异构平台上基于OpenCL的FFT实现与优化

李 焱[1,2,3] 张云泉[1,2] 王 可[1,2] 赵美超[1,2,3]
(中国科学院软件研究所并行软件与计算科学实验室 北京 100190)[1]
(中国科学院软件研究所计算机科学国家重点实验室 北京 100190)[2]
(中国科学院研究生院 北京 100190)[3]

摘 要 快速傅立叶变换作为20世纪公认的最重要的基础算法之一,在大规模科学计算处理、数字信号处理、图形图像仿真等众多领域有着广泛的应用。OpenCL是首个面向异构系统通用的并行编程标准,为软件开发人员提供了统一的面向异构系统的并行编程环境。首先,在异构平台Cell和GPU上使用OpenCL实现了基于2的幂一维FFT,并对其进行了测试和分析,在Cell平台上当数据规模适中时它能够达到SDK性能的65%,当数据规模继续增大时,相对性能有所降低。此外,针对Nvidia Fermi平台,手工调优了小因子的FFT,使其性能接近于CUFFT的140%。

关键词 FFT,OpenCL,Cell,CUDA,GPU,快速傅立叶变换
中图法分类号 TP311 **文献标识码** A

Implementation and Optimization of the FFT Using OpenCL on Heterogeneous Platforms

LI Yan[1,2,3] ZHANG Yun-quan[1,2] WANG Ke[1,2] ZHAO Mei-chao[1,2,3]
(Lab. of Parallel Software and Computational Science, ISCAS, Beijing 100190, China)[1]
(State Key Lab. of Computer Science, CAS, Beijing 100190, China)[2]
(Graduate University of Chinese Academy of Sciences, Beijing 100190, China)[3]

Abstract Fourier methods have revolutionized fields of science and engineering, from astronomy to medical imaging, from seismology to spectroscopy. A fast Fourier transform (FFT) is an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse. OpenCL (Open Computing Language) is a new framework for writing programs that execute across heterogeneous platforms consisting of CPUs, GPUs, and other processors, and it provides parallel computing using task-based and data-based parallelism. In this paper, we first implemented FFT with OpenCL, then tested and analyzed the performance of it on heterogeneous multi-core platforms like Cell, NVIDIA GPU. The performance we achieved is about 65% of Cell SDK, and 75% of CUDA CUFFT, and it needs to improve in the near further. Furthermore, we acquire unprecedented performance results that nearly 140% of CUFFT on Fermi GPU by exploiting hardware features when the size of FFT is small.

Keywords FFT, OpenCL, Cell, CUDA, GPU, Fast fourier transform

1 引言

1.1 FFT的原理和应用

快速傅里叶变换[1-4](Fast Fourier Transform, FFT),是离散傅里叶变换(Discrete Fourier Transform, DFT)的快速算法,具有广泛的应用,常被用于数字滤波、求解偏微分方程和信号分解。近年来具有FFT特点的谱方法已被成功地应用于解决计算流体力学(CFD)问题,而求解CFD问题广泛地被应用于气象预报、模拟地球物理学以及量子力学等方面。FFT在数据分析领域中的应用包括计算机断层成像、数据过滤和重构以及流体结构相互间作用的分析和可视化等。FFT算法设计的基本思想,就是充分利用DFT的周期性和对称性,减少重复的计算量;并把N点长序列分成几个短序列,递归地减少每个序列长度从而降低计算量和计算复杂度。

1.2 GPU和Cell

硬件方面,集成电路产业一直按照摩尔定律(Moore's Law)发展,计算机硬件环境日新月异,高性能计算机不断挑战更高难度数量级的性能。作为高性能计算机的基本组成部件,CPU、GPU、IBM Cell等各种高性能处理器的发展也十分迅猛。其中,GPU在通用计算方面的发展尤为引人注目。2006年11月,AMD开始研发"close to metal",后来演进为ATI Stream技术。同时,NVIDIA的计算统一设备架构(Compute Unified Device Architecture, CUDA)与G80显卡同时公开,并于2010年3月发布了CUDA 3.0。

到稿日期:2010-09-14 返修日期:2010-12-22 本文受国家863计划项目(2006AA01A125,2009AA01A129,2009AA01A134),国家重大项核高基项目(2009ZX01036-001-002)资助。

李 焱(1985—),男,博士生,主要研究领域为并行计算,E-mail:liyan08@iscas.ac.cn;张云泉(1973—),男,研究员,主要研究领域为高性能计算及并行数值软件、并行计算模型、并行数据库、海量数据并行处理。

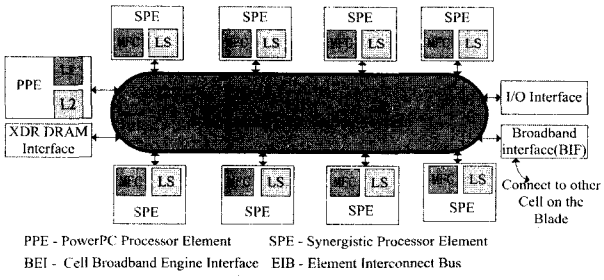

图 1 Cell 硬件结构图

Cell[5-7]是由 IBM、东芝(Toshiba)和 Sony 共同研发的异构多核处理器,具备多工、多效以及庞大的浮点运算功能。Cell 处理器(如图 1 所示)包括 1 个基于 PowerPC 架构的控制处理单元 Power Processing Element (PPE)以及 8 个 SIMD 的协处理器单元 Synergistic Processing Elements (SPE)。此外,采用高速环形数据总线 Element Interconnect Bus (EIB)连接 PPE、输入输出单元和 SPE,同时还提供了 DMA 指令和控制机制以用于在不同处理单元之间提供高效的通信。SPE 访问主存储器的方法则是利用 DMA 命令在主存储器和没有高速缓存(Cache)的私有本地内存(LS,Local Store)之间移动数据和指令,而不是共享主存储器。

1.3 OpenCL 简介

OpenCL[8](Open Computing Language,开放计算语言)是首个面向异构系统通用的并行编程标准,具有免费、跨平台的特点和很好的互操作性。OpenCL 为软件开发人员提供了统一的编程环境,便于为高性能计算服务器、桌面计算系统、手持设备编写高效轻便的代码,而且广泛适用于多核处理器(CPU)、图形处理器(GPU)、Cell 类型架构以及数字信号处理器(DSP)等其他并行处理器,因此在游戏、娱乐、科研、医疗等各种领域都有广阔的发展前景。

OpenCL 的架构如图 2 所示,一个主机(Host)可以连接到多个 OpenCL 设备(即 Compute Device)上,这些设备可以是 CPU、GPU、Cell 甚至 DSP 等。每个 OpenCL 计算设备被看作一组计算单元(Compute Unit)的集合,每个计算单元又划分为多个处理部件(Compute Element)。对于 GPU 而言,一个计算单元是指流多处理器(Streaming Multiprocessor),一个处理部件是指一个流处理器(Streaming Processor)。OpenCL 程序由主机程序(Host program)和内核程序(Kernel)组成。主机程序为 Kernel 定义了上下文并管理 Kernel 代码的执行,Kernel 则在支持 OpenCL 的计算设备(Compute Device)如 GPU、Cell 等上执行具体的计算任务。

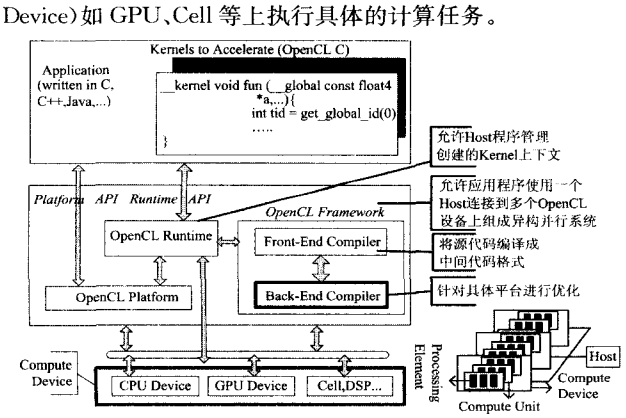

图 2 OpenCL 的框架图[8]

2 基于 OpenCL 的 FFT 算法设计与实现

对于复数序列 $x_0, x_1, \cdots, x_{n-1}$,离散傅里叶变换 DFT 的计算公式为:

$$y_k = \sum_{j=0}^{N-1} \omega_N^{jk} x_j \tag{1}$$

式中,$\omega_N = \exp(-\frac{2\pi i}{N})$,$i = \sqrt{-1}$;$K = 0, 1, \cdots, N-1$

计算每个 y_k 须作 N 次复数乘法及 $N-1$ 次复数加法,要完成整个 DFT 变换共需 N^2 次复数乘法及 $N(N-1)$次复数加法。经典的 FFT 算法是由 Cooley-Tukey[4]提出的,该算法考虑到 FFT 的周期性,以分治法为策略递归地将长度为 $N = r \times m$ 的 DFT 分解为长度分别为 N_1 和 N_2 的两个较短序列的 DFT。

$$j = j_1 r + j_2 \quad 0 \leqslant j_2 < r, 0 \leqslant j_1 < m$$
$$k = k_1 + k_2 m \quad 0 \leqslant k_1 < m, 0 \leqslant k_2 < r$$

于是:

$$y_{k_1+k_2 m} = \sum_{j_2=0}^{r-1} \sum_{j_1=0}^{m-1} \omega_N^{(k_1+k_2 m)(j_1 r + j_2)} x_{j_1 r + j_2}$$
$$= \sum_{j_2=0}^{r-1} \left(\left(\sum_{j_1=0}^{m-1} \omega_N^{k_1 j_1} x_{j_1 r + j_2} \right) \omega_N^{k_1 j_2} \right) \omega_r^{k_2 j_2} \tag{2}$$

用张量积表示为:

$$Y_l = (W_r \otimes I_m) T_m^N (I_r \otimes W_m) \Pi_{N,rX} \tag{3}$$

式中,T_m^N 是一个对角线"旋转因子"矩阵,$\Pi_{N,rX}$是一种模 r 置换矩阵,这是 FFT 的核心思想,以上式(3)就是著名的 Cooley-Tukey 混合基分解算法。

采用混合基分解 DFT 可将其复数计算量由 N^2 下降到 $N(r+m)$,当 N 为 r 的幂时,计算复杂度降低为 $2r\log_r N$。当 $N=8$ 时,DFT 的计算示意图如图 3 所示。

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & u & -i & v & -1 & -u & i & -v \\ 1 & -i & -1 & i & 1 & -i & -1 & i \\ 1 & v & i & u & -1 & -v & -i & -u \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -w & -i & -v & -1 & u & i & v \\ 1 & i & -1 & -i & 1 & i & -1 & -i \\ 1 & -v & i & -u & -1 & v & -i & u \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix}$$

图 3 8 个点 DFT 计算示意图[3]

其中,$u = \omega_8 = (1-i)/\sqrt{2}$,$v = \omega_8^3 = (-1-i)/\sqrt{2}$。对于以上 $N=8$ 时的计算序列,可以将其分解为 Radix-2 FFT(见图 4),所对应的蝶式计算图如图 5 所示。

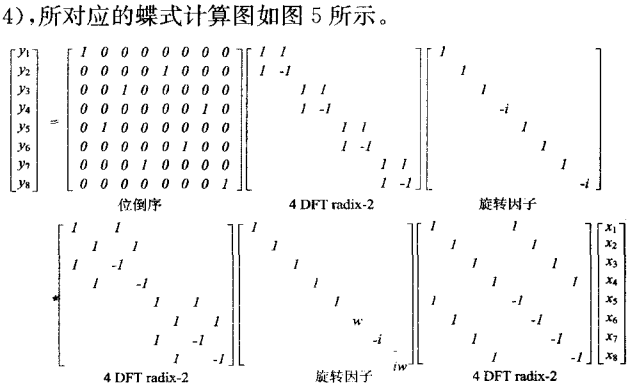

图 4 $N=8$ 时 FFT 计算分解示意图[3]

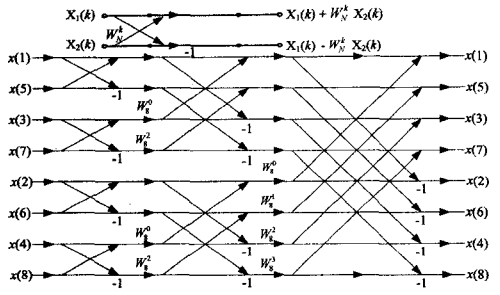

图 5　$N=8$ 时 FFT 蝶式计算图

目前，基于 CPU 版的 FFT 库的研究与发展比较成熟，各大硬件厂商 Intel、IBM 以及 AMD 也相应推出了针对各自平台优化的 FFT 库。FFTW(Fast Fourier Transform in the West)是由麻省理工学院开发的开源免费的自适应优化 FFT 软件包，同时支持共享存储多线程并行和分布式存储 MPI 并行，其运算性能远远领先于目前已有的其它 FFT 软件，甚至优于商用软件。FFTW 遴选了过去近 40 年的各种好的 FFT 算法，包括 Cooley-Tukey 算法以及各种变式、素因子、Rader 算法、分裂基算法等。在软件包安装时通过 Ocaml 语言针对安装平台的硬件特性产生一系列优化了的小因子 FFT(codelets)，例如 Radix-2、Radix-3、Radix-5、Radix-8 等。在运行时通过搜索和组合这些小因子 FFT 来选择较优或者最优的针对指定规模的 FFT 算法[13,14]。

但是，针对 GPU 优化的 OpenCL 版 FFT 库在各个平台(包括 Cell、GPU)基本上还处于开发阶段，OpenCL 的应用和使用还处于起步阶段。本文中基于 OpenCL 的 FFT 算法实现采用的是时间抽选 FFT 算法 (Decimation in frequency, DIF)，先将输入元素重新排列成位序颠倒的次序，然后进行蝶形算法输出自然序列。数据的实部和虚部采用混合存储模式即实部和虚部连续存储。本文所采用的 Cooley-Tukey 算法框架如下：

- 将输入数据按照列优先的顺序存储在 $r\times m$ 的数组中。
- 对矩阵中的每一行数据并行做 FFT radix-r 计算。
- 将矩阵乘以相关的旋转因子。
- 对矩阵进行转置操作。
- 对矩阵中的每一行数据并行地进行 FFT radix-m 计算。

算法框架所对应的图示如图 6 所示。

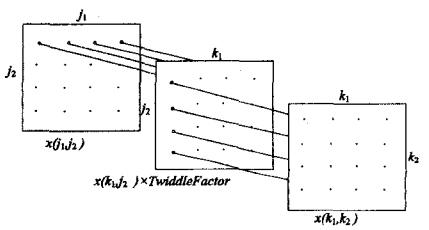

图 6　Cooley-Tukey 算法框架

3　实验结果及分析

3.1　测试环境

本节通过实验分别进行了 Cell SDK 以及 NVIDIA(CUFFT[10])的 FFT 库与实现的 OpenCl FFT 性能的比较，实验的平台如表 1 所列。AMD 中 SDK 2.1 的 Sample 中 FFT 只支持 1024 规模，而且其对应 ATI GPU(AMD Core Math Library for Graphic Processors, ACML-GPU)上没有可用的 FFT 函数库[9]，由于缺少可比性，因此实验平台暂时没有考虑 AMD GPU。

表 1　实验平台

平台	参数
Cell	Cell 3.2GHZ,2GB Memory,256K LS
	32/32 KB L1 (data/instruction) Cache,512KB L2 Cache
	Cell OpenCL 0.1.1+Cell SDK 3.1
	编译器:XLC 9.0
GPU	Tesla C1060
	NVIDIA OpenCL 1.0+ CUDA SDK 3.0
	编译器:NVCC 3.0
	Fermi (Tesla C2050)
	NVIDIA OpenCL 1.0+ CUDA SDK 3.1.1
	编译器:NVCC 3.1

测试结果中的时间包括 IO 传输时间、执行时间以及 OpenCL Kernel/CUFFT/Cell libfft 执行上下文的创建时间。由于 Cell 体系结构的异构特点[6,7]譬如支持 16M 的大页表，我们在 Host program 端使用了大页表来减少 TLB 缺失，并且数据内存分配采用 128bytes 地址对齐，MFC 虽然支持自然对齐的 DMA 传输，大小分别为 1,2,4,8 和 16 字节，以及 16 字节整数倍，但当 DMA 传输的源地址和目的地址都是 128 字节对齐且传输数据的大小是 128 字节的倍数时，传输的性能可以达到峰值(DMA 访问带宽是 128bytes/cycle)。考虑到 Radix-2 FFT 的特点，结合 Cell 上 SPE 大容量向量寄存器的特点，在 OpenCL kernel 中数据类型尽量使用 float2 和 float4 类型来提高寄存器的利用率。

3.2　测试结果与分析

OpenCL 在 Cell 平台上的测试结果如图 7 所示，若数据规模为 1024 至 8192，OpenCL 版的 FFT 的性能约为 SDK 的 65%，当数据规模较大如 size 为 65536 时，性能只有 SDK 的 35%左右。在 Kernel 端，虽然 SPE 上没有 Cache，SPE 从容量为 256kB 的 LS 中读取指令和数据，但是，其内部包含一个有 128 条目的 128 位寄存器堆(Register file)，而且具有两条(奇、偶)执行流水线，可在单周期内发射并完成两条指令，因此，在 Kernel 中应该尽量使用 float4 类型进行运算和操作。但是 FFT 的实现无论采用 DIT 还是先迭代后进行位序颠倒的频率抽选法(Decimation in frequency, DIF)算法，其内部实现的还是两个元素进行运算和操作，OpenCL 1.0 标准中没有类似于 SPE SIMD 指令中的 shuffle 指令来重组元素，因此寄存器的使用率最多只能达到 50%。由于一个 128 位寄存器刚好能够容纳 2 个 double 类型数据，即一个 double complex (对应于 double2 类型)，但是，目前 Cell 上该 OpenCL 版本对 OpenCL 标准中 double2 类型的操作的支持还不是很完善，譬如三角函数，故只能期待在下一版本中进行测试。另外，在 Kernel 端不能有效地使用双缓冲机制(Double buffer)来分摊通信开销，因此，随着计算数据量(size)的增大，势必会影响性能，从图 9 可以看出当数据规模增加时，虽然 Kernel 所占的时间比例明显增加，但性能与 SDK 的差距反而增大。

在 NVIDIA Tesla C1060 平台上的测试结果如图 8 所示，平均性能接近 CUFFT 的 75%。Fermi 是 Nvidia 专门为高性能计算优化的新一代处理器架构，基于该架构的 Tesla GPU 处理器拥有 30 亿颗晶体管、512 个计算内核，双精度性能相

(下转第 296 页)

[10] Wolfram S. Cellular automata as models of complexity[J]. Nature,1984,311:419-424

[11] Pinheiro E, Bianchini R, Dubnicki C. Exploiting redundancy to conserve energy in storage systems[J]. ACM SIGMETRICS Performance Evaluation Review, ACM,2006:15-26

[12] Colarelli D, Grunwald D. Massive arrays of idle disks for storage archives[C]// Proceedings of the 2002 ACM/IEEE Conference on Supercomputing. ACM,2002:1-11

[13] Pinheiro E, Bianchini R. Energy conservation techniques for diskarray-based servers[C]// Proceedings of the ACM/IEEE Conference on Supercomputing. ACM,2004:88-95

[14] Narayanan D, Donnelly A, Rowstron A. Write Off-loading: Practical Power Management forEnterprise Storage[C]// the 6th USENIX Conference on File and Storage Technologies (FAST'08). USENIX Association,2008:253-267

[15] Adami C. Introduction to artificial life[M]. New York: Springer-Verlag,1998:94-98

[16] Lee W, Scheuermann P, Vingralek R. File Assignment in Parallel I/O Systems with Minimal Variance of Service Time[J]. IEEE Trans. Computers,2000,49(2):127-140

(上接第 286 页)

对前一代产品 GT200 提升 8 倍，带有 ECC 校验功能，增加了 L1 和 L2 两级缓存，内存带宽提高两倍。Fermi (Tesla C2050)每个 SM 中的寄存器文件数量为 32K[11,12]，对于小规模的 FFT，每个线程拥有的寄存器数量非常充足。因此，在 Fermi 平台，我们对小规模的 FFT 完全展开。此外，Fermi 支持高精度的乘加指令 FMA(fused multiply-add)[11]，FFT 中复数相乘的操作非常频繁，有效使用 FMA 指令优化复数运算能够改善运算精度和运算效率。线程组成 warp 进行访存优化，减少线程之间的同步，同时通过转置操作优化全局内存的对齐访问，对于 block 内部线程之间尽量使用 local memory (对应于 CUDA 中的 Shared memory)来进行 block 内全局通信。在 Fermi 上通过采用如上技术优化 Radix-8，Radix-64 以及 Radix-512，使其性能大大提升，实验结果如图 10 所示。

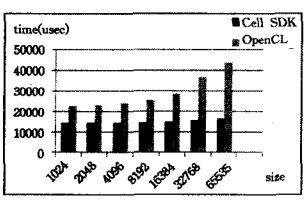

图 7 Cell 平台上的测试结果

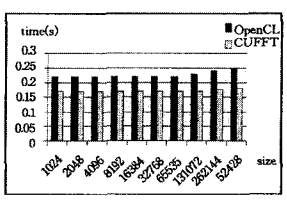

图 8 Tesla C1060 平台上的测试结果

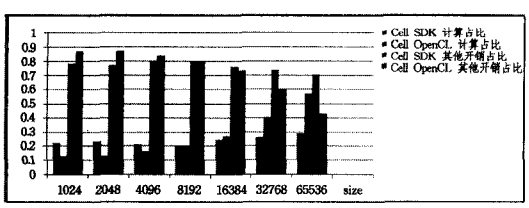

图 9 OpenCL FFT 在 Cell 平台上的计算和开销占比

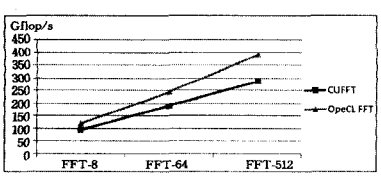

图 10 Fermi 平台上的小规模 FFT 优化结果

结束语 本文测试和分析了面向异构平台基于 OpenCL 的一维 FFT，实验结果表明，在 Cell 平台上当数据规模适中时能达到 SDK 中提供的 FFT 性能的 65%，当数据规模继续增大时，性能有所降低了，仅为 SDK 的 35% 左右。在 NVIDIA 平台上能够到达 CUFFT 性能的 75%，并对小规模的 FFT 在 Fermi 进行了手工调优，性能接近 CUFFT 的 140%。另外，在 Cell 上由于 SPE 具有大容量向量寄存器的特点，下一步工作会利用循环展开和寄存器分块来有效利用寄存器；在 NVIDIA 平台上借鉴 CUDA 程序的优化技术来改善 OpenCL Kernel 中的性能，例如流技术。此外，将针对 Fermi 平台实现和优化更多的小因子 FFT(codelets)，并对一定规模大小的 FFT 采用多种搜索策略和匹配 codelets 算法来得到较优的性能。

总之，OpenCL 为软件开发人员提供了统一的面向异构系统的并行编程环境，尤其在 Cell 上，屏蔽了基于 SDK 编程的很多复杂的硬件细节，提高了并行编程的生产率，因此，OpenCL 在并行计算领域具有良好的应用前景。

结束语 本文在异构平台 Cell 和 Nvidia GPU 上使用 OpenCL 实现和优化了一维 FFT，并测试和分析了该版本 FFT 的性能。此外，针对 Fermi 的存储层次和硬件特点，通过手动调优小因子 FFT，使其性能接近 CUFFT 的 140%。

参 考 文 献

[1] Bracewell R N. The Fourier Transform and Its Applications(3rd ed)[M]. McGraw-Hill,1999

[2] Govindaraju N K, Lloyd B, Dotsenko Y, et al. High Performance Discrete Fourier Transforms on Graphics Processors[C]// SC08. November 2008

[3] Volkov V, Kazian B. Fitting FFT onto the G80 architecture[R]. http://www.cs.berkeley.edu/~kubitron/courses/cs258-S08/projects/reports/project6_report.pdf, May 2008

[4] Cooley J W, Tukey J W. An algorithm for the machine calculation of complex Fourier series [J]. Mathematics of Computation,1965,19(90):297-301

[5] IBM Corporation. Cell Broadband Engine Programming Handbook[S]. Version 1.12, Apr. 2009

[6] IBM Corporation. Software Development Kit 2.1 Programmer's Guide[S]. Version 2.1, March 2007

[7] IBM Corporation. Cell Broadband Engine Programming Tutorial [S]. Version 2.0, Dec. 2006

[8] Khronos OpenCL Working Group. The OpenCL Specification [S]. Version 1.0, Oct. 2009

[9] AMD Corporation. AMD Core Math Library for Graphic Processors Release Notes for Version 1.0[S]. March 2009

[10] CUDA Programming Guide Version 3.0: NVIDIA Corporation [S]. Feb. 2010

[11] NVIDIA's Next Generation CUDA Compute Architecture: Fermi. NVIDIA Corporation[S]. 2009

[12] Glaskowsky P N. NVIDIA's Fermi: The First Complete GPU Computing Architecture[Z]. September 2009

[13] Frigo M, Johnson S G. The Design and Implementation of FFTW3 [J]. Proceedings of the IEEE,2005,93 (2):216-231

[14] Frigo M. A Fast Fourier Transform Compiler[C]// Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation. Feb. 1999