软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(8):2352−2364 [doi: 10.13328/j.cnki.jos.006007]

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

SW26010 众核任务并行调度系统及其嵌套并行算法应用*

孙 乔, 黎雷生, 赵海涛, 赵 慧, 吴长茂

(中国科学院 软件研究所 并行软件与计算科学实验室,北京 100190)

通讯作者: 吴长茂, E-mail: changmaowu@foxmail.com

摘 要: 任务并行是并行程序设计的基础设计模式.但由于算法本身的复杂性及目标平台的特殊性,设计实现高效率的任务并行程序对程序员来说往往充满挑战.基于新兴的SW26010众核CPU,提出了支持任务嵌套并行模式的通用运行时框架 SWAN.SWAN 对任务并行程序的实现提供了高层次的抽象,使程序员能够专注于算法逻辑本身而提高开发效率.在性能方面,SWAN 框架对诸多共享资源进行了细粒度的划分,从而有效地避免了众多线程间对共享资源的高强度争用.充分利用平台的高速访存机制、高速可控缓存和原子操作等特性,对 SWAN 框架的核心数据结构进行优化设计以降低其本身的性能开销.SWAN 还具备动态负载均衡能力,使各个处理器核心的资源得以充分利用.基于 SWAN 框架,在目标平台上实现了若干典型的具有递归特性的嵌套并行算法,包括 N-皇后问题、二叉树遍历、快速排序和凸包求解.实验结果表明,这些通过使用 SWAN 框架得以并行化的算法相对于其串行版本取得了4.5~32 倍的加速,充分说明了 SWAN 框架具有较高的实用性及性能.

关键词: 任务并行框架;并行计算;嵌套并行算法;SWAN;SW26010 众核 CPU

中图法分类号: TP303

中文引用格式: 孙乔,黎雷生,赵海涛,赵慧,吴长茂.SW26010 众核任务并行调度系统及其嵌套并行算法应用.软件学报,2021,32(8):2352−2364. http://www.jos.org.cn/1000-9825/6007.htm

英文引用格式: Sun Q, Li LS, Zhao HT, Zhao H, Wu CM. Task parallel framework and its application in nested parallel algorithms on the SW26010 many-core platform. Ruan Jian Xue Bao/Journal of Software, 2021,32(8):2352−2364 (in Chinese). http://www.jos.org.cn/1000-9825/6007.htm

Task Parallel Framework and Its Application in Nested Parallel Algorithms on the SW26010 Many-core Platform

SUN Qiao, LI Lei-Sheng, ZHAO Hai-Tao, ZHAO Hui, WU Chang-Mao

(Laboratory of Parallel Software and Computational Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: Task parallelism is one of the fundamental patterns for designing parallel algorithms. Due to algorithm complexity and distinctive hardware features, however, implementation of algorithms in task parallelism often remains to be challenging. On the newly SW26010 many-core CPU platform, a general runtime framework, SWAN, which supports nested task parallelism is proposed in this study. SWAN provides high-level abstractions for programmers to implement task parallelism so that they can focus mainly on the algorithm itself, enjoying an enhanced productivity. In the aspect of performance, the shared resources and information manipulated by SWAN are partitioned in a fine-grained manner to avoid fierce contention among working threads. The core data structures within SWAN take advantage of the high-bandwidth memory access mechanism, fast on-chip scratchpad cache as well as atomic operations of the platform to reduce the overhead of SWAN itself. Besides, SWAN provides dynamic load-balancing strategies in runtime to ensure a full occupation of the threads. In the experiment, a set of recursive algorithms in nested parallelism, including the *N*-queens problem, binary-tree traversal, quick sort, and convex hull, are implemented using SWAN on the target platform. The experimental results reveal

* 基金项目: 中国科学院战略性先导科技专项(C 类)(XDC01030200)
Foundation item: Strategic Priority Research Program of the Chinese Academy of Sciences (Category C) (XDC01030200)
本文由“国产复杂异构高性能数值软件的研制与测试”专题特约编辑孙家昶研究员、李会元研究员推荐.
收稿时间: 2019-08-22; 修改时间: 2019-12-05; 定稿时间: 2020-01-22

that each of the algorithms can gain a significant speedup, from 4.5x to 32x, against its serial counterpart, which suggests that SWAN has a high usability and performance.

Key words: task parallel framework; parallel computing; nested parallel algorithm; SWAN; SW26010 many-core CPU

任务并行模式[1,2]是并行程序设计的一种基本设计模式,在并行计算领域中有着广泛的应用.一个任务是由相关数据及其操作形成的集合[1,3].相互独立的任务可以同时被分派到不同的处理器得以并行处理,从而使程序获得较为理想的并行加速比.相对于朴素的数据并行模式,采用任务并行模式可以更高效地并行化诸如快速排序,二叉树遍历及凸包计算等一系列具有递归结构的算法[4,5].然而在实际应用中,不同任务间往往具有复杂的前驱后继关系,一些任务还可能需要衍生出其他任务,形成任务嵌套(nested task)[1].因此任务并行程序的正确性不仅依赖于各个任务的正确定义,还依赖于任务间恰当的时序.因此若要全盘考虑上述要素,则任务并行程序的设计和实现会有着相当的难度.从而一个通用的支持任务并行模式的运行时框架[6,7](下称:任务并行框架)成为程序员开发任务并行程序的必需.

SW26010(申威 26010)CPU 是我国自主研发的一款高性能众核 CPU[8].如图 1 所示,一片 SW26010 CPU 由 4 个 CG(core group,核组)构成.每组 CG 包括 1 个控制核心 MPE(management processing unit)和 64 个 CPE (computing processing element).从本质上讲,MPE 是一个通用处理器,负责处理程序的逻辑密集部分和系统资源的控制;众多 CPE 则是一些轻量级计算核心,主要负责加速程序的计算密集部分.64 个 CPE 被排布成 8×8 阵列,每个 CPE 可通过唯一的标识或其在阵列中所处的行(列)号进行索引.每个 CPE 有程序可控的 64KB 高速 LDM (local data memory,本地数据内存)作为其数据缓存.一组 CG 中的 MPE 和 CPE 阵列能够共享进程的内存空间.除了常规的访存方式外,单个 CPE 与内存间还能通过 DMA(direct memory access)机制进行数据块的高效传输.

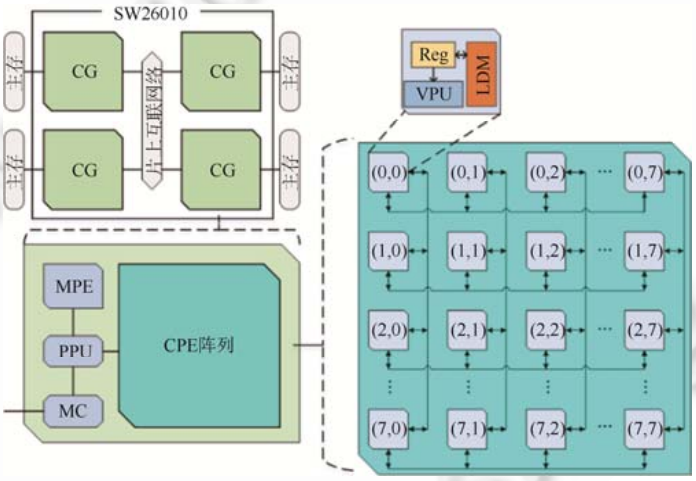


Fig.1 Architecture of a SW26010 many-core CPU

图 1 SW26010 众核 CPU 体系结构

一片 SW26010 CPU 的 4 组 CG 通过片上高速互联网络相连,其间实现了缓存一致性协议.但由于每组 CG 拥有各自临近的内存空间,因此在常规情况下,一片 SW26010 CPU 的 4 个 CG 需分别运行单独的进程,以避免延迟较高的远端内存访问.运行于 SW26010 CPU 的进程由 CPE 阵列对计算密集但逻辑简单的部分进行并行加速处理,从而一系列科学计算类应用能够得以有效加速.然而随着计算机应用的日益广泛,一系列非规整算法也亟待通过并行化得以有效加速.因此本文针对上述情况,面向 SW26010 CPU 的单组 CG,设计实现任务并行框架 SWAN.其对降低 SW26010 CPU 的使用难度及扩大平台的适用性有着重要的现实意义.

本文第 1 节将进一步描述对 SWAN 的设计实现有着重要影响的目标平台特性,并针对性地提出 SWAN 框架的设计目标.第 2 节具体描述 SWAN 的静态结构与和动态行为.第 3 节进一步介绍 SWAN 框架采用的关键技

术.第 4 节拟通过若干具有代表性的嵌套并行算法对 SWAN 的实用性及性能进行测试.第 5 节回顾在主流的多(众)核平台上常用的诸多任务并行框架,并探讨其与 SWAN 的区别与联系.第 6 节对本文的工作进行总结.

1 平台特性

一组 CG 的诸多硬件特性对 SWAN 框架的设计实现提出了要求.首先,一个 CPE 目前只能运行单一线程,无法进行线程切换.因此 CPE 上的任务调度环境应当具备上下文切换功能,旨在确保当前任务因依赖关系尚未满足而挂起导致的 CPE 忙等.其次,由于所有 CPE 共享统一的内存空间,这虽有利于简化共享数据结构的设计实现,但若不对共享数据加以适当处理,则会导致大量线程由于对共享数据的争用而导致的巨额开销.因此 SWAN 框架需对任务池等共享资源进行细粒度划分以提高 SWAN 框架本身的并发性.再次,单个 CPE 虽可以通过 load/store 指令直接访问内存单元,但其效率低下.因此为了增大内存吞吐率,CPE 应尽可能地采用 DMA 机制传输成块数据.这就意味着 SWAN 框架中的关键数据结构需有合适的物理结构,能够通过 DMA 进行高效的访问.最后,SWAN 还需利用各个 CPE 的 LDM 以缓存任务管理所需的各种信息,以提高任务管理的效率.

在平台的软件特性方面,单个 CG 的线程控制、DMA 机制等功能通过调用“athread”库实现.athread 库提供基本的线程发起(“athread_spawn()”)和线程集合(“athread_join()”)接口操纵 CPE 阵列.单个 CPE 上,athread_get()/athread_put()接口通过 DMA 实现数据在 LDM 和内存间的交换.另外,单个 CPE 上拥有可作用于从核阵列的“取并加一(fetch-and-add)”原子操作.从现有 API 上看,目前单个 CG 可方便地部署以“Fork-and-Join”方式并行的程序,但对具有嵌套并行特性的算法,平台尚未向程序员提供更高层的接口及更高级的细粒度线程同步手段,如锁和信号量等.

综上所述,在 SW26010 上实现通用的支持任务并行的运行时框架,需以现有的并行接口为基础,采用软件方式跨越硬件限制并充分利用平台的各种资源.因此 SWAN 框架的设计和实现具有相当的难度.

2 相关工作

虽然 SWAN 框架为 SW26010 众核核组定制,结合了诸多平台的软硬件特性,但当今国内外在任务并行模式及任务并行框架方面的研究对 SWAN 框架的设计与实现有着重要的借鉴意义.

任务并行模式主要应用在 SMP(对称多处理器)或 NUMA(非对称多处理器)环境下[1].在这种环境下 OpenMP[4,5,9,10]是一套指导性编译处理标准,用户能够通过编译制导语句方便地对程序实施并行化.OpenMP 标准已被广泛接受,在学术界及开源社区有着众多实现,比如 OpenUH[11].早先,OpenMP 被广泛应用于并行化大型科学计算程序中结构规整的循环,但由于在版本 3.0 之前,OpenMP 并不支持任务并行,因此其应用范围受到了极大的制约.在版本 3.0 之后,OpenMP 集成了任务并行功能[12],用户可以通过编译制导语句方便地实施任务并行及任务嵌套并行.遗憾的是,OpenMP 在当前 SW26010 核组上没有得到有效支持.因此本文中提出的支持任务并行及任务嵌套并行的 SWAN 框架能够有效地拓展 SW26010 核组在任务并行领域的应用.于 2012 年提出的 OpenMP 4.0 标准增加了以任务间依赖有向无环图为基础的任务调度策略[13],这亦为 SWAN 框架的进一步发展指明了方向.除 OpenMP 之外,在当前主流的多核、众核平台上还有一些专门支持任务并行的并行编程框架.Cilk[14,15]及 Cilk-5[16]是由 MIT 于 2001 年提出的多核平台任务并行解决方案.与 OpenMP 不同,Cilk 扩展了 C 语言关键字以实现任务并行及任务嵌套并行.Cilk 采用了经典的工作窃取[17]调度策略并对其性能和内存使用行为进行了理论评估.本文结合 SW26010 平台的特性,将工作窃取策略整合于 SWAN 框架中,构成任务管控逻辑的核心.相似地,X10[18,19]定义了新的并行语言,能够使程序员高效地表达程序的并行性.X10 主要面向 NUMA 架构,以库所(place)概念为核心管理程序数据,缓解由平台访存不均匀性带来的程序性能损耗.Intel TBB[20]及 Microsoft 推出的 TBL[21]以 C++语言为基础,为程序员提供了丰富的并行模板及数据结构.它们的出现标志着任务并行编程模型走入了工业界.但是它们主要面向 C++程序,其实现也结合了诸多面向对象特性,因此它们的高效使用需要程序员对 C++语言中的模板、泛型等概念有着深刻的理解.

随着异构平台的流行,任务并行编程的应用得到了进一步拓展.如 Intel 为 MIC 协处理器定制的 OpenMP 扩

展编译器,能够通过分载(offload)子句将任务指派到 MIC 协处理上予以执行[22].另外,StarPU[23]是一套支持主流异构平台(如 CPU+MIC 和 CPU+GPGPU)上的通用任务并行的编程框架.StarPU 需要将任务透明地通过互联总线映射到协处理器予以计算.由于 SW26010 的 MPE 与 CPE 阵列能够统一访存,因此 SWAN 框架无需负责任务在不同设备间的来回传输.值得一提的是,在典型的 SIMD 平台 GPGPU 上,任务并行也日渐受到重视,文献[24]阐述了如何在 GPGPU 平台上以线程束(warp)为逻辑处理器核心,实现多任务并行的实现机制.由于 GPGPU 和 SW26010 在处理器架构,内存层次结构和访存特性上有较大差异,因此在 GPGPU 上实现的任务并行框架难以在本文的目标平台上予以直接应用.

3 SWAN 框架的结构与行为

如图 2 所示,SWAN 系统由 4 个模块组成,分别是 CI(concurrent infrastructure,并发基础结构)模块、MC-Modeling(MPE/CPE modeling,MPE/CPE 建模)模块、QPP(queuing and parallelization policy、排队及并行策略)模块和 UI(user interface,用户接口)模块.其中,CI 模块与 MC-Modeling 模块描述 SWAN 框架的静态结构,QPP 模块定义 SWAN 框架的动态行为,UI 模块为用户提供 API 以屏蔽底层的并行处理的细节.

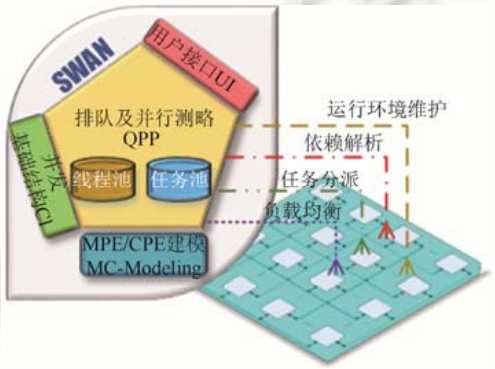


Fig.2 Software architecture of SWAN

图 2 SWAN 软件体系结构

3.1 SWAN框架的静态结构

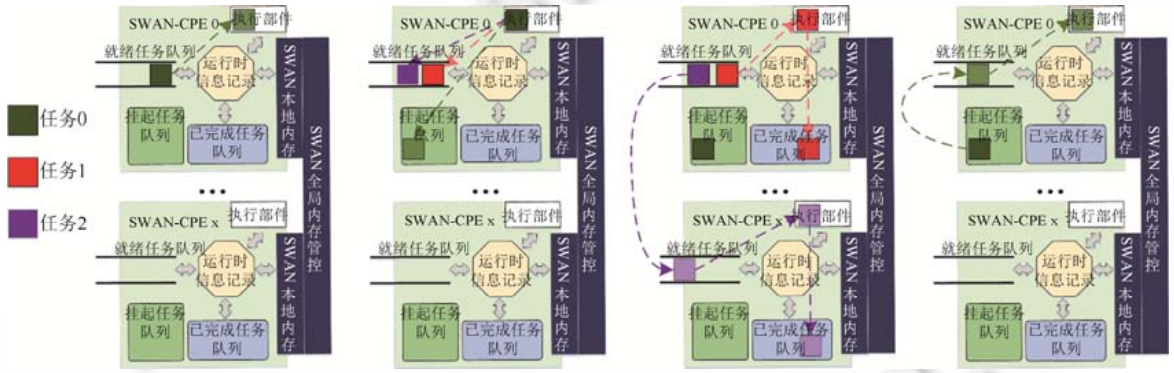
鉴于当前 CPE 阵列缺少灵活的细粒度线程同步机制,CI 模块基于平台的“取并加一”原子操作实现面向 CPE 阵列全体的互斥锁.由于被锁保护的临界区一次只能容纳一个 CPE 线程进行访问和 CPE 无法进行线程切换等原因,其余想要访问该临界区的 CPE 不得不处于“忙等”状态——反复读取并判别存储在内存中的同步信号以获取临界区访问权限.大量线程的忙等导致访存量剧增,极大地影响了临界区中工作线程的访存能力.因此,为了减少加锁导致的冗余访存,CI 模块中的互斥锁还具备睡眠功能:获取锁访问权限失败的线程将进入“睡眠”状态以避免对内存的高频访问.睡眠时间可由程序员自主调优.此外,CI 模块中包含了能够让众多 CPE 线程并发申请内存空间的内存管理子模块.在互斥锁和内存管理子模块基础上,CI 模块中还使用泛型技术实现了并发循环队列和并发 Hash 表等关键数据结构.出于性能考虑,这些并发数据结构需结合平台特性予以实现,具体内容在第 3.2 节中详述.

在 CI 模块基础上,本文对单组 CG 的 MPE 和各个 CPE 分别进行建模,形成 SWAN-MPE 和 SWAN-CPE,旨在使得一组 CG 的所有处理核心形成一个具有生产、交换、维护和处理任务的有机整体.由于程序的主进程运行于 MPE 上,SWAN-MPE 对象需要管理整个 SWAN 框架.因此,其需要包含一个由指定数量的 SWAN-CPE 对象形成的集合和诸多共享数据:任务总量及完成任务总量,任务参数总表及各个 SWAN-CPE 线程的内存使用量、处理器时间等运行时信息以为 SWAN 框架的负载均衡行为提供依据.SWAN-CPE 对象用来对一个 CPE 进行建模,其包括了若干私有及公共的数据结构.其中,就绪任务队列负责存放当前已经就绪的可执行任务,在必要情况下,就绪任务队列中的任务可以被其他线程偷取[17,19,25],以保证负载平衡.挂起任务队列属于一个 SWAN-CPE 的私有队列,当一个任务由于依赖关系未被满足而中止时,该 SWAN-CPE 将其加入挂起队列并获取其他任务继续执行.待某一时刻该

任务的所有前驱依赖关系全部满足后,SWAN-CPE 可将该任务再度放入就绪任务队列中等待执行.每个 SWAN-CPE 还有私有的已完成任务集,在该集合中的任务需要进一步被处理,以解除它们所有后继任务的依赖.每个 SWAN-CPE 还负责记录自己的内存使用量及处理器运行时间,在其空闲时更新 SWAN-MPE 对象中对应数据条目.一个进程的所有共享内存空间被划分成若干部分,使得每个 SWAN-CPE 线程拥有独立的私有内存空间来存放相应的数据结构.在私有内存空间不足的情况下,一个 SWAN-CPE 可将自己的内存分配请求发送给其他 SWAN-CPE.这样,SWAN 框架在充分利用内存资源的同时还能有效地避免多个线程争相申请内存而带来的开销.综上所述,MC-Modeling 模块的设计分散了 SWAN 框架的关键数据结构,尽可能地避免了共享资源的争用.

3.2 SWAN动态行为

SWAN 支持任务的动态生成及依赖关系的解析,这一关键过程被实现在 SWAN 的 QPP 模块中.如图 3 所示,一个简单的尾递归程序的并行执行由“任务-0”开始.初始时,任务-0 被“SWAN-CPE 0”执行,在执行过程中,由于任务-0 的进一步执行依赖于由它产生的两个子任务(“任务-1”和“任务-2”)的完成.因而,任务-0 被 SWAN-CPE 0 挂起(加入挂起任务队列).同时,衍生出的任务-1 和任务-2 被加入 0 号 SWAN-CPE 的就绪任务队列中.此时,由于 SWAN-CPE x 处于线程饥饿状态,由上文所述,它能够在 SWAN-CPE 0 的就绪任务队列中窃取某一任务(假设为任务-1)以进行处理.当任务-1 和任务-2 被各个 SWAN-CPE 执行完毕之后,它们被放入相应的终止任务队列,等待后续处理.处理一个已被完成的任务包括释放其占用的内存资源及解除与它相关的任务依赖关系.此时在 SWAN-CPE 0 的中挂起的任务-0 由于其所有的依赖条件皆被满足,它将被重新放回 SWAN-CPE 0 的就绪队列中得以进一步执行.



(a) 初始时任务 0 被放置于就绪队列 (b) 由任务 0 产生的任务 1 和任务 2,任务 0 挂起 (c) 任务 1 在本地执行,任务 2 被 SWAN-CPEx 窃取 (d) 子任务结束后,任务 0 继续得以执行

Fig.3 Process of task dependency resolvement in SWAN

图 3 SWAN 任务依赖关系解析过程

3.3 SWAN用户接口

SWAN 向用户提供了方便的编程 API,程序员并不需要显示操作“任务”这一数据结构.附录 1 和附录 2 分别展现了由 SWAN API 编写的并行快速排序和并行凸包求解程序,用户可以与函数调用相似的方式向 SWAN 框架动态地插入任务,程序的并行细节对程序员来说是透明的.

4 SWAN 框架的关键技术

4.1 任务挂起和上下文保存

在嵌套并行程序中,父任务的执行往往依赖其子任务执行的结果,在这种情况下父任务不得不暂停以等待所有子任务的完成.在没有线程切换功能的 CPE 上,SWAN 框架为用户提供了任务断点以中止当前任务的继续执行.在该任务的所有依赖关系被满足后,SWAN-CPE 可从任务断点开始继续处理先前被挂起的任务.在任务挂

起之前,用户还能将 LDM 中的关键数据移交至 SWAN 框架进行保存,以便在任务恢复执行时取回 LDM 加以使用.我们发现,任务的挂起和重启功能不仅能妥善处理任务间的依赖关系,在一些算法(如凸包求解算法,见附录2)中还有利于提高算法的并发度.

4.2 基于DMA和LDM的循环队列

在运行时,SWAN-CPE 需要不断地从就绪任务队列或挂起任务队列中获得任务进行执行,因此 SWAN-CPE 对这些队列的访问效率将影响 SWAN 框架本身的性能.由于目标平台拥有 DMA 高速访存机制和程序可控 LDM,我们采用了具有线性存储结构的环形链表,并将处于表头和表尾部分的若干任务指针缓存于拥有这些队列 SWAN-CPE 的 LDM 中,形成缓存结构.该 SWAN-CPE 若需要获得位于表头的任务时,事先会向该队列的 LDM 缓存进行索取;若获取失败,则使用 DMA 将批量任务指针加载到缓存.与此类似,若该 SWAN-CPE 需要向队尾插入任务时,可直接将该任务的指针放入进队缓存;当缓存队列放满时,则使用 DMA 向位于内存中的队尾进行批量插入.由于工作窃取机制的存在,某一就绪任务队列在运行时将可能被多个 SWAN-CPE 并发访问.为了保持队列中数据的一致性,我们限定来自其他 SWAN-CPE 的工作窃取请求只能作用于未被缓存入 LDM 的任务,而当前处于缓存中的任务被认为是当前 SWAN-CPE 私有的.值得注意的是,任意循环队列的都有指定的容量,因此 SWAN-CPE 不能无限制地向某一队列加入任务而不及时获取任务进行处理.

4.3 负载均衡策略

任务并行框架需要采取一定的任务调度策略[26,27]确保程序的并发度和处理单元的负载均衡.SWAN 目前提供两种工作窃取[17,19,25,28,29]手段保证 CPE 阵列的动态负载均衡:轮询窃取及基于动态信息的窃取.轮询窃取策略的实现相对简单:每个 SWAN-CPE 持有一个私有的轮询计数器,每次线程饥饿时对计数器指定的 SWAN-CPE 进行任务窃取并使计数器指向下一个 SWAN-CPE 的线程标志.轮询窃取机制实现简单,运行时开销较小并能够确保 SWAN 框架产生的所有窃取行为均匀地分布于所有 SWAN-CPE.基于动态信息的窃取机制需要 SWAN-CPE 动态地维护自身运行时信息,如已执行的任务数.需要窃取任务的 SWAN-CPE 通过向 SWAN-MPE 对象查询以得知至此最为繁忙(已完成任务数量最大)的 SWAN-CPE,并在它的就绪任务队列中窃取任务.基于动态信息的工作窃取机制由于需要维护全局运行信息,因此开销较大,但其窃取行为目标较为明确.一般而言,基于动态信息的工作窃取在任务粒度较大时具有较高调度效率,而轮询窃取在任务粒度较小但数量多时效率较高.

5 实 验

我们在 SW26010 CPU 的一组 CG 上对 SWAN 的可用性和性能进行测试.一组 CG 拥有大小为 7.7GB 的内存空间.MPE 的主频为 1.25GHz,理论带宽大约为 5GB/s.单个 CPE 的主频为 1.45GHz,DMA 的理论聚合带宽为 34GB/s(实际峰值为 22GB/s).实验所使用的程序均采用 sw5cc 编译器编译,产生相应的 MPE 代码及 CPE 代码,所有编译过程均采用“-O3”优化选项.我们在多个任务并行基准测试集如 Barcelona OpenMP Task Suite[5]和 Clik Task Suite[14−16]中选取了 4 个具有代表性的嵌套并行算法(算例),并使用 SWAN 框架在 CPE 阵列上实现它们.这 4 个算例分别是 *N*-皇后问题,二叉树遍历,快速排序和凸包求解.我们采用运行在 MPE 上的各个算法的串行版本作为性能参考基准,来衡量使用 SWAN 框架并行化带来的性能提高.并行程序运行的负载均衡率由运行时单个 CPE 处理的平均任务数量和单个 CPE 处理的最大任务数量的比决定[22].对于 SWAN 的可用性,我们拟在附录 1 和附录 2 中分别展示使用 SWAN 实现的具有尾递归特点的并行快速排序算法和具有首递归特点的并行凸包求解算法.

5.1 *N*-皇后问题

N-皇后问题要求在一个 $N\times N$ 的国际象棋棋盘上放置 N 个“皇后”棋子,并保证每个皇后不能直接攻击其他任意一个皇后.根据国际象行棋的规则,这些皇后棋子不能同时处于同一行、列及斜对角线.由于下一步棋子摆放的方式依赖于当前已摆放的棋子的形态,因此 *N*-皇后问题通常使用递归方式进行求解.然而在具体的算法执

行过程中,该算法有着大量的并行性值得挖掘:假设已经成功摆放了 K–1 个棋子,那么验证第 K 个棋子的各种摆放方法是否合法的计算间是相互独立的.通过 SWAN 可以让该算法的并行性充分地实现在 CPE 核组上.

对棋子合法性进行判断的计算代码可以得到针对性的简化,但这并不影响 SWAN 框架的使用.因此在图 4 中我们分别展示了使用经过计算简化的程序和未经计算简化的程序的可扩展性.但该算法无论是否经过计算简化,CPE 阵列并行版本相对于使用同样计算代码的 MPE 串行版本都有显著的加速比.而且加速比随着问题规模的增大而显著提高.这是因为随着问题规模的增大,更多独立的任务能够被分配到各个 CPE 得以并行处理.进一步的测试表明在执行 14-queens 算法的过程中,各个从核的负载均衡率能够达到了 91%.但值得我们注意的是:第一,计算简化对 MPE 串行程序带来了更优的加速效果,进而使得对应的并行加速比整体较低.第二,虽然在实验中我们开启了 64 个 CPE 线程,但程序整体的并行加速比并不超过 4.5.这是因为一方面 MPE 的数据缓存能够容纳整张棋盘,因此 MPE 串行实现的访存效率很高;另一方面,每一个 CPE 任务的访存规模太小而导致众核并行版访存时 DMA 性能较低.

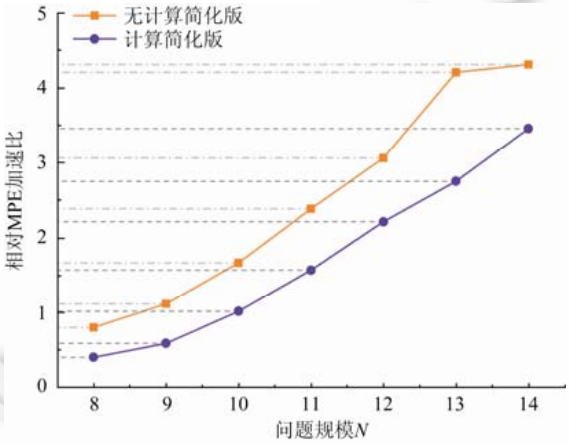


Fig.4 Speedup of the parallel N-Queens problem against the serial reference on the MPE

图 4 并行版 N-皇后问题相对于 MPE 串行版的加速比

5.2 二叉树遍历

二叉树遍历算法要求(并行地)访问二叉树的每一个节点,并保证每个节点仅被访问一次.在实际应用中各个二叉树的节点可代表不一样的处理逻辑.不失一般性,我们拟统计二叉树各个节点所含字符串中含有给定字符的数量.在本例中,我们采用二叉树的后根遍历算法,以验证 SWAN 框架确保任务执行顺序的能力.为了让 SWAN 框架体现其在大规模计算中带来的加速效果,各个节点的字符串的长度被设置为 5 000~10 000;为了进一步验证 SWAN 框架的负载均衡能力,对于每一个节点,我们产生随机长度的字符串并使其左右子节点的字符串总长度的比值达到 30%.

如图 5 所示,以 SWAN 框架实现的并行二叉树遍历程序的性能大幅度高于 MPE 串行版本,随着树的规模的增大,加速比可提高到 35 倍左右.这一方面由于 SWAN 能够将计算负载分配到各个 CPE 进行并行处理,另一方面是因为在各个任务的访存量较大,DMA 机制传输效率高.我们还发现,在问题规模较小的测试条件下(树高度小于 15 时),基于运行时信息的工作窃取策略优于基于轮询的工作窃取策略.在此用例中,我们还发现使用 LDM 缓冲的并发循环队列能够使性能进一步提高约 30%.经过进一步测试我们还发现即便在非规整的数据结构上,整个并行程序的负载均衡率也到达了 85 以上,这说明 SWAN 框架有着较强的负载均衡能力.

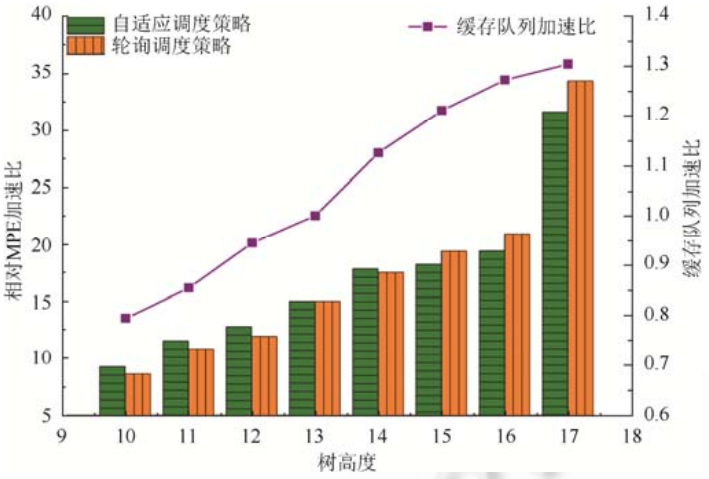


Fig.5 Speedup of parallel binary tree traversal against the MPE sequential binary tree traversal

图 5 并行版二叉树遍历程序相对于 MPE 串行版的加速比

5.3 快速排序

快速排序在目标数据集中选取一个“主元(pivot)”之后,将原数据序列以该主元为基准按照大小关系一分为二.对划分出的数据子列以同样的方式处理,直到子列只含有不多于 1 个元素.原始无序数据序列在经过如是处理之后变为有序.在实际的应用中,我们还可以设定一个阈值 *K*,当数据子列的长度小于 *K* 之后,使用高效的串行排序核心处理当前数据子列,以避免任务粒度过小.为考察使用 SWAN 实现的并行排序在实际应用中的性能表现,我们选取实现在 C 语言标准库中的快速排序(函数 qsort)作为性能对比基准.

如图 6 所示,在同样的测试数据序列上,以 SWAN 为基础实现的并行快速排序相对 qsort 有高于 13 倍的加速比.由于 SWAN 为程序员屏蔽了并行调度的细节,程序员可以集中注意力于核心排序函数的性能优化上.比如在当前算例中,我们对长度小于 K 的整型数据子列采用查表的方式进行排序,可使计算复杂度变为线性.经过计算核心的优化,基于 SWAN 的并行排序算法的性能又提升了大约 2 倍.与此类似,对于其他数据类型的排序,还可以使用向量化等手段提高排序核心函数的性能,在此不赘述.

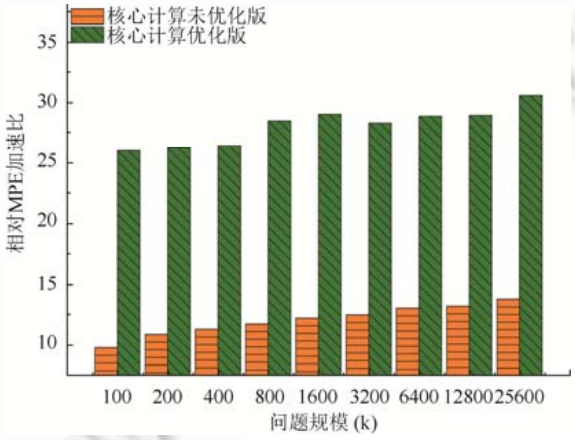


Fig.6 Speedup of the parallel quick-sort algorithm against the sequential qsort in C standard library

图 6 并行版快速排序算法对 C 标准库中串行 qsort 函数的加速比

5.4 凸包求解

在一个实数向量空间 V 中,对于给定点集 X,所有包含 X 的凸集的交集 S 被称为 X 的凸包.本文只讨论二维空间中点集的情况.图 7 中红色连线确定了一个含有 12 个点的平面点集形成的凸包.图 7(a)~图 7(f)分步展示了该凸包的快速求解算法的递归步骤:首先确定含有最小横坐标值和最大横坐标值的点 a 和 b 并进入递归步骤:以有向线段 ab 为基准,分别在其两侧进一步确定凸包中的其他点.以 ab 上侧为例,点 c 为距离 ab 最远点,将 c 加入凸包并由此形成有向线段 ac 和 cb.之后分别以 ac 和 cb 为基准线段,分别在它们外侧进一步寻找凸包中的点.

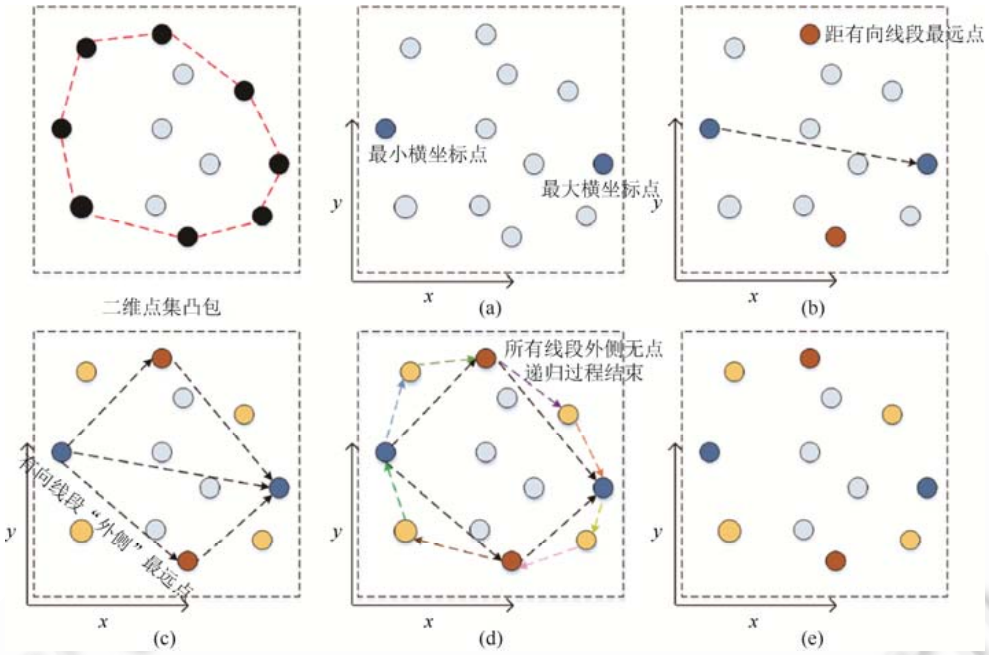


Fig.7 Demonstration of the recursive convex-hull algorithm

图 7 凸包问题的递归求解算法示意

给定一条基准线段和对应点集,将源任务定义为求在基准线外侧距该基准线段距离最远的点.以源任务为基础使用 SWAN 可轻易实现并行的凸包求解程序(附录 2).但由图 7 可以看到,在算法的初始阶段,由于有向基准线段的数量较少导致了程序的并行度很低.但在该阶段,少量的线程需要计算大量的点到基准线段的距离,因此形成了程序的性能瓶颈.针对这种情况,我们使用 SWAN 的任务挂起和上下文保存功能,使程序在初始阶段就派生诸多距离计算子任务以使更多的处理核心参与计算.在所有距离计算子任务结束后,由源任务进行归约产生目标点.如图 8 所示,在初始阶段提高并发度后,凸包求解程序的性能最终达到 MPE 串行版本性能的 23.6 倍.

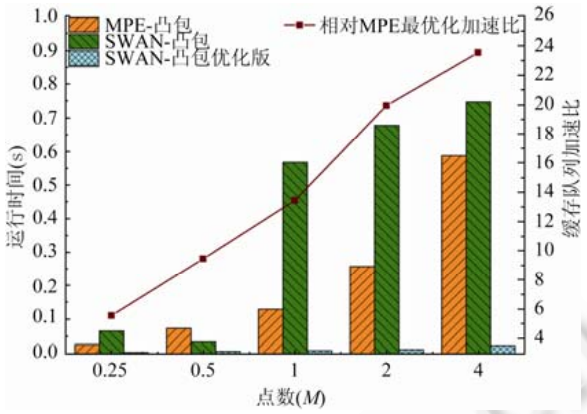


Fig.8 Speedup of the parallel convex-hull algorithm against the MPE sequential convex-hull algorithm

图 8 并行版凸包算法相对于 MPE 串行版算法的加速比

6 结论及未来的工作

在新兴的 SW26010 众核处理器上,本文提出并实现了支持任务并行模式的 SWAN 框架,并成功将之应用于若干典型的嵌套并行算法.在目标平台上,SWAN 框架为用户实现任务并行提供了高层次的抽象,能够大幅度降低用户开发任务并行程序的难度.在现有 CPE 功能的基础上,SWAN 能够挂起并恢复任务的执行,使得具有递归特性的嵌套并行算法能够得以有效并行.结合 SW26010 的访存特性,SWAN 框架中关键数据结构采用了高效的 DMA 访存机制和 LDM 缓存以有效地降低框架本身的执行开销.在并行程序执行过程中,SWAN 还能够确保任务在各个处理单元上的负载平衡以充分发挥众核阵列的计算效能.实验表面,在若干典型嵌套并行程序算例中,SWAN 能够有效加速目标程序,并随着问题规模的增大,加速效果更加明显.值得一提的是,通过对 SWAN 框架的灵活应用,我们可将集中在凸包问题初始阶段的大量计算负载均分到各个可用的 CPE 核心上,缩短程序执行的关键路径长度以大幅度提高程序的性能.

今后的工作将从两个方面展开.一方面,基于 SWAN 框架,我们将在 SW26010 众核核组上进一步研究各类嵌套并行算法,在此过程中设计新颖的动态负载均衡策略以提高程序的执行效率.另一方面,我们将进一步丰富和完善 SWAN 框架的功能,将基于任务有向无环图的任务调度技术整合于 SWAN 框架中以拓展其使用领域.

References:

[1] Wang L, Cui HM, Chen L, Feng XB. Research on task parallel programming model. Ruan Jian Xue Bao/Journal of Software, 2013, 24(1):77−90 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4339.htm [doi: 10.3724/SP.J.1001.2013.04339]

[2] An H, Chen GL. Parallel programming models and languages. Ruan Jian Xue Bao/Journal of Software, 2002,13(1):118−124 (in Chinese with English abstract). http://www.jos.org.cn/jos/ch/reader/create_pdf.aspx?file_no=20020117&year_id=2002&quarter_id=1&falg=1

[3] Ian F, Krishnaiyer R, Choudhary A. A library-based approach to task parallelism in a data-parallel language. Journal of Parallel & Distributed Computing, 1997,45(2):148−158.

[4] Blikberg R, Srevik T. Nested parallelism: Allocation of threads to tasks and OpenMP. Scientific Programming, 2001,9(2):185−194.

[5] Duran A, Teruel X, Ferrer R, Martorell X, AyguadE. Barcelona OpenMP tasks suite: A set of benchmarks targeting the exploitation of task parallelism in OpenMP. In: Proc. of the 2009 Int'l Conf. on Parallel Processing. Vienna: IEEE, 2009. 124−131.

[6] Wang QX, Sun SX, Shang MS, Liu YB. Research of parallel computing model. Computer Science, 2004,31(9):130−133.

[7] Blumofe RD, Joerg CF, Kuszmaul BC, Leiserson CE, Randall KH, ZhouY. Cilk: An efficient multithreaded runtime system. Journal of Parallel & Distributed Computing, 1996,37(1):55−69.

[8] Sun Q, Zhang CY, Wu CM, Zhang JJ, Li LS. Bandwidth reduced parallel SpMV on the SW26010 many-core platform. In: Proc. of the 47th Int'l Conf. on Parallel Processing. 2018. Article No.54.

[9] Ayguad E, Copty N, Duran A, Hoeflinger J, Lin Y, Massaioli F, Su E, Unnikrishnan P, Zhang G. A proposal for task parallelism in OpenMP. In: Proc. of the Int'l Workshop on OpenMP. Berlin: Springer-Verlag, 2010. 1–12.

[10] Mller MS, Baron J, Brantley WC, Feng H, Hackenberg D, Henschel R, Jost G, Molka D, Parrott C, Robichaux J. SPEC OMP2012—An application benchmark suite for parallel systems using OpenMP. In: Proc. of the Int'l Conf. on OpenMP in a Heterogeneous World. Berlin: Springer-Verlag, 2012. 223–236.

[11] Duran A, Corbaln J, Ayguad E. Evaluation of OpenMP task scheduling strategies. In: Proc. of the OpenMP in a New Era of Parallelism Int'l Workshop. West Lafayette: Springer-Verlag, 2008. 100–110.

[12] OpenMP 4.0 Specification. 2012. http://www.openmp.org/uncategorized/openmp-40/

[13] Blumofe RD, Leiserson CE. Scheduling multithreaded computations by work stealing. Parallel & Distributed Computing, 1997, 45(2):148–158.

[14] Leiserson CE. Programming irregular parallel applications in Cilk. In: Proc. of the Int'l Symp. on Solving Irregularly Structured Problems in Parallel. Berlin: Springer-Verlag, 1997. 61–71.

[15] Frigo M, Leiserson CE, Randall KH. The implementation of the Cilk-5 multithreaded language. ACM Sigplan Notices, 1999,33(5): 212–223.

[16] Cav V, Zhao J, Shirako J, Sarkar V. Habanero-Java: The new adventures of Old X10. In: Proc. of the 9th Int'l Conf. on Principles and Practice of Programming in Java. New York: ACM, 2011. 51–61.

[17] Berenbrink P, Friedetzky T, Goldberg LA. The natural work-stealing algorithm is stable. SIAM Journal of Computing, 2003,32(5): 1260–1279.

[18] Tardieu O, Wang H, Lin H. A work-stealing scheduler for X10's task parallelism with suspension. ACM Sigplan Notices, 2012, 47(8):267–276.

[19] Task Parallel Library (TPL). 2017. https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-parallel-library-tpl

[20] Addison C, LaGrone J, Huang L, Chapman B. Openmp 3.0 tasking implementation in OpenUH. In: Proc. of the Open64 Workshop. 2009. 1–10.

[21] Robison A, Voss M, Kukanov A. Optimization via reflection on work stealing in TBB. In: Proc. of the IEEE Int'l Symp. on Parallel and Distributed Processing. Miami: IEEE, 2008. 1–8.

[22] Chatterjee S, Grossman M, Sbrlea A, Sarkar V. Dynamic task parallelism with a GPU work-stealing runtime system. In: Proc. of the 24th Int'l Workshop on Languages and Compilers for Parallel Computing. Berlin: Springer-Verlag, 2013. 8–10.

[23] Cédric A, Samuel T, Raymond N, Pierre-André W. StarPU: A unified platform for task scheduling on heterogeneous multicore architectures. In Proc. of the Euro-Par 15th Int'l Conf. on Parallel Processing. LNCS 5704, Delft, 2009. 863–874.

[24] Tzeng S, Patney A, Owens JD. Task management for irregular parallel workloads on the GPU. In: Proc. of the ACM SIGGRAPH/ EUROGRAPHICS Conf. on High Performance Graphics. Saarbrücken: ACM, 2010. 29–37.

[25] Thread Building Blocks. 2021. https://www.threadingbuildingblocks.org/

[26] Copty N, Duran A, Hoeflinger J, Massaioli F, Massaioli F, Teruel X, Unnikrishnan P, Zhang G. The design of OpenMP tasks. IEEE Trans. on Parallel & Distributed Systems, 2009,20(3):404–418.

[27] Fu HH, Liao JF, Yang JZ, *et al*. The Sunway TaihuLight supercomputer: System and applications. SCIENCE CHINA: Information Sciences, 2016,59(7):1–16.

[28] Acar UA, Blelloch GE, Blumofe RD. The data locality of work stealing. Theory of Computing Systems, 2002,35(3):321–347.

[29] Hamidzadeh B, Lilja DJ. Dynamic scheduling strategies for shared-memory multiprocessors. In: Proc. of the Int'l Conf. on Distributed Computing Systems. Hong Kong: IEEE, 1996. 208–215.

附中文参考文献:

[1] 王蕾,崔慧敏,陈莉,冯晓兵.任务并行编程模型研究与进展.软件学报,2013,24(1):77−90. http://www.jos.org.cn/1000-9825/4339.htm [doi: 10.3724/SP.J.1001.2013.04339]

[2] 安虹,陈国良.并行程序设计模型和语言.软件学报,2002,13(1):118−124. http://www.jos.org.cn/jos/ch/reader/create_pdf.aspx?file_no=20020117&year_id=2002&quarter_id=1&falg=1

附录 1:基于 SWAN 框架实现的并行快速排序

在图 A 中,我们展示了使用 SWAN 框架编写的并行快速排序程序.图 A 中文件“Qsort_SWAN_MPE.c”和“Qsort_SWAN_CPE.c”分别记录了运行于 MPE 和 CPE 上的代码.红色高亮部分的语句是 SWAN 框架提供的 API.我们看到 SWAN 可以帮助用户清晰地表达程序逻辑,使得并行快速排序的整体代码结构与串行递归程序高度相仿.

Qsort_SWAN_MPE.c

```
1 #include "swan_qsort.h"
2
3 extern void slave_swan_partition_task(Swan *, void *);
4 void swan_qsort(Swan *swan, double* array, int length){
5   double *work_space = malloc(sizeof(double)*length);
6   PartitionArgNew argPartition
7           = partition_arg_new(work_space,array, length,0);
8   swan_MPE_insert_async_task
9           (swan, slave_swan_partition_task, argPartition);
10 swan_spawn(swan);
11 swan_join(swan);
12 free(qsortArg);
13 free(work_space);
14 }
```

Qsort_SWAN_CPE.c

```
1 #include "swan_qsort.h"
2
3 void swan_partition_task(Swan *swan, void *arg){
4   PartitionArg *arg_par = arg;
5   double *array = arg_par->src;
6   double *work_space = arg_par->work_space;
7   int len = arg_par->length;
8   int pivot_idx = arg_par->pivot_idx;
9   int nsmaller, nlarger_eq;
10  nsmaller = bipartition(array, work_space, len, pivot_idx);
11  nlarger_eq = len - nsmaller;
12  if(nsmaller < TASK_SORT_NELEM_THRES){
13    QsortArg *arg_sort = qsort_arg_new(array, nsmaller);
14    swan_CPE_insert_async_task(swan, swan_sort_task, arg_sort);
15  }
16  else{
17    PartitionArg *arg_par = partition_arg_new(work_space, array, nsmaller, 0);
18    swan_CPE_insert_async_task(swan, swan_partition_task, arg_par);
19  }
20  if(nlarger_eq < TASK_SORT_NELEM_THRES){
21    QsortArg *arg_sort = qsort_arg_new(&array[nsmaller+1], nlarger_eq-1);
22    swan_CPE_insert_async_task(swan, swan_sort_task, arg_sort);
23  }
24  else{
25    PartitionArg *arg_par = partition_arg_new(&work_space[nsmaller+1],
26                                              &array[nsmaller+1], nlarger_eq-1, 0);
27    swan_CPE_insert_async_task(swan, swan_partition_task, arg_par);
28  }
29 }
```

Fig.A Parallel quick sort programme using SWAN framework

图 A 用 SWAN 实现的并行快速排序程序

附录 2:基于 SWAN 实现的并行凸包求解算法

在图 B 中,我们展现了使用 SWAN 框架实现的并行凸包程序.与附录 1 中的并行快速排序类似,SWAN 帮助程序员并行化了凸包求解算法的递归结构.但值得注意的是,在图 B 展现的“SWAN_ Convex_Hull_CPE.c”文件中,为了提高程序的并行性,当前任务 slave_convex_hull_task 需要派生众多子任务去寻找距当前基准有向线段距离最远点(行 17).此时,父任务需挂起并等待所有子任务的完成.通过调用 SWAN 框架负责上下文保存及任务挂起的接口(行 14~行 19),该过程能够得以实现.在挂起任务之前,用户使用“swan_save_variable()”函数记录关键上下文信息(行 16).当该任务被重新执行时,程序控制流将从行 14 直接跳转到行 18 继续执行,并取回上下文信息(行 19).

SWAN_Convex_Hull_MPE.c

```c
#include "convex_hull_swan.h"

extern void slave_convex_hull_task(Swan *swan, void *task);
Point_set *g_ret;
swan_mutex_t *g_set_mutex;
Point_set* swan_convex_hull(Swan *swan, Point_set *src_set){
    CH_task_arg arg_1, arg_2;
    Point min_point, max_point;
    Point_set *ret = malloc(sizeof(Point_set));
    Point_set *set_upper = malloc(sizeof(Point_set));
    Point_set *set_lower = malloc(sizeof(Point_set));
    g_ret = ret;
    minmax_x_point(src_set, &min_point, &max_point);
    point_set_MPE_init(ret, src_set->length);
    point_set_MPE_init(set_upper,  src_set->length);
    point_set_MPE_init(set_lower,  src_set->length);
    point_set_split(src_set, set_upper, set_lower, min_point, max_point);

    g_set_mutex = swan_mutex_new();
    point_set_MPE_add(ret, min_point);
    point_set_MPE_add(ret, max_point);
    CH_task_arg_init(&arg_1, &min_point, &max_point, set_upper, 1);
    CH_task_arg_init(&arg_2, &max_point, &min_point, set_lower, 1);
    if(set_upper->length)
        swan_MPE_insert_async_task(swan, slave_convex_hull_task, &arg_1);
    if(set_lower->length)
        swan_MPE_insert_async_task(swan, slave_convex_hull_task, &arg_2);
    swan_spawn(swan);
    swan_join(swan);
    swan_mutex_destroy(g_set_mutex);
    point_set_MPE_destroy(set_upper);
    point_set_MPE_destroy(set_lower);
    return g_ret;
}
```

SWAN_Convex_Hull_CPE.c

```c
#include "convex_hull_swan.h"
extern Point_set  *g_ret;
extern int g_max_layer;
extern swan_mutex_t *g_set_mutex;

void slave_convex_hull_task(Swan *swan, void *arg){
  CH_task_arg *ch_arg = arg;
  Point *A, *B, *C, illegal_point = {ILLEGAL_CONDINATE, ILLEGAL_CONDINATE};
  Point_set *src_point_set = ch_arg->set;
  A = &ch_arg->A;
  B = &ch_arg->B;
  /*Find the farthest point according to the baseline*/
  /*Insert dependent child tasks and suspend*/
  swan_reenter_section_prelude(swan);
  C = swan_malloc(swan, sizeof(Point));
  swan_save_variable(swan, (Swan_byte*)&C, sizeof(Point*));
  spawn_task_find_farthest_point(swan, ch_arg, C, A, B);
  swan_reenter_section(); /*The task resumes from here*/
  swan_get_variable(swan, 0, (Swan_byte*)&C);
  /*Compact the resultant point into sets*/
  if(!point_CPE_is_equal(C, &illegal_point)){
    Point_set *set_1, *set_2;
    Point buf_1[POINT_SET_BUF_LEN];
    Point buf_2[POINT_SET_BUF_LEN];
    swan_mutex_lock(g_set_mutex);
    point_set_CPE_add(g_ret, C);
    sw_mutex_unlock(g_set_mutex);
    set_1 = swan_malloc(swan, sizeof(Point_set));
    set_2 = swan_malloc(swan, sizeof(Point_set));
    point_set_CPE_init(swan, set_1, src_point_set->length, buf_1, POINT_SET_BUF_LEN);
    point_set_CPE_init(swan, set_2, src_point_set->length, buf_2, POINT_SET_BUF_LEN);
    point_set_compact(src_point_set, set_1, A, C, set_2, C, B);
    if(set_1->length){
      CH_task_arg *arg_new_1 = swan_malloc(swan, sizeof(CH_task_arg));
      CH_task_arg_init(arg_new_1, A, C, set_1, cur_layer+1);
      swan_CPE_insert_async_task(swan, slave_convex_hull_task, arg_new_1);
    }
    if(set_2->length){
      CH_task_arg *arg_new_2 = swan_malloc(swan, sizeof(CH_task_arg));
      CH_task_arg_init(arg_new_2, C, B, set_2, cur_layer+1);
      swan_CPE_insert_async_task(swan, slave_convex_hull_task, arg_new_2);
    }
  }
}
```

Fig.B Parallel conve-hull programme using SWAN framework

图 B 使用 SWAN 实现的并行凸包程序

孙乔(1989－),男,博士,高级工程师,主要研究领域为并行编程模型,并行算法.

赵慧(1984－),女,博士,助理研究员,主要研究领域为高性能计算.

黎雷生(1981－),男,博士,副研究员,主要研究领域为并行计算.

吴长茂(1974－),男,博士,副研究员,CCF专业会员,主要研究领域为并行算法与并行软件,大规模渲染算法,异构平台数值计算.

赵海涛(1981－),男,博士,副研究员,CCF专业会员,主要研究领域为高性能工程,科学计算.