面向 OpenCL 模型的 GPU 性能优化

陈 钢[1,2], 吴百锋[1]

[1] (复旦大学计算机科学技术学院 上海 200433)

[2] (AMD 先进微处理机技术实验室 上海 201203)

(071021049 @fudan.edu.cn)

摘 要: GPU 的高性价比吸引了越来越多的通用计算. 为充分发挥异构处理平台下 GPU 的通用计算能力,提出面向 OpenCL 模型的性能优化方法. 该方法建立源程序的多面体表示,分别对 GPU 的全局存储器和快速存储器进行优化与分配;通过检测存储访问模式发掘可向量化的存储访问实例,利用数据空间变换对存储访问模式进行转换,进而使用向量数据类型提高片外存储器的带宽利用率;通过检测程序中的数据重用,根据数据的访问属性和 OpenCL 存储模型的特性实现快速存储器的有效分配与优化,提高了片上存储器的使用效率. 采用文中方法对 6 个测试程序进行实验的结果表明,程序的性能提高了 1.6～8.4 倍,证实了其有效性.

关键词: OpenCL; GPU;性能优化;异构处理;通用计算;多面体表示

中图法分类号: TP391

GPU Performance Optimization Targeting OpenCL Model

Chen Gang[1,2] and Wu Baifeng[1]

[1] *(School of Computer Science, Fudan University, Shanghai 200433)*

[2] *(AMD Advanced Microprocessor Technique Laboratory, Shanghai 201203)*

Abstract: Graphic processing units attract more and more general-purpose computing due to high performance/cost ratio. In order to fully exploit the capability of GPU for general-purpose computing under heterogeneous processing platforms, this paper proposes performance optimization methods targeting OpenCL model. Polyhedral representation of a source program is built to optimize and allocate GPU memory system. By checking memory access patterns of the source program, access instances those can be grouped together are discovered by means of graph coloring. Subsequently, data space transformation is utilized to alter irregular memory access patterns for the sake of improving the off-chip memory bandwidth by taking advantage of vector data types. Meanwhile, data reuse information is detected to allocate data into distinct fast memory regions according to both the properties of data accesses and the characteristics of the OpenCL memory model, with the purpose of making best usage of the fast on-chip memory. Experimental results on benchmarks showed that the optimized programs achieved a speedup of 1.6X～8.4X in comparison with the un-optimized versions, demonstrated the effectiveness of the proposed methods.

Key words: OpenCL; GPU; performance optimization; heterogeneous processing; general-purpose computing; polyhedral representation

现代图形处理器(graphics processing unit, GPU)采用相对简单的控制逻辑,大量晶体管被用于算术逻辑单元(arithmetic logic unit, ALU)参与数据处理[1].凭借强大的计算能力和卓越的性价比,GPU 吸引了越来越多的通用计算(general-purpose computation on graphics processing units, GPGPU)[2-3],涉及的领域包括流体模拟[4]、视频检测[5]、序列比对[6]和蛋白质分子场[7]等.为克服传统依靠图形接口对 GPU 编程的缺陷,AMD 和 NVIDIA 分别推出了各自的编程模型——Brook + ①和计算统一设备架构(compute unified device architecture, CUDA)②.但是在程序可移植性方面,AMD GPU 与 NVIDIA GPU 互不兼容.为充分发挥异构处理平台下各种设备的性能潜力,同时使得程序具备可移植性,Khronos Group 推出了开放式计算语言(open computing language, OpenCL)③.

高级编程模型在一定程度上降低了程序设计的学习曲线,但编写高性能的通用计算程序仍需考虑如何将代码有效映射至 GPU 硬件上加以执行,尤其是充分利用 GPU 高效的存储带宽[8].GPU 通常采用分散式设计,其存储访问延迟高达数百个时钟周期[9].与图形应用相比,通用计算程序中存在大量循环以稀疏、非对齐的方式访问数据,影响它们在 GPU 上的执行性能.由于 GPU 主要面向图形应用,需要快速编译以满足 3D 游戏的实时性;加之 GPU 底层硬件的复杂性,编译器并没有对程序进行充分的优化[10].为使通用计算程序在异构处理平台下充分利用 GPU 的处理能力,本文针对 OpenCL 模型提出基于程序多面体表示的性能优化方法,分别对 GPU 的全局存储器和快速存储器进行优化与分配.对 6 个常用的通用计算程序实施文中优化方法后的实验结果表明,优化后的程序可以映射至 GPU 上高效执行,并取得了 1.6～8.4 倍的加速比.

1 相关工作

为指导通用计算程序有效映射至 GPU 体系结构,GPGPU 性能优化技术至关重要.Ryoo 等[11-12]针对 NVIDIA GPU 定义了效率和利用率模型,通过优化空间的修剪获得应用程序的最优配置;同时,他们指出片外存储器的访问效率对于程序的性能至关重要.为创建足够多的线程以隐藏片外存储访问延迟,他们建议使用较小的线程块、减少线程动态执行的指令数目、重新分配线程的计算任务等方式,以均衡 GPU 上各种资源的利用率;但其不足之处在于这种优化假设 GPU 上可用的资源不受限制,仅适用于计算受限型的应用.Jang 等[13]针对 AMD GPU 提出了 ALU 利用率、纹理单元利用率和线程利用率 3 个优化空间,他们建议尽可能使用内建函数、合并子函数调用、同时输出多个执行结果等方式,以增加程序的算术密度,优化 GPU 上各种资源的利用率.他们也指出创建足够多的线程以隐藏片外存储访问延迟,是程序获取高性能的关键手段;然而却并没有针对 GPU 的存储系统进行优化.Baskaran 等[14]针对 CUDA 程序中的循环嵌套进行优化,进而改善片外存储器的接合访问;同时采用基于模型驱动的经验搜索以确定应用程序最佳的变换参数,如循环展开次数和循环分块大小.Lee 等[15]实现了 OpenMP 到 CUDA 程序的自动转换,其使用循环交换和循环分裂优化循环嵌套中 GPU 的片外存储访问.然而,循环变换往往受制于数据相关和循环携带依赖,并且基于模型驱动的方法难以适应日趋复杂的 GPU 硬件平台和应用程序.韩博等[16]通过分析传统 GPU 的硬件架构和运行机制,提出了一个简单的 GPU 性能模型,以计算强度和访问密度作为性能优化的基本准则;但这种模型是否适用于新型 GPU 架构还没有得到充分的验证.

2 OpenCL 模型

作为开放的工业标准,OpenCL 架构可以用 4 个模型来描述:平台模型、执行模型、存储模型和编程模型.

2.1 平台模型

在平台模型中,一个主机(host)连接一个或多个 OpenCL 设备,这些设备可以是 GPU,CELL BE 或 DSP 等.本文的异构处理平台是 CPU + GPU,因此设备在本文中指 GPU.一个 OpenCL 设备被划分为若干个计算单元(compute unit, CU),每个 CU 进一步被划分为若干个处理元件(processing element, PE),设备上的计算在 PE 中完成.OpenCL 应用程序通过 host 提交命令,驱动设备上的 PE 执行核心程序.

2.2 执行模型

执行模型定义了核心的执行方式.当 host 提交

核心执行时,系统生成一个 N 维($1 \leq N \leq 3$)的索引空间 $NDRange$. 从软件的角度看,运行在 PE 上的核心实例称为工作项,可以通过 $NDRange$ 中的全局 ID 对其标识. 为提供对 $NDRange$ 粗粒度的分解,若干个工作项被组织为一个工作组;同样,每个工作组也有其唯一的工作组 ID. 此外,在每个工作组中,工作项还拥有一个局部 ID,此 ID 在其所在的工作组中是唯一的. 因此,单个工作项可以通过其全局 ID 或通过其所在的工作组 ID 加局部 ID 来唯一标识. 一个给定工作组中的工作项被分配到单个 CU 中的多个 PE 上并发执行,如图 1 所示.

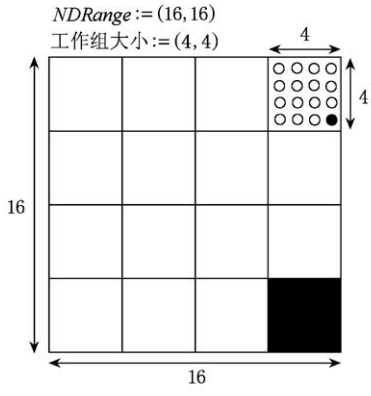

图 1　二维 OpenCL 执行模型索引空间

2.3　存储模型

OpenCL 存储模型指定执行核心的工作项可以访问设备上 4 块不同的存储区域,如图 2 所示. 全局存储器由主机动态分配,所有工作组中的工作项都可以读写其中的任意数据. 通常情况下,该存储器的容量较大,但访问延迟较高. 在 OpenCL 执行模型中,GPU 可以同时维持大量的工作项,以零开销的硬件切换隐藏存储访问延迟. 如果程序中存在过多的不规则存储访问,同一工作组中的工作项会同时访存从而造成工作项暂停,浪费了宝贵的 ALU 资源. 因此,减少工作项对全局存储器的不规则存储访问对于提升程序的性能至关重要. 常数存储器是全局存储器中的一块区域,由主机动态分配并初始化,其在核心执行的过程中保持不变. 局部存储器是一个隶属于特定工作组的存储区域,可用来分配一些由该工作组中所有工作项共享的数据,它通常位于片上,可以较快的速度访问,但容量非常有限. 因此,合理地利用局部存储器对于程序性能的提升同样重要. 私有存储器是一个工作项的私有存储区域,该区域中的数据对于其他工作项是不可见的.

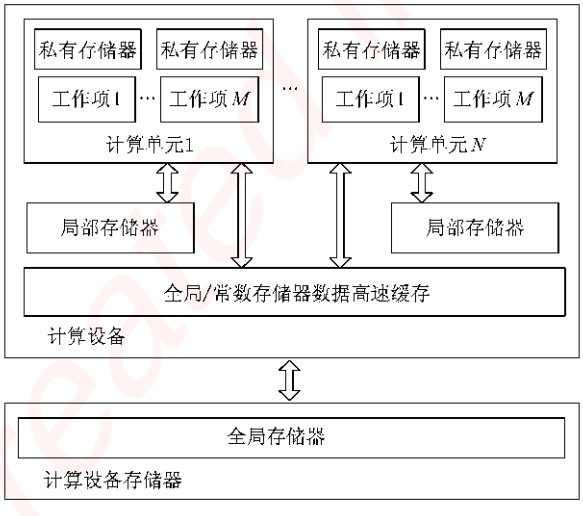

图 2　OpenCL 存储模型

2.4　编程模型

OpenCL 编程模型支持数据级并行和任务级并行,同时也支持这 2 种方式的混合. 为充分发挥异构处理平台中设备的计算能力,OpenCL 的设计重点在于数据并行(任务并行主要针对多核 CPU). OpenCL 提供了 2 种数据并行编程模型:在显式模型中,程序员定义并行执行的工作项的总数,并指定如何将工作项划分为工作组;在隐式模型中,程序员仅指定前者,后者由 OpenCL 的具体实现来管理.

3　GPGPU 性能优化空间

为统一起见,约定 OpenCL 存储模型中除全局存储器之外的存储区域统称为快速存储器. 本文采取的优化框架如图 3 所示,在建立源程序多面体模型的基础上,分别对全局存储器和快速存储器进行优化与分配.

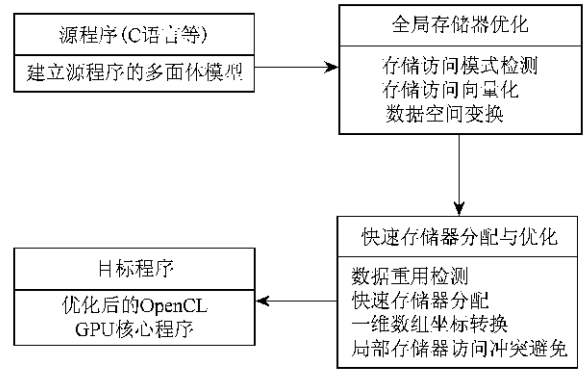

图 3　性能优化框架

3.1　程序的多面体表示

多面体模型是一种程序表示的代数框架,它通过迭代域、访问函数和仿射调度 3 种操作表示代码,近似地给出程序的属性[17-18],该模型可以处理具有

仿射数组访问和循环边界的循环结构. 因此,本文假设循环边界和数组访问是循环索引和全局参数的仿射函数,这类代码在科学计算、媒体处理等领域中占有相当大的比例[19].

一个 n 重嵌套循环定义了一个 n 维空间里的多面体,称为迭代空间多面体. 其中的每个点对应循环体 S 的一次执行实例,可由一个 $n \times 1$ 维的列向量 $\mathbf{x}_S$ 表示,即 $\mathbf{x}_S = (x_1, x_2, \cdots, x_n)^{\mathrm{T}}$, $x_1, x_2, \cdots, x_n$ 从左至右分别代表最外层循环索引变量直至最内层循环索引变量, $\mathbf{x}_S$ 称为迭代向量. 迭代空间多面体可以用一组仿射不等式来说明,即 $D(\mathbf{x}_S) \geq 0$. 使用齐次方程的形式,迭代空间多面体可表示为

$$\begin{matrix}\mathbf{x}_S \\ \mathbf{D}_S \cdot \\ 1\end{matrix}\begin{matrix}\\ \mathbf{n} \\ \\ \end{matrix} \geq 0;$$

其中,矩阵 $\mathbf{D}_S$ 表示循环边界限制,向量 $\mathbf{n}$ 表示全局参数(例如问题规模).

一个 m 维数组定义了一个 m 维数据空间里的多面体,其中的每个点代表了一个数组元素;同样,数组访问函数也可以用齐次方程形式表示. 循环体 S 对数组 A 的访问可以表示为

$$F_A(\mathbf{x}_S) = \mathbf{f}_A \begin{matrix}\mathbf{x}_S \\ \mathbf{n} \\ 1\end{matrix}.$$

其中,访问矩阵 $\mathbf{f}_A$ 表示从 S 的迭代空间到 A 的数据空间的仿射映射,矩阵的每一行反映了数据空间中每一维的访问模式. 如果一个数组 A 的访问矩阵的秩小于访问该数组的迭代空间的维数,即 $\mathrm{rank}(\mathbf{f}_A) < \dim(\mathbf{x}_S)$,则数组 A 在迭代空间 $\mathbf{x}_S$ 中存在数据重用.

在多面体模型中,仿射调度可以是时间调度或空间调度[20]. 时间调度为语句 S 的每次执行分配一个时间戳. 为保证数据的依赖关系,语句实例必须按照时间戳的递增次序依次执行,具有相同时间戳的语句实例可以并行执行. 在 OpenCL 模型中,同一工作组中的所有工作项可以同时执行,时间调度由 GPU 设备硬件和 OpenCL 软件系统来保证. 空间调度将迭代空间分配到不同的处理单元中,对于 S 的一个语句实例,空间调度返回执行该实例的处理单元的序号,在 OpenCL 模型中,即返回每个工作项的全局 ID.

3.2 全局存储器访问模式检测

GPU 存储系统的设计目标是维持高吞吐量,而非低延迟. 因此,为充分利用 GPU 高效的存储带宽,应尽可能保持工作组具有较大的全局存储器访问粒度,即工作组中的工作项对全局存储器中数据的连续访问. 在这种访问模式下, GPU 存储系统可以把多个小而分散的存储访问合并为一个大而集中的存储访问.

定义 1. 对数据空间(数组)的连续访问是指访问该数据空间的下标表达式从其每一维的起始位置以步长为 1 的方式进行访问,即单调递增地访问全局存储器中连续排列的相邻数据单元.

稀疏存储访问模式和非对齐存储访问模式会降低全局存储器访问的性能. 如图 4 所示,假设一次存储器请求可以读入 4 个数据单元,图 4a 中访问数据单元 0 和 4 需要 2 次存储器请求,但每次只用到了其中的一个数据单元,存储带宽的有效利用率仅为 25 %;图 4b 中访问数据单元 1,2,3,4 同样需要 2 次存储器请求,此时带宽利用率为 50 %. 所以,不连续的存储访问在较大程度上浪费了全局存储器带宽. 由于工作组中的所有工作项以单指令流多数据流的方式执行,它们的存储访问指令在同一时钟周期执行,因此如果所有工作项的存储访问以顺序方式执行,势必会同时等待,影响程序的执行性能. 为最大化程序映射为 OpenCL 核心后的存储带宽利用率,同一工作组中的不同工作项应具有图 4c 所示的理想存储访问模式:第一个工作项访问对齐存储段①的第一个元素,第二个工作项访问对齐存储段的第二个元素等. 如此,所有工作项产生连续的存储器访问地址,最大限度地利用了全局存储器带宽.

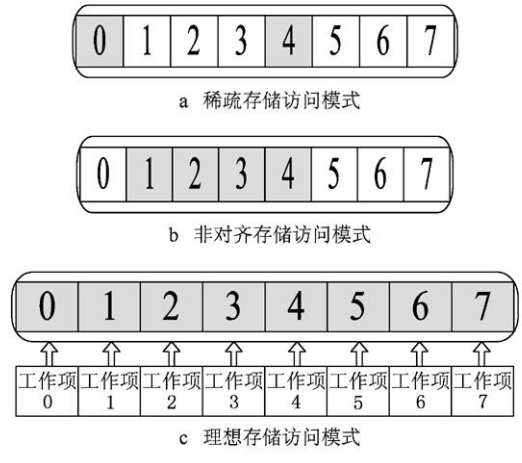

图 4 不同的存储访问模式

为满足图 4c 中的理想存储访问模式,一个工作

① 为保证兼容各种厂商不同型号的 GPU 设备,本文作一个充分但非必要条件的假定:对齐的存储段开始于数据空间每一维的起始地址

组中的工作项需要满足如下条件:

条件 1. 工作项必须同时执行;

条件 2. 工作项的全局 *ID* 在 *NDRange* 中必须连续;

条件 3. 工作项所访问的数据元素必须在全局存储器中连续存放;

条件 4. 具有最小 *ID* 的工作项所访问的数据地址必须是数据空间某一维的起始地址.

考虑循环体语句 S 对数组 A 的访问,W_x 和 W_y 代表工作组中的 2 个工作项,它们执行的语句实例分别为 x_S 和 y_S.用 θ_t 表示时间调度,θ_s 表示空间调度,上述 4 个条件用多面体模型可以分别表示为

$$\theta_{tW_x}(x_S) = \theta_{tW_y}(y_S),$$

$$\theta_{sW_x}(x_S) = \theta_{sW_y}(y_S) + 1,$$

$$F_{AW_x}(x_S) = F_{AW_y}(y_S) + (0\cdots1)^{\mathrm{T}},$$

$$lastrow(f_{W_xAS}) \cap lastcolumn(f_{W_xAS}) = 0,$$

$$lastrow(f_{W_yAS}) \cap lastcolumn(f_{W_yAS}) = 0.$$

3.3 全局存储器访问向量化

为充分利用 OpenCL 数据并行编程模型,可以在程序中显式地使用向量数据类型.相对标量数据类型,向量数据类型可由 2～4 个 32 位的标量数据类型组合而成,其大小由向量数据类型的后缀数字指示.例如,使用 float4 向量数据类型,可以在一次存储访问请求中取入 4 个 float 类型的标量数据.与标量数据类型相比,使用向量数据类型后的存储访问次数有所减少,较大限度地利用了存储带宽.

为发掘程序中可向量化的存储访问实例,本文在文献[20]的基础上定义几种较为常见的存储访问模式,如图 5 所示.

正向访问	逆向访问	偏移访问	倾斜访问	跳步访问	按列访问
$\begin{matrix}1&0&0&0\\0&1&0&0\end{matrix}$	$\begin{matrix}1&0&0&0\\0&-1&0&0\end{matrix}$	$\begin{matrix}1&0&0&0\\0&1&0&2\end{matrix}$	$\begin{matrix}1&0&0&0\\1&1&0&0\end{matrix}$	$\begin{matrix}1&0&0&0\\0&5&0&0\end{matrix}$	$\begin{matrix}0&1&0&0\\1&0&0&0\end{matrix}$
for(i=0;i<N;i++)	for(i=0;i<N;i++)	for(i=0;i<N;i++)	for(i=0;i<N;i++)	for(i=0;i<N;i++)	for(i=0;i<N;i++)
for(j=0;j<N;j++)	for(j=N;j>0;j--)	for(j=0;j<N;j++)	for(j=0;j<N;j++)	for(j=0;j<N;j++)	for(j=0;j<N;j++)
…A[i][j]…	…A[i][j]…	…A[i][j+2]…	…A[i][i+j]…	…A[i][5j]…	…A[j][i]…

图 5 常见存储访问模式的访问矩阵与代码示例

以行优先存储系统为例,由于数据空间的最后一维代表了其在存储器中的物理存储布局,因此可向量化的存储访问需要保证它们对数据空间的最后一维具有与正向访问兼容的访问模式.由第 3.1 节可知,访问矩阵的最后一行代表了数据空间最后一维的存储访问模式.因此,本文对可向量化的存储访问做如下定义:

定义 2. 在一条循环体语句 S 中,如果多个数组的访问矩阵的最后一行与正向访问兼容,则对这些数组的存储访问是可向量化的.

定义 3. 在一条循环体语句 S 中,如果多个存储访问实例访问相同的存储单元,则称这些存储访问实例在语句 S 中存在数据依赖关系.

值得注意的是,本文定义的数据依赖关系与传统基于语句的数据依赖关系不同.考虑如下语句 S: *a[i][j] = a[i][j] + b[i][j];*在传统的数据依赖定义中,语句 S 内部不存在依赖.然而按照本文的定义,语句 S 存在先读后写的数据依赖.基于定义 2 与定义 3,本文采用图着色的方法对程序中的存储访问实例进行可向量化检测,如算法 1 所示.首先,考察循环体语句 S 中所有数组的访问矩阵(代表数组的存储访问模式);其次,建立无向图并连接具有不同存储访问模式的访问实例;再次,检测无向图中相邻节点与正向访问模式的兼容性;最后,通过寻找与正向访问模式具有相同颜色的节点,发掘可向量化的存储访问实例.

算法 1. 存储访问向量化检测

输入.循环体语句 S 中所有数组的访问矩阵集合 $f = \{f_1, \cdots, f_n\}$.

输出.可向量化的存储访问集合 f_{VEC}.

$f_{\mathrm{VEC}} = color = \mathrm{NULL}$

考察 f 中每个 f_i 的存储访问模式

建立无向图 $G = \{V, E\}$,其中,

$V = \{f_1, \cdots, f_n\}$

E 连接 2 个存储访问模式不相同的 f_j 和 f_k

$temp = \{f_1\}$ // f_1 的存储访问模式与正向访问兼容

$color = \{c_1\}$

for f 中所有与 $temp$ 相邻的节点 f_p do

  $temp = temp \cup \{f_p\}$

  if (f_p 不与正向访问兼容)

    对 f_p 着色 c_2

    $color = color \cup \{c_2\}$

  endif

```
    else if(f_p 在 S 中存在数据依赖关系)
        对 f_p 着色 c_3
        color = color ∪{c_3}
    endif
    else
      对 f_p 着色 c_1
    endif
  endfor
  f_VEC = temp ∩{c_1}
```

例 1. 下列代码中语句 S 的可向量化存储访问集合 $f_{VEC}=\{B,C,D\}$,如图 6 所示.

```
for(i=0;i<N;i++)
  for(j=0;j<N;j++)
    S:A[i]=A[i] + B[i][j] + C[i][2j] * D[i][j+10]
```

数组 A 为正向访问模式,但在语句 S 中存在数据依赖关系. 数组 B 的访问模式为正向访问模式,数组 C 的访问模式为跳步访问模式,数组 D 的访问模式为偏移访问模式. 经算法 1 检测后,对数组 C, D, E 的存储访问可向量化.

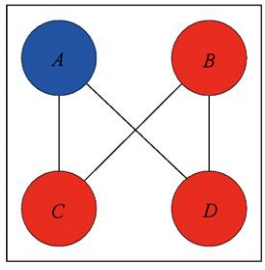

图 6　图着色的存储访问向量化检测实例

3.4　数据空间变换

存储访问向量化测试发掘了程序中可向量化的存储访问实例,而使用向量数据类型的前提是数据必须在存储器中连续存放. 为此,本文采用数据空间变换[20]把不连续的存储访问(非正向访问模式)转化为连续的存储访问(正向访问模式),即图 4c 所示的理想存储访问模式.

定义 4. 若一个 m 维数组的访问矩阵为 f(大小为 $m\times n$),对其实施数据空间变换(T,ε)后,其访问矩阵变为 f′,则有 $f'=Tf+\varepsilon$,其中 T 为 $m\times n$ 的变换矩阵,ε 为 $m\times n$ 的偏移校准矩阵.

以图 5 为例,数据空间变换(T,ε)需要将非正向访问模式的访问矩阵变为正向访问模式的访问矩阵

$$f'=\begin{matrix}1&0&0&0\\0&1&0&0\end{matrix};$$

令 $Tf+\varepsilon=f'$,得到

$$Tf+\varepsilon=\begin{matrix}1&0&0&0\\0&1&0&0\end{matrix}.$$

通过高斯变换很容易求得(T,ε)的值. 值得注意的是,由第 3.2 节中的条件 4 可知,偏移校准矩阵具有如下特征:

1) 仅在变换偏移访问模式的情况下不为零矩阵;

2) 在任何情况下,除了最后一列以外,其他列全为 0.

当一个存储访问实例存在多个访问模式的组合时,可以分别针对每个访问模式进行变换,最终转化为正向访问模式. 算法 2 描述了数据空间变换的过程.

算法 2. 数据空间变换

输入. 可向量化的存储访问集合 f_{VEC}.

输出. 经过数据空间变换后的向量化存储访问集合 f'_{VEC}.

```
f'_VEC = NULL
for each f_i ∈ f_VEC do
  if(f_i 是正向访问模式)
    T为单位矩阵,ε=0   //不需要对其变换
  endif
  else if(f_i 是逆向访问模式)
    T为单位矩阵中对应逆向访问的元素取反(以 -1 代替 1),ε=0   //按列反转
  endif
  else if(f_i 是偏移访问模式)   //偏移量为 c
    T为单位矩阵,ε为零矩阵最后一列(0,0,…,c)^T
    //按行移位,移位量为 c
  endif
  else if(f_i 是倾斜访问模式)
    T为单位矩阵中对应倾斜访问的元素扩展 X 倍后取反(以 -X 代替 X, X=1,2,3…),ε=0
    //按行动态移位,移位量为相应外层循环索引变量值
  endif
  else if(f_i 是跳步访问模式)   //步长为 K
    T为单位矩阵中对应跳步访问的元素扩展 K 倍后取倒数(以 1/K 代替 K, K=2,3,4…),ε=0
    //按行聚集压缩
  endif
  else if(f_i 是按列访问模式)
    T为单位矩阵的转置矩阵,ε=0   //行列交换
  endif
  f_i′ = Tf_i + ε
  f'_VEC = f'_VEC ∪{f_i′}
endfor
```

3.5　快速存储器的分配与优化

由第 2.3 节可知,OpenCL 存储模型包含一些可以快速访问的存储区域,如常数存储器、局部存储

器和私有存储器,合理地利用这些快速存储器对于程序的性能至关重要.在 OpenCL 存储模型中,还有一种可供使用的 IMAGE 存储对象.在某些特定的场合中,IMAGE 存储对象通过使用 GPU 设备的纹理高速缓存提供额外的存储带宽.IMAGE 对象可以存储二维或三维的结构化数组,可通过采样器访问 IMAGE 中的数据.在通用计算中,IMAGE 非常适用于实现图像处理和查找表,对大量数据的随机访问有着良好的加速效果.为使一维数组能够有效地利用 IMAGE 对象,本文将其转换为具有二维坐标的数组,如图 7 所示.对于一个一维数组 A,定义宽度为 W,则 A 中的元素 p 的二维坐标为

$$(x,y) = (p\%W, p/W) \tag{1}$$

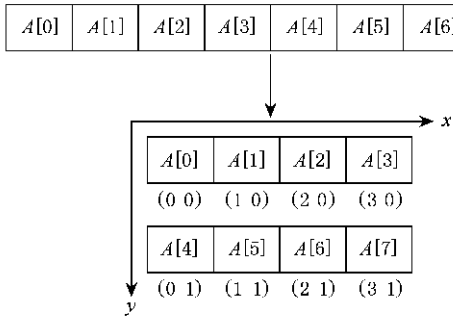

图 7　一维数组的坐标转换($W=4$)

为有效地使用各个快速存储器,本文对数据访问属性做如下定义:

定义 5. 若一个数组在循环体中多次被访问,或该数组的访问矩阵的秩小于访问该数组的迭代空间的维数,则称该数组存在数据重用.

定义 6. 若一个数组的访问矩阵不随迭代空间的循环索引变量发生变化,则称该数组为只读的.

定义 7. 若一个数组的访问矩阵存在随机数,即存储访问模式没有固定的规律,则称该数组的存储访问模式为不可预知的(例如对数组的间接访问).

为了提供较高的带宽,局部存储器被划分为大小相等、能够被同时并行访问的存储器模块.在一个时钟周期内,每个存储器模块只能响应一次访问请求.如果工作组中不同的工作项都访问不同的存储器模块,此时局部存储器带宽的有效利用率最高.如果多个工作项请求访问的多个数据位于同一个存储器模块时,就会出现模块冲突.由于存储器模块在同一时刻无法同时响应多个访问请求,这些存在冲突的请求必须在几个连续的时钟周期内串行完成.由此可见,有效地使用局部存储器的关键是避免访问冲突.以 AMD/ATI HD Radeon™ 5850 GPU 为例,其局部存储器的大小为 32 KB,组织为 32 个存储器模块,每个模块大小为 1 KB(256 ×4B).当各工作项以间隔为 1 的方式访问局部存储器时,此时没有模块冲突,如图 8a 所示.但当各工作项以间隔为 2 的方式访问局部存储器时,此时存在 2-路模块冲突,如图 8b 所示.不难看出,当工作项对局部存储器的访问间隔为奇数时不存在冲突;为偶数时则存在冲突,冲突的路数是访问间隔的数目.

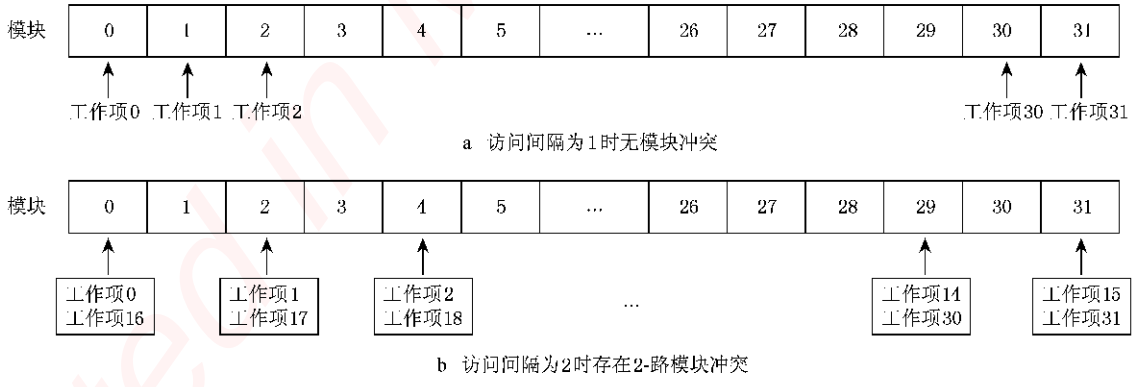

图 8　局部存储器访问冲突实例

定义 8. 如果 K 个工作项因间隔 K 步访问局部存储器而发生一个 K-路冲突,则称这 K 个等待串行访问的工作项为一个大小为 K 的冲突槽.

本文通过增加偏移量的方法改变同一工作组中不同工作项的映射方式,以避免访问局部存储器时的模块冲突.假设局部存储器的存储模块数为 N,冲突槽的大小为 K,冲突槽中 K 个工作项映射到局部存储器的模块索引为 idx_i^{INIT},增加偏移量后的模块索引为 idx_i^{AFTER},对该冲突槽中的每个工作项增加偏移量 $offset_i$,其中 $1 \leq i \leq K$,则有

$$offset_i = i \times K \times \frac{localID_i}{N} \tag{2}$$

$$idx_i^{\mathrm{AFTER}} = idx_i^{\mathrm{INIT}} + offset_i \tag{3}$$

例 2. 图 9 所示为一个通过增加偏移量避免 4-路

冲突的示例,其中 $N=32$. 局部 ID 为 0,8,16 和 24 的工作项同时访问局部存储器模块 0,构成了第一个冲突槽. 由式(2)可以计算出,该冲突槽中各工作项所需的偏移量分别为 0,1,2 和 3. 因此,由式(3)可知,加入偏移量后的存储器模块索引分别为 {0,0,0,0} + {0,1,2,3},即 {0,1,2,3},后续的冲突槽以此类推. 可以看出,偏移量的增加改变了工作项的映射索引,有效地避免了访问局部存储器时的模块冲突.

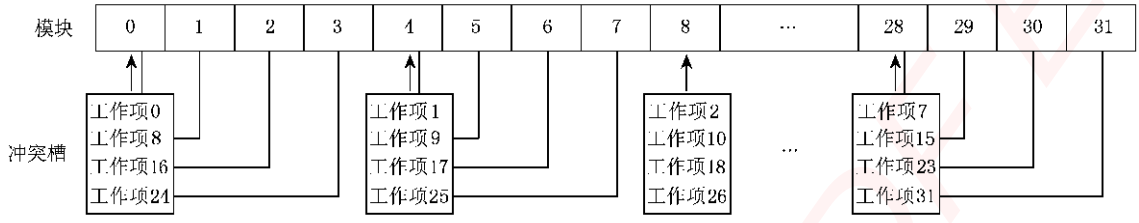

图 9　4-路冲突的避免

根据 OpenCL 存储模型中硬件的物理特性与源程序中数据访问的属性,对快速存储器区域进行分配和优化的过程如算法 3 所示.

算法 3. 快速存储器的分配与优化

输入. 循环体语句 S 中所有数组的访问矩阵集合 $f=\{f_1,\cdots,f_n\}$.

输出. 快速存储器区域 FMR 的分配方案.

```
FMR = NULL
TS = f - f'_VEC
for each f_i ∈ TS do
  if (f_i 存在数据重用)
    if (f_i 只读 && 存储访问的起始地址相同)
      FMR = FMR ∪ {常数存储器}
    endif
    if (f_i 只读 && 存储访问的起始地址不相同 ‖ f_i
    读取 && 写回 && 存储访问模式可预知)
      FMR = FMR ∪ {局部存储器}
    endif
    if (f_i 访问间隔为偶数)
      使用式(2)(3)增加偏移量以避免模块冲突
    endif
  endif
  if (f_i 存储访问模式不可预知)
    FMR = FMR ∪ {IMAGE}
    if (f_i 行数为 1)   //一维数组
      采用式(1)转换为二维坐标
    endif
  endif
endfor
```

4　实验及结果分析

为验证本文优化方法的有效性,我们使用 OpenCL 编程语言对以下 6 个程序进行了测试:Hydro 片段(取自 Livermore Kernel 1),ADI 积分(取自 Livermore Kernel 8),矩阵乘法 MM(取自科学计算核心程序),N-body 模拟(取自分子动力学程序),绝对差值和 SAD(取自图像处理程序),稀疏矩阵与向量乘法 SpMV(取自 NAS CG Benchmark). 每个测试程序各有 2 个不同的版本:未经优化的版本和采用本文方法优化后的版本. 实验软硬件环境配置如下:

1) AMD/ATI Radeon™ HD 5850 GPU(显存为 1 GB),AMD Phenom 9950 Quad-Core 处理器(系统内存为 3 GB);Windows XP 操作系统,AMD Stream SDK 2.1 版本.

2) NVIDIA GeForce 9600 GT(显存为 512 MB),Intel Core2 Quad CPU Q8200 处理器(系统内存为 1 GB);Windows XP 操作系统,NVIDIA OpenCL SDK 3.0 版本.

4.1　全局存储器优化

为验证全局存储器优化方法的有效性,我们选取 Hydro 片段、ADI 积分和 MM 作为测试程序. Hydro 片段由一个二重循环迭代多面体空间构成,访问 3 个一维的数据空间. 其中,对前 2 个数据空间的访问(数组 x 和数组 y)为正向访问模式,对第 3 个数据空间(数组 z)的访问为 2 次不同的偏移访问模式. 按照第 3.3 节中的方法,检测到该程序片段中的所有存储访问均可向量化;随后,按照第 3.4 节中的方法,对第 3 个数据空间的 2 次偏移访问模式通过 2 次常量移位的方法转化为正向访问模式,进而利用 float4 向量数据类型提高全局存储器带宽的利用率. MM 程序由一个三重循环迭代多面体空间构成,访问 3 个二维的数据空间. 其中,对 2 个数据空间的访问(数组 A 和数组 C)为正向访问模式,对一个数据空间(数组 B)的访问为按列访问模式. 因此,

对该数据空间采取行列转置的变换方法转化为正向访问模式. 但由于目标数据空间(待写回的数组 C)在循环体语句中存在数据依赖关系,因此MM 程序的存储访问实例不可全向量化,使用 float4 数据类型向量化数组 A 和数组 B 的存储访问实例. ADI积分程序由一个三重循环迭代多面体空间构成,访问 3 个一维的数据空间(数组 du_1, du_2, du_3)和 3 个三维的数据空间(数组 u_1, u_2, u_3). 其中,对 3 个一维的数据空间的访问都是正向访问模式,但对 3 个三维的数据空间分别存在 2 次不同形式的按列访问模式:第一次在列的方向上存在偏移访问模式(如 $u_1[n_{l1}][k_y+1][k_x]$和 $u_1[n_{l1}][k_y-1][k_x]$),第二次在列的方向上为正向访问模式(如 $u_1[n_{l1}][k_y][k_x]$). 因此,为全向量化所有的存储访问实例,需要对 3 个三维数据空间进行 2 次不同的变换. 为了变为正向访问模式,对它们实施转置(最低维和次低维交换). 然而对于转置后的数据空间,列方向上存在偏移的访问变为了行方向上的偏移访问,为此需要对其进一步实施常量移位,最终转化为正向访问模式. 值得注意的是,在 ADI 积分程序中,3 个三维数据空间虽然在循环体语句的两边同时出现,但并非访问同一存储单元. 按照本文的定义,该循环体语句并不存在数据相关. 因此,ADI 积分程序中所有的存储访问都是可向量化的. 3 个测试程序在使用 float4 向量数据类型优化前与优化后的执行时间与加速比如表 1 所示.

表 1　全局存储器优化前后的执行时间和加速比

		Hydro			MM			ADI		
		1024	2048	4096	1024	2048	4096	1024	2048	4096
优化前/ms	HD5850	19.3	25.8	61.9	36.7	120.0	685.2	262.4	749.5	2235.9
	9600GT	215.0	356.9	1067.2	281.8	1070.2	4892.7	2190.9	5848.4	12205.1
优化后/ms	HD5850	5.6	8.5	21.4	17.5	42.6	214.2	97.2	202.7	496.8
	9600GT	119.4	178.5	426.9	176.1	535.1	2575.1	995.9	2088.7	5306.6
加速比	HD5850	3.5	3.0	2.9	2.1	2.4	3.2	2.7	3.7	4.5
	9600GT	1.8	2.0	2.5	1.6	2.0	1.9	2.2	2.8	2.3

4.2　快速存储器分配与优化

为验证快速存储器分配与优化方法的有效性,本文选取 N-body 模拟、SAD 和 SpMV 作为测试程序. 其中测试程序优化前的版本仅使用全局存储器,优化后的版本采用了第 3.5 节中的快速存储器分配与优化方法.

N-body 模拟在天体物理学、分子动力学等众多领域有着广泛的应用,其核心程序片段由一个二重循环迭代多面体空间构成,访问 4 个一维的数据空间:粒子加速度数组 acc、粒子间距离数组 $dist$、粒子位置数组 pos 和粒子速度数组 vel. 其中,数组 acc 和数组 $dist$ 的大小为 3,并且在程序中既被读取又被写回. 在第 3.5 节中提到,共享存储器的使用会影响系统生成的工作项数目. 因此对于这 2 个规模极小的数组,本文采用 float3 向量数据类型保存它们以进行访问. 数组 pos 和数组 vel 在整个程序中存在数据共享,其中数组 pos 的访问属性为既被读取又被写回,而数组 vel 的访问属性为写回. 按照本文方法将数组 pos 分配至局部存储器,由于源程序中对数组 pos 是以间隔为 2 的方式进行访问,因此需要对访问该数组的工作项增加偏移量,以避免访问局部存储器时的模块冲突. 访问局部存储器中的数据时,必须使用 OpenCL 同步原语. 绝对差值和程序在图像处理、视频压缩等领域有着广泛的应用,例如用于检测图像块之间的相似性. 本文以 16 ×16 像素的图像块为例,其相似度计算公式为

$$SAD(U,V) = \sum_{x=0}^{15}\sum_{y=0}^{15} | U(x,y) - V(x,y) | .$$

该核心片段由一个四重循环迭代多面体空间构成,访问模板图像块数组 $template\_img$、查询图像块数组 $search\_img$ 和相似度数组 sad 3 个二维的数据空间. 其中,对于数组 $template\_img$ 的访问为可预知的只读访问,对于数组 $search\_img$ 的访问为不可预知的只读访问(随机访问). 按照本文方法,将数组 $template\_img$ 分配至常数存储器,并将数组 $search\_img$ 分配至 IMAGE 存储对象. 由于 $search\_img$ 是二维数组,因此不需要实施坐标转化. 稀疏矩阵与向量乘法作为基础的通用计算核心程序,在有限元建模、

EDA、机器学习等众多领域有着广泛的应用. 以CSR格式存储的稀疏矩阵为例,一个二维矩阵被压缩至一个一维数组中,全部非零元素记录在数组 *elem* 中,数组 *row ptr* 记录该矩阵每行第一个非零元素在数组 *elem* 中的索引,每个非零元素对应的列坐标记录在 *col* 数组中. 该程序片段由一个二重循环迭代多面体空间构成,访问5个一维的数据空间;其中,数组 *row ptr* 和数组 *p* 存在数据共享. 对数组 *row ptr* 的访问为只读,但每次访问的起始地址并不相同,因此将其分配到局部存储器中. 对数组 *p* 的访问为不可预知的只读访问(随机访问),故将其分配到IMAGE存储对象. 由于数组 *p* 是一维数组,因此需要对其实施坐标转化(实验中取 $W = 64$). 3个测试程序在使用快速存储器分配和优化前与优化后的执行时间与加速比如表2所示.

表2　快速存储器分配与优化前后的执行时间和加速比

		N-body			SAD			SpMV		
		1024	2048	4096	1024	2048	4096	1024	2048	4096
优化前/ms	HD5850	1251.5	1718.1	2676.4	578.2	1080.5	1623.8	79.5	213.9	344.2
	9600GT	7634.2	13588.1	36307.5	3184.7	7406.2	13252.0	640.2	1897.8	5346.9
优化后/ms	HD5850	368.1	429.5	514.7	137.7	180.1	242.4	20.9	46.5	71.7
	9600GT	1777.5	2516.3	6601.4	539.8	949.5	1577.6	246.2	632.6	1304.1
加速比	HD5850	3.4	4.0	5.2	4.2	6.0	6.7	3.8	4.6	4.8
	9600GT	4.3	5.4	5.5	5.9	7.8	8.4	2.6	3.0	4.1

5　结　　语

本文针对OpenCL模型提出GPU通用计算的性能优化方法,在建立源程序多面体模型的基础上,分别对全局存储器和快速存储器进行优化与分配. 对于全局存储器的优化,检测源程序的存储访问模式并采用图着色的方法以发掘可向量化的存储访问实例;利用数据空间变换对不规则的存储访问模式进行转换,进而使用向量数据类型提高片外存储器的带宽利用率. 对于快速存储器的分配,通过检测程序中的数据是否存在重用,并根据数据的访问属性和OpenCL存储模型的特性,实现快速存储器的有效分配. 此外,还采用坐标转换和增加偏移量的技术分别对IMAGE存储对象和局部存储器进行优化,进而提高片上存储器的使用效率. 在AMD/ATI Radeon™ HD 5850 GPU和NVIDIA GeForce 9600 GT GPU上采用6个测试程序对本文优化方法的有效性进行了验证,实验结果表明,采用本文优化方法可使测试程序的性能提高1.6～8.4倍.

通常,程序中的分支语句可能会引起一个工作组内多个工作项执行不同的分支路径. 当分支发生时,不同的执行路径以串行方式执行,严重影响程序的执行性能. 我们下一步将研究如何通过分支提取的方法,结合运行时工作项重定向技术,尽可能从源程序级和目标程序级消除分支语句的负面影响,进一步提高通用计算程序在GPU上的执行性能.

参考文献(References):

[1] Owens J D, Houston M, Luebke D, *et al*. GPU computing [J]. Proceedings of the IEEE, 2008, 96(5): 879-899

[2] Owens J D, Luebke D, Govindaraju N, *et al*. A survey of general-purpose computation on graphics hardware [J]. Computer Graphics Forum, 2007, 26(1): 80-113

[3] Wu Enhua, Liu Youquan. General purpose computation on GPU [J]. Journal of Computer-Aided Design & Computer Graphics, 2004, 16(5): 601-612 (in Chinese)
(吴恩华, 柳有权. 基于图形处理器(GPU)的通用计算[J]. 计算机辅助设计与图形学学报, 2004, 16(5): 601-612)

[4] Wen Chanjuan, Ou Jiawei, Jia Jinyuan. GPGPU-based smoothed particle hydrodynamic fluid simulation [J]. Journal of Computer-Aided Design & Computer Graphics, 2010, 22(3): 406-411 (in Chinese)
(温婵娟, 欧嘉蔚, 贾金原. GPU通用计算平台上的SPH流体模拟[J]. 计算机辅助设计与图形学学报, 2010, 22(3): 406-411)

[5] Ren Huamin, Zhang Yongdong, Lin Shouxun. GPU-accelerated video copy detection based on incremental clustering [J]. Journal of Computer-Aided Design & Computer Graphics, 2010, 22(3): 449-456 (in Chinese)
(任化敏, 张勇东, 林守勋. GPU加速的基于增量式聚类的视频拷贝检测方法[J]. 计算机辅助设计与图形学学报, 2010, 22(3): 449-456)

[6] Lin Jiang, Tang Min, Tong Ruofeng. GPU accelerated biological sequence alignment [J]. Journal of Computer-Aided Design & Computer Graphics, 2010, 22(3): 420-427 (in Chinese)
(林 江, 唐 敏, 童若锋. GPU 加速的生物序列比对[J]. 计算机辅助设计与图形学学报, 2010, 22(3): 420-427)

[7] Zhang Fan, Wang Zhangye, Yao Jian, *et al*. Accelerating the calculation of protein molecular field using GPU cluster [J]. Journal of Computer-Aided Design & Computer Graphics, 2010, 22(3): 412-419 (in Chinese)
(张 繁, 王章野, 姚 建, 等. 应用 GPU 集群加速计算蛋白质分子场[J]. 计算机辅助设计与图形学学报, 2010, 22(3): 412-419)

[8] Fatahalian K, Houston M. GPUs: a closer look [J]. ACM Queue, 2008, 6(2): 18-28

[9] Jang B, Mistry P, Schaa D, *et al*. Data transformations enabling loop vectorization on multithreaded data parallel architectures [C] //Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York: ACM Press, 2010: 353-354

[10] Liu Y X, Zhang E Z, Shen X P. A cross-input adaptive framework for GPU program optimizations [C] //Proceedings of IEEE International Symposium on Parallel & Distributed Processing. Los Alamitos: IEEE Computer Society Press, 2009: 1-10

[11] Ryoo S, Rodrigues C I, Stone S S, *et al*. Program optimization space pruning for a multithreaded GPU [C] //Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization. New York: ACM Press, 2008: 195-204

[12] Ryoo S, Rodrigues C I, Stone S S, *et al*. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA [C] //Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York: ACM Press, 2008: 73-82

[13] Jang B, Do S, Pien H, *et al*. Architecture-aware optimization targeting multithreaded stream computing [C] //Proceedings of the 2nd Workshop on General Purpose Processing on Graphics Processing Units. New York: ACM Press, 2009: 62-70

[14] Baskaran M M, Bondhugula U, Krishnamoorthy S, *et al*. A compiler framework for optimization of affine loop nests for GPGPUs [C] //Proceedings of the 22nd Annual International Conference on Supercomputing. New York: ACM Press, 2008: 225-234

[15] Lee S, Min S J, Eigenmann R. OpenMP to GPGPU: a compiler framework for automatic translation and optimization [C] //Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York: ACM Press, 2009: 101-110

[16] Han Bo, Zhou Bingfeng. A performance model for general-purpose computation on GPU [J]. Journal of Computer-Aided Design & Computer Graphics, 2009, 21(9): 1219-1226 (in Chinese)
(韩 博, 周秉锋. GPGPU 性能模型及应用实例分析[J]. 计算机辅助设计与图形学学报, 2009, 21(9): 1219-1226)

[17] Pouchet L N, Bastoul C, Cohen A, *et al*. Iterative optimization in the polyhedral model: Part I, one-dimensional time [C] //Proceedings of the International Symposium on Code Generation and Optimization. Los Alamitos: IEEE Computer Society Press, 2007: 144-156

[18] Pouchet L N, Bastoul C, Cohen A, *et al*. Iterative optimization in the polyhedral model: Part II, multidimensional time [C] //Proceedings of the ACM Conference on Programming Language Design and Implementation. New York: ACM Press, 2008: 1-11

[19] Bastoul C. Code generation in the polyhedral model is easier than you think [C] //Proceedings of the 13th International Conference on Parallel Architecture and Compilation Techniques. Los Alamitos: IEEE Computer Society Press, 2004: 7-16

[20] O'Boyle M F P, Knijnenburg P M W. Non-singular data transformations: definition, validity and applications [C] //Proceedings of the 11th International Conference on Supercomputing. New York: ACM Press, 1997: 309-316

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用
33. 高可用的磁盘阵列 Cache 的设计和实现
34. RAID 阵列中高速 Cache 管理的优化

35. 一种新的基于 RAID 的 CACHE 技术研究与实现
36. 基于 PCIE-104 总线的高速数据接口设计
37. 基于 VPX 标准的 RapidIO 交换和 Flash 存储模块设计
38. 北斗卫星系统在海洋工程中的应用
39. 北斗卫星系统在远洋船舶上应用的研究
40. 基于 CPCI 总线的红外实时信号处理系统
41. 硬件实现 RAID 与软件实现 RAID 的比较
42. 基于 PCI Express 总线系统的热插拔设计
43. 基于 RAID5 的磁盘阵列 Cache 的研究与实现
44. 基于 PCI 总线的 MPEG2 码流播放卡驱动程序开发
45. 基于磁盘异或引擎的 RAID5 小写性能优化
46. 基于 IEEE1588 的时钟同步技术研究
47. 基于 Davinci 平台的 SD 卡读写优化
48. 基于 PCI 总线的图像处理及传输系统的设计
49. 串口和以太网通信技术在油液在线监测系统中的应用
50. USB30 数据传输协议分析及实现
51. IEEE 1588 协议在工业以太网中的实现
52. 基于 USB30 的设备自定义请求实现方法
53. IEEE1588 协议在网络测控系统中的应用
54. USB30 物理层中弹性缓冲的设计与实现
55. USB30 的高速信息传输瓶颈研究
56. 基于 IPv6 的 UDP 通信的实现
57. 一种基于 IPv6 的流媒体传送方案研究与实现
58. 基于 IPv4-IPv6 双栈的 MODBUS-TCP 协议实现

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理

13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发
23. Vxworks 下的冗余 CAN 通讯模块设计
24. WindML 工业平台下开发 S1d13506 驱动及显示功能的实现
25. WindML 中 Mesa 的应用
26. VxWorks 下图形用户界面开发中双缓冲技术应用
27. VxWorks 上的一种 GUI 系统的设计与实现
28. VxWorks 环境下 socket 的实现
29. VxWorks 的 WindML 图形界面程序的框架分析
30. VxWorks 实时操作系统及其在 PC104 下以太网编程的应用
31. 实时操作系统任务调度策略的研究与设计
32. 军事指挥系统中 VxWorks 下汉字显示技术
33. 基于 VxWorks 实时控制系统中文交互界面开发平台
34. 基于 VxWorks 操作系统的 WindML 图形操控界面实现方法
35. 基于 GPU FPGA 芯片原型的 VxWorks 下驱动软件开发
36. VxWorks 下的多串口卡设计
37. VxWorks 内存管理机制的研究
38. T9 输入法在 Tilcon 下的实现
39. 基于 VxWorks 的 WindML 图形界面开发方法

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译

10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通
12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现
24. 掌握 Android 7.0 新增特性 Quick Settings
25. Android 应用逆向分析方法研究
26. Android 操作系统的课程教学
27. Android 智能手机操作系统的研究
28. Android 英文朗读功能的实现
29. 基于 Yocto 订制嵌入式 Linux 发行版
30. 基于嵌入式 Linux 的网络设备驱动设计与实现
31. 如何高效学习嵌入式
32. 基于 Android 平台的 GPS 定位系统的设计与实现
33. LINUX ARM 下的 USB 驱动开发
34. Linux 下基于 I2C 协议的 RTC 驱动开发
35. 嵌入式下 Linux 系统设备驱动程序的开发
36. 基于嵌入式 Linux 的 SD 卡驱动程序的设计与实现
37. Linux 系统中进程调度策略
38. 嵌入式 Linux 实时性方法
39. 基于实时 Linux 计算机联锁系统实时性分析与改进
40. 基于嵌入式 Linux 下的 USB30 驱动程序开发方法研究
41. Android 手机应用开发之音乐资源播放器
42. Linux 下以太网的 IPv6 隧道技术的实现
43. Research and design of mobile learning platform based on Android
44. 基于 linux 和 Qt 的串口通信调试器调的设计及应用
45. 在 Linux 平台上基于 QT 的动态图像采集系统的设计
46. 基于 Android 平台的医护查房系统的研究与设计
47. 基于 Android 平台的软件自动化监控工具的设计开发
48. 基于 Android 的视频软硬解码及渲染的对比研究与实现

Windows CE:

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究
4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究
17. Windows CE 环境下无线网卡的自动安装
18. 基于 Windows CE 的可视电话的研究与实现
19. 基于 WinCE 的嵌入式图像采集系统设计
20. 基于 ARM 与 WinCE 的掌纹鉴别系统
21. DCOM 协议在网络冗余环境下的应用
22. Windows XP Embedded 在变电站通信管理机中的应用
23. XPE 在多功能显控台上的开发与应用
24. 基于 Windows XP Embedded 的 LKJ2000 仿真系统设计与实现
25. 虚拟仪器的 Windows XP Embedded 操作系统开发

PowerPC:

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置

8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植
12. 基于 PowerPC 双核处理器嵌入式系统 UBoot 移植
13. 基于 PowerPC 的雷达通用处理机设计
14. PowerPC 平台引导加载程序的移植
15. 基于 PowerPC 嵌入式内核的多串口通信扩展设计
16. 基于 PowerPC 的多网口系统抗干扰设计
17. 基于 MPC860T 与 VxWorks 的图形界面设计
18. 基于 MPC8260 处理器的 PPMC 系统
19. 基于 PowerPC 的控制器研究与设计
20. 基于 PowerPC 的模拟量输入接口扩展
21. 基于 PowerPC 的车载通信系统设计
22. 基于 PowerPC 的嵌入式系统中通用 IO 口的扩展方法
23. 基于 PowerPC440GP 型微控制器的嵌入式系统设计与研究

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现
16. 基于 ARM 的嵌入式 Linux 无线网卡设备驱动设计
17. ARM S3C2440 Linux ADC 驱动
18. ARM S3C2440 Linux 触摸屏驱动
19. Linux 和 Cortex-A8 的视频处理及数字微波传输系统设计

20. Nand Flash 启动模式下的 Uboot 移植
21. 基于 ARM 处理器的 UART 设计
22. ARM CortexM3 处理器故障的分析与处理
23. ARM 微处理器启动和调试浅析
24. 基于 ARM 系统下映像文件的执行与中断运行机制的实现
25. 中断调用方式的 ARM 二次开发接口设计
26. ARM11 嵌入式系统 Linux 下 LCD 的驱动设计
27. Uboot 在 S3C2440 上的移植
28. 基于 ARM11 的嵌入式无线视频终端的设计
29. 基于 S3C6410 的 Uboot 分析与移植
30. 基于 ARM 嵌入式系统的高保真无损音乐播放器设计
31. UBoot 在 Mini6410 上的移植
32. 基于 ARM11 的嵌入式 Linux NAND FLASH 模拟 U 盘挂载分析与实现

Hardware：

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究
13. 基于 UEFI 固件的恶意代码防范技术研究
14. MIPS 架构计算机平台的支持固件研究
15. 基于 UEFI 固件的攻击验证技术研究
16. 基于 UEFI 的 Application 和 Driver 的分析与开发
17. 基于 UEFI 的可信 BIOS 研究与实现
18. 基于 UEFI 的国产计算机平台 BIOS 研究
19. 基于 UEFI 的安全模块设计分析
20. 基于 FPGA NiosⅡ的等精度频率计设计
21. 基于 FPGA 的 SOPC 设计
22. 基于 SOPC 基本信号产生器的设计与实现
23. 基于龙芯平台的 PMON 研究与开发

24. 基于 X86 平台的嵌入式 BIOS 可配置设计
25. 基于龙芯 2F 架构的 PMON 分析与优化
26. CPU 与 GPU 之间接口电路的设计与实现
27. 基于龙芯 1A 平台的 PMON 源码编译和启动分析
28. 基于 PC104 工控机的嵌入式直流监控装置的设计
29. GPGPU 技术研究与发展
30. GPU 实现的高速 FIR 数字滤波算法
31. 一种基于 CPUGPU 异构计算的混合编程模型

Programming:

1. 计算机软件基础数据结构 - 算法
2. 高级数据结构对算法的优化
3. 零基础学算法
4. Linux 环境下基于 TCP 的 Socket 编程浅析
5. Linux 环境下基于 UDP 的 socket 编程浅析
6. 基于 Socket 的网络编程技术及其实现
7. 数据结构考题 - 第 1 章 绪论
8. 数据结构考题 - 第 2 章 线性表
9. 数据结构考题 - 第 2 章 线性表 - 答案