

# 基于软硬件的协同支持在众核上对 1-D FFT 算法的优化研究

周永彬 张军超 张 帅 张 浩

(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** 随着高性能计算需求的日益增加,片上众核(many-core)处理器成为未来处理器架构的发展方向.快速傅立叶变换(FFT)作为高性能计算中的重要应用,对计算能力和通信带宽都有较高的要求.因此基于众核处理器平台,实现高效、可扩展的 FFT 算法是算法和体系结构设计者共同面临的挑战.文中在众核处理器 Godson-T 平台上对 1-D FFT 算法进行了优化和评估,在节省几乎三分之一 L2 Cache 存储开销的情况下,通过隐藏矩阵转置,计算与通信重叠等优化策略,使得优化后的 1-D FFT 算法达到 3 倍以上的性能提升.并通过片上网络拥塞状况的实验分析,发现对于像 FFT 这样访存带宽受限的应用,增加 L2 Cache 的访问带宽,可以缓解因为爆发式读写带给片上网络和 L2 Cache 的压力,进一步提高程序的性能和扩展性.

**关键词** 众核; Godson-T; 快速傅立叶变换; 计算与通信重叠

**中图法分类号** TP302 **DOI 号:** 10.3724/SP.J.1016.2008.02005

## Software/Hardware Co-Design for 1-D FFT Optimization on Many-Core Architecture

ZHOU Yong-Bin ZHANG Jun-Chao ZHANG Shuai ZHANG Hao

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

**Abstract** As the increasing demand of high performance computing, many-core architecture becomes to the trend of future processor architecture. Fast Fourier Transform (FFT), both computing intensive and bandwidth intensive, is one of the most important applications of the high performance computing. For both software and hardware developers, it is a challenge to implement high efficiency and scalable FFT algorithm on many-core processor. Based on Godson-T processor, the authors developed an optimized implementation of 1-D FFT through implicitly matrix transpose hidden as well as overlapping computation and communication. The performance of optimized 1-D FFT algorithm achieves more than 3 times better and reduces almost 1/3 L2 Cache consumption. After the analysis of on-chip network congestion problem, the authors suggest that increasing the access bandwidth of L2 cache can alleviate the negative impact on on-chip network and L2 Cache which is brought by burst L2 Cache access. As a result, the performance and scalability of memory bandwidth limited applications, such as FFT, can be further improved.

**Keywords** many-core; Godson-T; fast Fourier transform; computation/communication overlapping

收稿日期:2008-05-31;最终修改稿收到日期:2008-09-17.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)和国家自然科学基金重点项目(60736012)资助.周永彬,男,1981年生,博士研究生,主要研究方向为体系结构以及并行算法的研究. E-mail: ybzhou@ict.ac.cn.张军超,男,1976年生,博士,主要研究方向为体系结构以及编译优化技术.张帅,男,1985年生,硕士研究生,主要研究方向为体系结构.张浩,男,1980年生,博士,主要研究方向为体系结构.

## 1 引言

半导体工艺的发展,使得微处理器设计面临的功耗、热扩散等问题日趋严重,片上多核处理器(CMP)已经逐渐代替单处理器成为通用处理器的主流产品.而面对日益增多的高性能计算需求,片上众核处理器(many-core)也开始成为研究的重点.随着处理器核个数的增加,计算资源也相对增多,在芯片面积的约束下,片上存储资源显得更加珍贵.在片上众核处理器平台下,提高片上存储资源和通信带宽的利用率,实现较高的计算效率是体系结构设计者的主要任务.对于算法设计人员,需要充分挖掘应用行为的并行性,并根据众核体系结构的特点实现高效、可扩展的算法.只有两者紧密结合起来,才能真正发挥出片上众核处理器平台的计算能力.

快速傅立叶变换(Fast Fourier Transform, FFT),是离散傅立叶变换的快速算法,广泛运用于通信、数字信号和图像处理、生物计算等领域.1965年 Cooley-Tukey FFT 算法<sup>[1]</sup>就已经被提出,至今仍然被 FFTW<sup>[2]</sup>等高效的 FFT 运算库所采用.而高效的并行 FFT 算法研究和实现一直是高性能计算中的关键问题,需要考虑到计算复杂度、存储开销、通信带宽和延迟等多种因素,但在面向通用的片上众核处理器平台上对 FFT 的研究工作正起步不久. Williams 等<sup>[3]</sup>基于 IBM 的 Cell 处理器,利用 DMA 和 Local Memory 对 FFT 算法进行优化,峰值达到 41.8Gflop/s. Govindaraju 等<sup>[4]</sup>基于图形处理器实现了 GPUFFT 库,峰值达到 29Gflops/s. Chen 等<sup>[5]</sup>基于 IBM Cyclops-64 众核处理器平台上对 FFT 算法进行了优化,峰值达到 20.7Gflops/s. Six-Step FFT<sup>[6]</sup>作为并行测试集 SPLASH-2<sup>[7]</sup>中 FFT 测试算法,是一种常用的 1-D FFT 并行算法,我们以此为基础,从软硬件协同支持的角度出发,在片上众核处理器 Godson-T 上对 1-D FFT 算法进行了评估.

这篇论文的主要贡献有 3 点:(1)在 Godson-T 的结构中增加了数据传输执行单元(Data Transfer Agent, DTA)作为处理器核的协处理器,通过其编程接口,可以将矩阵的行列转置隐藏在和 L2 Cache 的数据传输过程中,从而使得 Six-Step FFT 算法在 L2 Cache 中的存储开销减少近三分之一.(2)片上存储结构采用 Cache+SPM(Scratch-pad Memory)的混合存储结构,通过 DTA 操作可以在提高片上

存储资源和通信带宽利用率的基础上,实现计算与通信重叠的编程模式,相对于未经优化的 Six-Step FFT 算法,达到了 3 倍以上的性能提升.(3)通过分析优化后 1-D FFT 算法的片上网络延迟,发现增加 L2 Cache 的访问带宽能够缓解爆发式读写请求造成的网络拥塞和 L2 Cache 竞争,从而提高像 FFT 这种访存带宽受限程序的性能和扩展性,对 Godson-T 结构的优化以及其它片上众核处理器的结构设计具有指导意义.

本文第 2 节简单描述 Six-Step FFT 算法;第 3 节主要介绍 Godson-T 的结构以及 DTA 的功能和编程接口;第 4 节详细介绍我们基于 Godson-T 平台对 Six-Step FFT 算法进行的优化;第 5 节主要介绍实验方法和结果;第 6 节对全文进行总结.

## 2 Six-Step FFT 算法

这一节里我们对 Six-Step FFT 算法进行简单的介绍.对于离散傅立叶变换:

$$X_k = \sum_{j=0}^{n-1} x_j e^{-ijk \frac{2\pi}{n}}, \quad 0 \leq k < n \quad (1)$$

其中  $i = \sqrt{-1}$ , 令  $\omega_n = e^{-\frac{2\pi i}{n}}$  可得到

$$X_k = \sum_{j=0}^{n-1} x_j \omega_n^{jk}, \quad 0 \leq k < n \quad (2)$$

如果存在因数  $n_0$  和  $n_1$  使得  $n = n_0 n_1$ , 则可以将  $j, k$  写成:

$$\begin{cases} j = j_1 + j_0 n_1 \\ k = k_0 + k_1 n_0 \end{cases} \quad (3)$$

因此我们用两个 2 维矩阵定义  $x_j$  和  $X_k$ :

$$\begin{cases} x_n = x(j_0, j_1), \quad 0 \leq j_0 < n_0, \quad 0 \leq j_1 < n_1 \\ X_k = X(k_1, k_0), \quad 0 \leq k_0 < n_0, \quad 0 \leq k_1 < n_1 \end{cases} \quad (4)$$

将式(3)、(4)代入式(2)中,得

$$X(k_1, k_0) = \sum_{j_1=0}^{n_1-1} \sum_{j_0=0}^{n_0-1} x(j_0, j_1) \omega_{n_0}^{j_0 k_0} \omega_n^{j_1 k_0} \omega_{n_1}^{j_1 k_1} \quad (5)$$

将式(5)的计算分为两步,第 1 步计算  $n_0$  次长度为  $n_1$  的 FFT 得

$$X^{(1)}(k_0, j_1) = \omega_n^{j_1 k_0} \sum_{j_0=0}^{n_0-1} x(j_0, j_1) \omega_{n_0}^{j_0 k_0} \quad (6)$$

再从式(7)进行  $n_1$  次长度为  $n_0$  的 FFT 计算:

$$X^{(2)}(k_0, k_1) = \sum_{j_1=0}^{n_1-1} X^{(1)}(k_0, j_1) \omega_{n_1}^{j_1 k_1} \quad (7)$$

最后通过式(5)可以得到

$$X(k_1, k_0) = X^{(2)}(k_0, k_1) \tag{8}$$

从式(5)开始,一共要经过 6 步来计算 FFT:

- 1. 在式(6)中需要对  $n_0 \times n_1$  矩阵  $x(j_0, j_1)$  做多列的 FFT 计算,如果我们在内存中使用的是行存储的方式,则需对该矩阵做转置变换.
- 2. 对步 1 变换后得到的  $n_1 \times n_0$  矩阵做  $n_1$  次独立的一维  $n_0$  点 FFT 计算. 对应于式(6).
- 3. 将步 2 计算后得到的矩阵乘以旋转因子  $\omega_n^{j_1 k_0}$ . 对应于式(6).
- 4. 将步 3 计算后得到的矩阵做转置变换得到  $n_0 \times n_1$  矩阵,其作用与步 1 相同.
- 5. 对步 4 得到的  $n_0 \times n_1$  矩阵做  $n_0$  次独立的一维  $n_1$  点 FFT 计算. 对应于式(7).
- 6. 对步 5 计算后得到的矩阵做转置变换,即可得到  $n_1 \times n_0$  矩阵  $X(k_1, k_0)$ . 对应于式(8).

在步 2 中,radix-2 FFT 算法是经常被采用的 FFT 算法. 其时间复杂度为  $O(n \log_2 n)$ , 计算过程包含  $\log_2 n$  步,每一步包含  $\frac{n}{2}$  次蝶形运算,如图 1 所示,每个蝶形运算都需要一个复数乘法 and 2 个复数

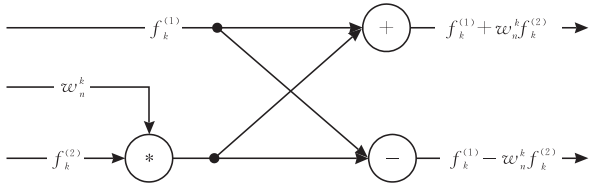


图 1 蝶形运算

加法,因此计算一个 radix-2 FFT 需要  $(n/2) \log_2 n$  次复数乘法 and  $n \log_2 n$  次复数加法.

对于前面所述的  $n$  输入的矩阵,我们假设一种简单情况,即  $\sqrt{n} = n_0 = n_1$ , 则输入矩阵为  $\sqrt{n} \times \sqrt{n}$ . 在步 2 中需要对各列各做一次 radix-2 FFT,在步 5 中需要对各行各做一次 radix-2 FFT,此外在步 3 中还要对矩阵每一项乘以一个旋转因子. 因此完成 Six-Step FFT 需要  $\left(\frac{\sqrt{n}}{2} \log_2 \sqrt{n}\right) \times 2 \sqrt{n} + n = n(\log_2 \sqrt{n} + 1)$  次复数乘法 and  $(\sqrt{n} \log_2 \sqrt{n}) \times 2 \sqrt{n} = 2n \log_2 \sqrt{n}$  次复数加法. 1 次复数乘法相当于 4 次实数乘法 and 2 次实数加法,1 次复数加法相当于 2 次实数加法,总的计算量又相当于  $4n(\log_2 \sqrt{n} + 1)$  次实数乘法 and  $6n \log_2 \sqrt{n} + 2n$  次实数加法.

### 3 多核处理器 Godson-T 结构

片上多核处理器 Godson-T(图 2)采用了 2D Mesh 结构,共有 64 个处理器核,片上网络采用静态 X-Y 选路策略,保证了片上网络的有序性,数据总线宽度为 16Bytes. 片上存储层次包括处理器核内的 Dcache 和 SPM 以及被分成 16 块的 L2 Cache,采用域存储一致性模型<sup>[8]</sup>.

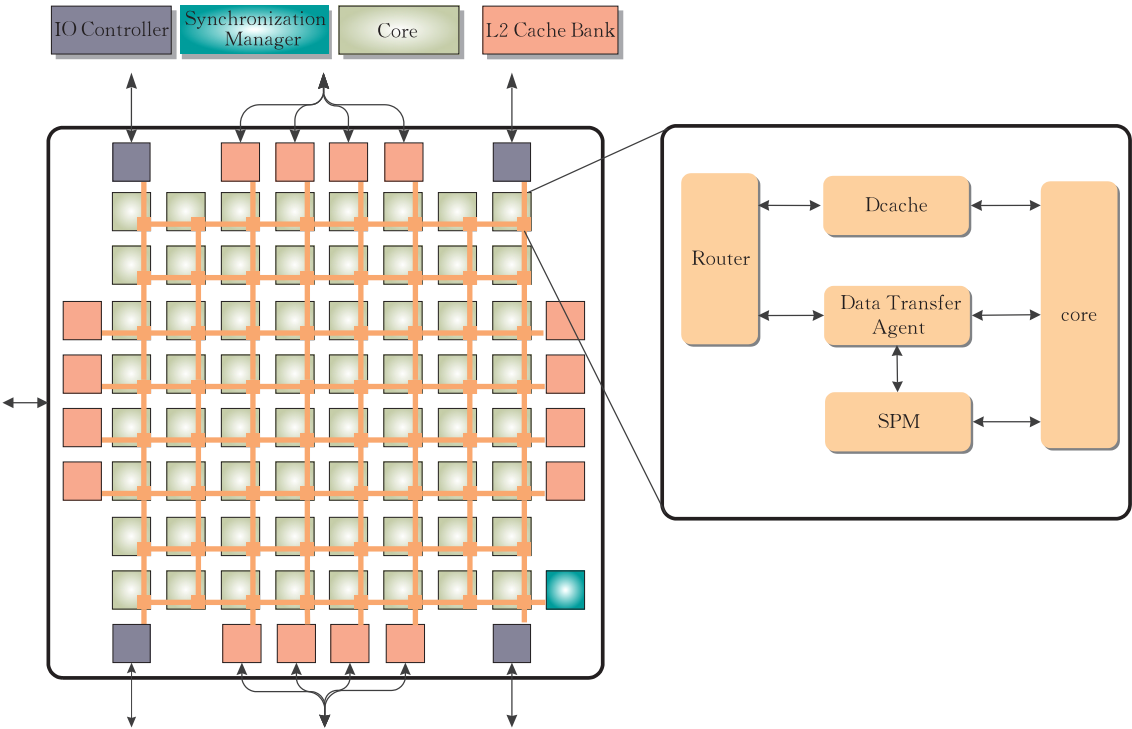


图 2 Godson-T 结构图

Godson-T 中的处理器核采用顺序双发射,包含一个浮点加法部件和一个浮点加法部件,工作频率为 1GHz,使得 Godson-T 的理论峰值可以达到 192Gflops/s. 处理器核内包含了 Dcache 和 SPM,通过全局地址空间进行划分,DTA 的主要作用是在远程和本地 SPM 之间进行连续数据块(图 3 所示)或者跨步长数据块(图 4 所示)的传输,其中阴影表示需要进行传输的数据块(block),数据块是由一个或

多个数据包(packet)组成,每个数据包的大小是 16Bytes. 在现有的设计中,如果数据块是不连续的,那么步长(stride)必须恒定.

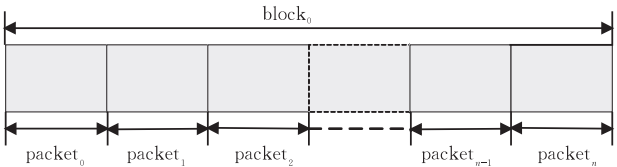


图 3 连续数据块

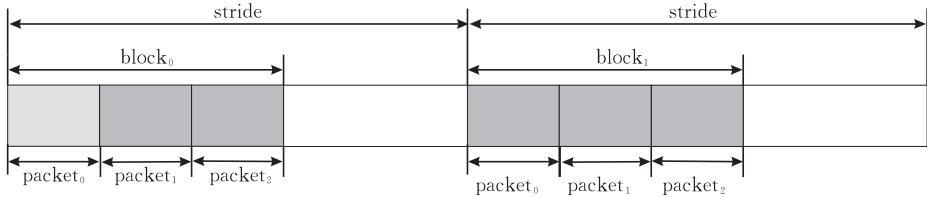


图 4 跨恒定步长数据块

作为 Core 的协处理器,DTA 为程序员提供了如下 API 编程接口:

```
godsont_dta_status (dta_status);  
    查询 DTA 的工作状态. 因为处理器核暂时没有实现中断功能,所以采用轮询的方式来查询 DTA 是否完成,dta_status 表明了当前 DTA 的状态,只有在 DTA 空闲的时候,才能进行新的 DTA 操作.  
godsont_dta_get (src_addr, src_stride, dst_addr,  
    dst_stride, packet_cnt, block_cnt);
```

从地址 *dst\_addr* 开始的远程存储空间读取数据,并将其存入从地址 *src\_addr* 开始的本地 SPM 存储空间. 如果远程存储空间中数据块是连续的,那么步长 *dst\_stride* 为 0,需要传输的数据块个数 *block\_cnt* 为 1, *packet\_cnt* 记录了需要传输的数据包的个数. 如果远程存储空间中数据块是不连续的,那么步长 *dst\_stride* 记录了前后两个数据块起始地址的间隔, *block\_cnt* 记录了需要传输的数据块个数, *packet\_cnt* 记录了每个数据块中需要传输的数据包的个数. 同理,在本地 SPM 存储空间中,步长 *src\_stride* 记录了前后两个数据块的间隔.

```
godsont_dta_put (src_addr, src_stride, dst_addr,  
    dst_stride, packet_cnt, block_cnt);
```

将从地址 *src\_addr* 开始的本地 SPM 存储空间中的数据送到从地址 *dst\_addr* 开始的远程存储空间中,其参数意义与 *godsont\_dta\_get* 相同.

4 对 Six-Step FFT 算法的优化

SPLASH-2 中的 FFT 对 Six-Step FFT 算法,

做了 $\sqrt{n}=n_0=n_1$ 的简化. 在 Godson-T 上,我们对这个算法的优化和实现做了研究. 我们假设处理器核数为  $p$ ,所有需要进行 FFT 的数据  $x[n]$ 包括它的转置  $x^T[n]$ 都存储在 L2 Cache 中,同时 L2 Cache 也存储了提前计算出结果的基于  $n$  的旋转因子  $w_n$ 和基于 $\sqrt{n}$ 的旋转因子  $w_{n_0}$ . 首先,我们对各个存储层次的开销进行分析,表 1 列出处理器核数为 64 时, Six-Step 单精度 FFT 算法在 Dcache/SPM 和 L2 Cache 中所需的存储空间,其中  $x_p[n]$ 表示每个处理器核分配的 $\sqrt{n}/p$ 行需要做 $\sqrt{n}$ 点 FFT 的矩阵元素,  $w_{pn}$ 表示分配到的矩阵元素所对应的基于  $n$  的旋转因子.

表 1 Six-Step FFT 算法在 Godson-T 上的存储开销

存储层次	数据	2 <sup>12</sup> (4K)	2 <sup>14</sup> (16K)	2 <sup>16</sup> (64K)
		FFT/KB	FFT/KB	FFT/KB
L2 Cache	$x[n]$	32	128	512
	$x^T[n]$	32	128	512
	$w_n$	32	128	512
	$w_{n_0}$	0.5	1	2
Dcache/SPM	$x_p[n]$	0.5	2	8
	$w_{pn}$	0.5	2	8
	$w_{n_0}$	0.5	1	2

Six-Step FFT 算法中的步 1、步 4 和步 6 都涉及到矩阵的转置,这个代价非常大,在后面第 5 节中我们可以看到. 如果将  $x_p[n]$ ,  $w_{pn}$  和  $w_{n_0}$  存储在处理器核的 SPM 空间中,通过调用 DTA 的 *godsont\_dta\_get* 或者 *godsont\_dta\_put* 编程接口,矩阵转置的过程则可以隐藏在 L2 Cache 和 SPM 之间的数据传输过程之中. 如图 5 所示,相同灰度的数据块是表示与同一个处理器核相关的. 处理器核对 SPM 中

的元素是以行为单位进行 FFT 计算的,同一行前后两个数据块是连续的,当这些数据块写入 L2 Cache

时,需要被映射到同一列上,才能达到转置的效果,相邻的步长为 $\sqrt{n}$ 点所占存储空间的大小.

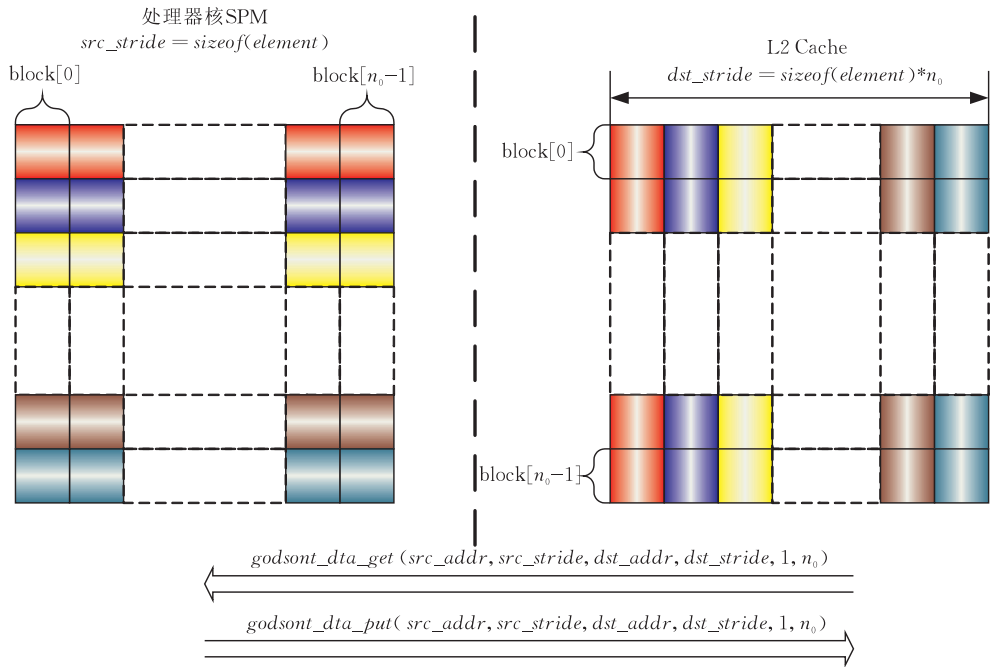


图 5 DTA 完成矩阵行列转置示意图

通过调用 DTA 进行数据传输,可以把 Six-Step FFT 算法按图 6 所示的方法实现,即 Six-Step FFT 的步 1、步 4 和步 6 的转置操作都被 DTA 隐藏在处理器核与 L2 Cache 的通信之中.在 DTA 第一次将 SPM 中的数据以列的方式写入 L2 Cache 中时,需要所有参与工作的处理器核进行一次同步,保证 DTA 再次从 L2 Cache 中取回的数据是正确的.

从图 6 所示策略中发现 DTA 和处理器核是串行工作方式,处理器核进行 $\sqrt{n}$ 点 FFT 的时候,需要

处理的各行之间也没有数据依赖,所以通过调用 DTA 进行数据预取,可采用计算与通信重叠的编程模式对其进一步优化,如图 7 所示.设处理器核 ID 为  $MyNum$ ,处理器个数为  $p$ ,每个处理器核对 $\sqrt{n} \times \sqrt{n}$ 矩阵  $x$  的  $(MyFirst: MyLast - 1)$  列进行 $\sqrt{n}$ 点 FFT 变换,其中  $MyFirst = \sqrt{n} \times MyNum / p$ ,  $MyLast = \sqrt{n} \times (MyNum + 1) / p$ .图 7 中以 SPM 中保存两列矩阵  $x$  的元素 ( $_x[0]$  和  $_x[1]$ ) 为例,实际应用中可以根据 SPM 的大小来确定.

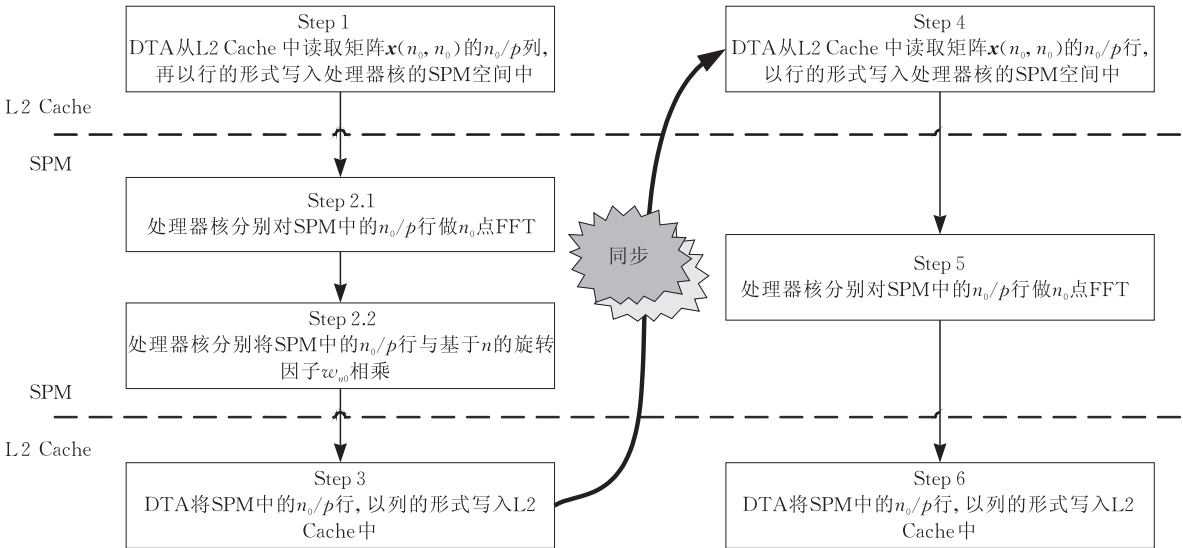


图 6 通过 DTA 对转置优化后的 Six-Step FFT 算法

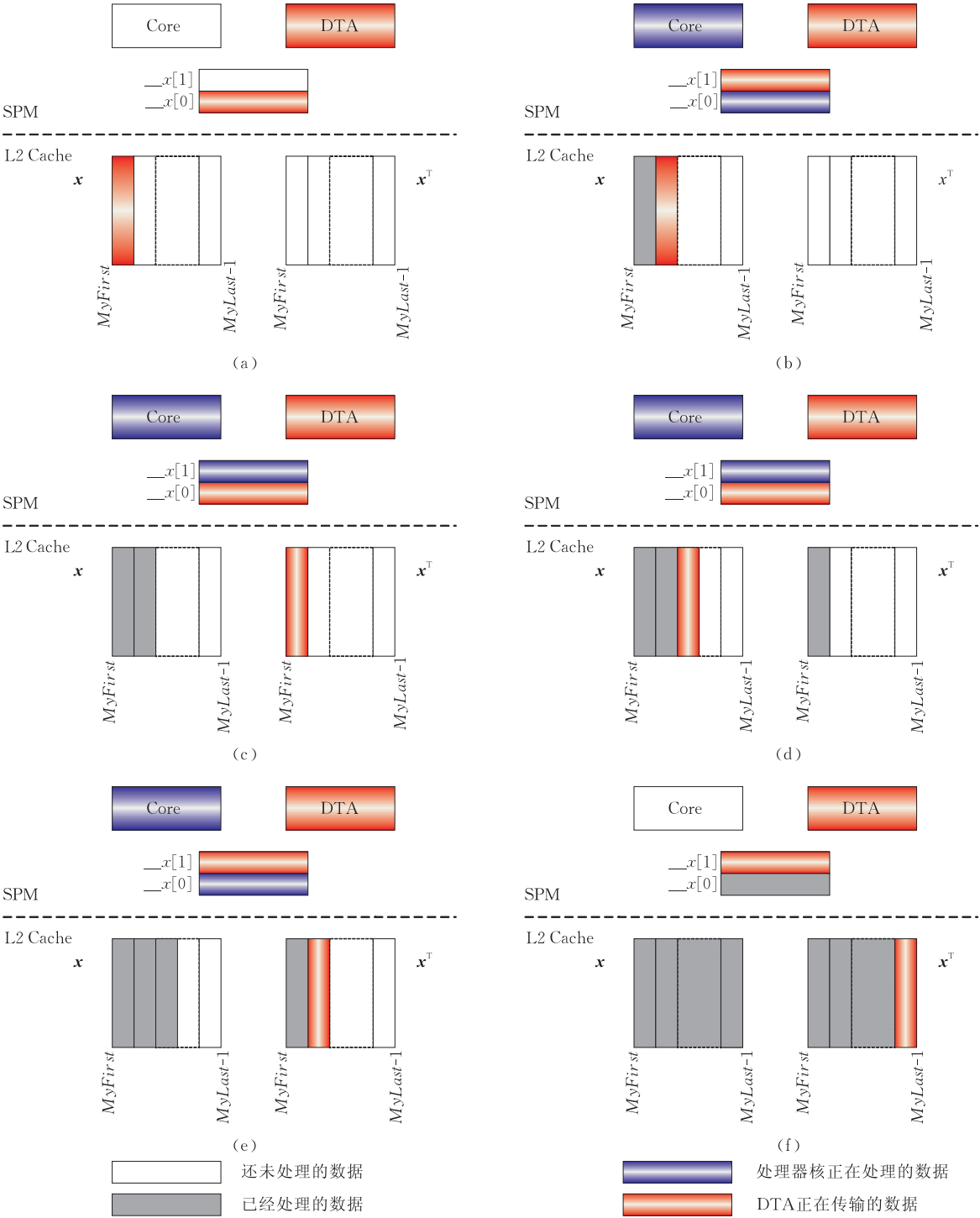


图 7 通过 DTA 实现计算与通信重叠的编程模型示意图

图 7 的(a)表示 DTA 首先将  $x[MyFirst]$ 列元素从 L2 Cache 取入  $x[0]$ 中,并在此过程中完成转置的任务,处理器处于空闲状态;

图 7 的(b)表示  $x[MyFirst]$ 列元素已全部存放在  $x[0]$ 中,处理器核开始对  $x[0]$ 进行 FFT 变换,而 DTA 开始将  $x[MyFirst+1]$ 列元素从 L2 Cache 预取到  $x[1]$ 中;

图 7 的(c)表示  $x[MyFirst+1]$ 列元素已全部存储到  $x[1]$ 中,处理器核也已经完成了对  $x[0]$ 的 FFT 变换,开始对  $x[1]$ 进行 FFT 变换,此时 DTA 将  $x[0]$ 元素存储到转置矩阵  $x^T[MyFirst]$ 列中.

图 7 的(d)表示  $x[0]$ 中的数据已全部存储到转置矩阵  $x^T[MyFirst]$ 列中,DTA 开始预取  $x[MyFirst+2]$ 列元素到  $x[0]$ 中,此时处理器核还

在对  $\_x[1]$  进行 FFT 变换 (也可能出现  $\_x[1]$  的 FFT 变换已经完成, 处理器核处于等待的情况);

图 7 的(e)表示  $x[MyFirst+2]$  列元素已经全部存储到  $\_x[0]$  中, 处理器核完成了  $\_x[1]$  的 FFT 变换, 开始对  $\_x[0]$  进行 FFT 变换, 此时 DTA 将  $\_x[1]$  的元素存储到转置矩阵  $x^T[MyFirst+1]$  列中;

图 7 的(f)表示 DTA 将  $\_x[1]$  中的元素写入到转置矩阵  $x^T[MyLast-1]$  列中, 此时处理器核处在空闲状态, 矩阵  $x(MyFirst, MyLast-1)$  所有列的 FFT 变换都已经完成。

图 7 描述了图 6 所示算法的步 1 到步 3 的实现过程, 该算法步 4 到步 6 也可以用类似的方法实现。通过这种模式, 处理器核和 DTA 的工作在一定程度上重叠在一起, 有利于减小或者消除处理器核和 L2 Cache 之间的通信开销。

设处理器核完成  $\sqrt{n}$  点 FFT 需要的时间为  $T_F$  拍, DTA 从 L2 Cache 取回  $\sqrt{n}$  点需要的时间为  $T_L = \tau_L \times \sqrt{n}$ , 其中  $\tau_L$  为 DTA 从 L2 Cache 读取数据的平均延迟, DTA 将  $\sqrt{n}$  个经过 FFT 变换的点写入 L2 Cache 的时间为  $T_S = \tau_S \times \sqrt{n}$ , 其中  $\tau_S$  为 DTA 向 L2 Cache 写数据的平均延迟。从图 7(c) 和图 7(d) 可以看出, 要将访问 L2 Cache 的延迟尽可能隐藏在计算之中, 需要满足  $T_S + T_L \leq T_F$ , 即

$$\tau_L + \tau_S \leq T_F / \sqrt{n} \quad (9)$$

由第 2 节的介绍可知, 进行  $\sqrt{n}$  点 FFT 需要  $2\sqrt{n}\log_2\sqrt{n}$  实数乘法和  $3\sqrt{n}\log_2\sqrt{n}$  实数加法 (为了简单, 没有考虑进行 bit-reverse 和完成  $\sqrt{n}$  点 FFT 后与基于  $n$  的旋转因子相乘的处理器核开销), 如果采用单精度浮点, 则 FLOPS 为  $5\sqrt{n}\log_2\sqrt{n}$ 。假设理想情况下, 处理器核每拍能完成 3 个 FLOPS, 则  $T_F = 1.67\sqrt{n}\log_2\sqrt{n}$ , 代入式(9)得

$$\tau_L + \tau_S \leq 1.67\log_2\sqrt{n} \quad (10)$$

当处理器核和 L2 Cache 的平均访问延迟满足式(10)的时候, 处理器核和 L2 Cache 的通信延迟则能完全隐藏在  $\sqrt{n}$  点 FFT 变换的计算过程之中。

对图 7 进一步观察可以发现, 通过 DTA 实现矩阵的转置后, 矩阵元素的位置在矩阵  $x$  和转置后矩阵  $x^T$  中是完全一致的, 例如图 7 中的(a)和(c)所示。在此过程中, 处理器核之间完全独立, 没有任何竞争, 即经过优化后的 Six-Step FFT 算法的步 1 到步 3, 可以不需要转置矩阵  $x^T$  来存放中间结果, 继续分析算法的步 4 和步 6 却发现矩阵元素在矩阵  $x$  和

转置矩阵  $x^T$  中的位置发生了变化, 处理器核之间存在竞争的可能, 但是当 SPM 空间能够存储其处理器核所需处理的数据时, 可以在算法的步 4 让所有的处理器核都将所需的数据取入各自的 SPM 中, 避免发生竞争, 因此转置矩阵  $x^T$  可以和  $x$  共享同一地址空间, 从表 1 可以看出这几乎减少了 FFT 算法在 L2 Cache 中三分之一的存储开销, 大大提高了片上存储资源的利用率。

## 5 实验方法和结果

我们的实验是基于 Godson-T 时间精确的事件驱动模拟器完成的, 假设工作频率为 1GHz。为了评估基于软硬件支持的优化策略, 首先将 Godson-T 的存储结构配置成传统方案 (以下图例中用“Dcache+L2(16 Banks)”表示) 和优化方案 (以下各图图例中用“Dcache+SPM+L2(16 Banks)”表示) 两种, 如表 2 所示。传统方案没有使用 DTA 和 SPM, 采用 SPLASH-2 中对 Cache 进行了优化的 Six-Step FFT 算法, 其 Dcache 大小设置为 32KB。优化方案使用了 DTA 和 SPM, 采用本文图 7 所示的对 Six-Step FFT 的优化策略, SPM 设置为 24KB, Dcache 设置为 8KB。这两种方案中, L2 Cache 均被分为 16 个体, 与 Godson-T 的处理器核通过片上网络周围的片上路由节点相连 (如图 2 所示), 总容量为 2MB。Dcache 和 SPM 的访问延迟均为 1 拍, L2 Cache 全流水设计, 读写延迟均为 4 拍, 片上路由延迟为 2 拍。我们对单精度浮点的 4K FFT, 16K FFT 和 64K FFT 进行了评估。

表 2 实验方案的配置

	L2 Cache/ MB	Dcache/ KB	SPM/ KB	片上存储资源 大小/MB
传统方案	2	32	无	4
优化方案	2	8	24	4

首先, 我们基于传统方案对 Six-Step FFT 算法中步 1、步 4 和步 6 矩阵转置的开销 (转置过程耗费时间占 FFT 总运算时间的比例) 进行了测试, 如图 8 所示。当工作的处理器核数逐渐增大的时候, 矩阵转置的开销在 17% 到 23% 之间, 这从侧面说明了利用 DTA 操作消除转置过程的优势。

图 9 是传统方案和优化方案在 4K FFT, 16K FFT 和 64K FFT 变换时的性能对比。随着工作的处理器核数增加, 两种方案的性能都有不同程度的提升。当工作处理器核数为 64 的时候, 优化方案在



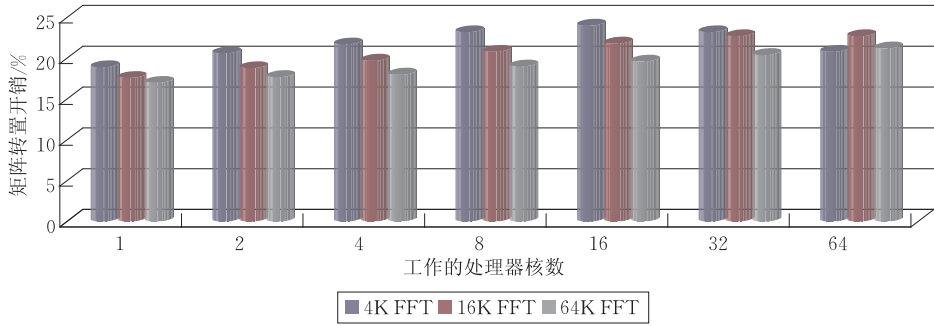


图 8 基于 Godson-T 的矩阵转置开销

计算 4K FFT 时性能达到 14.8Gflops/s, 16K FFT 的时性能达到 21.7Gflops/s, 64K FFT 时性能达到 33.6Gflops/s. 由此可以看出,FFT 规模增大使得并行的粒度增粗,图 6 算法中的同步开销所占比重降低,因此计算效率相应得到提高. 在相同数据规模和工作处理器核数的时候,相对于传统方案,通过 DTA 支持的软件预取、计算与通信折叠等优化策略,优化方案性能在 4K FFT 时提高 3.6 到 5.4 倍,在 16K FFT 时提高 3 到 5.1 倍,在 64K FFT 时提高 3.6 到 5.5 倍. 但工作的处理器核数为 64 的时候,优化方案的性能在 4K FFT 时提高 4.3 倍,16K FFT 时提高 3 倍,64K FFT 时提高 3.6 倍. 为此,我们以一个工作处理器核运行的性能作为基准,通过测试其加速比,对两种方案下的算法可扩展性进行分析,如图 10 所示.

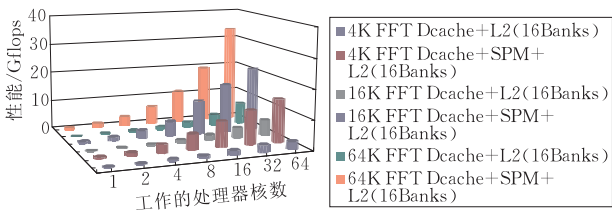


图 9 传统方案和优化方案下 FFT 性能比较

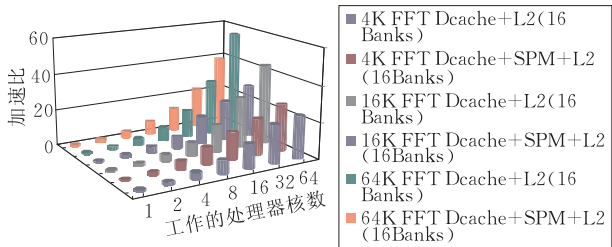


图 10 传统方案和优化方案下 FFT 可扩展性比较

从图 10 中发现,当工作的处理器核数较少(小于 16)时,无论 FFT 的数据规模如何,传统方案和优化方案的加速比都很接近. 当工作的处理器核数较多(多于 16)时,传统方案的加速比开始高于优化方案的加速比,随着数据规模和工作的处理器核数

增多,这个差距越来越明显. 因为优化方案通过 DTA 以爆发式(burst)读写的方式进行数据传输,相对于传统方案,数据传输速率加快,增加了对片上网络和 L2 Cache 的压力. 我们将通过片上网络和 L2 Cache 进行数据通信的消息分为三种类型:L2 Cache 读请求,L2 Cache 写请求以及 L2 Cache 读返回. 图 11 描述了这 3 种类型消息通过片上网络的平均延迟. 从图中可以看出,随着处理器核数和数据规模的增大,传统方案中这三种类型的消息在网络上的平均延迟差别不大,在 15~20 拍之间. 而优化方案中,这三种类型的消息通过片上网络的平均延迟却出现了较大的差异. 其中 L2 Cache 读请求的平均延迟恶化得最严重,不仅比对应传统方案的平均延迟增加了 2~3 倍,而且随着工作处理器核数增多,增长也较为明显. 而 L2 Cache 读返回消息的平均延迟受到的影响最小,随工作处理器核数增多变化不明显,几乎和对应的传统方案相当. L2 Cache 写请求的平均延迟介于两者之间,相对于对应的传统方案增加了接近 1 倍. 由此可以推断,因为分成 16 体的 L2 Cache 输入带宽限制,造成了当工作的处理器核数较多时,爆发式读写请求使得 L2 Cache 访问端口竞争加剧,导致消息在片上网络上拥堵,网络平均延迟增加,最终表现为图 10 优化方案的加速比相对于传统方案有所下降的现象.

为了解决爆发式读写给网络和 L2 Cache 带来的负面影响,我们在优化方案的基础上,将 Godson-T 原来分成 16 块的 L2 Cache 改为分成 64 块的组织结构,从分布在片上网络四周的路由节点改为每个路由节点上面分配一个 L2 Cache 块,同时每块的容量减少为原来的四分之一,保证 L2 Cache 的总容量仍为 2MB. 这样的 L2 Cache 组织方式(以下图例中用 Dcache+SPM+L2(64 Banks)表示)不仅使得 L2 Cache 的带宽变为原来的 4 倍,同时使处理器核与 L2 Cache 块之间局部性更加直观,有利于根据程



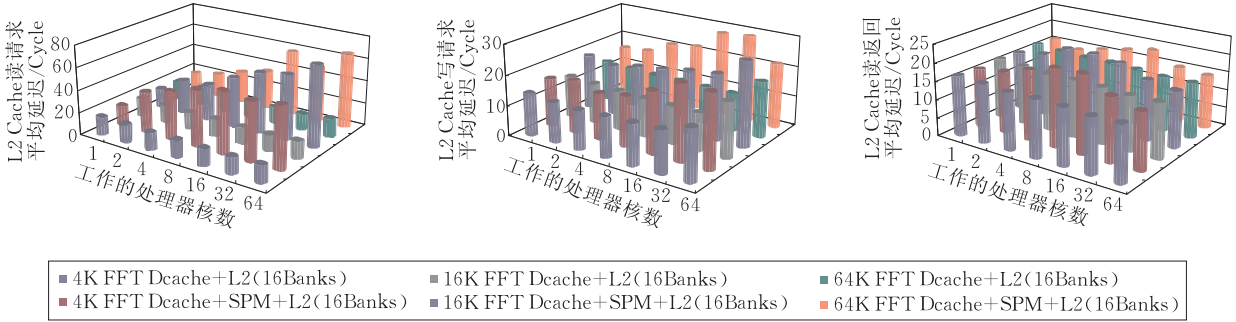


图 11 片上网络平均延迟

序和数据的亲近性来进行数据分布. 图 12 和图 13 分别显示了在进行 64K FFT 变换时,传统方案,分别基于 16 块和 64 块 L2 Cache 的优化方案在性能和加速比两方面的比较. 很明显,当采用 64 块 L2 Cache 组织方式时,优化方案的性能和扩展性都得到了更进一步得改善,当工作的处理器核数达到 64 时,性能达到了 47.2Gflops/s, 相当于一个处理器核工作性能的 56.4 倍,和传统方案的加速比相当. 因此,增加 L2 Cache 的访存带宽,可以缓解爆发式读写对 L2 Cache 和片上网络的压力,使得程序获得更好的性能和扩展性.

Cyclops-64,以 24.6%位居第二.

表 3 64K 单精度 1-D FFT 性能比较

	性能/Gflops	峰值效率/%
IBM Cell <sup>[3]</sup>	41.8	20.4
NVIDIA GeForce 8800 GTX <sup>[4]</sup>	29.0	7.4
IBM Cyclops-64 <sup>[5]</sup>	20.7	25.8
Godson-T	47.2	24.6

6 结 论

在对 1-D FFT 算法进行优化的过程中,通过 SPM 和 DTA 的共同支持,不仅使得 Six-Step FFT 算法中的转置操作完全隐藏在数据传输过程之中,而且当处理器核的 SPM 空间能够存储其进行 FFT 变换的所需数据时,可以将存放转置矩阵的地址空间和原矩阵空降共享,将 L2 Cache 的开销减少近三分之一,大大提高了片上存储资源的利用率.

另一方面,通过 DTA 的编程接口,采用计算与通信重叠的编程模式,通信的延迟被隐藏在计算过程之中,容忍了因为 DTA 的爆发式读写造成的片上网络压力增大,网络延迟增加的负面影响,使得其相对于原始的 Six-Step FFT 算法,相同处理器核数的性能提升了 3 倍以上.

除此以外,通过片上网络平均延迟和扩展性分析,我们发现在采用软硬件协同支持的优化策略之后,各个处理器核对 L2 Cache 访问频率提高,竞争加剧,而 L2 Cache 带宽的限制使得对 L2 Cache 的访问造成了网络拥塞. 当将 L2 Cache 由 16 块变为 64 块时,L2 Cache 的访问带宽也相应增加为原来的 4 倍,程序的性能和扩展性都得到了较大程度的提高.

最后,我们将 64K 单精度 FFT 的最终优化结果和 NVIDIA 的 GeForce 8800GTX、IBM 的 Cell 以及 Cyclops-64 进行了比较,在取得 47.2Gflops/s 最高性能的同时,保持了 24.6%的峰值效率,证明

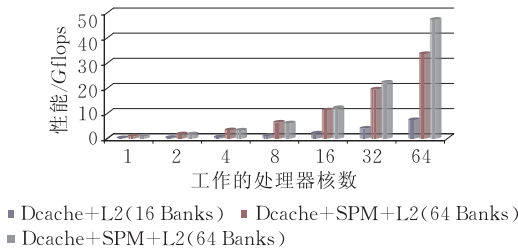


图 12 64K FFT 性能比较

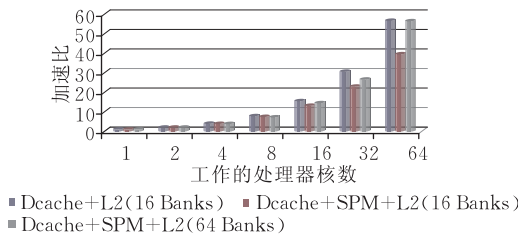


图 13 64K FFT 加速比较

最后,我们基于 64 块 L2 Cache 组织方式,在 Godson-T 上将优化方案得到的 64K 单精度 1-D FFT 性能和 IBM 的 Cell 处理器,IBM 的 Cyclops-64 众核处理器以及 NVIDIA 的 GeForce 8800 GTX 进行了比较,如表 3 所示. 从表中我们可以看出,基于 Godson-T 软硬件协同支持的 1-D FFT 性能超过了 IBM 的 Cell 处理器,以 47.2Gflops/s 位居第一,而峰值效率(即实际性能/峰值性能)仅次于 IBM 的

我们提出的基于软硬件协同支持的 1-D FFT 优化算法是一种在众核平台上行之有效的方法,对众核处理器的体系结构设计也具有一定的参考意义.

参 考 文 献

[1] Cooley J W, Tukey J W. An algorithm for the machine computation of the complex fourier series. *Mathematics of Computation*, 1965, 19(90): 297-301

[2] Frigo M, Johson S G. The design and implementation of FFTW3. *Proceedings of the IEEE*, 2005, 93(2): 216-231

[3] Williams Samuel, Shalf John, Olikier Leonid, Kamil Shoaib, Husbands Parry, Yelick Katherine. Scientific computing kernels on the Cell processor. *International Journal of Parallel Programming*, 2007, 35(3): 263-298

[4] Govindaraju Naga K, Larsen Scott, Gray Jim, Manocha Dinesh. A memory model for scientific algorithms on graphics

processors//*Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*. Tampa, Florida, 2006

[5] Chen Long, Hu Ziang, Lin Jun-Min, Gao Guang R. Optimizing fast fourier transform on a multi-core architecture//*Proceedings of the IEEE International Parallel and Distributed Processing Symposium*. California, USA, 2007: 499

[6] Bailey D H. FFTs in external or hierarchical memory. *Journal of Supercomputing*, 1990, 4(1): 23-35

[7] Woo Steven Cameron, Ohara Moriyoshi, Torrie Evan, Singh Jaswinder Pal, Gupta Anoop. The SPLASH-2 programs: Characterization and methodological considerations//*Proceedings of the 22nd International Symposium on Computer Architecture*. S. Marghenta Ligure, Italy, 1995: 24-36

[8] Iftode Liviu, Singh Jaswinder Pal, Li Kai. Scope consistency: A bridge between release consistency and entry consistency//*Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*. Padua, Italy, 1996: 277-287



**ZHOU Yong-Bin**, born in 1981, Ph.D. candidate. His research interests include processor architecture and parallel algorithm.

**ZHANG Jun-Chao**, born in 1976, Ph.D.. His research interests include processor architecture and compiler

**ZHANG Shuai**, born in 1985, M. S. candidate. His research interest is in processor architecture.

**ZHANG Hao**, born in 1980, Ph.D.. His research interest is in processor architecture.