高技术通讯 2022 年 第32 卷 第9 期:927-936

doi:10.3772/j.issn.1002-0470.2022.09.006

面向神威·太湖之光的多核组协同的 OpenCL 编译方法①

伍明川②*** 刘 颖③* 李立民* 冯晓兵***

(*中国科学院计算技术研究所计算机体系结构国家重点实验室 北京 100190)

(**中国科学院大学 北京 100049)

摘 要 近年来,科学领域对高性能计算的需求与日俱增,如何有效利用新型超算架构的计算能力成为研究重点。我国自主研制的神威·太湖之光超算平台,采用了国产异构众核处理器 SW26010,其包含 4 个核组,但未提供核组间的同步机制。为了增加其易编程性,本文提出了面向神威·太湖之光的核组间同步方法,并在 SWCL OpenCL 编译器中实现了该核组间同步方法。该方法利用跨 OpenCL 主机内核的数据依赖分析来标识必要的同步操作位置,并通过 SW26010 的交叉段进行低开销的核组间通信,程序员在不使用消息传递接口(MPI)进行显式控制同步的情况下,可以自动地将一个 OpenCL Kernel 程序部署到多个核组上。使用 SPEC ACCEL 1.2 中的 OpenCL 测试用例在神威太湖之光平台的实验表明,本方法的加速效果明显优于传统的 MPI 实现版本。

关键词 OpenCL;国产众核处理器;异构;同步;数据依赖分析

0 引 言

现如今,气象建模[1-2]、地质建模和反演[3]、大气模拟和相场模拟[4]等科学领域对高性能计算的需求日益增高。同时,超级计算机的计算能力也随着新型架构(例如 GPU 和 Xeon Phi 等异构加速器)的发展而迅速增长。在这些功能强大的超级计算机中,于 2016 年投入使用的神威·太湖之光超级计算机是世界上第一个峰值性能超过 100 PFlops 的超算系统。该系统的核心组件是采用申威架构的 SW26010 异构多核处理器[5-6],与现有的图形处理器(graphic processing unit, GPU)和集成众核(many integrated core, MIC)在架构设计上有着较大的差异。因此,在神威·太湖之光上提供对主流异构并行编程框架(如 OpenCL[7])的支持,对其软件通用性以及易编程性的增强是至关重要的。

在 SW26010 处理器上建立以及优化 OpenCL 编译系统时,存在着一项重要的挑战,即 SW26010 处理器包含 4 个核组,但是硬件上并不提供跨核组的全局同步机制。SW26010 把 260 个核心组织到 4 个核组(core groups, CGs)中,每个 CG 均由一个管理核心(management processing element, MPE)和 64 个计算核心(computing processing element, CPE)组成,其提供的硬件同步只能对属于同一 CG 的核心有效。这与 GPU 不同,GPU 的所有计算单元(例如 Nvidia GPU 的流式多处理器 SM)均由同一管理核心(与其通过 PCIE 连接的通用处理器 CPU)管理,并且硬件同步对 SM 中的所有核心有效。因此,GPU 可以轻松地将 OpenCL Kernel 启动到 GPU 的所有计算单元。但是,对于 SW26010 而言,软件开发人员需要通过显式地插入 CG 间同步,来把 OpenCL Kernel 部署到多个 CG。

① 国家重点研发计划(2016YFB0200803),国家自然科学基金面上项目(61872043)和国家自然科学基金青年科学基金(61802368)资助项目。

② 男,1992 年生,博士生;研究方向:编译技术;E-mail: wumingchuan@ict.ac.cn。

③ 通信作者,E-mail: liuying2007@ict.ac.cn。

(收稿日期:2021-05-13)

近年来,学术界针对同步的识别和优化进行了研究。针对如何有效识别同步操作的问题,SherLock[8]使用无监督推理(unsupervised inference)的方法来识别同步,它使用精心设计的假设和同步属性,在反馈积累的帮助下,可以有效地识别各种类型的同步。但是 SherLock 只能分析 C#等同构程序,并不支持 OpenCL 等异构程序。针对核间同步的优化研究工作中,pLock[9]是利用 SW26010 的核间通信机制将同步操作转换为异步通信,即设计一个或者几个 CPE 作为服务器并通过消息传递将相同数据的更新操作发往同一服务器,使得存在数据竞争的操作可以在服务器上序列化。但是 pLock 只能处理单一核组内的同步操作,并不支持多个核组间的同步。文献[10]则是通过 GPU 上的事务内存(transactional memory)来保持一致性进而提高 GPU 上同步操作的性能。由于申威处理器的 4 个核组是异步控制且核组间没有硬件同步机制,因此该研究并不适用于申威处理器的核组间同步。

针对这项挑战,本文对于异步控制的核组,自动插入核组间同步,从而在多个核组之间分配一个 OpenCL Kernel,并且通过对主从核(Host-Kernel)间数据依赖性的分析,将插入的同步次数最小化。具体来说,首先提取跨核组共享的数据并将其分配到虚拟共享内存空间,然后利用参数制导的过程间分析来解析跨主机内核的数据依赖关系,来确定需要在核组间插入同步操作的位置。最后,自动生成绑定到 4 个管理核心的 Host 代码和相应的消息传递接口(message passing interface, MPI)进程(1 个主 MPI 进程和 3 个影子 MPI 进程)。因此,本文实现了将一个 OpenCL Kernel 程序部署到多个核组,并且不需要程序员使用 MPI 对其进行显式控制的功能。

1 相关工作

1.1 神威·太湖之光超级计算机

神威·太湖之光超级计算机采用国产自主研发的异构众核处理器:SW26010[6-7],其架构如图 1 所示。每个 SW26010 处理器都拥有 4 个核组(CGs),每个核组都包含 1 个管理核心(MPE)、1 个运算核心簇(computing processing element cluster,CPE cluster)和 1 个 DDR3 内存控制器(memory controller,MC),其中每个运算核心簇包含 64 个运算核心(CPE),簇内的 CPE 之间采用拓扑结构为 8×8 阵列的簇通信网络进行连接。CG 间由片上网络连接起来,每个 SW26010 处理器都包含一个系统接口(system interface,SI),用于连接片外系统。

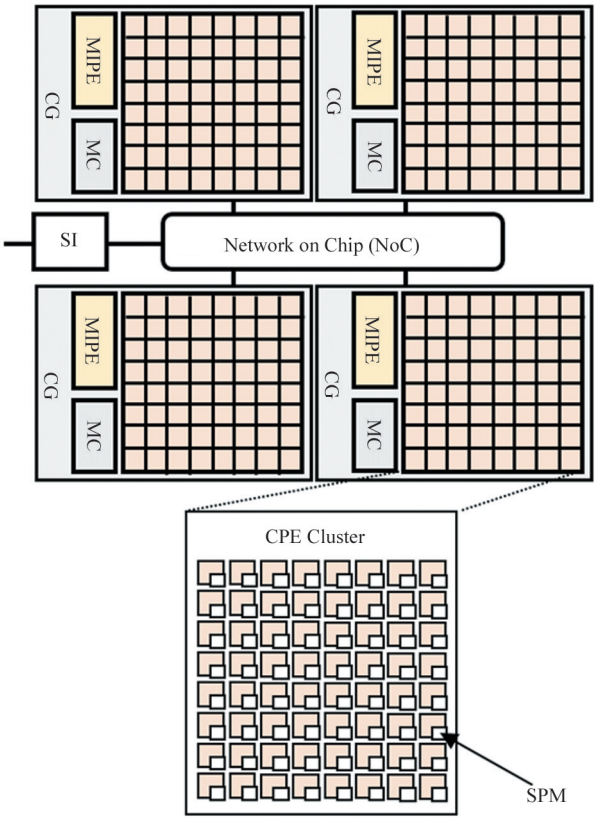

图 1 SW26010 处理器架构

SW26010 的 MPE 和 CPE 都是字长 64 位的全功能精简指令集(reduced instruction set computer, RISC)核心。其中 MPE 采用两级片上存储层次结构,包括 32 kB 的一级数据缓存(L1 data cache)和 256 kB的二级指令数据共享缓存(L2 instruction/data cache)。而 CPE 只包含 16 kB 的 L1 指令缓存和 64 kB 的便笺式存储器(scratchpad memory, SPM)。CPE 的片上存储 SPM 也被称为局存(local data memory,LDM),拥有 L1 数据缓存的速度,但是其数据存储的组织直接开放给了软件管理。这种方式简化了传统缓存的控制开销,可以实现空间的精确使用。SPM 可以通过直接存储器访问(direct memory

access, DMA)与主存进行数据移动,每个 CPE cluster 的带宽大约为 30 GB/s(SW26010 处理器的总内存带宽为 136 GB/s),而 CPE 的寄存器也可以通过全局加载/存储指令(gload/gstore)和主存进行数据传输,但这种方式拥有 177/278 周期的高延迟。此外,SW26010 引入了一种新的寄存器通信机制,使得同一行或同一列中的 CPE 可以相互通信。

在编程环境方面,大多数情况下每个 CG 对应一个 MPI 进程。在每个 CG 中,都可以使用 Sunway OpenACC 或 Athread 进行从核加速,其中 Athread 是专门为申威架构设计的轻量级加速线程库,它也是 Sunway OpenACC 的基础实现。与 Pthread[11] 类似,Athread 提供了用于创建/加入 CPE cluster 和 DMA 传输的应用程序接口(application programming interface, API),用于管理 CPE cluster 的计算和访存。

1.2 SW26010 的 OpenCL 编译器 SWCL

OpenCL 作为用于异构架构的统一并行编程框架,提供了独立于平台的抽象平台模型(platform model)、使程序员可以设计控制计算和数据引用的执行模型(execute model)和内存模型(memory model)[5]。

OpenCL 平台模型由配备有多个 OpenCL 设备(Device)的主机(Host)组成,每个设备被划分为多个计算部件(compute unit,CU),这些计算部件又被划分为多个处理单元(processing element,PE)。在神威·太湖之光上,之前的工作设计并实现了支持 SW26010 的 OpenCL 编译系统:SWCL[12-13],该编译系统把一个 CG 的 MPE 映射为 Host,并将相应的 CPE cluster 映射为 OpenCL Device。

OpenCL 执行模型定义内核代码(Kernel)在多个 OpenCL 设备上执行,而主机代码(Host Program)则在主机上运行。在提交 Kernel 以供执行时,OpenCL 需要定义一个索引空间(NDRange),其中每个 Kernel 线程都是在一个 PE 上运行的工作项(work item)。多个工作项可以被组织为一个工作组(work group),每个工作组都在 CU 上运行。根据 SWCL 的平台模型映射关系,SWCL 用过一种串行执行工作组中所有工作项的方式在 CPE cluster 上执行每个 Kernel 程序,并在一个 CPE 上执行多个工作组,即 SWCL 把所有工作组静态分配给每个 CPE,从而使得每个 CPE 通过嵌套循环的方式串行执行所分配的工作组。这种嵌套循环的方法被多种 OpenCL 编译器采用[14-15],因此 SWCL 同样拥有处理同步功能(barrier)和工作项专用变量问题的功能。

OpenCL 内存模型定义了 2 个内存区域,即主机内存(host memory)和设备内存(device memory),它们分别用于 Host 代码和 Kernel 代码。其中,device memory 由全局/常量内存(global/constant memory)、局部内存(local memory)和私有内存(private memory)组成,其中 global/constant memory 在所有工作项之间共享,local memory 在一个工作组中共享,而 private memory 仅由一个工作项使用。基于此,SWCL 将 host memory 和 global/constant memory 映射到主存,将 local memory 映射到 CPE 独有的 SPM 上,并将 private memory 映射为 CPE 的寄存器。

与 Sunway OpenACC 类似,SWCL 将 OpenCL 源代码转换为调用 Athread 加速线程库的C/C++代码,再通过神威·太湖之光的本地编译器 SWCC 得到可执行文件。

2 核组间同步生成方法

如 1.1 节所述,SW26010 处理器由 4 个异步控制的核组组成,每个核组具有独立的内存地址空间,并且核组间没有硬件同步机制。因此,程序员必须采用“MPI+X”的模式在一个处理器上对所有的核组进行编程[16]。其中 MPI 主要用于核组之间的显式通信和同步,X(例如 OpenMP、OpenACC、OpenCL、Athread 等)则用于编写一个执行在加速设备(即核组内的 CPE cluster)上的 Kernel 代码。

但是,OpenCL 所定义的模型是在全局内存上分配所有的加速设备。为了弥补 OpenCL 模型和 SW26010 处理器架构之间的差异,本文提出了面向神威·太湖之光的核组间同步生成方法,并基于此方法设计实现了核组间同步生成器,使其能为 SW26010 处理器上的所有核组分配一个 OpenCL Kernel。

核组间的同步生成方法有 2 个关键点：首先，为了便于编程，需要通过编译器自动插入核组间的同步语句，并且通过跨主机内核的数据依赖分析来确保不会插入不必要的同步语句。至此，程序员将无需在编写 OpenCL 的同时还显式编写 MPI 代码。其次，为了降低通信开销，本文利用 SW26010 处理器所提供的虚拟共享内存（即内存交叉段，memory intersection）进行核组间通信，以实现核组间的数据传输。

SW26010 处理器在虚拟内存地址空间中提供了 3 个区域，即线程专用内存，同核组上线程共享的内存和内存交叉段。其中内存交叉段是 SW26010 处理器架构所提供的特殊区域，可以由处理器上运行的所有进程共享。本文利用此区域为 OpenCL Kernel 生成核组间的同步语句，并且在生成通信代码时解决数据竞争问题。

核组间同步生成器的工作流程如图 2 所示。首先，它通过跨主机内核数据依赖分析确定必要的核组间同步位置，以确保由不同核组发起的并发读/写正确性，从而避免潜在的数据竞争（如“识别并插入核组间同步”方框所示）。其次，SWCL 会自动生成用于将 Kernel 分配给多个核组的代码，它会将 MPI 进程组分为一个主 MPI 进程和其他的影子 MPI 进程。每个 MPI 进程都绑定到一个 MPE 上，并在交叉段上分配 Kernel 的参数对象（如“代码生成”框所示）。最后，再把标识的核组间同步语句插入到生成的 4 核组代码里。

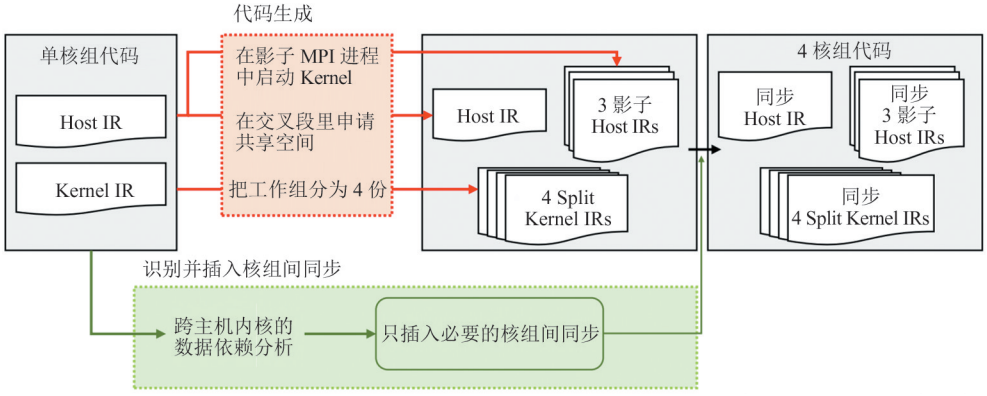

图 2　核组间同步生成器框架图

2.1　识别必要的核组间同步

2.1.1　潜在的数据竞争（data race）

对于 OpenCL 程序而言，如果在不同的工作项或主机程序中所包含的 2 个操作满足下述条件，将产生数据竞争问题。条件包括（1）一个操作对某一内存位置进行了修改，而另一个操作读取或修改了相同的内存位置；（2）这些操作中至少有一个不是原子操作，或者对应的内存作用域不是包含性的；（3）这些操作是不满足全局先行发生（global-happens-before）关系的全局操作，或者是不满足局部先行发生（local-happens-before）关系的局部操作[5]。例如有 2 个操作 A 和 B，如果 A 是全局先行发生/局部先行发生于 B 的，即表示 A 对全局/局部内存所做的任何写操作，都是对 B 可见的。

当将没有数据竞争的 OpenCL 程序分发到多个核组并发执行时，可能会引入 2 种类型的数据竞争：（1）Kernel 代码中的数据竞争，即由多个核组并发执行的工作项引起的数据竞争；（2）Host 代码和 Kernel 代码之间的数据竞争，即由并发执行的主 MPI 进程和其他核组上执行 Kernel 代码的工作项引起的。产生的原因是无法保证 OpenCL 中某些操作的先行发生语义（happens-before semantic），例如原子操作和具有 memory _ scope _ device 的同步操作，诸如 clFinish( ) 和 clEnqueueNDRangeKernel( ) 之类的 API 函数。因此，需要识别并添加必要的核组间同步操作，用以消除潜在的数据竞争，来保证上述特定操作的内存顺序。

图 3（b）和（c）展示了为图 3（a）生成核组间同

步时可能出现的 2 种数据竞争类型。其中,如果 A 对于 B 是全局先行发生的并且它们不是来自同一工作组或同一 MPI 进程,那么当这些操作在不同的核组上执行时,可能会违反全局先行发生的关系,即当 0 号核组中的 M 操作和 n 号核组的 P 操作同时执行,就会在 Kernel 代码中发生数据竞争。而当主 MPI 进程中的 N 操作和 n 号核组中的 M 或 P 操作同时执行时,则会出现 Host 和 Kernel 代码之间的数据竞争问题。

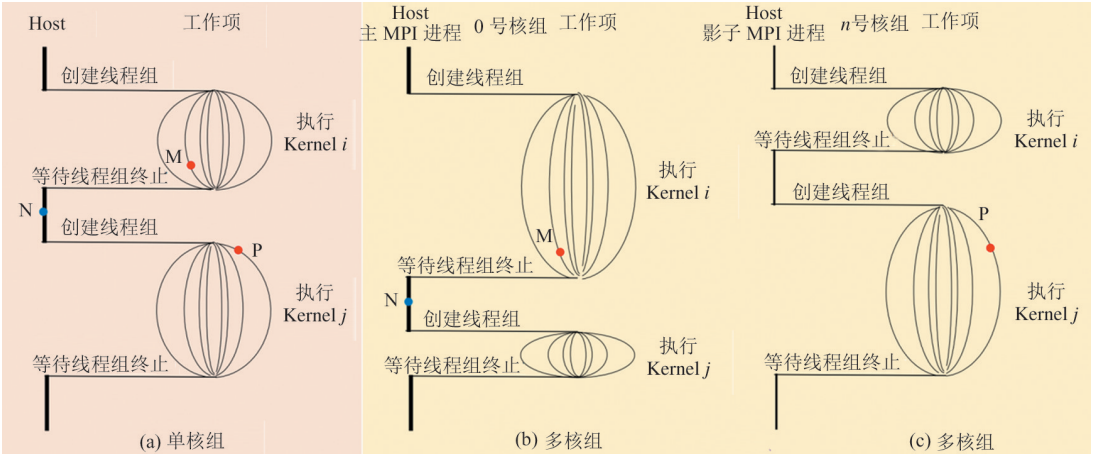

图 3　将 Kernel 分发给所有核组时,由于违反了 M、N 和 P 中任意两个之间的全局先行发生关系而产生的数据竞争问题

2.1.2　确保先行发生语义

如果 OpenCL 程序包含一些能够确保先行发生语义的操作,那么在跨多个核组分发 OpenCL Kernel 时,需要特别注意以下 2 类操作:

(1)加速设备上的操作(device-side actions)。在 OpenCL Kernel 中,使用 memory _ scope _ device 的同步语句或原子操作,都暗含了全局先行发生语义。因此,本文实现了软件的全局同步操作和原子操作,以实现所有核组的全局先行发生关系。其中,全局同步操作是通过使用核组间信号和核组内同步控制函数(athread _ syn)来实现的。全局原子操作则是使用锁和原子指令来实现。实现细节可参考第 2.2 小节。

(2)主机端命令(host-side commands)。在 Host 代码里,可以显式调用一组 OpenCL API 来控制 Kernel 的调用顺序,包括 Kernel 入队/结束(enqueue/finish)和基于事件的执行依赖关系。由于存在多个核组,本方法需要增强这些命令的功能,用以同时控制所有的核组。例如入队 API clEnqueueNDRangeKernel()需要同时将 Kernel 分发给所有核组并启动执行,其中每个 Kernel 都访问特定的内存区域,而结束 API clFinish()则需要等待所有核组上的 Kernel 程序执行结束。此外,可以利用跨主机内核的数据依赖分析来消除不必要的同步等待操作。

2.1.3　跨主机内核的数据依赖分析

确保先行发生语义的直接方法是同时控制所有核组 OpenCL Kernel 的启动和结束。但是,当 2 次 Kernel 执行之间没有依赖关系时,这种方法会引入不必要的同步等待操作。为此,本文提出了跨主机内核的数据依赖分析方法,来消除不必要的同步等待操作。

跨主机内核分析主要用于提取 Host 和 Kernel 代码之间的 RAW、WAR 和 WAW 依赖关系。本文把一个依赖关系表示为(src _ bb, dst _ bb),其中 src _ bb 或者 dst _ bb 是具有内存对象读/写操作的主机基本块(host basic block,Host BB),或者是 Kernel 启动基本块(kernel launch BB)。具体来说,如果 src _ bb 和 dst _ bb 中至少有一个是 Kenrel 启动基本块,那么依赖关系是跨主机内核的,本文将其分为 3 类,即(1)单内核依赖关系(one-kernel dependence),那么 src _ bb 和 dst _ bb 是同一个 Kernel 启动基本块,这意味着它是位于循环嵌套中的 Kernel;(2)多内核依赖关系(kernel-kernel dependence),即其中 src _ bb 和 dst _ bb 是不同的 Kernel 启动基本

块;(3)主机内核依赖关系(host-kernel dependence),即 src _ bb 和 dst _ bb 中只有一个是 Kernel 启动基本块。

算法 1　跨主机内核依赖分析

1: PROCEDURE *HKDATADEP* (KernelLaunchBB k)
2: $kernel \leftarrow k$.callee()
3: $bits_{waw} \leftarrow$ Ctx(k)& Sum($kernel$).MOD
4: $set_{waw} \leftarrow$ CtxLoc(k).get($bits_{waw}$)
5: $set_{raw} \leftarrow$ Sum($kernel$).REF
6: $set_{war} \leftarrow bb$.users() for $\forall bb \in set_{waw}$
7: Dep_i.add(bb, k) for $\forall bb \in set_i (i = waw, raw, war)$
8:END PROCEDURE

为了检测这 3 种类型的跨主机内核依赖关系,本文提出了算法 1,该算法遍历 Host-Kernel 融合后的中间表示(intermediate representation, IR),并为所有的 Kernel 启动基本块创建集合,并使用以下规则来分析调用节点上的数据依赖关系:

如果基本块 k 里存在对变量 M 的定值或使用,基本块 bb 中有对 M 的写操作,当 bb 里 M 的定值可以到达 k,那么 M 的 WAW/RAW 依赖为(bb, k)。

如果基本块 k 包含了对变量 M 的定值,基本块 bb 里包含对 M 的读操作,当在 bb 中 M 配对的定值可以到达 k,则存在一个 M 的 WAR 依赖(bb, k)。

表 1 展示了对图 4 代码进行跨主机内核数据依赖分析后的结果。首先分析基本块 B12,对 Kernel 函数 k2 而言,可以得到 Workset(k2) = {T,S}, Sum(k2).MOD = 01 和 Sum(k2).REF = 11。对调用节点 B12 而言,可以得到 Ctx(B12) = 11 以及 CtxLoc(B12) = {{B16, B3, B19}, {B8}}。因此,本方法可以得到主机内核依赖关系(B16, B12)以及(B3, B12),多内核依赖关系(B19, B12)。

```
    cl_mem A;
    main ()
    {
     cl_mem B;
B1: (void *)p=(void *)&B;

B2: if(…) f();
     else
B3: clEnqueueWriteBuffer(A,…);

B4: g(p);

B5: if(…) p=(void *)&A;
B6: h(p);

B7: clEnqueueReadBuffer(A,…);
    }
```

```
    h(void *y)
    {
     cl_mem C;
B8:  clEnqueueWriteBuffer(C,…);
B9:  (void *)z=cond?y:(void*)&C;

B10: clSetKernelArg(k2,0,y);
B11: clSetKernelArg(k2,1,&C);
B12: clEnqueueNDRangeKernel(k2);

B13: clSetKernelArg(k1,0,z);
B14: clSetKernelArg(k1,1,&A);
B15: clEnqueueNDRangeKernel(k1);
    }
```

```
    f() {
B16: clEnqueueWriteBuffer(A,…);
    }
```

```
    g(void *x)
    {
B17: clSetKernelArg(k1,0,&A);
B18: clSetKernelArg(k1,1,x);
B19: clEnqueueNDRangeKernel(k1);
    }
```

```
__kernel k1(__global int *M,N)
{
  if(…)
    N[tid]=M[tid]…;
}
```

```
__kernel k2(__global int *T,S)
{
  S[tid]=S[tid]+T[tid]…;
}
```

图 4　OpenCL 示例代码

表 1　图 4 的数据依赖分析

Kernel 执行基本块 k	B19 (g → k1)	B12 (h → k2)	B15 (h → k1)
set_{waw}	Ø	{B16, B3, B19}	{B16, B3}
set_{raw}	{B16, B3}	{B16, B3, B19, B8}	{B16, B3, B19, B12}
set_{war}	Ø	{B19}	{B19}

借助跨主机内核的数据依赖分析,遍历每个 Kernel 执行基本块 k,来决定是否需要在 Kernel 启动前后(即 kernel:entry()之前和 kernel:exit()之后)进行同步操作。因此,只有在基本块满足以下条件时,才有必要在启动内核之前进行同步:(1)存在依赖关系(bb,k);(2)sync 是出现在从 bb 到 k 的每个路径上的唯一同步。而有必要在内核执行之后进行同步的条件为:(1)存在依赖关系(k,bb);(2)sync 是出现在从 k 开始的每个路径上的唯一同步操作。图 5 展示了对 3 种依赖关系进行分析识别后得到的必要同步操作。

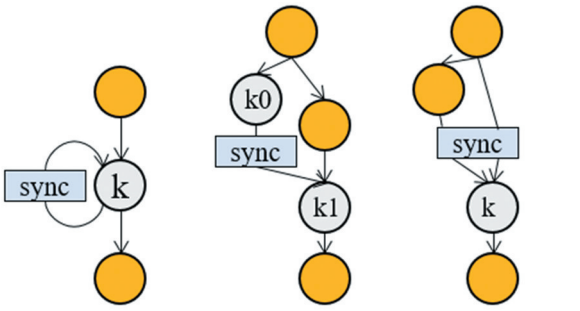

图 5　基于跨主机内核的依赖分析来标识必要的同步操作

2.2　代码生成

对于主机代码,需要自动生成 3 部分的主机代码段,即用于访问内存交叉段、管理 MPI 进程和维护同步操作的代码段。首先,需要在主机 0 号 MPI 进程中插入 cl _ mem 对象的数据分配操作,并且控制只有主 MPI 进程才能访问这些对象。其次,其他的 MPI 进程(即影子 MPI 进程)同样绑定到每个核组里的 MPE 上,主要用来在各个核组内的运算核心簇上启动执行 Kernel 代码。最后,通过 2.1 节的分析,在各种主机代码上插入必要的同步操作。其中,OpenCL 的 Kernel 启动执行 API 可以使用 athread 加速线程库来实现。在 MPE 上运行的每个 MPI 进程都通过 athread 库中创建线程组函数(athread _ spawn)将 Kernel 分发到相应的 CPE 上。同时,在分析后确认的必要位置生成显示阻塞等待线程组终止函数(athread _ join)或者核组间同步函数(athread _ master _ sync)来执行同步操作。

对于 Kernel 代码,把工作组循环进行静态拆分,使工作组均匀地分布到所有核组上。此外,针对 OpenCL Kernel 中用于处理原子和 barrier 操作的 memory _ scope _ device,通过模拟跨核组的全局原子和 barrier 操作来实现。具体来说,全局原子操作是通过原子指令和对交叉段中的数据锁操作来实现的,因此确保了只有一个 CPE 可以访问数据,直到完成处理为止。而全局同步操作则通过使用每个核组中每个 CPE 所发出的信号,并配合核组内同步控制函数(athread _ syn)来实现同步每个 CPE cluster 的功能。

3　实　验

3.1　实验平台与测试用例

为了验证本文提出方法的有效性,在 SWCL[12-13] 中实现了跨核组同步方法,并在神威·太湖之光超算平台上进行实验,使用神威本地编译器 sw5cc(-O3)对生成的代码进行编译链接。实验使用 SPEC ACCEL[17] 1.2 中的 15 个 OpenCL 程序作为测试用例,运行时间记录为 Host 代码和 Kernel 代码的总执行时间。SPEC ACCEL 1.2 包含 19 个 OpenCL 用例,其中 4 个用例(bfs、histo 等)未被选取的原因是,它们包含原子操作,部署在多个核组上并不会获得比部署在单个核组上更好的性能。本文将基于这 15 个 OpenCL 程序进行常规的 MPI 扩展(即 MPI 版本),从而实现启用 4 个核组(包括各个核组的从核阵列)的功能。

3.2　实验结果与分析

为了验证本方法的性能,本文使用 SW26010 处理器的全部 4 个核组进行实验。如第 2 节所述,SW26010 处理器的 4 个异步控制的核组各自具有独立的内存地址空间,并且硬件没有核组间的同步机制。因此,采用 MPI 在一个处理器上对所有的核组进行编程成为了主流,其中 MPI 就是用于核组之间的显式通信和同步的主要手段。所以,本文使用 MPI 版本作为对照组,和在 SWCL 上实现的跨核组同步生成器进行比较。以单核组版本的时间作为基准值来计算 2 个版本的加速效果,实验结果如图 6 所示,MPI 版本在 15 个 OpenCL 程序上获得了平均 1.78 倍的加速效果(取几何平均值),而本文方法(即 SWCL-4CG)平均能获得 2.21 倍的加速效果,明显好于 MPI 版本。

本文方法的收益主要来源于两个方面。一方面来自跨主机内核数据依赖分析所减少的不必要同步操作。如 2.1 节所示,通过跨主机内核的数据依赖分析,本方法能识别出必须插入同步的位置。常规的 MPI 版本则只会对 Host 代码做数据依赖分析,为了保证程序的正确性,在不知道 Kernel 代码内数据

依赖关系的情况下会对每个 athread _ spawn/join 前后都插入同步操作函数。这种实现方法会出现多余的同步操作,造成性能的下降。其中在 hotspot、nw、stencil 等实验用例上效果明显,本方法相比于 MPI 版本分别有 1.08 倍、1.1 倍和 1.01 倍的加速效果。

本方法效果更好的第二个原因是交叉段的使用。为了降低通信成本,本文利用 SW26010 架构所提供的交叉段来实现跨核组的数据传输。由于交叉段上的数据对同片 4 个核组共享,4 个核组可以直接对其进行数据传输操作,进而减少了 MPI _ send/receive 所带来的额外开销。对此,本文针对 MPI 的通信开销进行了实验,对 15 个测试用例的 MPI 通信时间进行统计(即 MPI _ send 和 MPI _ receive 时间),并以 MPI 版本作为基准进行比较,结果如图 7 所示。

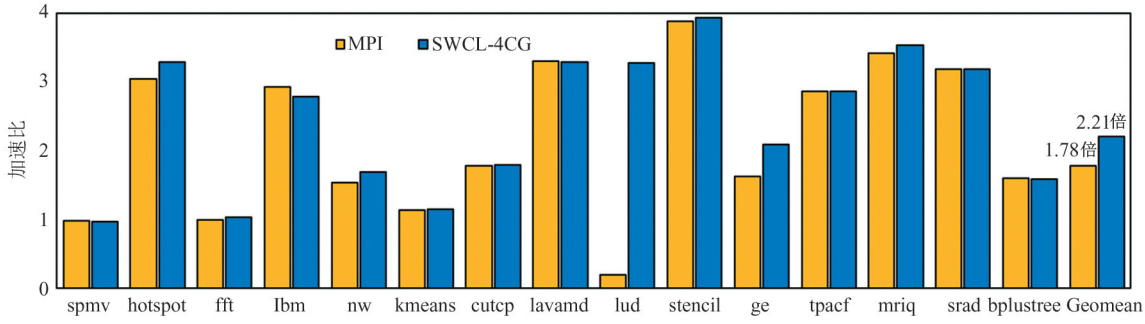

图 6　本方法在 4 核组上执行的加速效果

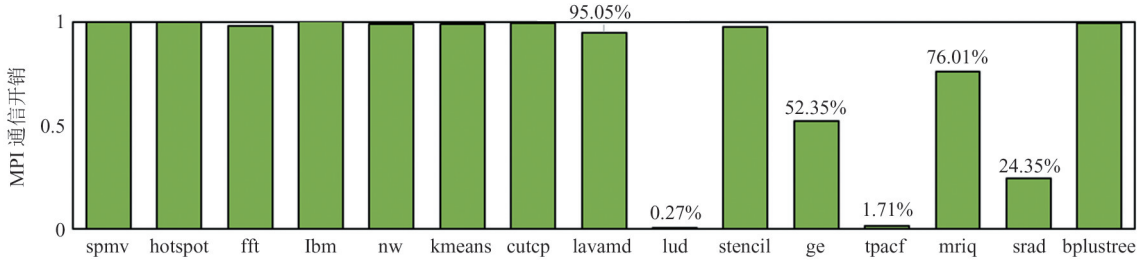

图 7　MPI 通信开销对比

实验表明,使用交叉段可以在 lavamd、lud、ge、tpacf、mriq 和 srad 等需要大量核组间数据通信的用例中取得效果。由于使用交叉段减少了 MPI 通信开销,本方法相比于 MPI 版本在 lud、ge 和 mriq 上分别可获得 15.89 倍、1.28 倍和 1.03 倍的加速比。

本文方法能在不需要程序员使用 MPI 对其进行显式控制的情况下,将一个 OpenCL Kernel 程序部署到多个核组。为了证明功能的有效性,在同一个 SW26010 处理器上,分别启动 1 个核组、2 个核组、3 个核组进行测试,并与 MPI 版本、4 个核组版本进行对比,结果如图 8 所示。本实验以单核组版本(SWCL-1CG)的时间作为基准值来计算各个版本

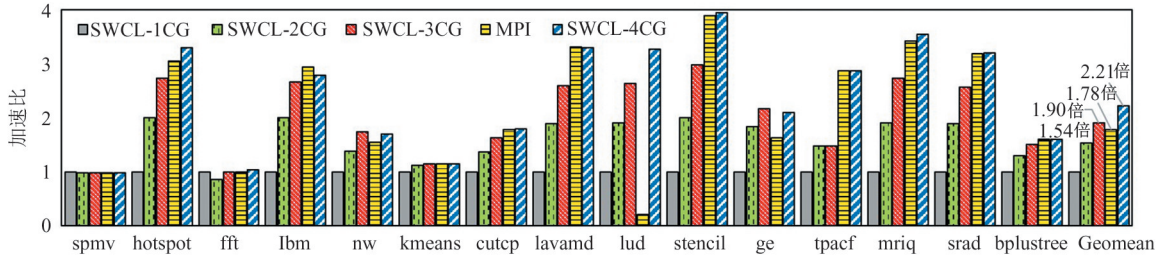

图 8　多核组的加速效果对比

的加速效果,启动 2 个核组(SWCL-2CG)可获得 1.54 倍的加速比,启动 3 个核组(SWCL-3CG)加速比达到 1.90 倍。实验证明本方法能发挥申威架构多核组的性能优势。

4 结 论

本文介绍了面向神威·太湖之光的核组间同步方法。为了提升申威平台的易编程性,在 SWCL 编译器中实现了自动插入核组间的同步语句,并利用跨主机内核的数据依赖分析来确保不会插入不必要的同步语句。同时,为了降低通信成本,该方法利用 SW26010 架构所提供的虚拟共享内存进行核组间通信,以实现跨核组的数据传输。

在神威·太湖之光超算平台上证明了本文方法的有效性。利用 SPEC ACCEL 1.2 的 15 个 OpenCL 测试用例进行实验,其结果表明本文方法的加速效果明显优于传统的 MPI 实现版本。实验结果表明,交叉段可以明显减少 MPI 通信开销,加速比可高达 15 倍,充分发挥了申威架构多核组的性能优势。

参考文献

[1] JOHNSEN P, STRAKA M, SHAPIRO M, et al. Petascale WRF simulation of hurricane Sandy deployment of NCSA's cray XE6 blue waters[C]//International Conference on High Performance Computing, Network, Storage and Analysis, New York, USA, 2013: 1-7

[2] FU H, LIU W, WANG L, et al. Redesigning CAM-SE for peta-scale climate modeling performance and ultra-high resolution on Sunway TaihuLight[C]//The International Conference on High Performance Computing, Networking, Storage and Analysis, New York, USA, 2017: 1-12

[3] RUDI J, MALOSSI A, ISAAC T, et al. An extreme-scale implicit solver for complex PDEs: highly heterogeneous flow in earth's mantle (ACM Gordon Bell Prize Winner 2015)[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Austin, USA, 2015: 1-12

[4] JIAN Z, ZHOU C, WANG Y, et al. Extreme-scale phase field simulations of coarsening dynamics on the Sunway TaihuLight supercomputer[C]//International Conference for High Performance Computing, Salt Lake City, USA, 2016: 1-12

[5] FU H H, LIAO J F, YANG J Z, et al. The Sunway TaihuLight supercomputer: system and applications[J]. *Science China: Information Science*, 2016, 59(7): 109-124

[6] DONGARRA J. Report on the Sunway TaihuLight System [J]. *UT EECS Technical Reports*, 2016:1-24

[7] OpenCL, Khronos Group. The open standard for parallel programming of heterogeneous systems[EB/OL], https://www.khronos.org/opencl/: Khronos,[2021-05-13]

[8] LI G, CHEN D, LU S, et al. SherLock: unsupervised synchronization-operation inference[C]//Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, New York, USA, 2021: 314-328

[9] TANG X, ZHAI J, QIAN X, et al. pLock: a fast lock for architectures with explicit inter-core message passing. [C]//2019 Architectural Support for Programming Languages and Operating Systems, New York, USA, 2019: 765-778

[10] REN X, LIS M. High-performance GPU transactional memory via eager conflict detection[C]//2018 IEEE International Symposium on High Performance Computer Architecture, Vienna, Austria, 2018:235-246

[11] BARNEY B. POSIX Threads Programming[EB/OL]. https://hpc-tutorials.llnl.gov/posix/:Lawrence Livermore National Laboratory,[2021-05-13]

[12] 伍明川, 黄磊, 刘颖, 等. 面向神威·太湖之光的国产异构众核处理器 OpenCL 编译系统[J]. 计算机学报, 2018, 41(10):2236-2250

[13] WU M C, LIU Y, CUI H M, et al. Bandwidth-aware loop tiling for DMA-supported scratchpad memory[C]//Proceedings of the 2020 International Conference on Parallel Architectures and Compilation Techniques, Virtual, 2020: 97-109

[14] LEE J, KIM J, SEO S, KIM S et al., An opencl framework for heterogeneous multicores with local memory[C]//Proceedings of the 19th ACM/IEEE/IFIP International Conference on Parallel Architectures and Compilation Techniques, Vienna, Austria, 2010: 193-204

[15] JAASKELAINEN P, LAMA C S, SCHNETTER E, et al.

Pocl: a performance-portable OpenCL implementation [J]. *International Journal of Parallel Programming*, 2015,43(5): 752-785

[16] 段晓辉. 基于"神威·太湖之光"的分子动力学算法优化[D]. 济南:山东大学计算机学院, 2020

[17] JUCKELAND G, BRANTLEY W C, CHANDRASEKARAN S, et al. 2014. SPEC ACCEL: a standard application suite for measuring hardware accelerator performance [C]// Proceedings of the 5th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, Austin, USA, 2015: 46-67

An inter-CG collaborative OpenCL compilation method on the Sunway TaihuLight supercomputer

WU Mingchuan[*,**], LIU Ying[*], LI Limin[*], FENG Xiaobing[*,**]

([*] State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**] University of Chinese Academy of Sciences, Beijing 100049)

Abstract

In recent years, demands for high performance computing has been increased significantly in various scientific domains. How to effectively utilize the computing power of the new supercomputing architecture has become a research focus. The homegrown Sunway TaihuLight supercomputer adopts the homegrown heterogeneous many-core processor SW26010. In order to efficiently use the computing power of the four core groups on the SW26010 and reduce the difficulty of programming, an inter-CG (core group) synchronization generation method on the Sunway TaihuLight is proposed, and the inter-core synchronization generator based on SWCL OpenCL is designed and implemented. This method proposes data dependency analysis across OpenCL host and kernel to identify the necessary synchronization operation, and uses memory intersection of SW26010 to communicate between core groups, which reduces communication overhead and ensures that programmers do not need to use the message passing interface (MPI) for explicit control synchronization. In this case, one OpenCL Kernel program is automatically deployed to multiple core groups. Experiments are carried out using the OpenCL test cases in SPEC ACCEL 1.2, and the results show that the acceleration effect of this method is significantly better than the traditional MPI implementation version.

Key words: OpenCL, homegrown many-core processor, heterogeneous system, synchronization, data dependency analysis