- 湍流直接数值模拟

湍流计算是物理过程模拟领域的重要应用,直接数值法湍流模拟计算的总数据量可达数十 TB,核心是三维大规模数组的 FFT 变换、频域的矩阵运算以及空间域的矩阵对应位置计算.

北京大学使用 Parallel C 语言,在神威太湖之光计算机系统上实现了目前全球最大规模的直接数值法湍流模拟应用,问题规模达到 16 384 的立方,主要使用了 Parallel C 的 alltoall、加速线程等功能.该课题的计算量大概与问题规模成正比,问题规模从 1024×1024×1024 增加到 16384×16384×16384,计算量增大 4 096 倍.表 5 给出了 16 384 进程的测试数据.

Table 5 Performance of turbulence numerical modeling (16 384 processes)

表 5 湍流直接数值模拟在 16 384 进程下的性能

问题规模(立方)	用时(秒)	每小时迭代步
1 024	1.2	3 000
2 048	2.6	1 384.62
4 096	7.4	486.49
8 192	25.7	140.08
16 384	97.0	37.11

从测试结果可以看出:当问题规模较小时,每次消息粒度小,持续性能较低,问题规模从 1024×1024×1024 到

16384×16384×16384,单次迭代时间仅增加了 48 倍,说明随着问题规模的扩大,持续性能增长明显,课题良好的性能主要得益于 Parallel C 优化的 alltoall 接口和加速支持.

- 蛋白质折叠

蛋白质折叠问题是分子动力学模拟的一类重要应用,被誉为计算生物化学研究中的圣杯.该问题通过模拟蛋白质分子链上原子之间的相互作用引起空间结构折叠变化过程,寻找满足特定生化特性的蛋白质三维结构,目前计算机的模拟结果已经在真实实验中得到了很好的印证.模拟过程要得到现实可用的结果,需要在总的模拟物理时间上达到毫秒级别,计算量非常大.

北京大学在神威太湖之光上以 Parallel C 加速运算模型编程,实现了高效、可扩展的蛋白质折叠应用,主要使用了单边消息、局部性描述、加速线程等功能.课题在 10034 进程下,基于 BSP 模型实现了规模为 21 万原子的蛋白质分子在无水环境下的折叠模拟,模拟计算的速度达到了每秒 1 931 步.该课题的并行算法需要频繁的不规则通信进行数据交换,Parallel C 在单边通信方面比 MPI 具有更好的优势,满足了课题对单边通信延迟尽量低的要求.同时,Parallel C 高效的局部性描述和加速线程机制,保证了大量计算的高效执行.

- 三维数据可视化

浙江大学使用 Parallel C,在国产众核系统上实现了三维数据可视化领域的面元可视化和体可视化,处理规模和效率分别可达亿级面片地形场景的交互级绘制以及 10 亿字节(GB)级三维体数据的交互级绘制,图 21 所示为大规模过程式地形场景渲染的效果.

Fig.21 Output of the large-scale procedural terrain rendering

图 21 大规模过程式地形场景渲染效果图

性能测试数据见表 6,统计两分钟输出的帧数,计算可得在 1.26 亿面片规模下可视化效率为 5.33 帧/s,在 1.3GB 体数据规模下可视化效率为 5.17 帧/s.由于申威众核处理器并非针对图形处理领域设计,也没有配备相关的图形专用库,因此该结果与 GPU 上的数据相比还有一定的差距,但已很好地发挥出了申威处理器的性能,达到了预期的效果.

Table 6 Results of 3D data visualization test

表 6 三维数据可视化测试结果

应用	输入文件规模	输出帧数(2min)	可视化效率
面可视化	1.26 亿面元	639	5.33 帧/s
体可视化	1.3GB 体数据	620	5.17 帧/s

- winrar crack

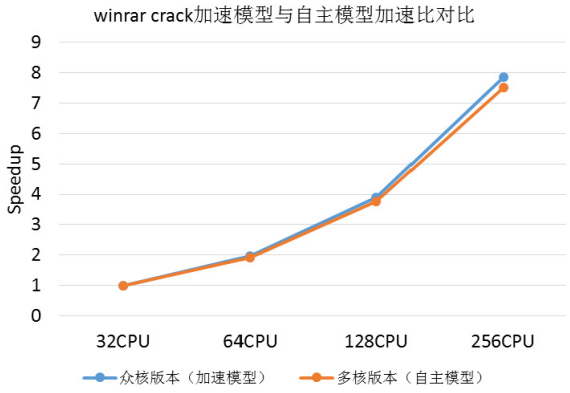


Fig.22 Results of the winrar crack with autonomic computing model

图 22 winrar crack 自主运算模型效果

winrar 是常用的压缩软件,其加密功能采用高强度的 Hash 算+分组密码,具有加密强度高的特点,是保护压缩文件数据安全的有效手段.winrar crack 是多核系统上尝试恢复 winrar 口令的成熟应用.

我们采用 Parallel C 的自主运算模型,直接将多核系统上的 winrar crack 应用不加修改地移植到众核系统上,取得了良好的效果.测试数据如图 22 所示,与采用加速运算模型优化的性能基本相当.多核版本的 Parallel C 程序不加修改即能直接在众核系统

上高效运行,主要得益于编译系统针对众核的数据布局与自动 DMA、线程代理框架和编译指导的代理优化.

7 结束语

本文针对众核时代面临的可编程性难题,提出了面向国产异构众核系统的多模式并行编程模型,设计了配套的 Parallel C 语言,能够有效地描述国产众核系统的异构并行性,并支持按照多核的方式使用众核系统.与其他众核系统上 MPI+X 的编程模式相比,程序员只需掌握一种语言,编程和编译系统优化都具有全局视角;Parallel C 语言采用了统一架构设计的思想,保持了进程和加速线程的扩展功能风格的一致性,支持多种局部性描述、并行任务调度框架、单边消息机制、加速线程相关扩展,对于编程和优化都有很好的帮助.基于 Open64 开源编译器,构建了 Parallel C 编译系统,全面支持加速运算模型和自主运算模型,提出并实现了数据布局与自动 DMA、编译指导的线程代理、拓扑位置感知的集合通信等优化.Micro Benchmark 和实际应用在神威太湖之光计算机系统上的测试数据表明:Parallel C 语言和编译系统具有良好的性能和可扩展性,能够有效支撑大型应用.

Parallel C 语言和优化实现的设计思想,可以推广到其他众核系统.例如,我们在 GPU 上实现了代理相关的部分功能和优化[24],取得了良好效果.下一步将继续探索针对国产众核系统的编译优化,不断完善和优化编译系统的性能,并使自主运算模型的适应面更宽.

致谢 在本文的研究工作中,得到了北京大学的陈一峯老师、崔翔老师、黄锟博士,浙江大学的候启明老师的热情指导和无私帮助,在此表示衷心的感谢.

References:

[1] TOP 500. TOP 500 supercomputer sites. 2015. http://www.top500.org/lists/2015/11

[2] Dongarra J, Beckman P, Moore T, *et al*. The Int'l exascale software project roadmap. Int'l Journal of High Performance Computer Applications, 2011,25(1):3–60. [doi: 10.1177/1094342010391989]

[3] Carter NP, Agrawal A, Borkar S, *et al*. An architecture for ubiquitous high-performance computing. In: Proc. of the 2013 Symp. on High- Performance Computer Architecture (HPCA 2013). Shenzhen, 2013. 198–209. [doi: 10.1109/HPCA.2013.6522319]

[4] EESI project—The European exascale software initiative. http://www.eesi-project.eu/

[5] MPI documents. http://www.mpi-forum.org

[6] OpenMP. http://www.openmp.org

[7] PGAS—Partitioned global address space languages. http://www.pgas.org

[8] OpenACC home. http://www.openacc.org

[9] Dolbeau R, Bihan S, Bodin F. HMPP: A hybrid multi-core parallel programming environment. In: Proc. of the Workshop on General Perpose Processing on Graphics Processing Units (GPGPU 2007). Boston, 2007. 28. https://www.researchgate.net/publication/240064180.

[10] NVIDIA CUDA home page. http://www.nvidia.com/object/cuda_home_new.html

[11] UPC Consortium. UPC language specifications v1.2. Lawrence Berkeley National Laboratory, 2005.

[12] Charles P, Grothoff C, Saraswat V, Donawa C, Kielstra A, Ebcioglu K, von Praun C, Sarkar V. X10: An objecti-oriented approach to no-uniform cluster computing. ACM Sigplan Notices, 2005,40(10):519–538. [doi: 10.1145/1103845.1094852]

[13] Lu XJ, Liu L, Jia HP, Feng XB, Wu CG. ParaC: A domain programming framework of image processing on GPU accelerators. Ruan Jian Xue Bao/Journal of Software, 2017 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5241.html [doi: 10.13328/j.cnki.jos.005241]

[14] BDEC: Home. http://www.exascale.org

[15] Patterson D. Overview of the UC Berkeley Par Lab. In: Proc. of the 2009 IEEE Hot Chips 21 Symp. (HCS). Stanford: IEEE, 2009. 1–38. [doi: 10.1109/HOTCHIPS.2009.7478356]

[16] Adve S, Adve V S, Agha G, *et al*. Parallel computing research at Illinois: The UPCRC agenda. 2008. http://www1.ju.edu.jo/ecourse/cpestcourse/readings/UPCRC_Whitepaper.pdf

[17] Duran A, Klemm M. The Intel® many integrated core architecture. In: Proc. of the 2012 IEEE High Performance Computing and Simulation (HPCS) Int'l Conf. 2012. 365−366.

[18] Lindholm E, Nickolls J, Oberman S, *et al*. NVIDIA Tesla: A unified graphics and computing architecture. IEEE Micro, 2008,28(2).

[19] Macri J. AMD's next generation GPU and high bandwidth memory architecture: FURY. In: Proc. of the Hot Chips 27 Symp. (HCS) 2015 IEEE. IEEE, 2015. 1−26.

[20] Fan DR, Yuan N, Zhang JC, Zhou YB, Lin W, Song FL, Ye XC, Huang H, Yu L, Long GP, Zhang H, Liu L. Godson-T: An efficient many-core architecture for parallel program executions. Journal of Computer Science and Technology, 2009,24(6): 1061−1073. [doi: 10.1007/s11390-009-9295-3]

[21] Fu H, Liao J, Yang J, *et al*. The sunway taihulight supercomputer: System and applications. Science China (Information Sciences), 2016,59:072001. [doi: 10.1007/s11432-016-5588-7]

[22] Steinke S, Wehmeyer L, Lee BS, Marwedel P. Assigning program and data objects to scratchpad for energy reduction. In: Proc. of the 2002 Design, Automation and Test in Europe Conf. and Exhibition. Paris: IEEE, 2002. 409−415. [doi: 10.1109/DATE.2002.998306]

[23] Chan S, Gao G, Chapman B, Linthicum T, Dasgupta A. Open64 compiler infrastructure for emerging multicore/manycore architecture. In: Proc. of the IEEE Int'l Symp. on Parallel and Distributed Processing (IPDPS 2008). Miami: IEEE, 2008. 1−2. [doi: 10.1109/IPDPS.2008.4536577]

[24] Wu W, Qi FB, He WQ, Wang SS. Using CUDA's mapped memory to support I/O functions on GPU. Tsinghua Science and Technology, 2013,18(6):588−598. [doi: 10.1109/TST.2013.6678904]

[25] MVAPICH2. http://mvapich.cse.ohio-state.edu/

附中文参考文献:

[13] 卢兴敬,刘雷,贾海鹏,冯晓兵,武成岗.ParaC:面向 GPU 平台的图像处理领域的编程框架.软件学报,2017. http://www.jos.org.cn/1000-9825/5241.html [doi: 10.13328/j.cnki.jos.005241]

何王全(1975−),男,四川丹棱人,高级工程师,CCF 专业会员,主要研究领域为高性能计算体系结构,并行语言设计,编译优化,运行时系统.

刘勇(1981−),男,博士,工程师,CCF 专业会员,主要研究领域为高性能计算体系结构,并行算法,性能监测工具.

方燕飞(1980−),女,工程师,主要研究领域为并行语言设计,并行编译,并行算法.

魏迪(1984−),男,工程师,主要研究领域为并行语言设计,并行运行时系统.

漆锋滨(1966−),男,博士,高级工程师,博士生导师,CCF 杰出会员,主要研究领域为高性能计算体系结构,编译优化.