LU 分解在 Godson-Tv1 众核体系结构上的并行化研究

龙国平 范东睿

(中国科学院计算技术研究所系统结构重点实验室 北京 100190)

摘 要 随着集成电路工艺的发展,众核体系结构成为人们日益关注的计算平台.LU 分解是科学和工程计算中被广泛使用的核心算法之一,尽管在传统的并行体系结构上已有大量的并行化研究工作,但是结合新型众核体系结构特征的工作还不多.文章从负载均衡、延迟容忍和性能分析模型 3 个方面系统研究了 LU 分解在众核体系结构上的并行化问题.该文的贡献在于:首先,针对二维卷帘负载分配方案难以达到良好负载均衡的缺点,提出一种新的"之"字形分配方案,实验表明不经任何优化的情况下性能比前者提高 20%,优化后达到了 40%;其次,提出了一个性能加速比的分析模型,并用实验定量研究了实测性能加速比和理论值之间的差距,发现在合理利用片上存储优化访存延迟,并恰当选择矩阵分块参数的情况下,实测加速效果能比较接近理论值;通过实验还证明实测性能难以达到理论预测值的两个主要原因:访存带宽有限和片上网络的资源竞争.

关键词 众核体系结构;LU 分解;并行化;延迟容忍;性能模型

中图法分类号 TP302 **DOI 号**: 10.3724/SP.J.1016.2009.02157

Parallelization of LU Decomposition on the Godson-Tv1 Many-Core Architecture

LONG Guo-Ping FAN Dong-Rui

(*Key Laboratory of Computer Systems and Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190)

Abstract The many-core architecture is increasingly becoming a promising computing platform due to the advancement of semi-conductor technology. LU decomposition is a widely used kernel in both scientific and engineering computations. Although there are a lot of related works on traditional parallel architectures, there is still little work focusing on parallelizing it on many-core architectures. This paper investigates this problem from three aspects: load balancing, latency hiding and performance modeling. There are three contributions of this work: Firstly, a novel load balancing technique has been introduced to overcome the limitations of 2D scatter decomposition. Experimental results show that the proposed scheme achieves 20% performance improvement without optimization and 40% improvement after optimization. Secondly, an analytical performance model is presented. Quantitative experimental study shows that by carefully hiding memory latency through on chip memory hierarchy and for a selected block size, the upper bound of theoretical performance can be approximated by experiments. Experimental results also reveal two primary causes which make theoretical speedup hard to achieve: limited DRAM bandwidth and resource contention of on-chip network.

Keywords many-core architecture; LU decomposition; parallelization; latency tolerance; performance model

---

收稿日期:2007-11-22;最终修改稿收到日期:2009-10-14.本课题得到国家"九七三"重点基础研究发展规划项目基金(2005CB321600)、国家自然科学基金重点项目(60736012)、国家"八六三"高技术研究发展计划项目基金(2009AA01Z103)、国家杰出青年科学基金和北京市自然科学基金(4092044)资助.**龙国平**,男,1982 年生,博士研究生,主要研究方向为计算机体系结构.E-mail: longguoping@ict.ac.cn.
范东睿,男,1979 年生,博士,主要研究兴趣包括体系结构、并行处理和运行时系统.E-mail: fandr@ict.ac.cn.

1 引 言

20 世纪 80 年代以来，宽发射及高主频的超标量设计曾持续推动着微处理器性能的稳步提升. 但随着半导体工艺尺寸的缩小，主频的持续提高变得非常困难，而功耗和散热问题却越来越严峻. 近年来，集成电路工艺技术的进展使得片上可集成的晶体管数目已达到数十亿量级，为了在平衡主频、功耗、散热等各方面设计空间的前提下充分利用海量的片上资源，学术界和产业界纷纷转向并行处理器体系结构的研究. 众核体系结构因其强大的片上并行能力和相对较合理的片上计算/存储比例日益引起学术界和产业界的关注[1-5]①. 和一般的商业多核处理器结构[6-7]不同，众核体系结构的目标是最大化片上并行性，每个核的功能设计得很简单，通过大量核的并行处理实现高性能. 已有的研究结果表明众核体系结构在分子动力学、数值计算等领域具有很大的潜力[8-9]. 本文研究立足于众核体系结构，探讨 LU 分解在此类结构上的并行化问题.

对于众核体系结构在物理上是否能实现以及对一些应用的加速效果等问题，目前已有一些研究结果[1,2,5,10-12]. 文献[11]基于 Cyclops-64 体系结构提出一种 LU 分解的并行实现，但是该实现过于依赖 Cyclops-64 体系结构，不适合基于 Cache 存储层次的处理器. 此外，鉴于 LU 分解问题在科学及工程计算中的重要作用，有大量的相关工作研究该算法在传统的集群、MPP 及 SMP 等系统中的并行化问题[13]②. 本文工作借鉴了现有并行实现的思想[13-15]，但是力图结合相对于传统并行机而言，众核体系结构独有的特征，如片上存储层次的高带宽和低访问延迟，在细粒度并行层次来研究 LU 分解的高效实现.

众核体系结构本质上是一个共享存储的并行处理系统. 本文的基本算法采取对矩阵分块的思想[11,13-15]，即首先将矩阵划分成若干大小相同的子块，然后按照一定的方法将各子块分配到各个核上计算. 由于众核体系结构侧重于挖掘细粒度并行，因此对处理器核负载的不均衡特别敏感，鉴于现有的二维卷帘(2D-Scatter Decomposition)子块分配方法[13-16]在众核体系结构上不能达到好的负责均衡效果，本文提出一种新的“之”字型子块分配方法，并在理论上证明该方法能获得比二维卷帘更好的负载均衡. 我们结合一个具体众核体系结构实现了提出的算法，并通过充分的实验验证“之”字形子块分配方案的性能优势. 结果表明在未经片上存储优化的情况下，比基于二维卷帘子块分配的实现性能提高了 20%，结合片上存储优化后性能提高达到了 40%以上. 实验结果还表明无论是“之”字形子块分配还是二维卷帘分配，合理利用片上存储进行优化对性能提高显著，达 1 倍以上.

对于并行算法设计而言，可预测的性能具有重要的指导意义，为此本文结合算法和片上网络的结构特征提出了一个性能加速比分析模型，并通过该模型得到了“之”字形子块分配和二维卷帘分配性能加速比的理论上限，分析模型的结果证明“之”字形子块分配能达到更好的性能加速. 为了验证分析模型的结论，我们用充分的实验分析实测加速比和理论加速比之间的差距，发现在合理使用片上存储优化并在子块大小为 16 时，试验测得性能加速效果能比较接近理论值；子块越小(块大小为 8 时)开销越大，性能也相应变差. 通过实验还证明加速比难以达到理论加速比的两个主要原因：访存带宽有限；片上网络的资源(如数据发送端口)竞争.

本文第 2 节介绍本文工作的实验平台——Godson-Tv1 众核体系结构；第 3 节介绍一种负载均衡的任务分配方法；第 4 节建立性能加速比分析模型；第 5 节是实验评估；第 6 节进一步介绍相关工作；最后一节总结全文.

2 Godson-Tv1 众核体系结构

Godson-T 是中国科学院计算技术研究所目前正在研究中的一个面向较大规模并行处理的众核体系结构[17]，本文的工作基于 Godson-T 第一版本的设计 Godson-Tv1，如图 1 所示. 目前 Godson-Tv1 支持 24 个计算节点(Tile Node)和一个专门用于支持高效片上同步的中央节点(Sync Node). 25 个片上节点由一个 5×5 的 MESH 网络互联，节点间采取静态路由策略. Tile Node 的结构如图中右半部分所示，每个计算节点由 4 个支持用户态 MIPS 指令的小核组成. 小核(TU)的设计采取了一个非常简单的顺序单发射 8 级流水线结构，包括一个定点部件、一个浮点部件和一个访存部件. 小核之间、小核和路由单元(router)之间通过交换开关(crossbar)互联.

① Asanovic K, Bodik R, Catanzaro B C et al. The landscape of parallel computing research: A view from berkeley. http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

② Dongarra J J, Bunch J R, Moler C B et al. LINPACK Users' Guide, SIAM, Philadelphia, PA, 1992. http://www.netlib.org/linpack/

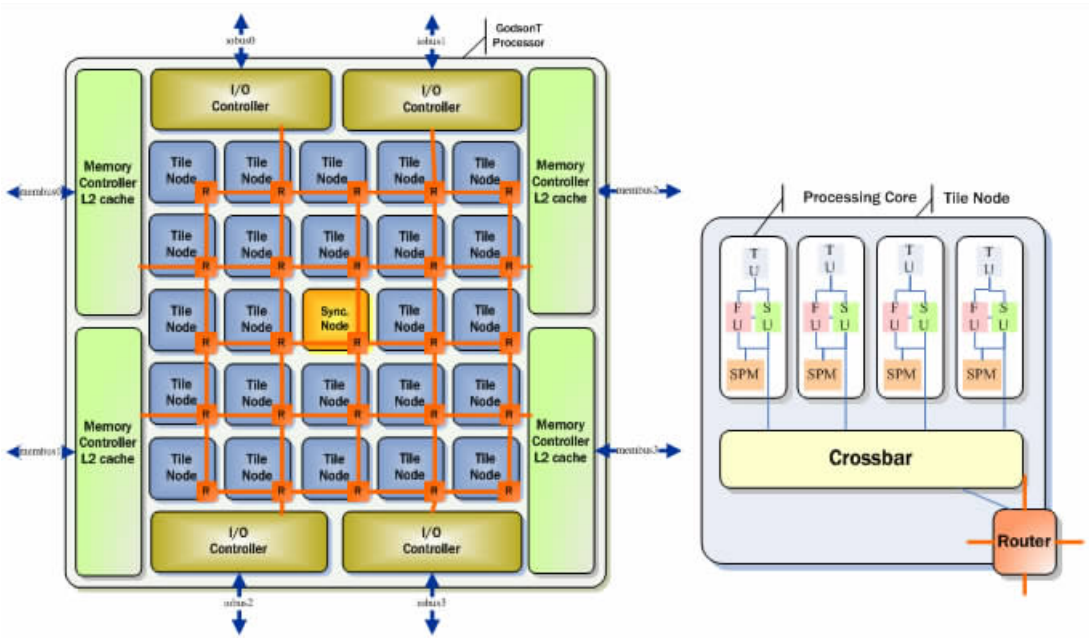

图 1　Godson-Tv1 众核体系结构

本文所依据的 Godson-Tv1 的存储层次是一个 L2 Cache 和 Scratch-Pad Memory(SPM)的混合结构. 首先,在芯片四周均匀分布 4 块 L2 Cache,每块容量为 512KB,取数延迟为 4 拍,每块 L2 Cache 和一个 Memory Controller 关联. 其次,每个小核内部署了一块 32KB 的私有的供程序员管理的 Scratch-Pad Memory,取数延迟为 1 拍. SPM 和 L2 Cache 之间的数据性由程序员维护. 采取 SPM+L2 Cache 混合设计具有一些重要的优点:(1) 去掉 L1 Cache 避免了复杂的硬件维护一致性的负担,使得设计更为简洁;(2) 利用 L2 Cache 来根据程序动态运行行为自动缓存最常使用的数据,同时给程序员足够的灵活性来利用 SPM 提高性能,这样充分结合了 Cache 和 SPM 各自的长处;(3) 由于 L2 Cache 被所有核共享,易于维护数据一致性,在性能要求不高的场合程序员不使用 SPM 很容易能保证程序的正确性.

然而也必须说明,Godson-Tv1 的这种存储层次和片上网络的设计也给程序员提出一些新的挑战. 一方面对于大部分性能要求较高的应用,程序员有必要仔细编程 SPM 来获得期望的性能;另一方面来自 MESH 网络,由于目前的设计采取静态路由策略,这使得当很多核同时访问相同 L2 Cache 的时候,大量的访存请求在 Router 上的竞争导致最终的取数延迟变长,同时工作的核越多、访问相同数据越频繁则问题越严重. 本文接下来详细讨论对于 LU 分解如何解决这些问题.

3　LU 分解在 Godson-Tv1 上的并行化

3.1　算法的控制结构

对规模为 N 的矩阵 $\boldsymbol{A}$ 进行 LU 分解,即求解下三角矩阵 $\boldsymbol{L}$ 和上三角矩阵 $\boldsymbol{U}$,使得 $\boldsymbol{A}=\boldsymbol{L}\times\boldsymbol{U}$. 为了能在 Godson-Tv1 这样的众核体系结构上高效进行 LU 分解,核心问题在于如何充分利用好片上存储,尤其是 SPM 空间. 由于片内存储容量的限制,处理器一次只能装下有限的矩阵规模. 当待分解矩阵规模 N 比较小时可以将所有数据取入片内 SPM 进行计算,但是当矩阵规模很大时,可考虑将问题分解成几个$\frac{N}{2}$规模的矩阵运算,令

$$\begin{pmatrix}\boldsymbol{A}_{00} & \boldsymbol{A}_{01}\\ \boldsymbol{A}_{10} & \boldsymbol{A}_{11}\end{pmatrix}=\begin{pmatrix}\boldsymbol{L}_{00} & \boldsymbol{0}\\ \boldsymbol{L}_{10} & \boldsymbol{L}_{11}\end{pmatrix}\times\begin{pmatrix}\boldsymbol{U}_{00} & \boldsymbol{U}_{01}\\ \boldsymbol{0} & \boldsymbol{U}_{11}\end{pmatrix},$$

则有

$$\begin{cases}\boldsymbol{A}_{00}=\boldsymbol{L}_{00}\times\boldsymbol{U}_{00}\\ \boldsymbol{A}_{01}=\boldsymbol{L}_{00}\times\boldsymbol{U}_{01}\\ \boldsymbol{A}_{10}=\boldsymbol{L}_{10}\times\boldsymbol{U}_{00}\\ \boldsymbol{A}_{11}=\boldsymbol{L}_{10}\times\boldsymbol{U}_{01}+\boldsymbol{L}_{11}\times\boldsymbol{U}_{11}\end{cases}.$$

由这个方程组可以得出对矩阵 $\boldsymbol{A}$ 分块进行 LU 分解的步骤:首先对子矩阵 $\boldsymbol{A}_{00}$ 进行 LU 分解得到子结果矩阵 $\boldsymbol{L}_{00}$ 和 $\boldsymbol{U}_{00}$;接下来用下三角子矩阵 $\boldsymbol{L}_{00}$ 计算 $\boldsymbol{U}_{01}$,并用上三角矩阵 $\boldsymbol{U}_{00}$ 计算 $\boldsymbol{L}_{10}$;最后一步把子矩阵 $\boldsymbol{A}_{11}$ 减去 $\boldsymbol{L}_{10}\times\boldsymbol{U}_{01}$ 的差进行 LU 分解,即得到 $\boldsymbol{L}_{11}$ 和 $\boldsymbol{U}_{11}$. 对问题照此分解下去,直到矩阵恰好可以

放入片内 SPM 空间时即可开始运算. 这样一个规模很大的矩阵分解可以分解成可直接计算的若干步,每一步计算是小规模的 LU 分解、三角方程组求解和矩阵乘法三者之一,步与步之间是串行,但是在计算每一步时充分利用片上存储调动所有小核并行计算. 对三角方程组和矩阵乘法求解超出了本文的范围,后文研究矩阵规模不超过片上存储容量的情况下如何实现高效 LU 分解.

3.2 负载均衡

本文以文献[14-15]的分块算法为基础进行研究. 如图 2 所示,设矩阵被划分成 8×8 个子块,根据 LU 分解计算的依赖关系,第 1 步对左上角标号为 1 的子块进行 LU 分解,第 2 步并行计算同行和同列(标号为 2)的所有子块,最后一步并行修正右下角(标号为 3)的所有子块. 由于每一步计算的结果在下一步被多个核使用,每一步之间用一个 BARRIER 操作同步[15].

1	2	2	2	2	2	2	2
2	3	3	3	3	3	3	3
2	3	3	3	3	3	3	3
2	3	3	3	3	3	3	3
2	3	3	3	3	3	3	3
2	3	3	3	3	3	3	3
2	3	3	3	3	3	3	3

图 2 一次迭代的处理过程图示

图 2 中标号相同的块表示对它们的计算可以并行完成. 由于除每一步以外,每次迭代之间都有 BARRIER 操作同步,为了获得尽可能好的加速性能,我们希望每一步分配到各个核上的计算任务是均衡的,特别是右下角子块的修正,因为当矩阵规模比较大时这一步耗时最多. 目前对这一问题常用解决方法是二维卷帘分解(2D-scatter decomposition),如图 3 所示,图中的数字表示处理器号,即把 8×8 个子块分配到 4 个处理器上的情形. 对照图 2 可以清楚地看出,在第 2 步修正同行/列子块时,处理器 4 始终是空闲的,处理器 1~3 分别需要修正 6、4 和 4 个子块;在第 3 步处理器 1~4 分别需要修正的子块数为 9、12、12 和 16. 可见,采用二维卷帘的方法并不能达到很好的负载均衡.

1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2
3	4	3	4	3	4	3	4

图 3 二维卷帘块分配

二维卷帘分配不能达到很好实现负载均衡的根本原因在于其没有很好地结合 LU 分解的依赖关系特征. 当然一种能实现负载均衡的方法是在每步计算开始前对所有核动态进行一次块分配,与此类似的思想体现在文献[11]中. 动态分配固然能保证很好的负载均衡,但是缺点在于每次重新分配都需要重新在所有核之间分配数据,这个过程会涉及核间大量的数据交换,这对于 Godson-Tv1 这样的结构会带来很大的通信开销. 为了能在使用静态块分配的同时还保证整个运行过程中每个迭代的负载均衡,本文提出一种"之"字形块分配方法,在介绍这种方法之前,我们先证明一个定理.

定理 1. 设有一个 $M\times M$ 方格阵,现用 P 种颜色对其中的方格着色,每个方格只能染一种颜色,则存在染色方案,使得所有方格染色完成后满足:对任意去掉左 $m(0\leq m<M)$ 列和上 m 行后的 $(M-m)\times(M-m)$ 子方格阵,令 C_i 为子方格阵中染了第 $i(1\leq i\leq P)$ 种颜色的方格数目,有 $\max\{C_1,C_2,\cdots,C_p\}-\min\{C_1,C_2,\cdots,C_p\}\leq 1$,其中 max 和 min 分别定义为对 P 个数求最大和最小值.

证明. 首先构造一种染色方案. 为讨论的方便,这里用坐标 $(i,j)(0\leq i<M,0\leq j<M)$ 表示每个方格在方格阵中的位置.

令

$$n=\min\{i,j\} \tag{1}$$

$$base=2\times M\times n-n^2 \tag{2}$$

$$inc=\begin{cases}M-i, & i\geq j\\ M+j-2\times i, & i<j\end{cases} \tag{3}$$

则将坐标位置为 $(i,j)(0\leq i<M,0\leq j<M)$ 的方格用颜色号 $((base+inc-1)\%P)+1$ 染色.

下面证明这种染色方案满足定理 1 的条件. 根据上述染色方案,令 $CN_{(i,j)}$ 为方格 (i,j) 的颜色号,对于 $i,j,k,l(0<i,j,k,l<M)$,只要下述条件(4)或(5)或(6)满足,则式(7)成立.

$$\begin{cases}k==i \text{ 且 } j-l=1 & (4)\\ i-k=1 \text{ 且 } l=j & (5)\\ k==j \text{ 且 } l-i=1 & (6)\end{cases}$$

$$(CN_{(k,l)}-CN_{(i,j)}+P)\%P=1 \tag{7}$$

现在来考虑去掉左 m 列和上 m 行后的子方格阵,按照如下顺序来遍历子阵中所有的方格并将遍历过方格的颜色号排成一列,即

$CN_{(M-1,m)},\cdots,CN_{(m,m)},\cdots,$

$CN_{(m,m+1)},CN_{(m,M-m-1)},\cdots,$

$CN_{(M-1,m+1)},\cdots,CN_{(m+1,m+1)},\cdots,$

$CN_{(m+1,m+2)},\cdots,CN_{(m+1,M-m-2)},\cdots,$

$CN_{(M-1,m+2)},\cdots,CN_{(m+2,m+2)},\cdots,$

$CN_{(m+2,m+3)},\cdots,CN_{(m+2,M-m-3)},\cdots,CN_{(M-1,M-1)}.$

在上述序列中任取两个位置连续的颜色号 CN_t 和 CN_{t+1},根据式(7)和条件(4)~(6)有

$$(CN_{t+1}-CN_t+P)\%P=1 \qquad (8)$$

这就是说,从方格$(M-1,m)$开始按照上述序列的顺序顺时针遍历右下角的方格阵,在遍历的过程中按照式(8)顺次轮换使用所有颜色对方格染色,那么任何颜色都不可能比其他颜色多使用两次,故定理 1 成立.　　证毕.

根据定理 1,只要把颜色号置换为处理器号即得到这里需要的子块分配方案,我们把对定理 1 证明过程中给出的分配方案叫"之"字形分配,一种可能的分配结果如图 4 所示. 对照图 3 可以清楚地看出,在第 2 步修正同行/列子块时,处理器 1~4 分别需要修正的子块数为 4、4、4 和 2 个子块;在第 3 步处理器 1~4 分别需要修正的子块数为 12、12、12 和 13. 比较图 4 和图 3 可以看出,"之"字形分配比二维卷帘能获得更好的负载均衡.

根据定理 1,之字形负载分配的一般结论是:在每次迭代的第 2 步(即修正同行/列子块),负载最重的处理器核最多比负载最轻的处理器核多计算 2 个子块(如图 4);每次迭代的第 3 步(即修正右下角的子块)负载最重的处理器最多比负载最轻的处理器核多计算 1 个子块.

4	1	2	3	4	1	2	3
3	2	3	4	1	2	3	4
2	1	2	3	4	1	2	3
1	4	1	4	1	2	3	4
4	3	4	3	4	1	2	3
3	2	3	2	3	2	3	4
2	1	2	1	2	1	2	3
1	4	1	4	1	4	1	4

图 4　"之"字形块分配

需要说明的是,尽管"之"字形分配方案理论上能实现比二维卷帘更好的负载均衡,实际实现未必总能得到更好的性能. 在传统的比较大型的并行机上,往往需要针对处理节点之间的通信代价进行特别的优化,二维卷帘简单直观,使通信优化变得容易,如文献[13]采用镜像二维卷帘来平衡负载. 但是在 Godson-Tv1 这样的众核体系结构中,核之间的数据交换通过共享的存储层次进行,不同核之间的物理距离对通信开销的影响很小,在这种情况下我们希望在每一步计算各个核计算的字块数目越平衡越好. 后面的实验结果也表明在 Godson-Tv1 这样的众核平台上"之"字形分配确实能获得比二维卷帘更好的性能.

3.3　体系结构相关的性能优化

对规模为 N 的矩阵进行 LU 分解需要的计算复杂度为 $O(N^3)$,而计算所需的数据规模为$O(N^2)$,可见数据重用度很高. 在计算的开始,各个小核依据"之"字形分配原则把矩阵中所有属于自己的子块取入到自己的 SPM 空间中,在接下来的每次迭代过程中,各小核根据上一次迭代的结果(只有这部分数据需要访问 L2 Cache)更新私有 SPM 中没有计算完成的所有子块,在每次迭代的结束,各小核把已经完成计算的子块写入到 L2 Cache 中供下次迭代使用,即相邻两次迭代之间的数据通信通过共享的 L2 Cache 进行,这样安排特别适合于 Godson-Tv1 这样的结构,原因在于:(1)根据 LU 分解的计算特征,每次迭代完成第 1 行和第 1 列所有子块的计算(参考图 1),根据定理 1 剩下的所有未完成计算的子块在处理器间分配仍然是均衡的,这样避免了迭代结束后重新分配数据的开销;(2)每次迭代所有核都需要到 L2 Cache 取相同的数据来更新私有 SPM 的子块,而 Godson-Tv1 采用静态 MESH 网络结构,由于路由是静态的,这样每次迭代多个核同时发出的大量请求会在网络上竞争,导致访问延迟变长,而且核越多竞争越严重. 缓解这一问题最简单的方法就是尽量减少每次迭代间的通信量,为此我们把计算需要的中间数据缓存在每个核的 SPM 中,在整个计算过程中都不需要移动,这样就完全避免了不必要的数据通信.

4　算法扩展性分析

4.1　性能加速比分析模型

本节分析基于"之"字形子块分配的并行 LU 实现和基于二维卷帘的实现的加速效果,即推导出 P 个处理器并行计算时的理论加速比公式. 了解一个并行算法的理论加速效果能够指导对算法的性能预测和算法本身的改进,例如工艺技术的更新换代使得将来处理器会集成更多的核,对算法本身扩展性的认识有助于解释算法在将来处理器上的行为. 算

法的实际加速效果和算法参数选取(如子块的大小)和处理器存储层次的优化有很大关系,我们将在下一节具体研究这些问题.

由前文第3节的讨论可以发现,基于分块思想的LU分解由4个基本子块运算组成(请参考图1):(1)对一个子块进行LU分解,时间设为 T_{lu};(2)根据左上角分解完成的子块修正同行的一个子块,本质上是解一组上三角方程组,时间设为 T_{upper};(3)根据左上角子块修正同列的一个子块,本质上是解一组下三角方程组,时间设为 T_{lower};(4)根据同行同列子块的结果修正右下角一个子块,核心计算是一个矩阵乘法,时间设为 T_{mod}. 对于Godson-Tv1这样的众核结构,所有核都是同构的,建模时认为同类子块运算有相同的时间. 这样我们就可以用 T_{lu}, T_{upper}, T_{lower}, T_{mod} 表示出二维卷帘分配和"之"字形分配情况下 P 个处理器的并行计算时间.

设待分解矩阵为 $\boldsymbol{A}$,规模为 $N\times N$,子块大小为 $B\times B$,则矩阵每行(列)有 $M=\dfrac{N}{B}$ 个子块,$A(i,j)$ 表示第 $i(0\leq i<M)$ 行第 $j(0\leq j<M)$ 个子块. 为讨论方便,将 P 个处理器中每个赋予一个标号 $p(1\leq p\leq P)$. 分块LU分解算法一共需要 M 次迭代,设 P 个处理器并行计算第 $k(0<k<M)$ 次迭代的时间为 TP_k,则 P 个处理器并行完成LU分解需要的总时间为

$$TP=\sum_{k=0}^{M-1}TP_k \tag{9}$$

为了得到 TP_k 的表达式,需要先定义几个函数:

$$f(i,j,p)=\begin{cases}1, & \text{块 } A(i,j)\text{由标号为 } p \text{ 的处理器计算}\\ 0, & \text{块 } A(i,j)\text{不由标号为 } p \text{ 的处理器计算}\end{cases},$$

$$g(i,j,k)=\begin{cases}1, & i=k \text{ 且 } j=k\\ 0, & i\neq k \text{ 或 } j\neq k\end{cases},$$

$$h(i,j,k)=\begin{cases}1, & i==k \text{ 且 } j>k\\ 0, & i\neq k \text{ 或 } j\leq k\end{cases},$$

$$e(i,j,k)=\begin{cases}1, & i>k \text{ 且 } j==k\\ 0, & i\leq k \text{ 或 } j\neq k\end{cases},$$

$$d(i,j,k)=\begin{cases}1, & i>k \text{ 且 } j>k\\ 0, & i\leq k \text{ 或 } j\leq k\end{cases},$$

其中 $g(i,j,k)$ 用来判断子块 $A(i,j)$ 是否是每次迭代左上角的对角块;$h(i,j,k)$ 用来判断子块 $A(i,j)$ 是否是和对角块同行的子块;$e(i,j,k)$ 用来判断子块 $A(i,j)$ 是否是和对角块同列的子块;$d(i,j,k)$ 用来判断是否是右下角的子块;$f(i,j,p)$ 用来判断子块 $A(i,j)$ 是否由标号为 p 的处理器来计算,这个判断函数因不同子块分配方案而不同,对于二维卷帘分配方案,首先需要对 P 进行因式分解,令 $P=rows\times cols$,则函数可以重写为

$$f_{\text{2d-scatter}}(i,j,p)=\begin{cases}1, & ((i\%rows)\times cols+j\times(cols))=p-1\\ 0, & ((i\%rows)\times cols+(j\times cols))\neq p-1\end{cases} \tag{10}$$

对于"之"字形分配,函数 $f(i,j,p)$ 可以重写为

$$f_{\text{z-shape}}(i,j,p)=\begin{cases}1, & ((base+inc-1)\%P)=p-1\\ 0, & ((base+inc-1)\%P)\neq p-1\end{cases} \tag{11}$$

注:$base$, inc 的表达式由式(2)和(3)给出.

于是在第 k 次迭代处理器 p 的计算时间为

$$\begin{aligned}T_k(p)=\sum_{i=k}^{M-1}\sum_{j=k}^{M-1}f(i,j,p)\times\{&g(i,j,k)\times T_{lu}+\\ &h(i,j,k)\times T_{\text{upper}}+e(i,j,k)\times T_{\text{lower}}+\\ &d(i,j,k)\times T_{\text{mod}}\}\end{aligned} \tag{12}$$

TP_k 由本次迭代任务最多的处理器消耗的时间决定,因此:

$$TP_k=\max\{T_k(1),T_k(2),\cdots,T_k(P)\} \tag{13}$$

至此,联立式(9)、(10)、(12)和(13)可得到二维卷帘子块分配方案下的并行处理时间 $TP_{\text{2d-scatter}}$;联立式(9)、(11)、(12)和(13)可得到"之"字形子块分配方案下的并行处理时间 $TP_{\text{z-shape}}$.

4.2 二维卷帘和"之"字形分配的加速效率比较

本节基于前一节的加速比模型来比较二维卷帘和"之"字形分配两种算法实现在Godson-Tv1上的理论加速能力. 在一阶分析时忽略存储层次不确定性(因为存在L2 Cache)和片上网络竞争等因素的影响,对于给定的块大小 B, T_{lu}, T_{upper}, T_{lower} 和 T_{mod} 的时间是确定的,因此可以直接用模拟器测量,根据4.1节给出的公式就可以计算出 $TP_{\text{2d-scatter}}$ 和 $TP_{\text{z-shape}}$. 图5给出了矩阵规模 $N=512$ 时不同块大

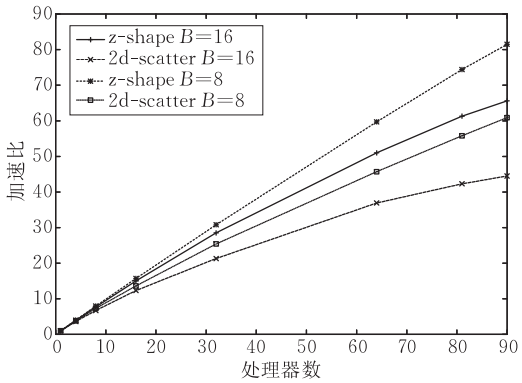

图5 二维卷帘和"之"字形分配的理论加速效果比较

小的情况下二维卷帘和“之”字形分配的理论加速效果.图中横坐标表示处理器核的数目,纵坐标表示算法的理论加速比.

图 5 中 z-shape $B=16$ 表示用“之”字形分配在子块大小为 16 时的加速特性曲线;2d-scatter $B=8$ 表示用二维卷帘分配在子块大小为 16 时的加速比曲线;另外为了研究加速特性和块大小的关系,还给出了两种算法实现下子块大小为 8 的曲线.由图中可以看出,基于“之”字形分配的加速效果相对二维卷帘有较大改善,而且处理器数 P 越大性能加速越明显.子块大小对加速效果也有一定影响,可以看出无论是“之”字形分配还是二维卷帘分配,子块越小加速特性越理想,“之”字形分配在子块大小 $B=8$ 时能基本得到线性关系的加速.另外,仔细比较图 5 基于二维卷帘分配在块大小为 16 时的曲线(2d-scatter $B=16$)和文献[14]中 LU 分解的加速比曲线可以看出,二者吻合得非常好,这说明文献[14]的结果符合 4.1 节的模型.

5　实验验证

由图 5 可以看出子块越小意味着并行粒度越小,越能够实现负载均衡,因此并行加速比最好.但是实际实现时块越小算法本身的控制结构越复杂,开销也越大,此外片上网络和存储层次的设计都会对最后的性能有影响.实际实现应该有一个合理的折中,本节结合在 Godson-Tv1 众核体系结构上的算法实现来研究这些因素和性能之间的关系.

5.1　实验平台

本文的实验平台是 Godson-Tv1 众核体系结构模拟器,该模拟器能精确模拟每个时钟周期的硬件行为.第 2 节描述了 Godson-Tv1 体系结构的概貌,这里补充一些和实验密切相关的内容.Godson-Tv1 提供类似 Cyclops 的多线程程序开发环境[1,8-9],每个核执行一个线程,线程在整个生命周期内是不可抢占的.所有线程共享同一地址空间,目前 Godson-Tv1 不支持虚存.Godson-Tv1 的 Runtime 实现了两个最基本的功能:一方面给用户程序提供必要的运行支持;另一方面抽象底层的实现细节,给程序员提供一个类似 Pthread 库的编程界面.Godson-Tv1 的运行环境启动后,其中一个小核为主线程代码.当执行到线程创建处时,由 Runtime 系统负责自动向子线程传递参数并启动子线程.子线程结束后,会通过 godsont_exit 系统调用通知主线程以完成一些资源释放之类的维护工作,目前 Godson-Tv1 的这种运行方式特别适合 SPMD 类型的并行程序.Godson-Tv1 的基本组成单元的参数指标如表 1 所示.

表 1　Godson-Tv1 的结构参数

组成单元	基本参数
Core	顺序单发射,8 级静态流水线,支持用户态 MIPS 指令集,定点、浮点和访存功能部件各一个.
Crossbar	每个节点(Tile Node)内有一个 7×7 的 Crossbar,仲裁延时为 1 拍,单个 Crossbar 最高带宽 56GB/s.
On-Chip-Network	片上网络为 5×5 MESH 结构,静态路由,小核到 L2 Cache 的请求和小核之间的通信请求按照 *X*-*Y* 选路路由,L2 Cache 到小核的 Refill 结果按照 *Y*-*X* 选路.
Router	4-stage 路由器,1 拍延时,2 个虚通道,每个路由器最高带宽 8GB/s.
SPM	集成在小核中的快速私有 SRAM 空间,读写延时为 1 拍.
L2 Cache	片上集成 4 块按照 Cache 行地址 interleave 的 L2 Cache,每块容量 512KB.Cache 行大小为 64 字节,访问延时为 2 拍,每块 L2 Cache 和一个 DDR2 Controller 对应.
DDR2 Controller	4 个 DDR2 Controller,Godson-Tv1 设计的工作频率为 500MHz.从 DRAM 取一个 Cache 行延迟为 52 拍,往 DRAM 写入一个 Cache 行延迟为 32 拍.DRAM 访问最高带宽(Peak Bandwidth)为 2.5GB/s.
Synchronization Unit	提供对 lock 和 barrier 的硬件支持,barrier 的同步代价为 10～100 拍,同时参与同步的线程越多,开销越大.

5.2　实验结果分析

5.2.1　片上 SPM 对性能的影响

图 6 给出了二维卷帘的实现和基于“之”字形子块分配的实现在使用 SPM 前后的性能效果比较结果($N=512$,$B=16$).图中 z-shape 和 2d-scatter 两条曲线表示加速的理论上限;z-L2 和 2d-L2 分别表示不使用 SPM 基于“之”字形分配的实现和二维卷帘实现的加速效果;z-SPM 和 2d-SPM 分别表示使用 SPM 缓存迭代中间结果数据后“之”字形实现和二维卷帘实现的加速效果.由图可以看出:(1)同样在没有使用 SPM 的情况下,基于“之”字形分配的实现因为能够更好地平衡负载,因此能获得比二维卷帘的实现更好的性能加速比.在两种实现都不使用 SPM 的情况下性能加速比差很多,主要原因来自于大量对 L2 Cache 的访存请求导致网络上竞争,从而使得访存延迟变得更加恶化;(2)使用 SPM 空间后,二维卷帘的实现能达到和理论值非常接近的加速比效果,说明通过 SPM 缓存中间计算结果能

够减少大量对 L2 Cache 的访问,从而缓解了片上网络的压力;(3)使用 SPM 空间"之"字形分配的实现在线程数目小于 64 的时候和理论值比较接近,但随后出现了一定的偏差,其中除了部分 L2 Cache 访问竞争的因素外,主要原因来自芯片本身的访存瓶颈,尽管如此"之"字形实现始终优于二维卷帘算法的加速特性.

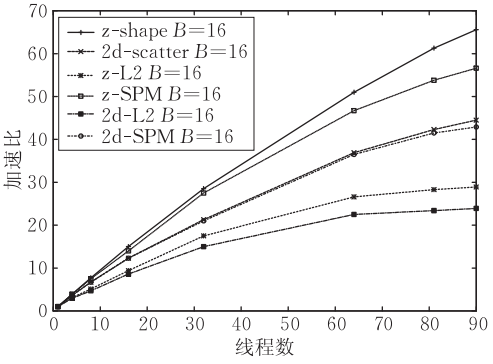

图 6 使用 SPM 和不使用 SPM 各种实现的加速特性比较

为了更好地看出几种算法实现在 Godson-Tv1 众核体系结构上的效果,图 7 给出了它们在线程数目分别为 64、81 和 90 时的性能比较.本文提及的性能定义为一种算法实现在 Godson-Tv1 模拟器上运行的时钟周期数.图 7 以二维卷帘的实现在不使用 SPM 情况下的性能为基准,其余几种实现参考该实现的性能归一化得到,每一组柱形条从左到右依次为 2d-L2、z-L2、2d-SPM 和 z-SPM.可以看出,使用 SPM 相对于不使用 SPM 至少有 1 倍左右的性能提高;使用 SPM 后,"之"字形分配的实现比二维卷帘的实现在线程数为 64、81 和 90 的情况下性能分别提高了 38%、43%和 48%,同时启动的线程数目越多性能改善越好;不使用 SPM 的情况下,"之"字形分配的实现比二维卷帘的实现在线程数为 64、81 和 90 的情况下性能分别有 19%、22%和 22%的性能提高.使用 SPM 空间后,"之"字形分配的实现优势

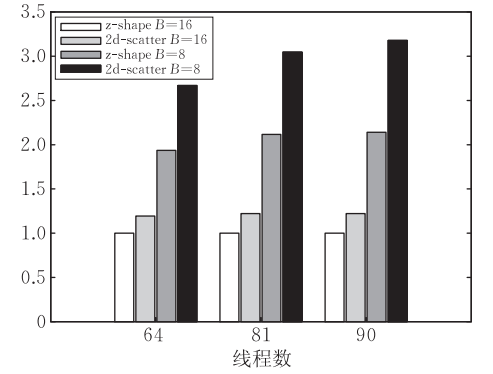

图 7 4 种实现的性能比较

更加明显,因为访存冲突的作用减弱,负载越均衡越能获得好的性能.尽管图 7 只给出了线程数大于 64 的结果,目的在于研究众核体系结构的潜能,但是在线程数目小于 64 的时候也能看到相同的性能趋势.

5.2.2 子块大小对性能的影响

在比较了块大小 $B=16$ 的情况下各种实现的效果后,一个自然的问题是如果块更小是否能获得更好的性能加速.为此图 8 给出了块大小 $B=8$ 时使用 SPM 获得的加速效果和理论值的比较.可以看出尽管理论上块越小加速效果越好,但是实测结果发现即使用上 SPM 加速比仍然和理论值有一定差距,特别是"之"字形分配的实现.尽管如此,我们还是可以清楚地看到"之"字形分配带来的负载均衡的优势:基于"之"字形分配的实现实测加速效果(z-SPM)甚至要略好于二维卷帘分配的理论预测值(2d-scatter).

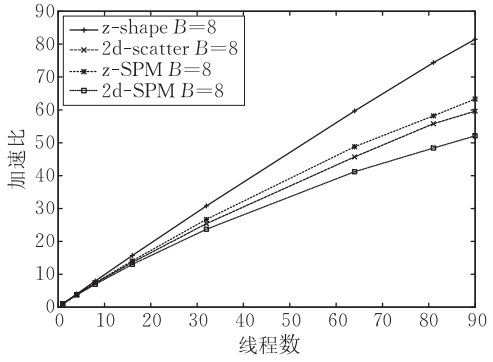

图 8 不同块大小的加速特性

影响加速比的因素是多方面的(如分块算法本身的开销(块越小算法控制结构越复杂)、同步开销、片上网络的竞争和访存延迟等等),块越小则对这些因素越敏感,相应获得理想加速比的难度就越大.

图 9 给出了使用 SPM 的情况下几种不同块大小的实现在线程数分别为 64、81 和 90 时的性能比较结果.和图 7 类似,这里以二维卷帘分配的实现在块大小 $B=8$ 的情况下测量的性能为基准来对其他

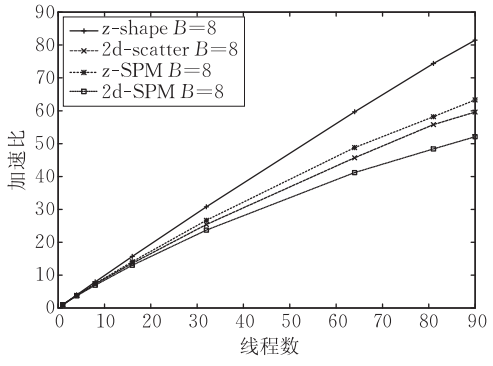

图 9 不同块大小的性能比较

3 种实现归一化,每组柱形条从左到右依次为 2d-SPM $B=8$、z-SPM $B=8$、2d-SPM $B=16$ 和 z-SPM $B=16$. 由图可知:(1) 无论是二维卷帘的实现还是“之”字形分配的实现,块大小为 16 时性能优于块大小为 8 时的性能,这说明当块大小为 8 时分块带来的开销已经完全抵消了其带来的负载均衡的优势;(2) 块大小为 8 的基于“之”字形块分配的实现(z-SPM $B=8$)的性能要优于块大小为 16 的基于二维卷帘的实现,这进一步说明“之”字形分配方案在 Godson-Tv1 这样的众核体系结构中优于通常的二维卷帘分配方案.

5.2.3　体系结构对性能的约束

从前面两个小节的实验分析能够看出体系结构本身对程序的实测性能有一定的制约作用. 对于 Godson-Tv1 而言,主要的体系结构约束一方面来自有限的访存带宽,另一方面来自片上网络.

由于面积、管脚数等物理实现的约束,众核体系结构无法在片上集成很多内存控制器,这使得片外内存访问带宽成为一个瓶颈. 本文用程序的计算时间和不能被隐藏的访存时间的比值,即计算访存比,来定量刻画 DRAM 带宽对性能加速比的影响. 现代处理器大多集成大容量的片上缓存,再加上预取机制使得大量的片外访存时间能够被容忍,能被容忍的延迟不会影响到程序的性能. 计算访存比越大说明访存时间所占的比重越小,越能接近峰值;反之访存时间的比重很大,计算的效率越低.

Godson-Tv1 目前暂不支持从 DRAM 到片上 SPM 直接数据传送的 DMA 通道,因此所有对 SPM 空间的存取都通过 Load/Store 指令实现. 由于本文的实现需要先将数据取入每个核的 SPM 空间才能开始计算,这样当同时启动的核数目很多时需要考虑到取数时间对性能的影响. 表 2 列出了二维卷帘和“之”字形分配两种实现分别在启动 64、81 和 90 个核时的计算访存比(矩阵规模为 512×512,块大小为 16×16). 综合表 2 和前面的讨论还可以看出,在同时启动的核数目很多的情况下,负载越均衡,单个核性能越好延迟容忍越重要,否则会影响芯片的运行效率. 这表明在众核体系结构上提供一些延迟容忍的硬件支持是必要的.

表 2　不同情况下的计算访存比

算法	计算访存比		
	64 核	81 核	90 核
“之”字形分配	69.8	63.7	61.4
二维卷帘	86.3	72.4	70.2

众核体系结构上另一个设计难点是片上网络. 实验表明,Godson-Tv1 的静态 X-Y 路由机制还不能很好地适应 LU 分解的算法结构. 对于 LU 分解算法而言,每次迭代的结束一部分核会将最终完成计算的子块写入到 L2 Cache,而在下次迭代开始时,所有的核需要访问上次迭代的结果来修正本地 SPM 中的数据,这样就会出现大量通往相同 L2 Bank 的请求,由于 L2 Cache 受理一个请求需要 2 拍,期间不能接受下一个请求,再加上网络是静态的 X-Y 路由,请求之间的资源竞争最终导致对 L2 Cache 的平均访问时间变长.

图 10 给出了“之”字形分配、块大小为 16 且不使用 SPM 的情况下随着线程数目的增加访问 L2 Cache 平均延时的变化,单位是 cycle. 在没有竞争的情况下大约需要 27 拍左右,当线程数增加到 4 的时候,由于竞争同一个 Crossbar 到 Router 的发送端口,使得平均延迟增加到 30 拍,当线程数目大于 4 的时候,不同的 Tile Node 的请求会在通往 L2 Cache 的 Router 上竞争相同的端口,这样会导致延迟进一步变长,当启动 90 个线程的时候平均访问延迟达到了 51.4 拍,接近没有竞争时候的 2 倍. 二维卷帘的实现在不使用 SPM 的情况得到的结论是类似的. 由图 10 可以看出,片上网络的资源(Crossbar 和 Router 端口)竞争是导致实测加速比难以接近理论值的主要原因,说明片上网络还有较大的改善空间. 在使用 SPM 优化后,由于大量计算用的数据缓存在 SPM 中,大大减少了 L2 Cache 的请求数量,片上网络竞争有所缓解,在子块大小为 16 的时候实测加速比能基本接近理论值(图 6).

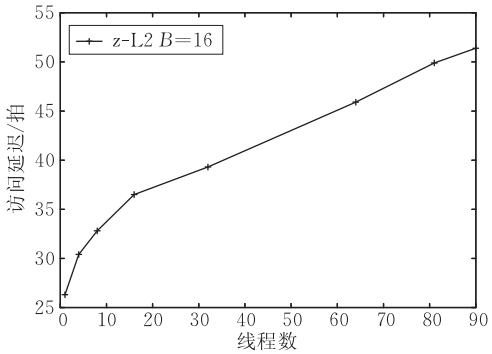

图 10　不使用 SPM 的情况下平均 L2 Cache 访问延迟和线程数目的关系

6　相关工作

文献[14-15]中描述了一种针对分布式共享存

储系统的 LU 分解算法,该算法是本文算法的基础,此外文献[11]也基于该算法在 C64 结构上进行了改进和优化.本文工作和文献[11]工作的相同点在于都着力改善负载均衡,但是区别在于:(1)文献[11]的方法需要在每次迭代重新调整子块大小,这意味着需要在核之间重新分配数据带来额外的通信开销;而本文的方法在静态分配的情况下就能实现较好的负载均衡;(2)文献[11]的方法非常适合于 C64 这样没有 Cache 的结构,本文的算法实现在两种结构上都适用.

关于 LU 分解有大量前人的研究,如文献[13,16,18]都有 LU 分解的 MPI 算法描述.和这些工作相比,本文工作的主要区别在于:针对 Godson-Tv1 这样的众核体系结构,在分析二维卷帘块分配的基础上,提出了"之"字形分配方案,并且用实验验证了该方案在众核体系结构上的优点."之"字形分配方案适合众核体系结构的根本原因在于核之间的物理距离很短,通过共享的 L2 Cache 就能高效交换数据,这是众核设计独有的优势,在大规模的集群或者 MPP 上是很困难的.必须要说明,在大型的分布式系统中,计算节点之间的通信代价差别很大,在那种情况下二维卷帘有其独特的优势.

近年来,众核体系结构越来越引起人们的兴趣[1-5].文献[19-22]提供了大量对 Tile 体系结构的研究结果.早期关于 Cyclops 的评估见文献[8-9],值得一提的是 Cyclops 最终演变成文献[1,5,10-12]中的结构.Intel 推出的 80 核众核芯片的研究见文献[2].

7 结论和将来的工作

LU 分解在科学和工程计算中有着重要的应用.本文立足于众核体系结构模型 Godson-Tv1,从负载均衡、延迟容忍和性能分析模型等方面研究了 LU 分解问题在众核体系结构上的并行化,结论如下:

(1)改进了二维卷帘子块分配方案难以实现良好的负载均衡的缺点,提出一种新的"之"字形分配策略,并证明该策略能获得比二维卷帘更好的负载均衡;

(2)在 Godson-Tv1 体系结构模型上用实验验证了"之"字形分配的有效性.用 Scratch-Pad-Memory(SPM)优化后实现"之"字形分配比二维卷帘分配性能高 40%,不使用 SPM 优化前者比后者高 20%;

(3)提出一个性能加速比的分析模型,并在 Godson-Tv1 上用实验分析了实测加速比和理论加速比之间的差距,发现在使用 SPM 优化并在块大小不小于 16 时,实测加速效果能比较接近理论值.通过实验证明了实测加速比难以达到理论加速比的两个主要原因为:访存带宽有限;片上网络竞争.

我们下一步的工作包括:一方面考虑增加一些结构支持来辅助程序员隐藏访存延迟和提高片上网络的利用效率;另一方面继续评估其它一些应用如在 Godson-Tv1 上的并行化效果.

参 考 文 献

[1] Cuvillo J D, Zhu W R, Hu Z A et al. FAST: A functionally accurate simulation toolset for the Cyclops64 cellular architecture//Proceedings of the Workshop on Modeling. Benchmarking and Simulation. Madison, Wisconsin, 2005: 11-20

[2] Vangal S, Howard J, Ruhl G et al. An 80-Tile 1.28TFLOPS network-on-chip in 65nm CMOS//Proceedings of the IEEE International Solid-State Circuits Conference. San Francisco, CA, 2007

[3] Borkar S Y, Mulder H, Dubey P et al. Platform 2015: Intel processor and platform evolution for the next decade. http:// www. cs. helsinki. fi/u/kerola/rio/papers/borkar _ 2015. pdf, 2005

[4] Dally W J. Computer architecture in the many-core era//Proceedings of the Keynote at the 24th International Conference on Computer Design. San Jose, California, 2006

[5] Zhu W R, Sreedhar V C, Hu Z A et al. Synchronization state buffer: Supporting efficient fine-grain synchronization for many-core architectures//Proceedings of the 34th International Symposium on Computer Architecture (ISCA2007). San Diego, CA, USA, 2007: 35-45

[6] McNairy C, Bhatia R. Montecito: A dual-core, dual-thread itanium processor. IEEE Micro, 2005, 25(2): 10-20

[7] Friedrich J, McCredie B, James N et al. Design of the POWER6 microprocessor//Proceedings of the IEEE International Solid-State Circuits Conference. San Francisco, CA, 2007

[8] Castanos C, Ceze J G, Denneau L et al. Evaluation of a multithreaded architecture for cellular computing//Proceedings of the 8th International Symposium on High Performance Computer Architecture. Boston, Massachusettes, 2002: 311-321

[9] Almasi G, Cascaval C, Castanos J G et al. Dissecting cyclops: A detailed analysis of a multithreaded architecture. ACM SIGARCH Computer Architecture News, 2003, 31(1): 26-38

[10] Tan G M, Sun N H, Gao G R. A parallel dynamic programming algorithm on a multi-core architecture//Proceedings of

the 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'07). San Diego, CA, USA, 2007: 135-144

[11] Venetis I E, Gao G R. Optimizing the LU benchmark for the cyclops-64 architecture. CAPSL Technical Memo 75, February, 2007

[12] Hu Z A, del Cuvillo J, Zhu Wei-Rong et al. Optimization of dense matrix nultiplication on IBM cyclops-64: Challenges and experiences//Proceedings of the 12th International European Conference on Parallel Processing (Euro-Par2006). Dresden, Germany, 2006

[13] Panziera J P, Baron J E. A highly efficient Linpack implementation based on shared-memory parallelism. Award Winning Paper, International SuperComputer Conference. Heidelberg, Germany, ISC 2005

[14] Woo S C, Ohara M, Torrie E et al. Methodological considerations and characterization of the SPLASH-2 parallel application suite//Proceedings of the 22nd Annual International Symposium on Computer Architecture. Santa Margherita Ligure, Italy, 1995: 24-36

[15] Woo S C, Singh J P, Hennessy J. The performance advantages of integrating block data transfer in cache-coherent multiprocessors//Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VI). San Jose, California, 1994: 219-229

[16] Zhang Lin-Bo, Chi Xue-Bin, Mo Ze-Yao, Li Ruo. Introduction to Parallel Computing. Beijing: Tsinghua University Press, 2006(in Chinese)
(张林波, 迟学斌, 莫则尧, 李若. 并行计算导论. 北京: 清华大学出版社, 2006)

[17] Advanced Micro-System Group. Godson-T Programming Manual Version 1.0. Technical Report, Institute of Computing Technology, Chinese Academy of Sciences, 2007(in Chinese)
(中科院计算所先进微系统组. GodsonT 编程手册 Version 1.0. 中国科学院计算技术研究所技术报告, 2007)

[18] Chen Guo-Liang, An Hong et al. Practice of Parallel Algorithms. Beijing: Higher Education Press, 2004(in Chinese)
(陈国良, 安虹等. 并行算法实践. 北京: 高等教育出版社, 2004)

[19] Sankaralingam K, Nagarajan R, Gratz P et al. The distributed microarchitecture of the TRIPS prototype processor//Proceedings of the 39th International Symposium on Microarchitecture (MICRO). Orlando, Florida, USA, 2006

[20] Sankaralingam K, Nagarajan R, Liu H et al. Exploiting ILP, TLP, and DLP using polymorphism in the TRIPS architecture//Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA). San Diego, California, USA, 2003: 422-433

[21] Taylor M B, Lee W, Miller J et al. Evaluation of the raw microprocessor: An exposed-wire-delay architecture for ILP and streams//Proceedings of the International Symposium on Computer Architecture. München, Germany, 2004

[22] Taylor M B, Kim J, Miller J et al. The raw microprocessor: A computational fabric for software circuits and general purpose programs. IEEE Micro, 2002, 22(2): 25-35

LONG Guo-Ping, born in 1982, Ph. D. candidate. His research interests include computer architecture, parallel programming, performance modeling and evaluation.

FAN Dong-Rui, born in 1979, Ph. D.. His research interests include computer architecture, parallel processing, run-time systems.

Background

The many-core architecture is increasingly becoming a promising computing platform due to the advancement of semi-conductor technology. Back in 2002, IBM researchers reported their early evaluation results of Cyclops many-core architecture. In 2007 ISSCC conference, Intel announced an 80-core TeraFlops prototype chip. The IBM Cyclops architecture had eventually evolved into the cache-less C64 architecture, which has been evaluated extensively by Prof. Guang R. Gao's group at University of Delaware.

This paper investigates the problem of parallelizing LU on Godson-T, a many core architecture. The authors study the problem from load balancing, latency hiding and performance modeling, specifically, propose a novel load balancing technique to overcome the limitations of 2D scatter decomposition, and propose an analytical model to understand the performance potential. The experimental results on Godson-T platform provide interesting observations regarding how to parallelize applications on many core architectures.