软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(12):2782−2796 [doi: 10.3724/SP.J.1001.2013.04386]

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

异构多核上多级并行模型支持及性能优化*

李士刚[1], 胡长军[1], 王 珏[2], 李建江[1]

[1](北京科技大学 计算机与通信工程学院,北京 100083)

[2](中国科学院 计算机网络信息中心 中国科学院超级计算中心,北京 100190)

通讯作者: 李士刚, E-mail: lsgwoody@gmail.co; 王珏, E-mail: wangjue@sccas.cn

摘 要: 低功耗及廉价性使得异构多核在超级计算机计算资源中占有重要比例.然而,异构多核具有高带宽及松耦合一致性等特点,获得理想的存储及计算性能需要更多地考虑底层硬件细节.实现了一种针对典型的异构多核Cell BE 处理器的多级并行模型 CellMLP,通过 C 语言扩展编译指导语句,实现了对数据并行、任务并行以及流水并行编程模型的支持,提高了并行程序生产率.运行支持优化方面,数据并行采用 SPE 并行数据传输、双缓冲等优化手段来提高数据传输带宽;任务并行使用一种新式混合任务队列以支持异步任务窃取,降低SPE线程间竞争,提高了任务并行的可扩展性;流水并行首次使用阻塞信号传输机制实现 SPE 线程间的低开销同步操作.实验对 Stream,NAS Benchmark 及 BOTS 等应用进行了测试,结果表明,CellMLP 可对多种典型并行应用进行高效支持.与目前同类编程模型 SARC 及 CellSs 进行性能对比,其结果表明,CellMLP 实际数据传输带宽以及非规则应用的支持方面具有明显优势.

关键词: 异构多核;数据并行;任务并行;流水并行;非规则应用;编译优化

中图法分类号: TP314 **文献标识码**: A

中文引用格式: 李士刚,胡长军,王珏,李建江.异构多核上多级并行模型支持及性能优化.软件学报,2013,24(12):2782−2796. http://www.jos.org.cn/1000-9825/4386.htm

英文引用格式: Li SG, Hu CJ, Wang J, Li JJ. Support for multi-level parallelism on heterogeneous multi-core and performance optimization. Ruan Jian Xue Bao/Journal of Software, 2013,24(12):2782−2796 (in Chinese). http://www.jos.org.cn/1000-9825/4386.htm

Support for Multi-Level Parallelism on Heterogeneous Multi-Core and Performance Optimization

LI Shi-Gang[1], HU Chang-Jun[1], WANG Jue[2], LI Jian-Jiang[1]

[1](School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China)

[2](Supercomputing Center of the Chinese Academy of Sciences, Computer Network Information Center, The Chinese Academy of Sciences, Beijing 100190, China)

Corresponding author: LI Shi-Gang, E-mail: lsgwoody@gmail.com; WANG Jue, E-mail: wangjue@sccas.cn

Abstract: Due to its lower power consumption and cost, heterogeneous multi-core makes up a major computing resource in the current supercomputers. However, heterogeneous multi-core processor features high bandwidth and loose memory consistency, programmers pay attention to hardware details to get ideal memory and computation performance. This paper introduces CellMLP, a multi-level parallelism model for Cell BE heterogeneous multi-core processor. Through extending compiler directives based on C, CellMLP supports data parallelism, task parallelism and pipeline parallelism programming model, and improves the programming productivity. In addition, runtime optimizations are used to improve the performance. Parallel SPEs data transfer and double-buffer mechanisms are used to improve memory bandwidth. A novel hybrid task queue is used in task parallelism to support asynchronous work stealing, reduce the

* 基金项目: 国家自然科学基金(61303050); "十二五"国家科技支撑计划(2011BAK08B04); 国家高技术研究发展计划(863)(2011AA01A205); 中国科学院计算机系统结构重点实验室开放课题(CARCH201108)

收稿时间: 2012-10-02; 修改时间: 2012-12-03; 定稿时间: 2013-02-05

contention between SPE threads and increase the scalability of task parallelism. For the pipeline parallelism, low-overhead synchronization operations are firstly implemented utilizing signal channels in Cell BE. Experiments are conducted on Stream, NAS Benchmark, BOTS and other typical irregular applications. Results show that CellMLP can support different typical parallel applications efficiently. Compared with similar programming model SARC and CellSs, CellMLP has obvious advantages in terms of practical data transfer bandwidth as well as the support of irregular applications.

Key words: heterogeneous multi-core; data parallelism; task parallelism; pipeline parallelism; irregular applications; compiling optimization

多核架构正在成为计算机领域的工业标准,旨在通过增加片上处理核数量以提高处理器性能.多核处理器从结构上可划分为两大类:同构多核和异构多核.在同构多核处理器中,各个处理核具有相同指令集.异构多核处理器则集成不同架构的控制核心与加速核心,其中,加速核心具有很高的浮点计算能力.异构多核的优势是低功耗及廉价性,但是,如何将串行程序或适用于同构多核的并行程序移植到异构多核上并充分利用异构架构特点,是研究人员面临的新挑战.

Cell BE 是一款典型的异构多核处理器,体系结构如图 1 所示.它包含一个基于 Power 架构的主处理器 PPE (powerpc processor element)和 8 个协同处理器 SPE(synergistic processor element).PPE 是一个 64 位通用的双线程处理器,包含两级硬件 Cache 结构.每个 SPE 具有 256KB 非一致性本地存储,并通过 DMA 控制器与其他 SPEs 以及主存储器进行通信.PPE 以及 SPEs 通过 EIB(element interconnect bus)总线与主存储器相连.此外,Cell BE 提供了片上信号及邮箱通信机制,以支持快速线程同步及小数据量通信.本文针对 Cell BE 异构多核体系结构特点,实现了一种多级并行模型 CellMLP 用于高效地表达多种并行应用程序.

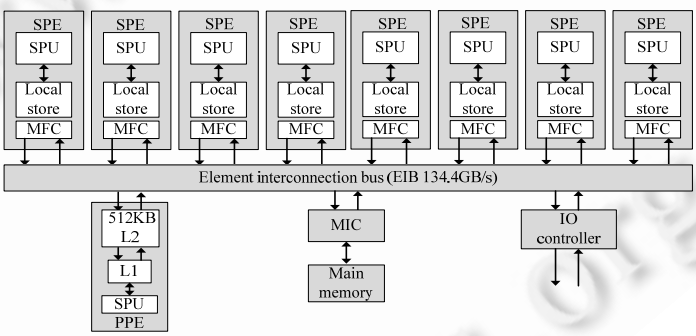

Fig.1 Cell BE architecture

图 1 Cell BE 体系结构

在数据并行的设计中,降低数据传输延迟是关键.CellMLP 采用 PPE 对数据进行划分,并通过 Cell BE 特有的邮箱通信机制将源地址信息广播给 SPE,然后由 SPE 并行发起 DMA 操作进行数据传输,以最大限度地利用 EIB 总线带宽.此外,使用双缓冲机制实现计算与通信重叠,进一步降低数据传输延迟.与 SARC[1]数据并行模型相比较,CellMLP 的有效数据传输带宽平均可提高 78.1%.

任务并行是 OpenMP 3.0 规范的重要内容[2],用于支持非规则应用的并行化.然而,体系结构的特殊性与复杂性使得非规则任务并行在异构多核上的实现难度很大.CellMLP 依照 OpenMP 3.0 规范,首次在异构多核上实现了对基本任务并行指导语句的支持.CellMLP 采用一种混合任务队列架构,避免了单一全局队列可能引发的性能瓶颈.本地任务队列被进一步划分为公有队列及私有队列以支持异步任务窃取机制.任务执行时,SPE 线程从私有队列中获取任务;任务窃取时,从牺牲线程的公有队列中窃取任务而并不干扰牺牲线程执行私有任务,从而降低了 SPE 线程间的竞争.为了降低任务生成开销,采用有界队列法控制任务生成数量及增大任务粒度.此外,双缓冲机制也应用到任务并行中,以降低任务输入数据传输延迟.与任务并行模型(CellSs)[3]相比较,CellMLP 非规则应用的任务并行执行可获得最高 38%的性能提升.

流水并行需要同步操作来保证计算结果的正确性.一种直接的方法是通过标记变量实现同步操作,即线程通过将标记变量置 0/置 1 通知其他线程等待或执行.但是,标记变量的访问一致性需要由硬件或软件机制来保证.SPE 协处理器没有硬件 cache,数据访问的一致性完全由显式的 DMA 操作来保证,这不仅增加了同步操作实现的复杂度,而且开销也较大.CellMLP 充分利用 Cell BE 处理器体系机构特点,首次实现了基于片上信号通信机制的同步操作,从而避免了 DMA 操作带来的开销.相对于标记变量实现的同步操作,LU[4]信号同步流水执行性能可提升 11.6%.

本文第 1 节为问题描述及相关工作.第 2 节介绍 CellMLP 模型的语言级支持.第 3 节~第 5 节分别介绍 Cell BE 异构多核处理器上数据并行、任务并行以及流水并行运行支持设计与优化.第 6 节应用多种测试用例对 CellMLP 模型性能进行评估.最后一节是结论及下一步工作.

1 问题描述及相关工作

图 2 列出了并行编程模型的 4 个应用实例:

- regular array 运算循环迭代之间没有依赖关系且连续访问地址空间,对其并行化可采用数据并行模型,即根据循环迭代数及线程数将计算负载尽可能平均地分配给各个线程并行执行;
- 对于 recursive function calls,编译器由于无法在编译时获知迭代次数等相关信息,则很难通过类似于数据并行的划分手段对此类应用进行并行化,因此需要通过任务并行模型在运行时动态生成并调度任务;
- 对 critical region 进行访问需要互斥锁保护,即在同一时刻只能有一个线程访问临界区,从而降低了程序的并行度;
- loop carried dependency 由于循环迭代之间具有输入/输出依赖关系,无法直接通过数据并行进行并行化.流水并行可对临界区间及循环依赖运算进行并行化,即将数据进行划块,同一时刻每个线程执行不同数据块,同步操作用来避免线程间数据访问冲突.

```
/******** regular array *********/
 for (i=0; i<N; i++)
   b[i]=c*A[i];

/*** recursive function calls ***/
 int fib(int n){
   if (n<=2) return 1;
   else return fib(n−1)+fib(n−2);}
```

```
/******** critical region *********/
 for (i=0; i<N; i++)
   shared_buff[i]+=private_buff[i];

/**** loop carried dependency ****/
 for (k=1; k<M; k++)
   for (i=1; i<N; i++)
     v[k][i]=v[k−1][i−1];
```

Fig.2 Application instances

图 2 应用程序实例

如何通过高效的并行编程模型提高并行应用程序的性能及生产率,一直是高性能计算领域的研究热点.OpenMP[2]对 C 及 Fortran 语言进行了扩展以支持循环级任务并行.Bikshandi 等人[5]提出了分层分块数组(HTA)模型,其中,“分块”被用来描述数据的并行性以及提高数据的局部性.异构多核上,Ferrer 等人[1]通过 SARC 编程模型来描述 Cell BE 处理器上的数据并行.程序员通过扩展语句将计算负载分配给 SPE 协处理器,并指定输入/输出数据信息以及数组分块大小.但是,该模型由于以下不足无法获得理想数据传输带宽:PPE 轮训发送数据块给空闲 SPE 容易导致 SPE 线程饥饿;不支持双缓冲/多缓冲技术进行计算通信重叠.

目前,已有一些并行语言或运行时库支持动态任务并行,例如 Cilk[6],TBB[7],Java’s Fork-join Framework[8],Microsoft Task Parallel Library[9]以及 OpenMP 3.0[2]等.任务并行的优化主要集中于以下几方面:

(1) 任务生成及任务粒度.懒惰任务生成策略(LTC)通常被用来降低任务生成开销,如 Cilk[6].Hiraishi 等人[10]提出了一种基于回溯算法的任务生成调度,程序首先串行执行,当有任务窃取发生时,程序回溯到最早任务生成点以保证任务窃取粒度最大;

(2) 在任务调度中,工作优先及求助优先是两种常用的任务生成策略[11].工作优先是指工作线程立即执行生成的任务,而将剩余任务留给其他工作线程窃取.此调度策略的优势是局部性较好,但是容易导致线程栈空间溢出.求助优先是指工作线程继续执行剩余任务,而将新生成任务留给其他工作线程窃取.此调度策略更适用于任务窃取较多的情况,但是会破坏数据的局部性;

(3) 任务窃取.Cilk[6]运行时,系统采用随机任务窃取策略,即窃取线程随机选取牺牲线程.但是,随机任务窃取被认为是一种缓存不友好策略[12].SLAW[11]是一种局部性可知任务窃取策略,它将工作线程通过 places 进行分组,并规定任务窃取只能在 places 之内发生.但是在 Cell BE 处理器上,由于开发工具不支持可控 SPE 线程布局[13],CellMLP 依然采用随机任务窃取策略.

Perez 等人[3]针对 Cell BE 提出一种任务并行模型(CellSs)用于简化 Cell BE 编程.CellSs 编译器为源到源编译器,它将带有注释语句的单一源程序翻译成 PPU 源程序及 SPU 源程序,然后调用传统编译器(如 gcc 或 IBM XLC)对这些源文件进行编译链接生成可执行文件.本文沿用这一编译策略来实现对多级并行模型的语言级支持.在 CellSs 模型中,PPE 在主存储器上维护全局任务队列,并根据任务依赖关系将任务分配给空闲 SPE.但是,全局任务队列容易成为性能瓶颈.CellMLP 采用一种新式混合任务队列结构以提高任务并行可扩展性.

Gonzalez 等人[14]通过扩展 OpenMP 编译指导语句使其可以有效表达流水并行.Michailidis 等人[15]提出了一种基于 OpenMP 的 LU 流水实现.针对流计算应用,Kessler 等人[16]实现了一种 Cell BE 处理器上流水执行模型.计算以流水线方式进行重组,中间结果不写回主存储器而是直接传给后继消费线程.与上述工作不同的是,CellMLP 中流水并行模型首次采用基于片上信号传输的低开销同步操作的实现流水同步.

此外,Jimenez 等人[13]通过一系列实验对 Cell BE 的存储性能进行了分析,并指出循环展开、向量化、双缓冲、DMA lists 以及推迟 DMA 数据传输同步操作对 SPEs 性能提升的重要性.上述优化手段也被应用到 IBM Cell BE 编译器[17]中.Chen 等人[18]系统地分析了 Cell BE 处理器上 DMA 数据传输双缓冲/多缓冲模型,并给出了性能模型.这些研究工作对本文 CellMLP 多级并行模型的设计及性能优化具有重要的参考价值.

2 CellMLP 多级并行模型语言级支持

CellMLP 对 C 语言进行了扩展,通过增加编译指导语句将负载划分到多个 SPEs 上,实现多级并行模型的语言级支持.目前,CellMLP 支持的编译指导语句包括:数据并行编译指导语句(#pragma cellmlp forall);任务并行编译指导语句(#pragma cellmlp task, #pragma cellmlp taskwait);流水并行编译指导语句(#pragma cellmlp IterWait, #pragma cellmlp GotoIter,#pragma cellmlp ThreadWait,#pragma cellmlp GotoThread).其中,#pragma cellmlp forall 及#pragma cellmlp task 指导语句支持 input/output 子句用于指定输入/输出数据.以流水并行为例,图 3 给出了基于 CellMLP 模型的 LU 流水并行程序片段.其中,第 2 行及第 11 行分别表明此程序具有外层循环依赖及内层循环依赖,第 4 行、第 5 行指定了在 SPEs 上并行执行的内层 for 循环及输入/输出数据,第 7 行、第 13 行及第 8 行、第 14 行分别为两对流水并行同步操作.每个同步操作有两个参数:第 1 个参数用于指定迭代依赖关系,第 2 个参数用于标记一对同步操作.LU 流水并行将在第 6.2 节做详细描述.

CellMLP 编译器采用源到源+运行支持库的构造方式:首先,将带有编译指导语句的单一源程序翻译成 PPU 源程序及 SPU 源程序,并插入相应运行支持接口,如运行支持库初始化与结束化、数据划分信息传输、输入/输出数据传输、任务生成、任务调度、任务等待及流水同步接口等.然后,调用 ppu-gcc 及 spu-gcc 对源程序文件进行编译生成 ppu 及 spu 目标文件,并提供提供循环展开、分支预测及自动向量化等编译优化,其中,spu 目标文件需要与 spu 运行支持库进行链接再封装成 ppu 目标文件.最后,将所有 ppu 目标文件与 ppu 运行支持库进行链接生成可执行文件.

```
1  for (k=1; k<=nz−2; k++){
2   v[i][j][k]=...v[i][j][k−1];
3   ...
4   #pragma cellmlp forall input(v[i−1][j][k])
5   output(v[i][j][k])
6   for (i=ist; i<=iend; i++) {
7    #pragma cellmlp IterWait(i−1,1)
8    #pragma cellmlp IterWait(i+1,2)
9
10    for (j=jst; j<=jend; j++)
11     v[i][j][k]=...v[i−1][j][k];
12      ...
13    #pragma cellmlp GotoIter(i+1,1)
14    #pragma cellmlp GotoIter(i−1,2)}}
```

Fig.3 Code segment of LU pipeline parallelism based on CellMLP

图 3 基于 CellMLP 模型的 LU 流水并行程序片段

3 多粒度数据并行

在 Cell BE 处理器中,PPE 通过 VMX 扩展指令支持向量计算,SPE 拥有与 PPE 不同的指令集架构以支持完全向量化,实现指令级细粒度并行(SIMD).应用程序的向量化需要满足地址对齐、数组偏移对齐等条件[19].CellMLP 采用 ppu-gcc 及 spu-gcc 提供的编译优化手段(如基本结构块聚合、短循环聚合、循环嵌套聚合、地址及数组偏移对齐等)实现自动向量化.向量指令集使得 Cell BE 具有高浮点运算速度,但是 PPE 及 SPEs 之间的实际数据传输带宽通常成为性能瓶颈.

为高效支持循环级数据并行,CellMLP 利用 Cell BE 特有的邮箱通信机制实现 SPEs 并行数据传输.具体说来,PPE 采用静态数据划分方法对数组进行划分.设共有 p 个 SPE 线程,数组共有 n 个元素,令 $q=\lceil n/p \rceil$,$r=p\times q-n$,则前 $p-r$ 个 SPE 线程分别处理 q 个数组元素,剩余的 r 个 SPE 线程分别处理 $q-1$ 个数组元素.PPE 将数组首地址以及各个 SPE 负责数据块的上下界通过邮箱发送给每个 SPE.每个 SPE 读取邮箱数据,然后并行发起 DMA_get 操作以获取主存储器中的相应数据.这样做出于以下 3 方面性能考虑:1) SPE 发起 DMA_get 操作比 PPE 开销低[20];2) SPE 并行数据传输可获得较高带宽;3) PPE 线程数较少容易引发 SPE 线程饥饿.单次 DMA 传输大小默认值为 16KB,也可由程序员指定.然而,本地存储和主存储器间的数据传输延迟很高,采用单缓冲技术(如图 4 所示),总的执行时间将等于计算时间与通信时间之和.双缓冲或多缓冲技术通常被用来隐藏数据传输延迟[18].图 5 给出了一个 DMA 通信受限双缓冲模型,数据传输通过非阻塞 DMA_get 操作发起,并交替使用 buffer_0 和 buffer_1 作为目的地址;计算则需等待上一次迭代中 DMA_get 操作的完成.因此在同一时刻,计算和通信作用于两个不同的缓冲上,从而实现计算与通信的重叠.在通信受限模型中,计算时间小于数据传输时间,因此最终总执行时间等于总通信时间.计算受限双缓冲模型与通信受限双缓冲模型类似,所不同的是,最终执行时间等于总计算时间.

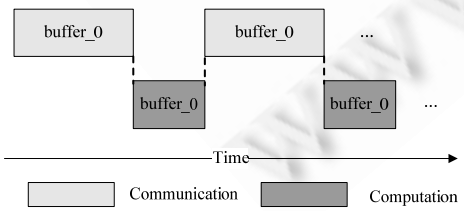

Fig.4 Single buffer model

图 4 单缓冲模型

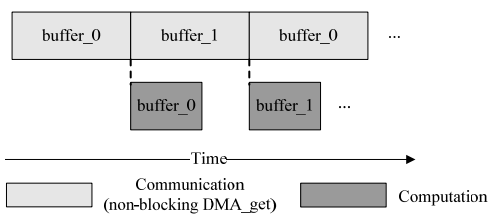

Fig.5 Communication bounded double buffer model

图 5 通信受限双缓冲模型

4 可扩展任务并行

任务是线程执行与调度的基本单位,它包括任务标示符、任务执行函数指针,输入/输出数据起始地址及大小,以及任务依赖关系等信息.SPE 线程在执行任务过程中需要将数据从主存储器传输到本地存储,任务执行结束将输出数据写回主存储器或发送给其他 SPE 线程.空闲 SPE 线程窃取任务则需要将任务相关信息拷贝到本地存储.以下将从任务队列、任务依赖及任务调度等方面介绍 CeleMLP 运行支持库中任务并行的设计与实现.

4.1 混合任务队列及异步任务窃取

同构多核通常采用分布式任务调度算法,即各个线程都处于同等地位.与同构多核不同的是,Cell BE 处理器中 PPE 与 SPEs 分别处于主从地位.PPE 线程作为生产者及管理者维护集中式任务队列,负责生成及调度任务,而 SPE 线程作为消费者负责执行任务.但是,单一维护集中式任务队列通常会成为性能瓶颈,原因有两方面:任务生成是串行执行的,容易导致 SPE 线程饥饿;频繁的主存储器与本地存储之间的数据传输会带来较大的通信开销.当并行应用生成大量细粒度任务时,这种性能瓶颈更为明显.CellMLP 采用了一种混合任务队列结构,在充分利用 Cell BE 主从架构的同时,避免了集中式任务队列可能带来的性能瓶颈.

混合任务队列结构如图 6(a)所示:PPE 线程在系统内存中维护一个全局任务队列,所有 SPE 线程在本地存储中共同维护一个分布式本地任务队列.在任务执行初期,PPE 负责在全局任务队列中以广度优先的方式生成一定数量的粗粒度任务,并轮训发送到各个 SPE 的本地队列中.此后,PPE 线程将负责输入/输出数据传输、任务依赖检测及监测 SPE 本地队列,以进行任务终止判断.SPE 负责执行本地队列中的任务.当线程空闲时,会尝试窃取其他 SPE 线程本地队列中的任务.图 6(b)给出了分布式本地任务队列执行模式图.每个本地队列保证 FIFO 执行模式.具体说来,head 指针指向队头,SPE 线程将新生成的任务插入到队头,任务执行结束时从队头获取任务,这样,SPE 线程总是执行最新生成的任务,有利于程序执行的局部性.tail 指针指向队尾,空闲 SPE 线程(如 SPE_2)从牺牲线程的队尾窃取任务.队尾任务是最先生成的任务,任务粒度通常最大,这样可以最大程度地偿还任务窃取带来的通信开销.为了进一步减少竞争,split 指针将任务队列分割成私有队列及公有队列.空闲 SPE 线程(如 SPE_2)从牺牲线程(如 SPE_3)的公有队列中窃取任务,牺牲线程可继续执行私有任务而无需等待,从而实现了异步任务窃取.当私有队列中的任务数量超过一定限制时,SPE 线程(如 SPE_0)通过移动 split 指针将私有任务转换为公有任务;当私有任务执行完毕时,则将部分公有任务转化为私有任务(如 SPE_1).在移动 split 指针以及窃取任务时,需要互斥锁保证在任意时刻最多有一个线程访问共享队列.

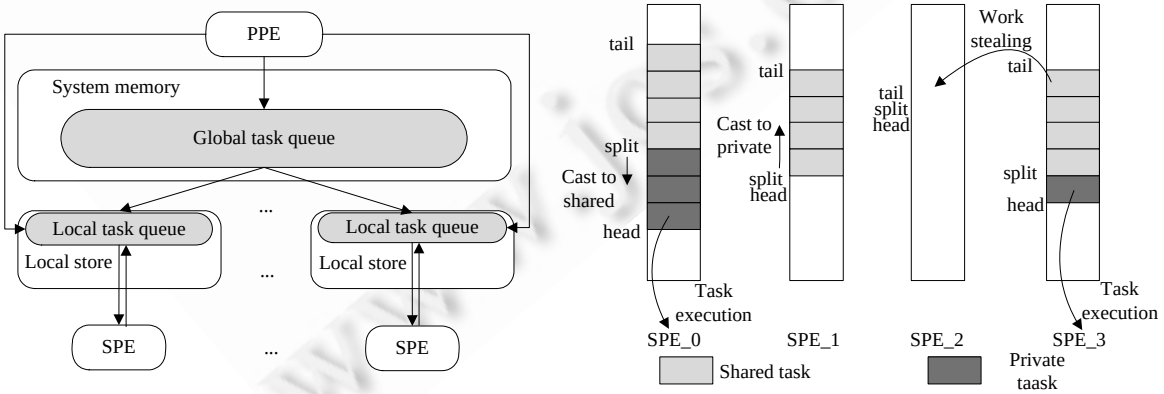

Fig.6 Hybrid task queue

图 6 混合任务队列

异步任务窃取如图 7 所示.分布式本地任务队列减少了线程竞争,任务窃取调度策略充分利用了 SPE 本地存储间的高速数据传输.

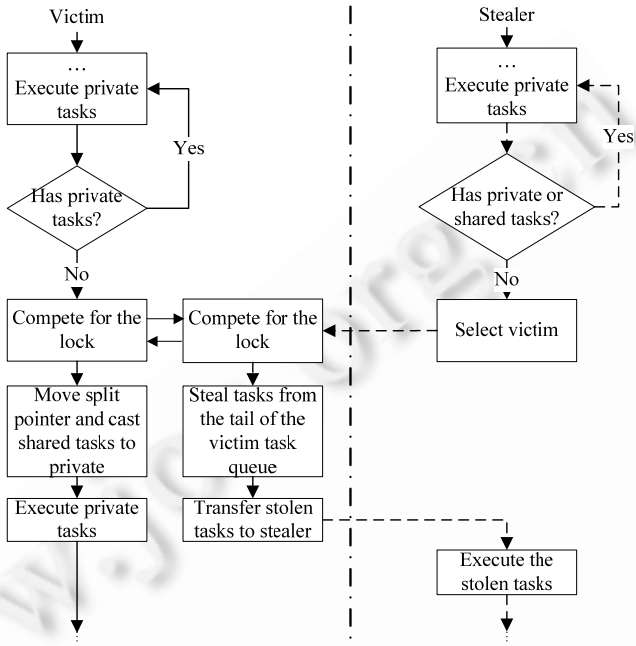

Fig.7 Asynchronous work stealing

图 7 异步任务窃取

4.2 任务依赖

对于任务依赖的支持实现了阻塞及非阻塞两种模式.图 8 给出了任务依赖的阻塞模式实现,其中,父任务的执行依赖于子任务的完成.此时,执行父任务的 SPE 线程在父任务执行函数内部调用执行任务或窃取任务操作以保证线程不空闲,直到子任务全部完成再执行父任务的剩余工作.这种模式的好处是实现简单并且有较好的局部性,但是由于函数嵌套,容易导致 SPE 线程栈溢出.图 9 给出了任务依赖的非阻塞模式实现,当父任务因依赖关系不能立即执行时,则将父任务封装成剩余任务放入到等待任务队列中,SPE 线程则继续执行其他任务或窃取任务.剩余任务中包含所依赖子任务个数以及输入/输出数据信息等,当子任务执行完毕时,则将输出写到剩余任务的输入中;当全部子任务执行完毕,则将剩余任务放到私有任务队列中进行调度.当然,子任务也有可能被其他 SPE 线程窃取,这种情况下,远程线程执行完子任务后,会将输出发送到父任务所在 SPE 线程的消息缓冲区中,父任务所在 SPE 线程会在特定时刻(如任务执行或任务窃取)刷新消息缓冲区,更新剩余任务的状态直到其可被 SPE 线程调度执行.非阻塞模式会带来较多的通信开销,但是可有效防止 SPE 线程栈溢出.

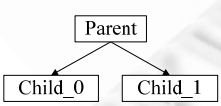

Fig.8 Blocking model for task dependency

图 8 任务依赖阻塞模式

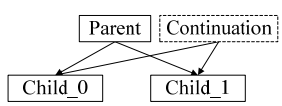

Fig.9 Non-Blocking model for task dependency

图 9 任务依赖非阻塞模式

4.3 任务生成及调度

工作优先及求助优先是两种常用的任务调度策略[11].在第 1 节已经讨论了两种调度策略具有互补作用,因此,CellMLP 采用工作优先及求助优先相结合的方案,根据运行时任务队列情况动态切换任务生成策略.具体说

来,CellMLP 采用有界队列(bounded queue)法对任务生成及任务调度进行优化.在任务执行初期,使用求助优先策略生成任务,以便快速分配负载给全部 SPE 线程;当某 SPE 任务队列达到预先设定界限时,则切换到工作优先任务生成策略.这样,可以在防止 SPE 线程饥饿的同时有效减少细粒度任务生成数量,从而降低任务生成开销.

在细粒度任务并行中,有界深度法(bounded tree depth)[21]是最为常用的任务生成优化策略.程序员手动指定最大任务生成深度,当到达界限时直接串行执行该任务,从而避免大量生成细粒度任务.此方法可以最大程度地降低任务生成开销,但是程序性能受深度界限的影响较大,程序员需要通过手动调优才能找到最优有深度界限,从而增加了编程复杂度.此外,对于不规则任务生成树,有界深度法由于过早将任务串行化,容易导致线程负载不均衡.而有界队列法通过运行时动态切换任务生成策略,可以有效避免以上不足.在具体实现当中,任务队列中任务数量上限被设定为 36,其中,私有任务数量上限为 6,共享任务数量上限为 30.

5 低同步开销流水并行

流水并行的关键是同步操作的实现.Cell BE 处理器支持高效的片上信号通信机制.每个 SPE 具有两个 32 位的信号通知寄存器,PPE 或其他 SPE 可以通过信号通知寄存器发送信息到一个 SPE.信号通知寄存器的地址可在 PPE 端获得并由 PPE 广播给所有的 SPE,当某个 SPE 得到其他所有 SPE 信号通知寄存器映射地址后,便可以向任意 SPE 发送信号.SPE 可采用阻塞模式读取信号通知通道,即如果相应通道中不存在可用信号,则阻塞等待.本节介绍基于信号通信机制的流水同步运行支持接口的实现与应用.

5.1 线程同步及迭代同步流水并行运行支持接口

图 10 给出了线程同步流水并行运行支持接口的实现.在 *goto_nxt_thread_sig* 函数中,线程号为 *curr_id* 的 SPE 线程向线程号为 *thrdid* 的 SPE 线程发送信号,其中,目的信号寄存器的映射地址为 *reg_add*[*thrdid*].在 *wait_prv_thread_sig* 函数中,线程号为 *curr_id* 的 SPE 线程以阻塞模式读取线程号为 *thrdid* 的 SPE 线程发送的信号.两个接口函数的形参为 SPE 线程号,从而实现了一对 SPE 线程间的同步操作.

```
void goto_nxt_thread_sig
(int thrdid){
  if (thrdid!=curr_id)
  /****Send signal***/
  pip_send_signal
  (reg_add[thrdid]);}
```

```
void wait_prv_thread_sig
(int thrdid){
  uint32_t sig = 0;
  if (thrdid!=curr_id)
  /***Receive signal***/
  sig=pip_read_signal();}
```

Fig.10 Thread synchronization for pipeline parallelism

图 10 线程同步流水并行运行支持

图 11 给出了迭代同步流水并行运行支持接口的实现.

```
void goto_nxt_loopindex_sig
(int index){
  int id=−1;
  if (lb<index && index<=ub) return;
  else {id=probe_threadId(index);
  if (id!=curr_id)
  pip_send_signal(reg_add[id]);}}
```

```
void wait_prv_loopindex_sig
(int index){
  uint32_t sig=0;
  if (lb<index && index<=ub)
    return;
  else sig=pip_read_signal();}
```

Fig.11 Iteration synchronization for pipeline parallelism

图 11 迭代同步流水并行运行支持

迭代同步流水并行运行支持接口与线程同步流水并行运行支持接口类似,也是通过信号的发送与读取实现同步操作.所不同的是,同步操作细化到每一次迭代,程序员无需了解每一个 SPE 线程上迭代划分的具体细节即可实现同步操作.在 *goto_nxt_loopindex_sig* 函数中,首先获得调用此接口的 SPE 线程负责执行的循环上下界(*lb* 与 *ub*),然后判断需要同步的迭代是否在该范围之内:如果在,表明此次迭代是在调用该接口的 SPE 上执行;

否则调用 *probe_threadId*,返回此迭代所在 SPE 线程号,并向这个 SPE 线程发送信号.在 *wait_prv_loopindex_sig* 函数中,同样是首先获得调用此接口的 SPE 线程负责执行的循环上下界,然后判断需要同步的迭代是否在该范围内,如果不在,则以阻塞模式读取信号.

5.2 流水并行应用实例

由临界区引起的串行执行是并行程序的性能瓶颈,而流水并行可以提高临界区的执行效率.具体做法是对共享数据进行分块,并通过线程同步操作保证每个 SPE 在同一时刻访问不同的共享数据块.图 12 给出了临界区流水执行模型.首先在 *T*1 时刻,SPE0 将 Block_1 从主存储器取到本地存储当中,然后进行计算,其他 SPE 则等待.SPE0 计算完毕后,Block_1 传给 SPE1,传送完毕后向 SPE1 发送信号,SPE1 接收到信号后开始执行计算.此时,SPE0 从主存储器中取出 Block_2 进行计算,并将 Block_1 覆盖.当 SPE0 执行完 Block_2 时,要等 SPE1 执行完 Block_1 再进行数据传输,以免引发数据冲突.因此,当 SPE1 执行完 Block_1 时,需要向 SPE0 发送信号,通知 SPE0 开始传输 Block_2.当 Block_1 依次流经所有 SPE 得到最终结果时,由最后一个 SPE 线程将其写回主存储器.关于临界区流水并行代码实现细节及底层运行支持接口调用请参考我们的前期工作[22].

LU 的实现是一个典型的高斯赛德尔迭代过程.在 LU 时间开销最大的核心计算部分,数组元素 *v*[*i*][*j*][*k*]的计算依赖于 *v*[*i*−1][*j*][*k*]的值.基于 OpenMP 实现的 NAS 并行测试程序[4]中,LU 的流水执行同步操作基于标记变量实现,而基于 CellMLP 实现的 LU 流水执行(如图 3 所示)底层同步操作是基于信号实现的.图 13 给出了基于 CellMLP 的 LU 流水执行模型.当 SPE0 及 SPE1 线程中 *k* 值相等时,即两个 SPE 线程都处于同一个最外层迭代,只有 SPE0 线程将低下标(*i* 值较小)的 *v* 数组元素计算完成之后,下一个 SPE 线程才会计算高下标(*i* 值较大)的 *v* 数组元素,这样就避免了 *v*[*i*][*j*][*k*]与 *v*[*i*−1][*j*][*k*]引起的读写冲突.当 SPE0 及 SPE1 中 *k* 值相差为 1 时,即两个 SPE 线程都处于相邻的最外层迭代,SPE1 线程第 1 个数组元素的计算需要在 SPE0 线程最后一个数组元素的计算之前完成.SPE1 告知 SPE0 已读取上次迭代中发送的信号,SPE0 可发送下一个信号,以免信号通知通道发生阻塞导致程序执行错误.

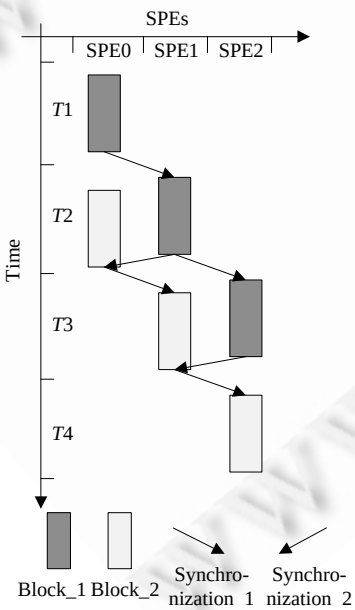

Fig.12 Critical region thread synchronization for pipeline parallelism

图 12 临界区线程同步流水并行

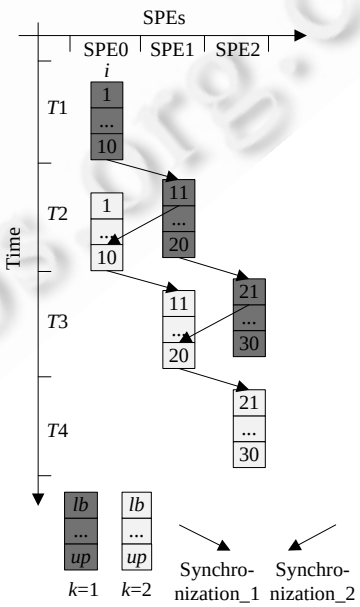

Fig.13 LU Iteration synchronization for pipeline parallelism

图 13 LU 迭代同步流水并行

6 实 验

实验的硬件环境是 Cell BE 刀片服务器.一个刀片服务器包含两个 Cell BE 处理器以及 1GB 的系统内存.实验平台的操作系统为 Fedora9(Linux Kernel 2.6.25-14),多级并行模型编译环境为 Cell SDK3.1.

6.1 数据并行

在数据并行模型方面,首先将 CellMLP 与 SARC[1]执行模型进行对比,其中,SARC 使用 4 个 PPE 线程(Cell BE 刀片服务器最多支持 4 个 PPE 线程)并行生成并调度计算任务,以最大程度缓解 SPE 线程饥饿[1].测试用例为 STREAMS[23],包括 Copy,Scale,Add 及 Triad,设定单次 DMA 传输数据量为 4Kbytes,实验结果如图 14 所示.

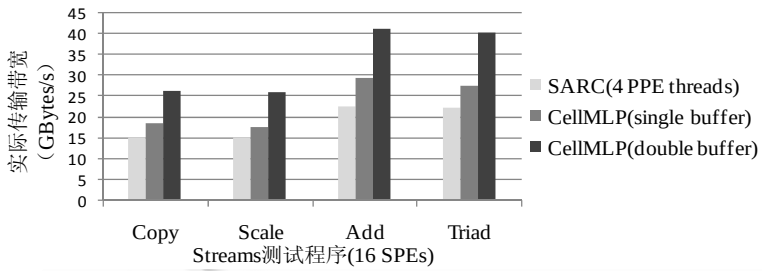

Fig.14 Practical data transfer bandwidth of data parallelism

图 14 数据并行实际传输带宽

当 CellMLP 采用单缓冲机制时,Copy,Scale,Add 以及 Triad 实际传输带宽相对于采用多 PPE 线程的 SARC 分别提高了 21.9%,17.2%,31.4%及 23.7%,这证明 CellMLP 采用 SPEs 并行发起 DMA 操作进行数据传输更有利于提高实际数据传输带宽.此外,由于 Copy 和 Scale 有一个输入向量及一个输出向量,Add 和 Triad 有两个输入向量及一个输出向量,Add 和 Triad 输入数据量更大,因此相对于 Copy 和 Scale 带宽提升较多.

当 CellMLP 采用双缓冲机制时,Copy,Scale,Add 以及 Triad 实际传输带宽在单缓冲机制的基础上进一步分别提高了 52.8%,55.4%,52.9%以及 56.7%.STREAMS 测试用例均属于通信受限双缓冲模型,即总执行时间由通信时间决定,如图 14 所示,Scale 相对于 Copy 计算量较大,会有更多的计算时间被覆盖,因此,Scale 相对于 Copy 带宽提升相对较多.同理,Triad 相对于 Add 带宽提升相对较多.

总体而言,SPE 访问主存储器理论峰值带宽为 51.2Gbytes/s,而基于 CellMLP 提供的数据并行模型,Add 实际传输带宽可以达到 41.3Gbytes/s,为峰值带宽的 80.6%.

6.2 任务并行

任务并行测试程序为 Barcelona OpenMP Task Suite(BOTS)[24],测试用例输入参数见表 1.为了避免非一致性内存访问(NUMA)对测试结果带来的影响(随机任务窃取会带来通信开销的不确定性),测试程序被绑定在刀片服务器中的一个 Cell BE 处理器上运行.

Table 1 Input parameters of benchmarks

表 1 测试程序的输入参数

Benchmarks	Input parameters
Alignment	100 protein sequences
N-Queens	14×14 chessboard
SparseLU	Matrix size 5000×5000
Multisort	Array of 33, 554, 432 integers
Fib	*N*=40
UTS	T3L

首先,在 CellMLP 中对有界深度及有界队列两种不同任务生成优化方法进行性能对比,结果如图 15 所示.

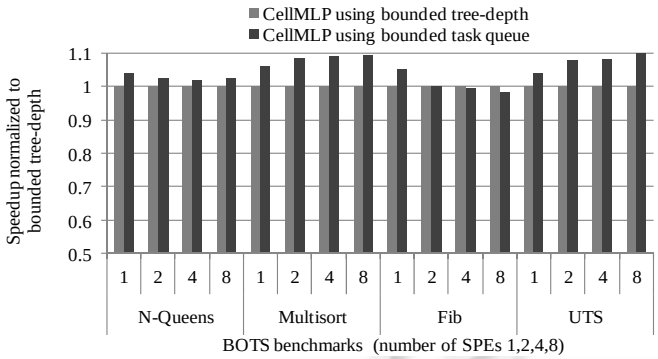

Fig.15 Performance Comparison between bounded tree-depth and bounded task queue

图 15 有界深度法与有界队列法性能对比

- 在 Multisort 及 UTS 中,有界队列法明显优于有界深度法.这是由于 Multisort 及 UTS 任务生成树很不规则,有界深度法由于过早将任务串行化,会导致线程负载不均衡;而有界队列法通过运行时动态切换任务生成策略,以保证生成足够的任务进行调度;
- 在 Fib 中:当 SPE 个数为 1 时,有界队列法优于有界深度法.这是因为当串行执行时,有界队列法生成任务数量明显小于有界深度法;但是随着 SPE 个数增多,有界队列法略微劣于有界深度法.这是因为 Fib 任务生成树十分规则且任务粒度很小,有界队列法会生成较多的细粒度任务从而带来额外开销;
- 由于 Alignment 和 SparseLU 为 *for* 循环结构不包含递归任务生成无法采用有界深度策略,因此并没有列在图中.此外,由于 Alignment 及 SparseLU 本身任务粒度都相对较大,任务生成优化效果并不明显.对于全部测试程序而言,有界队列法明显优于或接近有界深度法.

图 16 将 CellMLP 与 CellSs[3]任务并行模型进行了性能对比,其中,CellMLP 采用有界队列法对任务生成进行优化,CellSs 使用通过手动调优的最优有界深度对任务生成进行优化.

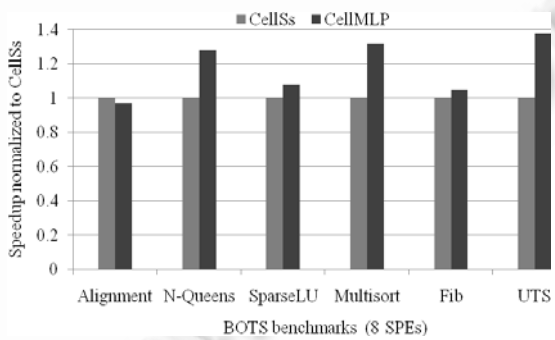

Fig.16 Performance Comparison between CellMLP and CellSs

图 16 CellMLP 与 CellSs 性能对比

除 Alignment 测试程序外,CellMLP 均优于 CellSs,平均性能提升为 22.1%.其中,*N*-Queen,Multisort 及 UTS 性能提升最为明显.这是因为 *N*-Queen,Multisort 及 UTS 任务生成树很不规则且任务粒度都较小,CellSs 采用 PPE 生成任务并维护集中式任务队列不仅容易导致 SPE 线程饥饿,而且无法实现较好地负载均衡.CellMLP 采用混合任务队列结构,PPE 在任务执行初期快速生成并调度任务到 SPEs 本地队列中,SPEs 执行本地队列任务并采用异步任务窃取机制实现负载均衡,使得不均衡的工作负载更快速地分布到其他 SPEs 上,并降低了 SPE 线程间的竞争.Fib 任务生成树很规则但是任务粒度很小,它的性能提升得益于混合任务队列.SparseLU 任务粒度较

大但是负载很不均衡,其性能提升得益于任务窃取机制.对于 Alignment,CellSs 略优于 CellMLP,这是因为 Aligment 任务粒度较大,CellSs 使用 PPE 线程可及时地将任务分配给 SPEs.总体而言,CellMLP 相对于 CellSs 具有更好的可扩展性.

图 17 为 CellMLP 可扩展性测试结果.Aligment,*N*-Queens,SparseLU 以及 Fib 都展现出了很好的可扩展性.Multisort 及 UTS 的任务生成树不规则,由任务窃取引起的通信开销越大,因此可扩展性相对较差.

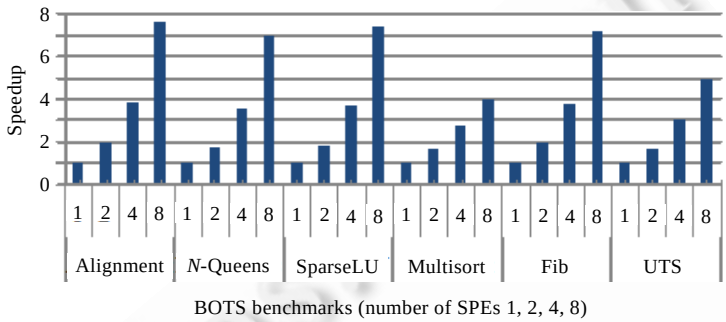

Fig.17 Scalability of task parallelism in CellMLP

图 17 CellMLP 任务并行可扩展性

6.3 流水并行

CellMLP 流水并行测试程序为 NAS benchmarks[4]中的 IS,LU 以及 SPEC[25]中的 MOLDYN.图 18 给出了 IS 及 MOLDYN 临界区间采用流水并行优化后的加速比,其中,基准程序为 OpenMP 版本的 IS 及 MOLDYN 的临界区间,基准程序的编译环境为 IBM XL CELL 单源编译器[26].在 CellMLP 流水优化中,为了测试数组划分块数对性能的影响,临界区间数组被分别划分为 4 块、8 块、16 块及 32 块.为了测试数据量级别对优化效果的影响,IS 分别采用 W 级、A 级及 B 级数据量,而 MOLDYN 通过手动方式达到与 IS 对应相等数据量.

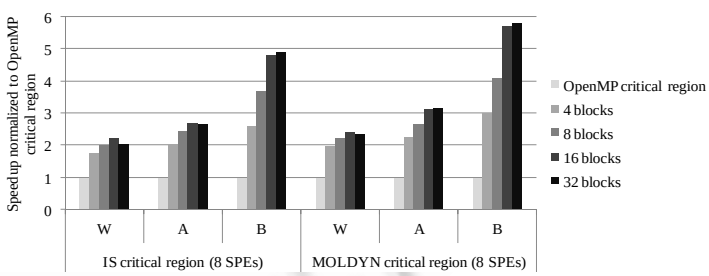

Fig.18 Pipeline parallelism optimization for critical region

图 18 临界区间流水并行优化

实验结果表明:

(1) 数据量级别越高,流水优化越明显.因为当计算量增加时,同步操作开销占临界区间总执行时间比例减小.例如,当 IS 临界区间划分块数为 16 且数据量级别从 W 级增加到 B 级时,流水优化加速比从 2.2 增加到 4.8;

(2) 当数据量级别一定且划分块数不大于 16 时,加速比会随临界区间划分块数而增加.例如,A 级 IS 划分块数从 4 增加到 16 时,加速比从 2.1 增加到 2.7.这是因为:假设同步开销为 0,划分块数为 x,SPE 线程数量为 y,加速比可被表示为 $x\times y/(x+y-1)$,加速比会随 x 的增加而增加.但是当划分块数增加到 32 时,加速比下降到 2.6.这是因为划分块数越多,同步开销越大,以至于加速比下降;

(3) 当数据量级别及划分块数均相同时,MOLDYN 优化效果优于 IS.因为 MOLDYN 临界区间相对于 IS 计算更加复杂,同步操作开销占临界区间总执行时间比例较小.

图 19 为 A 级 LU 标记变量同步流水并行及 CellMLP 信号同步流水并行执行时间对比.当 SPE 个数分别为 2,4,8 及 16 时,LU 性能平均提升 8.6%,这说明信号同步开销明显低于标记变量同步开销.此外,SPE 使用个数越多,LU 性能提升越过.当 SPE 个数从 2 增加到 16 时,信号同步流水并行性能提升从 5.5%增加到 11.6%.这是因为 SPE 个数越多,由同步操作引起的通信开销越大,因此信号同步优化效果越明显.

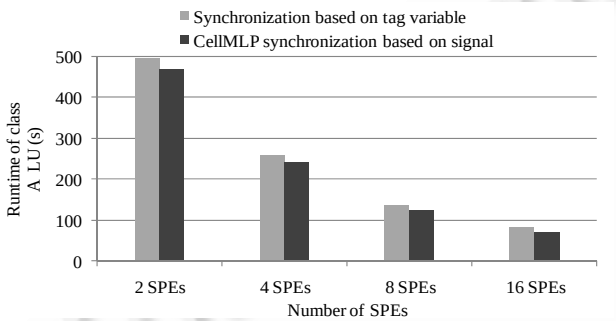

Fig.19 Scalability of task parallelism in CellMLP

图 19 CellMLP 任务并行可扩展性

7 结论及下一步工作

CellMLP 多级并行模型向程序员提供了一系列基于高级语言的扩展编译指导语句,实现了 Cell BE 异构多核处理器上多级并行的支持,提高了并行程序的生产率及可移植性.CellMLP 依照 OpenMP 3.0 规范,首次在异构多核上实现了对基本任务并行指导语句的支持.CellMLP 采用了一种新式的混合任务队列结构,在充分利用 Cell BE 主从架构的同时,又提高了任务并行可扩展性.流水并行首次利用片上信号传输机制,实现 SPE 线程间的低开销同步操作.在数据并行的设计中,SPEs 并行发起 DMA 操作获取数据及双缓冲机制的采用,明显提高了实际传输带宽.实验结果表明,基于 CellMLP 多级并行模型,数据并行应用可达峰值带宽的 80.6%,非规则应用的任务并行执行相对于 CellSs 模型可获得平均 22.1%的性能提升,LU 流水执行通过信号同步优化性能可提升 11.6%.

总体而言,CellMLP 充分利用 Cell BE 体系结构的特点,实现了对多级并行的高效支持.由于目前 Cell BE 处理器程序开发工具并没有提供给程序员相应的接口来获取线程布局信息,文中任务并行模型中仍采用随机任务窃取机制,并没有考虑局部性问题,这需要进一步完善.GPU+CPU[27]是另一种典型的异构计算架构.GPU 通过单指令多线程(SIMT)实现指令集数据并行,并通过零代价线程切换来隐藏设备端的数据传输延迟.双缓冲/多缓冲机制仍可用来降低 GPU 及 CPU 之间的数据传输延迟.与 Cell BE 不同的是,GPU 只能同步处于同一线程块(thread block)中的线程,这使得非规则任务并行的实现受到限制,例如,需要发起多个计算核心(kernel)以实现不同任务间的同步.实现 GPU+CPU 混合架构多级并行支持是我们的下一步工作.

致谢 感谢西班牙 Becelona Supercomputing Center 的 CellSs 编译组及 SARC 编程模型组提供的测试程序.感谢北京科技大学高性能计算实验室并行编译组全体成员.

References:

[1] Ferrer R, Gonzalez M, Silla F, Martorell X, Ayguadé E. Evaluation of memory performance on the Cell BE with the SARC programming model. In: Proc. of the 9th Workshop on Memory Performance: Dealing with Applications, Systems, and Architecture. New York: ACM Press, 2008. 77−84. [doi: 10.1145/1509084.1509095]

[2] OpenMP architecture review board. OpenMP Application Program Interface Version 3.0. http://www.openmp.org/mp-documents/

[3] Perez JM, Bellens P, Badia RM, Labarta J. CellSs: Making it easier to program the Cell broadband engine processor. IBM Journal of Research and Development, 2007,51(5):593−604. [doi: 10.1147/rd.515.0593]

[4] Jin H, Frumkin M, Yan J. The OpenMP implementation of NAS parallel benchmarks and its performance. Technical Report, NAS-99-011, Moffett Field: NASA Ames Research Center, 1999.

[5] Bikshandi G, Guo J, Hoeflinger D, Almasi G, Fraguela BB, Garzaran MJ, Padua D, von Praun C. Programming for parallelism and locality with hierarchically tiled arrays. In: Proc. of the ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. New York: ACM Press, 2006. 48−57. [doi: 10.1145/1122971.1122981]

[6] Frigo M, Leiserson CE, Randall KH. The implementation of the Cilk-5 multithreaded language. In: Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation. New York: ACM Press, 1998. 212−223. [doi: 10.1145/277650.277725]

[7] Intel corporation. Intel(R) threading building blocks. http://threadingbuildingblocks.org/documentation.php

[8] Lea D. A Java fork/join framework. In: Proc. of the ACM Conf. on Java Grande. New York: ACM Press, 2000. 36−43. [doi: 10.1145/337449.337465]

[9] Leijen D, Schulte W, Burckhardt S. The design of a task parallel library. In: Proc. of the 24th ACM SIGPLAN Conf. on Object Oriented Programming Systems Languages and Applications. New York: ACM Press, 2009. 227−242. [doi: 10.1145/1640089.1640106]

[10] Hiraishi T, Yasugi M, Umatani S, Yuasa T. Backtracking-Based load balancing. In: Proc. of the ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. New York: ACM Press, 2009. 55−64. [doi: 10.1145/1504176.1504187]

[11] Guo Y, Zhao JS, Cave V, Sarkar V. SLAW: A scalable locality-aware adaptive work-stealing scheduler. In: Proc. of the IEEE Int'l Symp. on Parallel and Distributed Processing. New York: ACM Press, 2010. 1−12. [doi: 10.1109/IPDPS.2010.5470425]

[12] Acar UA, Blelloch GE, Blumofe RD. The data locality of work stealing. In: Proc. of the 12th Annual ACM Symp. on Parallel Algorithms and Architectures. New York: ACM Press, 2000. 1−12. [doi: 10.1145/341800.341801]

[13] Jimenez-Gonzalez D, Martorell X, Ramirz A. Performance analysis of Cell broadband engine for high memory bandwidth applications. In: Proc. of the IEEE Int'l Symp. on Performance Analysis of Systems and Software. 2007. 210−219. [doi: 10.1109/ISPASS.2007.363751]

[14] Gonzalez M, Ayguadé E, Martorell X, Labarta J. Defining and supporting pipelined executions in OpenMP. In: Proc. of the 2nd Int'l Workshop on OpenMP Applications and Tools. Berlin, Heidelberg: Springer-Verlag, 2001. 155−169. [doi: 10.1007/3-540-44587-0_14]

[15] Michailidis PD, Margaritis KG. Implementing parallel LU factorization with pipelining on a multicore using OpenMP. In: Proc. of the 13th IEEE Int'l Conf. on Computational Science and Engineering. 2010. 253−260. [doi: 10.1109/CSE.2010.39]

[16] Kessler CW, Keller J. Optimized mapping of pipelined task graphs on the Cell BE. In: Proc. of the 14th Int'l Workshop on Compilers for Parallel Computing. 2009.

[17] Eichenberger AE, O'Brien K, O'Brien KM, Wu P, Chen T, Oden PH, Prener DA, Shepherd JC, So B, Sura Z, Wang A, Zhang T, Zhao P, Gschwind M, Archambault R, Gao Y, Koo R. Using advanced compiler technology to exploit the performance of the Cell broadband engine (tm) architecture. IBM Systems Journal, 2006,45(1):59−84. [doi: 10.1147/sj.451.0059]

[18] Chen T, Sura Z, O'Brien KM, O'Brien JK. Optimizing the use of static buffers for DMA on a CELL chip. In: Proc. of the Workshop on Language and Compiler for Parallel Computing. Berlin, Heidelberg: Springer-Verlag, 2006. 314−329. [doi: 10.1007/978-3-540-72521-3_23]

[19] IBM Corporation. Cell broadband engine programming handbook including the PowerXCell 8i processor version 1.12. https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/

[20] Brokenshire DA. Maximizing the power of the Cell broadband engine processor: 25 tips to optimal application performance. http://www-128.ibm.com-/developerworks/power/library/pacelltips1/

[21] Loidl HW, Hammond K. On the granularity of divide-and-conquer parallelism. In: Proc. of the Glasgow Workshop on Functional Programming. Berlin, Heidelberg: Springer-Verlag, 1995. 8−10.

[22] Li SG, Yao SC, He HH, Sun LL, Chen Y, Peng YF. Extending synchronization constructs in OpenMP to exploit pipeline parallelism on heterogeneous multi-core. In: Proc. of the 11th Int'l Conf. on Algorithms and Architectures for Parallel Processing. Berlin, Heidelberg: Springer-Verlag, 2011. 54−63. [doi: 10.1007/978-3-642-24669-2_6]

[23] Mccalpin JD. Stream: Sustainable memory bandwidth in high performance computers. http://www.streambench.org/

[24] Duran A, Teruel X, Ferrer R, Martorell X, Ayguadé E. Barcelona OpenMP tasks suite: A set of benchmarks targeting the exploitation of task parallelism in OpenMP. In: Proc. of the 38th Int'l Conf. on Parallel Processing. Washington: IEEE Computer Society, 2009. 124−131. [doi: 10.1109/ICPP.2009.64]

[25] SPEC: Standard performance evaluation corporation. http://www.spec.org

[26] Eichenberger AE, O'Brien K, O'Brien KM, Wu P, Chen T, Oden PH, Prener DA, Shepherd JC, So B, Sura Z, Wang A, Zhang T, Zhao P, Gschwind M. Optimizing compiler for the Cell processor. In: Proc. of the 14th Int'l Conf. on Parallel Architectures and Compilation Techniques. Washington: IEEE Computer Society, 2005. 161−172. [doi: 10.1109/PACT.2005.33]

[27] Wu EH. State of the art and future challenge on general purpose computation on GPU. Ruan Jian Xue Bao/Journal of Software, 2004,15(10):1493−1504 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1493.htm

附中文参考文献:

[27] 吴恩华.图形处理器用于通用计算的技术现状及其挑战.软件学报,2004,15(10):1493−1504. http://www.jos.org.cn/1000-9825/15/1493.htm

李士刚(1986－),男,河北定州人,博士生,主要研究领域为并行计算及并行编译技术.
E-mail: lsgwoody@gmail.com

王珏(1981－),男,博士,副研究员,主要研究领域为并行计算及并行编译技术.
E-mail: wangjue@sccas.cn

胡长军(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为并行计算及并行编译,并行软件工程,网格计算,网络存储体系结构,数据工程.
E-mail: huchangjun@ies.ustb.edu.cn

李建江(1971－),男,博士,副教授,CCF 会员,主要研究领域为并行计算及并行编译技术.
E-mail: jianjiangli@gmail.com