

N 体问题的 FPGA 求解和设计方法

傅丽丽 曾国荪

(同济大学计算机科学与技术系 上海 201804)

(国家高性能计算机工程技术中心同济分中心 上海 201804)

摘 要 N 体问题是一个经典动力学问题,在多个领域得到广泛的应用。但随着规模的增大,对求解计算性能的要求成为其研究的主要障碍。当前,FPGA 可重构技术由于具有硬件可编程结构和高度并行处理能力而成为高性能计算关注的热点。现以 FPGA 加速求解 N 体问题为例,阐述一种新型的求解计算密集型任务的方法。

关键词 N 体问题,FPGA,计算密集型,高性能计算

中图分类号 TP338 **文献标识码** A

Solution and Design of N-body Algorithm on FPGA

FU Li-li ZENG Guo-sun

(Department of Computer Science and Technology, Tongji University, Shanghai 201804, China)

(Tongji Branch, National Engineering & Technology Center of High Performance Computer, Shanghai 201804, China)

Abstract N-body algorithm is a classic problem in dynamics. It has been widely used in many fields. But in recent years, as its scale is much larger than before, the computing requirement for high performance has turn out to be a kind of bigger obstacle on its research. Right now the reconfigurable technology of FPGA therewith the hardware reconfigurable architecture and high parallel processing ability has become the focus of high performance computing industry. In this paper, with the example of using FPGA to accelerate solving N-body algorithm, we introduced a new method to solve computation-intensive task.

Keywords N-body algorithm, FPGA, Computation-intensive, High performance computing

1 引言

FPGA (现场可编程门阵列, Field Programmable Gate Array) 是一种大规模可编程门阵列的器件,不仅具有专用集成电路(ASIC)快速的特点,更具有很好的系统实现的灵活性。近年来,FPGA 技术正处于高速发展时期,新型芯片的规模越来越大,成本也越来越低;低端的 FPGA 已逐步取代了传统的数字元件,高端的 FPGA 则不断在争夺 ASIC 的市场份额。目前,FPGA 已经在通信、数据处理、网络、仪器、工业控制、军事和航空航天等众多领域得到了广泛应用。随着功耗和成本的进一步降低,FPGA 还将进入更多的应用领域。

随着 FPGA 器件可重构技术的快速发展,FPGA 已经成为高性能计算工业界关注的热点,而且将持续向更强的计算能力和低功耗、低成本、短开发周期等方面发展。FPGA 具有硬件可编程结构和高度并行处理能力,作为硬件加速器,它能够加速处理计算密集型应用任务。同时由于加速器的能效较高,降低了系统的整体能耗要求。对于计算密集型任务,其性能的提升主要依靠 CPU 及算法优化。众所周知,CPU 主频遵循摩尔定律发展。但是自 2003 年以来,CPU 的频率止步

于 4GHz,而且其功耗和运行速度也不尽人意。软件优化确实能在一定程度上提升复杂算法的性能,但其基于 CPU 串行的特点及对设计周期的依赖,使其在现阶段并无突出成就。利用 FPGA 并行结构来提高运行速度、减少功耗的高性能计算已经成为一种趋势。本文将以此为目的,希望提供一种新的解决计算密集型任务的方法。

本文以 N 体问题的求解为例来展开分析和讨论,因为 N 体问题是典型的计算密集型和通讯密集型应用,要求计算空间中的多个粒子之间的相互作用及其运动轨迹,是最具普适性的动力学问题之一。当粒子为宏观的天体时,天体多体模拟计算是当前研究星系以及宇宙结构形成的主要途径。当粒子为微观的分子、原子、电子时,多体问题即表现为广为人知的分子动力学问题。由于分子动力学可以预测纳米尺度上的材料动力学特性,它在物理、化学、生物、医药、新材料设计等领域有着广泛的应用。N 体问题对数据处理要求很高,最简单的 Particle-Particle (PP) 算法的算法复杂度为 $O(N^2)$,而一些优化算法如 Barnes-hut 算法其复杂度则降低到 $O(N \log N)$, Particle-Mesh (PM) 算法和快速多极子 FMM 算法的算法复杂度则为 $O(N)^{[1]}$ 。

到稿日期:2009-12-09 返修日期:2010-03-09 本文受 863 项目(2007AA01Z425, 2009AA012201), 973 计划课题(2007CB316502), 国家自然科学基金项目(90718015), NSFC-微软亚洲研究院联合资助项目(60970155), 教育部博士点基金项目(20090072110035), 上海市优秀学科带头人计划项目(10XD1404400), 高效能服务器和存储技术国家重点实验室开放基金项目(2009HSSA06)资助。

傅丽丽(1986—),女,硕士生,主要研究方向为异构计算、异构可重构计算,E-mail:lovejay_lili@hotmail.com;曾国荪(1964—),男,博士,教授,博士生导师,主要研究方向为异构计算、信息安全。

本文将采用 FPGA 实现 N 体问题的 PP 算法,并将运行速度与纯软件实现相比较,说明 FPGA 并行计算对于计算密集型任务的性能有显著提升。

2 FPGA 原理及设计流程

2.1 FPGA 结构与工作原理

FPGA 是一类高集成度的可编程器件,其内部的门数已经集成到数百万门甚至数千万门。主要采用可编程的查找表(Look Up Table, LUT)结构, LUT 是可编程的最小逻辑构成单元。大部分 FPGA 采用基于 SRAM 的查找表逻辑形成结构,即用 SRAM(静态随机存储器)来构成逻辑函数发生器。一个 N 输入查找表可以实现 N 个输入变量的任何逻辑功能。图 1 所示是 4 输入 LUT,其内部结构如图 1 右图所示^[2]。

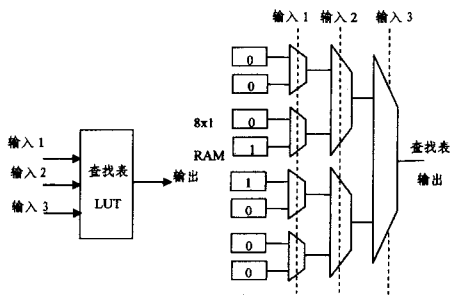


图 1 4 输入 LUT 及其内部结构

2.2 FPGA 编程语言

FPGA 中最常用的硬件描述语言有 VHDL, Verilog, AHDL, SystemC 和 SystemVerilog 等,其中 VHDL 和 Verilog 在 FPGA 设计中使用得最多。硬件描述语言 VHDL 具有很强的电路描述和建模能力,能从多个层次对数字系统进行建模和描述。VHDL 支持各种模式的设计方法:自顶向下、自底向上或两者混合的方法,大大缩短了电子产品的设计周期。设计者在使用 VHDL 设计系统时,可以专心致力于其功能的实现,不必对不影响系统功能却与工艺有关的因素花费过多的时间和精力^[2]。

2.3 FPGA 设计流程

典型的 FPGA 设计流程如图 2 所示,主要包括下面几个部分^[2]。

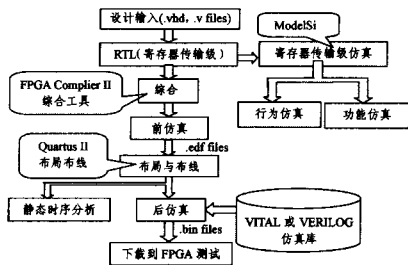


图 2 FPGA 设计流程图

(1)设计输入:设计输入主要有两种方式:图形输入和硬件描述语言文本输入。图形输入通常包括原理图输入、状态图输入和波形图输入等方法。而硬件描述语言输入使用 VHDL 或者 Verilog 语言进行编辑输入,是目前使用的主要方法。

(2)综合:整个综合过程就是将设计的 HDL 文本或者原理图等,依据给定的硬件结构组件和约束控制条件进行编译、优化、转换和综合,最终获得门级电路甚至更底层的电路描述网表文件。

(3)布局布线:将由综合器产生的网表文件配置于指定的目标器件中,使之产生最终的下载位流文件。布局是指从映射中取出定义的逻辑和输入输出块,并把它们分配到 FPGA 内部的物理位置;布线是指利用自动布线软件使用布线资源选择路径试着完成所有的逻辑连接。

(4)仿真验证:仿真分为功能仿真和时序仿真两种。功能仿真也称前仿真,是直接对 VHDL、原理图描述等形式的逻辑功能进行测试模拟,以了解其实现的功能是否满足原设计要求的过,仿真过程不涉及任何具体器件的硬件特性;时序仿真也称为后仿真,是接近于真实器件运行的仿真,其仿真文件中包含了器件硬件特性,如器件延迟、连线延时等时序参数。

(5)下载和硬件测试:在功能仿真与时序仿真正确的前提下,将布局布线后生成的位流文件下载到 FPGA 硬件上,进行实际器件的物理测试。若得到正确的验证结果,则证明设计正确。

3 N 体问题描述及其求解方法

3.1 N 体问题描述

N 体问题描述如下:在三维空间内给定 N 个天体,假设它们之间只有万有引力作用,根据给定的初始状态(包括位置和速度),模拟它们在空间内的运动轨迹。最简单的例子是三体运动,即太阳系中太阳、地球和月球的运动。在浩瀚的宇宙中,我们可以把它们看成质点,并假设它们的运动只是在万有引力的作用下产生的,这样就可以把它们的运动看成是一个三体问题。

3.2 N 体问题求解方法

迄今为止,已经有很多关于 N 体问题的求解方法:Particle-Particle (PP)算法、Barnes-Hut 算法、Particle-Mesh(PM)算法、快速多极子(Fast multipole method, FMM)算法等。

Particle-Particle(PP)算法是最直接、最原始的算法,即在 N 体系统中,根据各天体的初始位置和初始速度,通过在连续的短时间内进行迭代,利用计算机模拟各个天体的运动情况。通常时间步应尽可能小,以得到比较准确的值。在每个时间步中,根据万有引力公式,计算当前天体受到其他所有天体的合作用力,通过进一步计算得到天体下一时刻的位置和速度。在 PP 算法中,每个天体被看作一个独立的个体,对每个天体进行万有引力计算,则会产生 $O(N^2)$ 的计算复杂度。

BH 算法采用树结构进行编码,将距离很近的天体群近似地聚集到一个天体,而天体群对其它天体的作用力也近似为一个天体的作用力。其数据结构采用二叉树(二维)或八叉树(三维)来存储天体。采用树结构代码不仅减少了存储开销,而且更有利于快速计算和并行划分,其算法复杂度也降低为 $O(N\log N)$ 。

快速多极子 FMM 算法和 BH 算法一样,也是采用树结构编码,不同的是它使用位势来代替粒子之间的相互作用力。它的算法思想是对位势函数在远场作多极子展开,再转换为近场的局部展开。多级扩展的位势比简单的质点替代更准确,但是也使计算量增加,不过在每个叶结点上有多多个粒子可以减少计算量。FMM 算法尽管在计算之前需要进行大量的理论分析,但却能得到任意精度,而且通过树结构的并行划分能得到很高的并行效率,其算法复杂度也仅仅为 $O(N)$ 。

3.3 PP(Particle-Particle)算法

PP 算法的具体算法如下:

为了简化模拟,本文假定在二维空间中有 N 个天体,其

质量分别为 $m_i (i=1, 2, \dots, n)$, 初速度为 $\vec{v}_i$, 位置为 $p_i (x_i, y_i)$ 。质量为 m_i 和 $m_j (i \neq j)$ 的两个天体之间, 假设 r_{ij} 为 m_i 和 m_j 之间的距离, 则它们之间的万有引力为

$$F_{ij} = \frac{Gm_i m_j}{r_{ij}^2} \quad (1)$$

每个天体都会受到其余所有 $N-1$ 个天体对它的引力。将各个方向的引力进行矢量相加, 得到此物体受到的总合力 $\vec{F}_i$ (设 $F_{i,x}$ 为 x 轴方向的分力, $F_{i,y}$ 为 y 轴方向的分力)。然后根据公式 $\vec{F}_i = m_i \cdot \vec{a}_i$ 计算出第 i 个天体的加速度 $\vec{a}_i$ (设 $a_{i,x}$ 为 x 轴方向的加速度, $a_{i,y}$ 为 y 轴方向的加速度)。N 体问题主要研究天体在此加速度的作用下的运动轨迹, 即计算出天体在足够短的时间步后的位置和速度, 通过多次迭代从而模拟天体的运动轨迹。

假定该天体在时刻 t_1 的瞬时速度为 $\vec{v}_1$, 位置为 (x_1, y_1) , 记 $dt = t_2 - t_1$, 为一个足够小的时间步。在经过了 dt 时间后, 天体在 $t_1 + dt$ 时刻的瞬时速度为 $\vec{v}_2$, 位置为 (x_2, y_2) , 设 $dv_x = v_{2,x} - v_{1,x}$ 和 $dv_y = v_{2,y} - v_{1,y}$ 分别为该物体在经历时间间隔 dt 后速度在 x 轴和 y 轴方向的增量, 而 $dx = x_2 - x_1$, $dy = y_2 - y_1$ 分别为该物体在时间间隔 dt 后的 x 轴和 y 轴的位移。若 dt 足够小, 则我们可以认为天体在 dt 时间内做匀速运动, 满足以下公式:

$$dv_x = dx/dt \quad (2)$$

$$dv_y = dy/dt$$

$$dv_x = a_x \cdot dt = F_{i,x}/m \cdot dt \quad (3)$$

$$dv_y = a_y \cdot dt = F_{i,y}/m \cdot dt$$

$$v_{2,x} = v_{1,x} + F_{i,x}/m \cdot dt \quad (4)$$

$$v_{2,y} = v_{1,y} + F_{i,y}/m \cdot dt$$

$$x_2 = v_{1,x} \cdot dt + dv_x \cdot dt/2 \quad (5)$$

$$y_2 = v_{1,y} \cdot dt + dv_y \cdot dt/2$$

根据上述所有公式, 可以计算出每个时间步 dt 后天体的速度和位置。迭代一定次数 (例如 20000 次) 后, 将所有的位置都显示出来, 则可以模拟出天体的运动轨迹。

4 N 体问题 PP 算法的 FPGA 设计和实现

4.1 总体设计

整体设计主要分为 3 个模块: 初始化模块 (initial module)、计算模块 (calculation module)、显示模块 (display module)。初始化阶段从 FPGA 的 ROM 中读取相应的初始化数据, 包括各个天体的质量、初速度和初始位置。计算阶段根据本文第 3 节中提到的 PP 算法计算各个天体在单位时间步后的速度和位置。显示阶段则根据每次迭代产生的新位置模拟天体在空间的运行情况。整体设计框如图 3 所示。

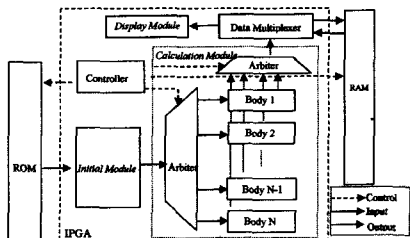


图 3 整体设计框图

4.2 初始化模块 (initial module)

每个天体的质量、初速度、初始位置分别存放在 FPGA 的 ROM 中, 在计算阶段开始前需要从 ROM 中分别读取相应

的数据, 存放在相应的变量中: $m, velocity_x, velocity_y, position_x, position_y$ 。同时, 还要预先定义公式中的一些常量: 万有引力常量 GRAVITY、空间范围大小 $space_x$ 和 $space_y$ 、天体个数 $number_of_bodys$ 以及极短的时间步 dt 。

基本设计步骤如下:

Step1 建立 ROM: package rom1 (其存储顺序依次为质量、初速度、初始位置)

Step2 从 ROM 中读取相应的数据

Step2.1 产生读取地址

if din(i)='1' then

result := result + 2 * i;

Step2.2 根据产生的地址对应读出相应的值

Step3 初始化一些常量, 包括 GRAVITY, $space_x, space_y, number_of_bodys, dt$

generic (number_of_bodys: integer; 10);

.....

4.3 计算模块 (calculation module)

计算模块对每个天体的计算是并行执行的, 主要分为两个部分: 一是根据式 (1) 计算所有天体中任意两个天体之间的引力大小及方向, 然后对每个天体受到的所有力求和, 得到单个天体受到的总力的大小及方向; 二是根据天体受到的力计算 dt 时间后的速度和位置, 假设 dt 足够小, 则可以认为天体在此时间段内做匀速运动。在根据式 (2) 到式 (5) 计算出第 i 个天体在 dt 时间后的位置和速度时要考虑当前天体的新位置是否超出规定的空间范围。如果没有, 则用新计算的速度和位置代替原来的速度和位置, 否则将位置不变、速度方向取反。

计算模块设计关键部分伪代码如下:

Step1 计算两两天体之间的引力

Step1.1 $F_{ij} = \frac{Gm_i m_j}{r_{ij}^2}, i \neq j, i=1, 2, \dots, N, j=1, 2, \dots, N$

If $j=i$ 则 $F_{ij}=0$

Step1.2 $F_i = \sum_{j=1, N, j \neq i}^N F_{ij}$

Step2 对任一天体, 计算其单位时间步后的速度

$F_i = \sum_{j=1}^N F_{ij}, i=1, 2, \dots, N$

$a_i = \frac{F_i}{m_i}$

$v_i(t+dt) = v_i(t) + a_i \cdot dt$

Step3 对任一天体, 计算其单位时间步后的位置

$x_i(t+dt) = x_i(t) + \frac{1}{2} \cdot a_i \cdot dt^2 + v_i(t) \cdot dt, i=1, 2, \dots, N$

Step4 根据 Step2 和 Step3 的结果更新速度和位置, 并把计算结果保存到 RAM

Step5 迭代执行 Step1—Step4 20000 次

在此模块中需要进行大量的加法、减法、乘法、除法和开平方运算, 但是 FPGA 本身并不自带乘法、除法和开平方的库函数, 所以需要自己编写相应的函数库。开平方运算采用的是二分逐次逼近法原理进行设计。根据文献 [4] 此设计方法硬件资源利用得少, 而且精度较高, 误差控制在 1 范围内。乘法、除法则根据最普通的竖式计算的方法编写。同时, 为了提高运行速度, 在所有设计中采用流水线方法。例如在进行 32 位乘法运算时, 如果不使用流水线方法, 则每隔 32 个时钟周期才能出一个计算结果; 而采用流水线方法, 只有在第一个结果出来之前需要 32 个时钟周期, 后面则每隔一个时钟周期都能出一个结果。

在计算模块中, 每次计算的结果都会存储在 RAM 中, 将它们的数据导出。然后利用 MATLAB 画图工具, 模拟出天体运动的轨迹。

4.4 设计实现

用 VHDL 设计实现该算法时,主要包括 3 个子模块:

- (1)readrom 模块:该模块从 rom 中读取 N 个天体的质量 *mass*、初速度 *velocity_x*,*velocity_y* 和初始位置 *position_x*,*position_y*;
- (2)CalForce 模块:该模块计算每个天体受到的总力的大小,最后输出 *force_x* 和 *force_y*;
- (3)MovBody 模块:该模块根据 CalForce 模块计算的天体受到的力的大小以及初始位置、初速度计算出的新速度和位置。

用 FPGA 开发工具 Quartus II 设计的总原理图如图 4 所示。

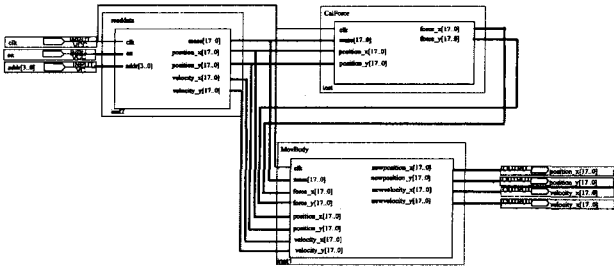


图 4 设计原理图

其中 MovBody 模块的 process 部分关键代码如下:

```
process(clk,force_x,force_y,mass,tempposition_x,tempposition_y,dvelocity_x2,dvelocity_y2)
begin
    if clk='1' and clk'event then
        dvelocity_x1<=force_x/mass;
        dvelocity_x2<=dvelocity_x1 * dt;
        dvelocity_x<=dvelocity_x2/10;
        --计算时间步 dt 为 0.1 后 x 方向的速度变化,y 方向类似
        .....
        dposition_x1<=velocity_x+tempdvelocity_x;
        dposition_x2<=dposition_x1 * dt;
        dposition_x<=dposition_x2/10;
        tempposition_x<=position_x+dposition_x;
        --计算时间步为 0.1 后 x 方向的位置,y 方向类似
        .....
        if tempposition_x<0 or tempposition_x>space_x or tempposition_y<0 or tempposition_y>space_y then
            newposition_x<=position_x;
            newposition_y<=position_y;
            newvelocity_x<=-velocity_x;
            newvelocity_y<=-velocity_y;
        else newposition_x<=tempposition_x;
            newposition_y<=tempposition_y;
            newvelocity_x<=velocity_x+dvelocity_x;
            newvelocity_y<=velocity_y+dvelocity_y;
        end if;
    end if;
end process;
```

5 实验结果展示

本实验考虑 10 个天体的运动轨迹。相关初始化数据如下:(1)单位时间步 *dt* 为 0.1;(2)空间的范围为(100000,100000);(3)迭代次数 *N*=20000;(4)万有引力常量假设为 6.67;(5)质量、初速度和初始位置如表 1 所列。表 2—表 4 分别表示 10 个天体运动 5000,10000,20000 个时间步后的位置

和速度。

表 1 初始数据

	质量	x 位置	y 位置	x 速度	y 速度
body 1	1E+07	50000	50000	10	10
body 2	1	75000	10000	0	-20
body 3	2000	80000	50000	0	-40
body 4	1	35000	95000	20	0
body 5	1	50000	90000	40	0
body 6	1	67000	34000	-15	-15
body 7	1	30000	30000	20	-20
body 8	1	75000	74000	-30	30
body 9	1	22000	78000	45	45
body 10	1	83000	18000	40	40

表 2 5000 个时间步后的运行结果

	质量	x 位置	y 位置	x 速度	y 速度
body 1	1.00E+07	55002	54999	10.007	9.9962
body 2	1	73454	2941.4	-5.483	-8.632
body 3	2000	70513	33062	-36.31	-21.207
body 4	1	46226	90973	25.078	-17.623
body 5	1	69165	84260	34.449	-24.221
body 6	1	51149	42138	-37.03	67.076
body 7	1	45634	27657	39.926	11.934
body 8	1	55750	82184	-44.03	-1.6989
body 9	1	47342	96190	54.396	27.157
body 10	1	99838	40252	26.867	47.944

表 3 10000 个时间步后的运行结果

	质量	x 位置	y 位置	x 速度	y 速度
body 1	1.00E+07	60006	59995	10.008	9.9878
body 2	1	69997	1091.5	-8.0007	1.0511
body 3	2000	49311	33042	-41.335	20.86
body 4	1	60412	72408	31.3	-77.603
body 5	1	79620	64476	-3.2051	-54.201
body 6	1	77944	59603	38.734	-36.725
body 7	1	66866	46476	35.444	76.576
body 8	1	38084	69702	-16.868	-44.727
body 9	1	37590	90344	-50.379	-35.034
body 10	1	83576	18860	-38.606	-38.345

表 4 20000 个时间步后的运行结果

	质量	x 位置	y 位置	x 速度	y 速度
body 1	1.00E+07	70010	69979	9.999	9.9822
body 2	1	61415	11168	-8.2104	19.142
body 3	2000	30622	74462	5.0153	48.64
body 4	1	57710	63334	91.02	-82.105
body 5	1	40955	76241	-3.9576	54.306
body 6	1	63027	62773	-17.493	100.05
body 7	1	51630	46125	26.864	-11.79
body 8	1	62731	34212	46.336	-12.103
body 9	1	2446.2	47604	-24.109	-43.376
body 10	1	75571	11482	41.339	32.349

将数据图形显示后结果如图 5 所示。* 代表原始位置,o 代表运行了 20000 个时间步后的位置,虚线表示天体的运动轨迹。

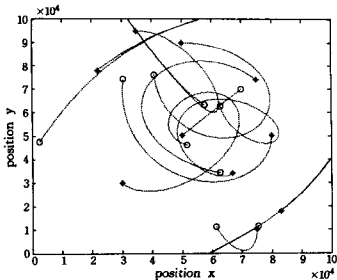


图 5 10 个天体运动了 20000 个时间步后的运动轨迹图

结束语 本文使用 FPGA 求解和实现了一个经典的动力学问题——N 体问题,从而提供了一种新型的解决计算密集型任务的方法。实验表明,此方法具有可行性,求解速度较一般软件编程实现要快,而且硬件资源占用率也不是很大,开发过程中编写的一些程序具有可重复使用性。

随着高性能计算的进一步发展,不仅是 FPGA,还有 GPU,CELL 等都令人关注。将它们混合集成到一起,构成异构重构计算系统,开展高性能并行计算,是本课题组将来的研究工作。

参考文献

- [1] Lindholm T. N-body algorithms [C]// Seminar presentation. 1999
- [2] 潘松,黄继业. EDA 技术与 VHDL[M]. 北京:清华大学出版社, 2006
- [3] 郑有泉. 现场可编程门阵列第一讲现场可编程门阵列 FPGA 概述[J]. 世界电子元器件, 2005, 9(1): 40-45
- [4] 张博. 基于逐次逼近的 VHDL 开平方算法[J]. 微计算机信息, 2008, 24(5): 196-197

- [5] 李旭. 基于 FPGA 的流水线技术应用研究[J]. 电子测量技术, 2007, 30(2): 131-132
- [6] Hamada T, Benkrid K, Nitadori K, et al. A comparative study on ASIC, FPGAs, GPUs and general purpose processors in the $O(N^2)$ gravitational N-body simulation[C]// NASA/ESA Conference on Adaptive Hardware and System. 2009: 447-452
- [7] Lienhart G, Kugel A, Manner R. Using floating-point arithmetic on FPGAs to accelerate scientific N-body simulations[C]// Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines. 2002: 182-191
- [8] Liu P F, Bhatt S N. Experiences with parallel N-body simulation [J]. IEEE Transactions on Parallel and Distributed Systems, 2000, 11(12): 1306-1323
- [9] Ho H, Chun C W, Yu P, et al. Floating-point FPGA: architecture and modeling[J]. IEEE Transactions on Very Large Scale Integration Systems, 2009, 17(12): 1709-1718
- [10] Valls J, Sansaloni T, Peiro M, et al. Fast FPGA-based pipelined digit-serial/parallel multipliers [C] // Circuits and Systems, ISCAS'99 Proceedings of the 1999 IEEE International Symposium. 1999: 482-485

(上接第 301 页)

同样,饱和减函数 sub()也可以用 QSUB16 指令来替换。

2 ARM 在视频通信中的应用

2.1 充分利用寄存器进行数据操作

H. 263 是 ITU-T(国际电信联盟)发布的视频编码标准,专为中高质量运动图像压缩设计。它采用 DCT(离散余弦变换)和 IDCT(反向离散余弦变换)进行视频压缩,可以得到很不错的视频效果,因此常应用于视频会议和可视电话等通信应用中^[2]。

ARM 内核有 16 个寄存器,它们可以作为通用寄存器,在视频编码中,编译器并不能有效、充分地使用这 16 个寄存器。比如 DCT 和 IDCT 函数有很多数组^[3],编译器一般直接编译成内存读取操作。DCT 和 IDCT 在运行过程中多次调用,造成频繁的访存操作,消耗较多的时间。如果能充分利用这 16 个寄存器,用汇编语言来精心编排,把经常用到的数组元素用寄存器来暂时存放,则可以有效提高程序的运行速度。这里有一个原则,就是必须确保最重要的和经常用到的数组元素被分配到寄存器。

2.2 把移位操作和其他操作作用一条指令实现

ARM 有一个特征,那就是寄存器可以在进入 ALU(算术逻辑单元)之前先通过桶式移位器进行移位操作,即预处理。这些移位操作包括 LSL(逻辑左移)、ASL(算术左移)、LSR(逻辑右移)、ASR(算术右移)、ROR(循环右移)以及 RRX(带扩展的循环右移)操作,它们可以和数据处理指令(ADC, ADD, AND, BIC, CMN, CMP, EOR, MOV, MVN, ORR, RSB, SBC, SUB, TEQ, TST)一起使用,不占用一个独立的指令,从而提高了代码密度,减少了对存储器的访问次数。

比如在 IDCT 函数中有一条语句:

$X0 = (blk[0] \ll 11) + 128$

数组元素 blk[0]先左移 11 位,然后加上立即数 128,假

设 R1 寄存器保存的是 X0, R2 寄存器保存的是 blk[0], R3 寄存器保存的是 128,则汇编语句如下:

ADD R1, R3, R2, LSL # 11

一条汇编指令就实现了两个操作,节省了取指令的时间。

2.3 实验分析

我们把 H. 263 的 DCT 和 IDCT 函数用汇编语言来实现,在汇编过程中采用上面的两种方法,把汇编实现后的函数部署到 HKC 智能手机中进行多次测试,并将结果与汇编前进行对比。HKC 的配置如下:

HKC G801:处理器内核是 ARMv5,主频是 416MHz,内存是 64MB。

实验结果如表 1 所列。

表 1 汇编前后的效率对比

	DCT 运行 1000 次的 时间	IDCT 运行 1000 次的 时间	H. 263 的 CPU 使用率
汇编前	20.5ms	18.7ms	90%
汇编后	10ms	9.3ms	52%
优化效率	51.2%	50.3%	42.2%

由该表可知,采用本文的方法使 H. 263 的 CPU 使用率下降了 40%左右,效果还是很明显的。

结束语 ARM 内核在音视频信号处理方面有很大的优势,因此特别适用于音视频编码中,它的性能在某些方面甚至超过了专门的 DSP。

参考文献

- [1] Sloss A N, et al. ARM 嵌入式系统开发:软件设计与优化[M]. 北京:北京航空航天大学出版社, 2005: 528-536
- [2] 余振建,周健,戴梅萼. 面向运动图像远程实时传输的 H. 263 压缩方法的分析与优化[J]. 电子技术应用, 2003, 29(1): 50-52
- [3] 郑晓博,陶品. ARM7 平台实时 MPEG-4 解码系统的实现与优化[J]. 计算机科学, 2008, 35(1): 246-249