龙芯3A上三个自适应FFT包的对比与分析

赵美超[1,2,3] 张云泉[1,2] 刘益群[1,2,3] 李 焱[1,2,3] 颜深根[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 FFT算法在计算机科学中具有广泛的应用,自适应FFT软件包以其良好的可移植性而备受研究人员和用户的青睐,龙芯3A是中科院计算所自主研发的四核CPU,采用RISC架构,兼容MIPS指令。主要对FFTW,UHFFT,SPIRAL这3类FFT自适应软件包进行研究。首先从搜索框架和代码产生器两方面总结了FFTW和UHFFT的异同,接着阐述了SPIRAL自动产生优化代码的三层架构实现原理,之后在国产CPU龙芯3A上对这3个软件包进行了性能测试,并结合龙芯的体系结构特点对结果作了分析对比。在最后总结了目前自适应FFT软件包的一般方法,为下一步开发自适应FFT软件包提供了思路。

关键词 FFTW,UHFFT,SPIRAL,Loongson 3A,FFT
中图法分类号 TP319 **文献标识码** A

Comparison and Analysis of Three Types of FFT Adaptive Libraries on Loongson 3A

ZHAO Mei-chao[1,2,3] ZHANG Yun-quan[1,2] LIU Yi-qun[1,2,3] LI Yan[1,2,3] YAN Shen-gen[1,2,3]
(Laboratory of Parallel Software and Computational Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)[1]
(State Key Laboratory of Computing Science, Chinese Academy of Sciences, Beijing 100190, China)[2]
(Graduate University, Chinese Academy of Sciences, Beijing 100190, China)[3]

Abstract FFT algorithm has a wide range of applications in computer science. Adaptive FFT software package with its excellent portability has been interested by many researchers and users. Loongson 3A is developed by institute of computing technology, Chinese academy of sciences. It is a quad-core CPU and compatible with MIPS instructions using RISC architecture. The article focused on three types of FFT adaptive libraries which are FFTW, UHFFT and SPIRAL. Firstly, we compared the difference between FFTW and UHFFT from two aspects of search framework and code generator. Then we elaborated SPIRAL's three layers schema which is used to produce optimized code automatically. Furthermore, we evaluated these libraries on the Loonson 3A platform and analyzed the results. Finally, we concluded the general method of current FFT adaptive software packages and provided a guideline for further development of adaptive FFT software package.

Keywords FFTW, UHFFT, SPIRAL, Loongson 3A, FFT

1 引言

随着计算设备的日益更新,如何充分有效地利用计算资源,使软件性能尽可能地逼近计算设备的理论峰值是人们关心的问题。针对特定硬件平台进行手工优化程序,或者单纯依赖编译器的优化技术,都存在人工介入。因此,软件性能的提升很难与硬件的更新保持同步,而且具有移植性和开发效率低的特点。自适应性能优化技术能够有效地解决这些问题,目前该技术已经成熟地运用于CPU高性能数值计算软件中,如PHiPAC(Portable High Performance Ansi C),ATLAS (Automatically Tuned Linear Algebra Software),OSKI(Optimized Sparse Kernel Interface)等,不仅缩短了软件的开发周期、降低了成本,而且提高了软件可移植性。使用自适应性能优化技术中,控制和优化耗时的参数搜索过程对数值软件开发和研究也具有重大的意义。

快速傅立叶变换(FFT)[21,22]是为计算离散傅立叶变换及其逆变换而设计的一系列算法。经典的FFT算法由Cooley和Tukey[1,2]于1965年所提出,它采用递归(recursive)和分治(divide-and-conquer)对离散Fourier变换求和,尽可能避免了乘法和加法的重复计算,把离散Fourier变换的浮

到稿日期:2012-02-04 返修日期:2012-06-11 本文受国家自然科学基金(61133005),国家高技术研究发展项目(863)(2009AA01A129,2009AA01A134),国家重大专项核高基项目(2009ZX01036-001-002)资助。
赵美超(1986—),女,硕士生,主要研究方向为并行计算,E-mail:zhaoangel@163.com;张云泉(1973—),男,研究员,博士生导师,CCF会员,主要研究方向为高性能计算及并行数值软件、并行计算模型;刘益群(1987—),女,博士生,主要研究方向为并行计算;李 焱(1985—),男,博士生,主要研究方向为并行计算;颜深根(1986—),男,博士生,主要研究方向为并行计算。

点运算量由 $8N^2$ 减少到 $5N\log_2 N$(这里 $N=2^p$ 为取样点的数目),从而大大提高了离散 Fourier 变换的计算效率。不同于 Blas(Basic Linear Algebra Subprograms)函数,FFT 算法具有带宽密集型的特点,计算所占比重相对较少,为了减缓内存带宽同 CPU 计算速率的差距,存储系统变得日益复杂,这样更加大了自适应优化的难度。伴随着多核(CMP)、多处理器(SMP)和混合共享内存多处理器结构的发展,在有限的时间内产生高效率的代码将更具挑战性。早期 FFT 算法的实现都是针对特定的机器手动调优。随着自适应优化技术的成熟,自适应 FFT 软件包逐渐发展起来,主要有 FFTW(Fast Fourier Transform in the West)[3-5],UHFFT[13,17] 以及 SPIRAL (Software/Hardware Generation for DSP Algorithms)[9-11]。

龙芯 3A 是中科院计算所自主研发的多核处理器,FFT 自适应软件包是否能够有效地支持国产 CPU 并且获得好的性能对于提高龙芯的市场竞争力具有重要的意义。本文第 2 节主要从搜索框架和代码产生器两方面分析了 FFTW 和 UHFFT 的异同;第 3 节阐述了 SPIRAL 自动产生优化代码的 3 层架构实现原理;第 4 节在龙芯 3A 上对这 3 个软件包进行性能测试,并结合龙芯的体系结构特点对结果进行分析对比;最后总结了 FFT 自适应软件包的一般方法。

2 FFTW 与 UHFFT 的分析

2.1 FFTW 与 UHFFT 简介

FFTW 用于计算一维及任意维实、复数据的离散傅立叶变换及其相关变式。FFTW 始创于 1997 年 3 月(V1.0),作者为 MIT 计算机科学实验室的 Matteo Frigo 及 MIT 物理系的 Steven G. Johnson,其主要由 ANSI C 写成,可移植性强。它大部分代码由一个名为 genfft 的 Objective Caml 程序自动生成。用户界面由低到高分为 3 个层次:Basic,Advanced 和 Guru。它支持 SSE,SSE2,3DNOW! 及 Altivec 等 SIMD 指令,实现了共享存储体统的多线程并行版本。此外,它还提供分布存储系统的 MPI 并行程序。

UHFFT 软件包在同一时期开发出来,它通过对微处理器结构、存储层次和并行化 3 方面进行优化,在不同的体系结构上保持其性能的可移植性,目前最新版本为 2.0.1 版,其支持一维复数 out-of-place 变换。

FFTW 与 UHFFT 的自适应框架类似,都是通过使用运行时性能分析和搜索技术使其适应不同的计算机硬件结构,如图 1 所示。自适应 FFT 库组合不同的代码块,每个代码块执行一部分变换并且高度优化,通常由代码产生系统产生,因此整个框架分为两大部分:底层的代码产生器(Code Generator)和上层的运行时框架(Execution Scheduler)。下面主要从这两部分分别介绍 FFTW 和 UHFFT 的实现原理。

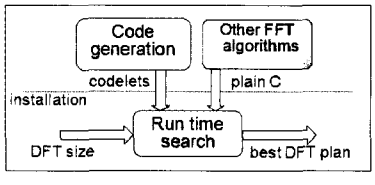

图 1 FFTW 结构图

2.2 代码产生器

代码产生器主要产生两类 DFT 片段:普通 DFT codelets 和 twiddle codelets。除此之外,FFTW 还提供支持 short-vector 指令的向量化 codelets;UHFFT 还包含了旋转所用的 PFA codelets[18],其在分解过程中作为素因子算法实现的基础片段。当变换规模大于 5 时,编码和调节 DFT 是一个非常繁琐的过程,这两个软件包都使用自动代码产生器有效地处理这个问题。两个包的代码产生器均分为 4 个步骤来产生 codelets,但是又有一些区别。

第一阶段为创建阶段。FFTW 针对每一种实例,采用 Caml Light 语言,通过 Objective Caml code 的形式定义了一种特定的 code data type,它可以产生某种类型的抽象语义树(AST),也就是通常说的 DAG(Directed Acyclic Graph)。UHFFT 的代码产生器选择最佳的因式分解算法,目的是产生最少的浮点操作数。通过产生一系列抽象的表达式进行蝶形计算。codelet 的大小被严格地限制以使数据和代码适合于缓存大小。

第二阶段为简化阶段。FFTW 对 AST 集进行一系列的优化,其中包括基本的编译优化技术(比如代数变换、子公共式提取等)和适用于 FFT 的专用优化技术(比如负号前置(propagating a minus)、简化转置的 DAG 等)。UHFFT 在这个阶段与 FFTW 类似。

第三阶段为调度阶段(scheduling phase)。FFTW 主要对简化后的 DAG 进行拓扑排序,目的是在编译阶段产生一个好的寄存器分配策略,采用的算法为 cache-oblivious。UHFFT 也做类似的工作,不同的是它主要采用的算法为 cache-obvious,主要目的是最优化内存复用距离(memory reuse distance)。

第四阶段为解析阶段(unparsed phase)。FFTW 将抽象语义树的形式解析为普通 C 语言代码。UHFFT 同样将抽象代码解析为期望的 C 代码。

2.3 运行时框架

FFTW 遴选了过去近 40 年的各种好的算法,包括 Cooley-Tukey 算法及其各种变式[1]、素因子算法[18]、Rader 算法、分裂基算法[19]等。每一种算法都实现成为一类特定的 solver。FFTW 表示问题规模的形式称为 I/O tensors,辅助采用 I/O dimension[4] 的形式,这不同于经常提及的张量积(tensor-product)[16]形式。每一种算法都是一种特定的 I/O dimension。

给定一个问题,FFTW 往往是将多维转化为一维、向量形式转化为循环形式、分析数据规模特点选定特定的算法这几步交叉进行来产生 plan 搜索空间。首先,planner 会初始化一系列的 solvers,得到一个给定的规模之后,顺序调用每一个 solver[4],接着每一个 solver 返回一个具体 plan 指针或者空指针。最后,planner 通过实际运行,选择一个最快的返回给用户。很多情况下,每一个 plan 都需要子问题的求解,此时产生这个 plan 的 solver 再递归地触发 planner 的运行。搜索空间往往是很大的,FFTW 的 planner 采用动态规划算法去加快搜索速度。整个搜索过程递归进行,具体执行采用迭代形式。

UHFFT 在运行时产生一个执行计划,其实际是一个模块链,每个模块描述具体分解后的规模大小,并且链表的顺序决定了整个变换的执行顺序。一个 FFT 任务被 FFT Schedule Specification Language(FSSL)准确描述,FSSL[8] 是一套

上下文无关文法产生式。UHFFT 将主要的分解算法归于 4 大类：混合基算法、分裂基算法、素因子算法和 rader 算法，每一类算法也作为一个模块(module)，以链表的形式连在一起，如图 2 所示。FSSL 基于输入向量和它的因子的属性，将不同的算法综合在一起，通过搜索来产生高性能的执行计划。当前主要包含 3 种不同的搜索策略：high、medium、low。

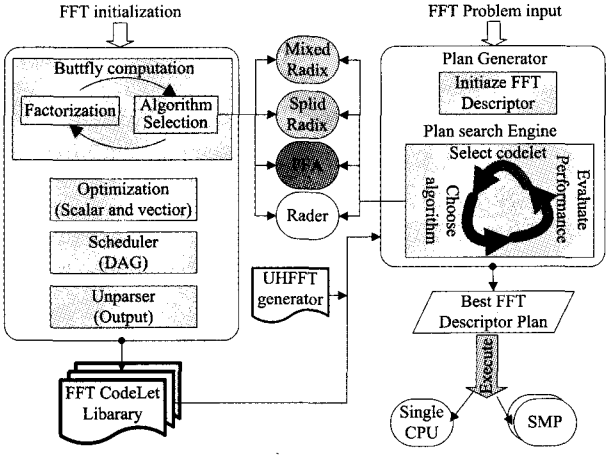

图 2　UHFFT 结构图

UHFFT 采用左递归或者右递归分解策略，如图 3 所示，首先确定第一步或者最后一步，接着对剩下的规模继续进行分解。相比 FFTW 的搜索空间，UHFFT 的搜索空间相对限制了很多。它的 high 策略采用的方法称为 context sensitive empirical search，其属于动态规划算法的范畴，事先维护一张 codelets 的执行时间表，将已经执行过的规模存下来，从而减少搜索时间。medium 搜索策略主要基于分解策略中的不同输入、输出步长的 codelets 执行时间和调用次数预估计整体性能，事实证明这个方法不仅可以减少搜索的时间，而且最终得到的性能也比较好。low 策略在 medium 策略的基础上，将所有 codelets 的执行时间按步长由低到高的顺序预先存在一个缓存中，更简便的做法是只考虑步长为 2 的幂次，并且假设输入步长和输出步长一致，因此搜索时间更小。总而言之，搜索会增加整个执行时间开销，但是当同一个问题规模重复使用时，它将会取得好的性能。

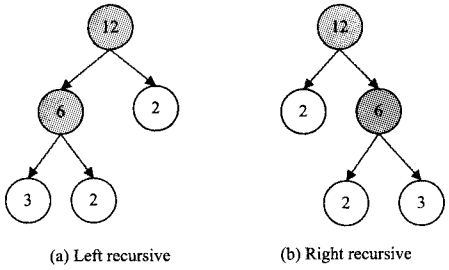

图 3　UHFFT 分解策略

3　SPIRAL 软件包分析

SPIRAL 是一个自动生成线性数字信号处理程序(DSP)的系统，包括 DFT、三角函数变换、滤波器的变换和离散小波变换等。图 4 为 SPIRAL 的结构图。与前两个软件包不同的是，它主要分 3 个层：算法层、实现层、评估层。在 SPIRAL 软件包中，需要注意 Set of implementations 和 Minimization of Costs 这两个概念。前者代表针对特定规模、特定软件平台、特定硬件平台所产生的代码实现集；后者代表得到最优代码所需要的最小成本。

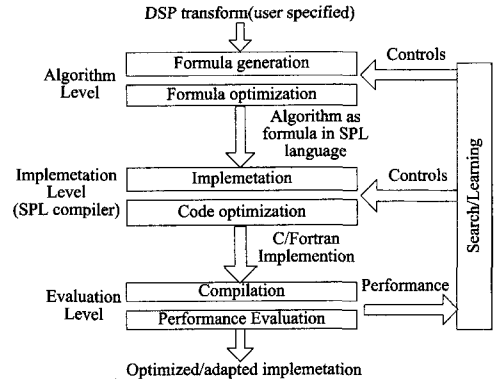

图 4　SPIRAL 结构图

3.1　算法层(Algorithm Level)

SPIRAL 软件包中，一个具体问题规模的算法主要是通过递归地采用分解规则(breakdown rules)和处理规则(manipulation rules)产生的，其中采用 SPL(signal processing language)[11]语言来表示问题规模和规则，称为 formula，如图 5 左边一栏所示，这种 241 语言类似于张量积形式。分解规则主要是递归地将大规模变成小规模直至不能再划分为止，整个过程由 formula generation block 控制。接着，formula optimization block 对每个 formula 施用 manipulation rules，以更好地适应硬件平台。这部分的优化不同于编译器针对代码的优化，主要是循环合并(loop merging)[9,15]和索引简化(index simplification)[9]等，它主要是在 SPL 的表示形式上进行优化，这样不仅可以更容易结合硬件的特点，也可避免冗余的底层细节。

SPL construct	code
$y=(A_nB_n)x$	t[0:1:n−1]=B(x[0:1:n−1]); y[0:1:n−1]=A(t[0:1:n−1]);
$y=(I_m\otimes A_n)x$	for (i=0;i<m;i++) y[i∗n:1:i∗n+n−1]=A(x[i∗n:1:i∗n+n−1]);
$y=(A_m\otimes I_n)x$	for (i=0;i<m;i++) y[i∗n:1:i∗m−1]=A(x[i∗n:1:i+m−1]);
$y=(\bigoplus_{i=0}^{m-1}A_n^i)x$	for (i=0;i<m∗n;i++) y[i∗n:1:i∗n+n−1]=A(i,x[i∗n:1:i∗n+n−1]);
$y=D_{m,n}x$	for (i=0;i<m;i++) y[i]=D$_{mn}$[i]∗x[i];

$y=L_m^{mn}x$	for (i=0;i<m;i++)
	for (j=0;j<n;j++)
	y[i+m∗j]=x[n∗i+j];

图 5　SPL 与代码的对应图

3.2　实现层(Implementation Level)

算法层的输出同样也是实现层的输入，是用 SPL 语言表示的一些 formulas，需要最终转换成 C 代码或者 Fortran 代码，因此，这一层又称为 SPL 编译器。首先将 SPL 表示的 formula 转变成具体代码，主要依照图 5 所示的对应规则。其次进行代码优化，这一层主要是对具体的代码进行优化，用到的技术主要是一些通用的优化技术，比如 loop tiling、loop exchange、loop interleaving 等。算法层和实现层主要给出了 Set of implementations，为之后的评估层提供了一个搜索空间。

3.3　评估层(Evaluation Level)

这一层主要解决 Minimization of Costs 的问题，需要自

动地搜索 Set of implementations，并对每一种实现做评估，类似于搜索过程，从而得到成本最小的实现，在具体评估中，可以使用动态规划算法或者启发式算法。这一层分解为两块：the compilation blocks 和 performance evaluation bocks。前者采用标准的编译器产生可执行代码；后者评估可执行代码的成本，然后反馈到前两层的实现和优化中。本节多次提到的成本概念主要是指给定平台上的实际执行时间。

4 3种软件包在龙芯3A上的性能评测

我们在龙芯3A上主要进行了3方面的测试：一是底层codelets的性能测试，这里主要指FFTW和UHFFT的codelets；二是从整体上对这3个软件包进行性能分析；三是测试自适应软件包在龙芯3A上多核的加速比。这里需要说明的是，目前UHFFT 2.0.1版没有多核部分的实现，因此只能对FFTW进行多核测试。由于SPIRAL软件包提供的代码有限，我们只能对它进行2的幂次并且规模不超过8192的单核性能测试，实验平台具体参数可参照表1。

表1 实验平台参数

CPU	1路龙芯3A(4核)，主频800MHz
内存	2条DDR-3 2GB内存，共4GB
操作系统	Debian 6.0，内核Linux-2.6.36
编译器	GCC-4.4.0 编译参数为 -O3

4.1 codelets性能对比

整体上，twiddle codelets的性能要低于非twiddle codelets性能，相差的部分主要是乘以旋转因子的开销。如图6、图7所示，随着步长增大，codelets的性能逐渐下降，并且在龙芯3A上这种下降趋势比较明显。由于龙芯3A的cache采用了随机替换策略，因此一级cache命中率的大小对codelets影响比较大。在测试过程中，每个规模都要循环很多次，随着步长逐步增加，一级cache的缺失和冲突的概率就会增加，从而影响性能。FFTW与UHFFT两者的codelets整体一致，也有些差异。首先素数规模，FFTW要好于UHFFT，个别规模例如13，两者差距接近一倍；其次规模为2的幂次，UHFFT性能略显优势，再者FFTW的twiddle codelets性能大部分要好于UHFFT，但随着规模增大，这种优势越来越弱。

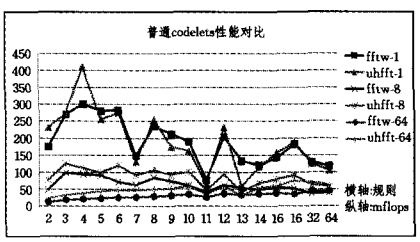

图6 龙芯3A-普通codelets性能对比

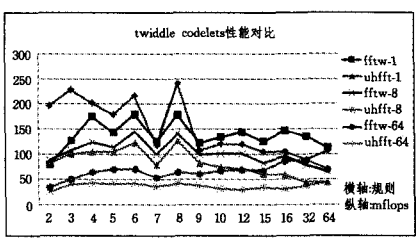

图7 龙芯3A-twiddle codelets性能对比

4.2 单核性能对比

本文主要对2的幂次、非2的幂次、素数3方面进行一维单核的性能测试，选用FFTW的estimate、measure 2种策略；UHFFT的high、medium、low 3种策略。

首先从性能上说，规模为2的幂次情况下，UHFFT3种策略性能差异不大，整体能达到FFTW measure策略的75%，明显优于FFTW estimate策略。SPIRAL性能处于FFTW measure策略和UHFFT之间。规模为非2的幂次情况下，UHFFT能达到FFTW的90%。

其次从搜索时间上说，UHFFT的low策略和medium策略基本上与FFTW estimate策略时间相当，FFTW measure策略搜索时间相对较高，UHFFT high策略搜索时间最高。

最后在素数规模下，UHFFT处于FFTW estimate策略和measure策略之间。

总结原因为：UHFFT运行时框架中没有考虑分裂基算法的实现，只是在codelets生成过程中使用了该算法，这对2的幂次有比较大的影响；UHFFT整个执行框架采用递归执行，FFTW采用迭代执行，递归函数调用开销不可避免。

龙芯3A-2的幂次3个软件包性能对比如图8所示；龙芯3A-素数性能对比如图9所示；龙芯3A-非2的幂次性能对比如图10所示；龙芯3A-2的幂次搜索时间对比如图11所示。

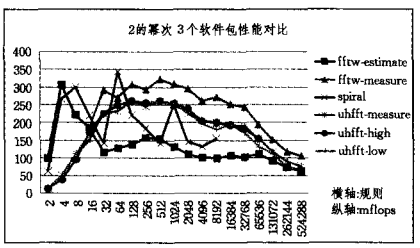

图8 龙芯3A-2的幂次3个软件包性能对比

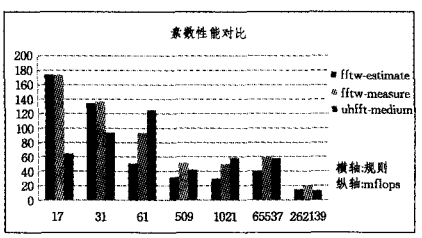

图9 龙芯3A-素数性能对比

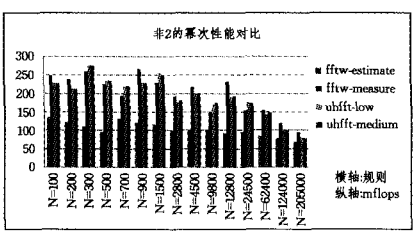

图10 龙芯3A-非2的幂次性能对比

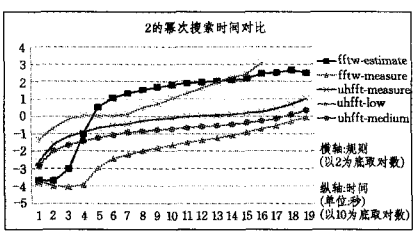

图11 龙芯3A-2的幂次搜索时间对比

4.3 FFTW多核性能

对FFTW进行了两种策略的多核性能测试，如图12所

示。FFTW estimate 策略下，2 核的加速比为 1.55，4 核的加速比为 2.55。FFTW measure 策略下，2 核的加速比为 1.50，4 核的加速比为 2.75。相比其它主流处理器，龙芯 3A 处理器浮点运算能力相对较强，但是访存能力较慢，然而 FFT 变换过程中多数都是对内存的不连续访问，因此很难得到很高的并行加速比。

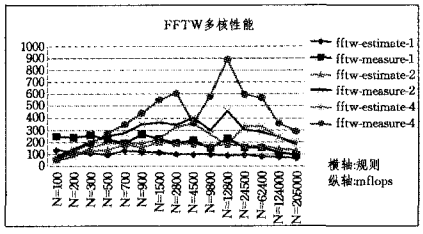

图 12　龙芯 3A-FFTW 多核性能

结束语　本文主要分析了 FFTW、UHFFT 和 SPIRAL 这 3 个软件包的自适应框架，并在龙芯 3A 上进行了性能分析和对比。总结起来，3 个软件包原理相通，主要分为高级别的优化和低级别的优化，也就是算法层和实现层的优化。不同点主要体现在问题规模的表示方式、搜索策略和片段代码这几方面。龙芯 3A 处理器作为国产多核 CPU，在一定程度上能够有效地支持自适应 FFT 软件包的运行，并且也取得了良好的多核加速比。

本文下一步要做的工作为进一步分析 3 个软件包的多核实现特点，并在当前主流多核平台上和龙芯 3A 平台上进行详细的测试分析，以更深入地理解自适应技术的特点，并特别针对龙芯 3A 平台进行算法的调优，将这些技术用在其它软件包中。

参 考 文 献

[1] Cooley J W. The re-discovery of the fast Fourier transform algorithm[J]. Mikrochimica Acta, 1987, 3: 33-45

[2] Cooley J W, Tukey J W. An algorithm for the machine calculation of complex Fourier series[J]. Math. Comp., 1965, 19: 297-301

[3] Frigo M. A fast Fourier transform compiler[C]// Proceedings of the ACM SIGPLAN 1999conference on Programming Language Design and Implementation, PLDI '99. New York, NY, USA: ACMPress, 1999: 169-180

[4] Frigo M, Johnson S G. The design andimplementation of FFTW3[C]// Proceedings of the IEEE Special Issue on Program Generation, Optimization, and Platform Adaptation. 2005: 216-231

[5] Frigo M, Johnson S G. The fastest fourier transform in the west [M]. technical report technical reportMIT-LCS-TR-728, Sep. 1997

[6] Frigo M, Johnson S G. Fftw: An adaptive software architecture for the fft[C]// Proc. ICASSP 3. 1998: 1381-1384

[7] Loan C V. Computational frameworks for the fastFourier transform. Society for Industrial and AppliedMathematics[M]. Philadelphia, PA, USA, 1992

[8] Mirkovic D, Johnsson S L. AutomaticPerformance Tuning in the UHFFT Library[C]// Proceedings of the International Conference onComputational Sciences-Part I, ICCS'01. London, UK: Springer-Verlag, 2001: 71-80

[9] Franchetti F, Voronenko Y, Püschel M. FFT program generationfor shared memory: SMP and multicore[C]// Proc. Supercomputing (SC). 2006

[10] Spiral Web site[OL]. http://www.spiral.net

[11] Xiong J, Johnson J, Johnson R, et al. SPL: A language andcompiler for DSP algorithms[C]// Proc. Programming Language Designand Implementation (PLDI). 2001: 298-308

[12] FFTW Web site[OL]. www.fftw.org

[13] UHFFT Web site[OL]. www2.cs.uh.edu/~ayaz/uhfft

[14] Franchetti F, Voronenko Y, Püschel M. A rewriting system forthe vectorization of signal transforms[C]// Proc. High PerformanceComputing for Computational Science (VECPAR). 2006

[15] Franchetti F, Voronenko Y, Püschel M. Loop merging forsignal transforms[C]// Proc. Programming Language Design andImplementation (PLDI). 2005: 315-326

[16] Johnson J R, Johnson R W, Rodriguez D, et al. Amethodology for designing, modifying, and implementing Fourier transformalgorithms on various architectures[J]. IEEE Trans. Circuits, Systems, and Signal Processing, 1990, 9(4): 449-500

[17] Ali A, Johnsson L, Subhlok J. Scheduling FFT computation onSMP and multicore systems[C]// Proc. Int'l Conf. Supercomputing (ICS). 2007

[18] Temperton C. A Note on Prime Factor FFT Algorithms[J]. Journal of Computational Physics, 1983, 52: 198-204

[19] Duhamel P, Hollmann H. Split Radix FFT Algorithms[J]. Electronic Letters, 1984, 20: 14-16

[20] Singleton R C. An algorithm for computing the mixed radix fast Fourier transform[M]. IEEE Trans. on Audioand Electroacoustics. 1969: 93-103

[21] Rockmore D N. The FFT: An algorithm the whole family can use[J]. Computing in Science & Engineering, 2000, 2(1): 60-64

[22] Brigham E O. The Fast Fourier Transform and Its Applications [M]. Prentice Hall Signal Processing Series, Englewood Cliffs. NJ, 1988

(上接第 254 页)

[5] Boudabous A, Ghozzi F, Kharrat M W, et al. Implementation of hyperbolic functions using CORDIC algorithm[C]// The 16th International Conference on Microelectronics, 2004(ICM 2004). Tunisia, 2004: 738-741

[6] 张华军，赵金，丛喆. CORDIC 在基于 FPGA 的神经网络设计中的应用[J]. 华中科技大学学报：自然科学版，2009，37(6)：36-39

[7] 张智明，张仁杰. 神经网络激活函数及其导数的 FPGA 实现[J]. 现代电子技术，2008(18)：139-142

[8] 夏欣，贾永刚，王素珍. RBF 神经网络中指数函数 e^x 的 FPGA 实现[J]. 微计算机信息，2005，21(7-2)：145-146

[9] Michael J S, James E S. The Symmetric Table Addition Method for Accurate Function Approximation[J]. Journal of VLSI Signal Processing, 1999(11): 1-12