HF 是水平滤波后的结果,VF 是垂直滤波后的图像,Down 是下采样后的图像,Up 是上采样计算后的图像,Dst 是完成差值计算后的图像,*Src*(*itx*,*ity*)表示图像 Src 中横坐标为 *ity*,纵坐标为 *itx* 的像素点.

1) 水平滤波:对于目标图像 HF 中的每个像素点,需要用到源图像 Src 上以对应像素点为中心的长度为 5 的一个向量 *SrcHVect*,然后再计算出 *SrcHVect* 与滤波算子 *HFilter* 的卷积值 *HVal*,该卷积值即为图像 HF 中该像素点的值.

2) 垂直滤波:对于目标图像 VF 中的每个像素点,需要用到图像 HF 中以对应像素点为中心的长度为 5 的一个向量 *HFVVect*,然后求 *HFVVect* 与滤波算子 *VFilter* 的卷积值 *VVal*,即为图像 VF 中该像素点的值.

3) 下采样:将图像 VF 的长宽缩小到源图像的 1/2,在计算目标图像的过程中,目标图像中对应像素点 *Down*(*itx*,*ity*)的值分别是源图像中坐标为 *VF*(2×*itx*,2×*ity*)的值.

4) 上采样:将图像 Down 的长宽分别扩大到原来大小的 2 倍,这时候得到的目标图像 Up 的长宽与源图像 Src 一致.

5) 求差值:用源图像 Src 中每个坐标对应的值减去图像 Up 中对应坐标的值,作为目标图像在该坐标上的值.

在拉普拉斯金子塔算法中,存在点操作和局部块操作两种类型,其中,步骤 1~步骤 4 是局部块操作,需要进行边界处理,步骤 5 是点操作.

3.1.2 图像锐化算法

如图 11 所示,图像锐化算法主要分为 4 个步骤:下采样、上采样、图像差值和锐化操作,其中,锐化操作本身又分为 3 个子步骤.

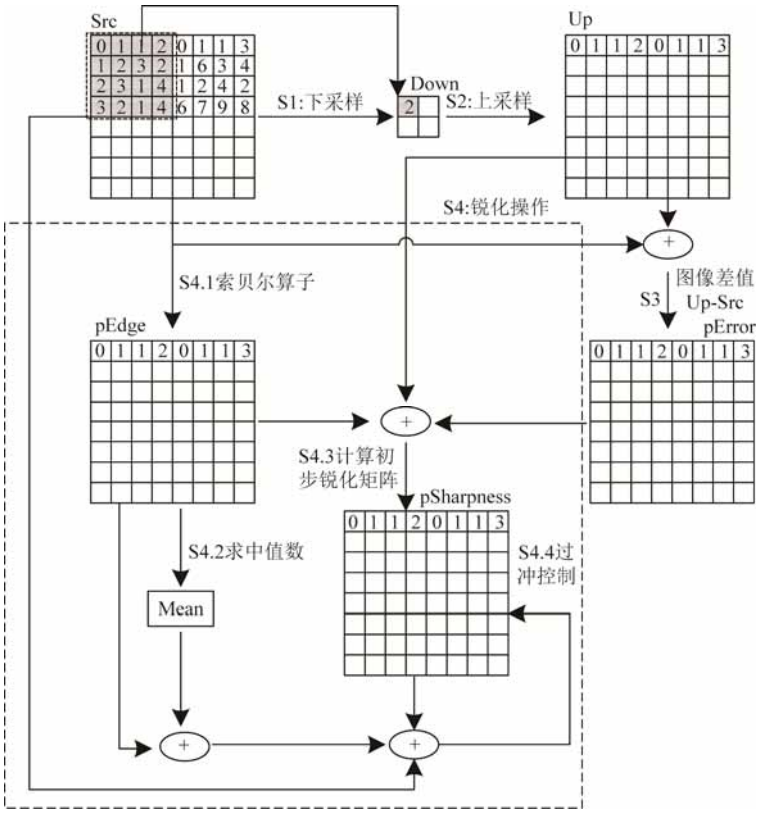

Fig.11 The work flow of sharpness

图 11 图像锐化算法的流程图

1) 下采样:目的是为了降低图像的大小,目标图像 Down 的长宽分别是源图像 Src 的 1/4.例如,图像 Down

中左上角灰色像素点的值是源图像 Src 中左上角相邻的 16 个灰色点的均值(按照四舍五入原则取整数).

2) 上采样:将下采样后的图像 Down 再通过上采样恢复到源图像大小,该过程的输出图像称为 Up.上采样在实现过程中分为 3 个部分,首先是图像 Up 的行边界;其次是计算列边界;最后是计算图像内部的像素点.

3) 图像差值:计算图像 Up 与源图像 Src 之间的差值矩阵 *pError*,该差值矩阵将会用于后续锐化操作中的索贝尔算子.

4) 锐化操作:该计算过程是包含多个子算法过程的复杂操作序列,包括索贝尔算子、求中值数操作、计算初步锐化矩阵和过冲控制操作.

a) 索贝尔算子:用于计算源图像矩阵亮度值的一阶梯度,完整的索贝尔算子包括两个步骤,分别是计算边界和图像内部像素点,索贝尔算子本质上是在源图像进行窗口为 3×3 大小的卷积计算,该算子的输出是 *pEdge* 图像矩阵.

b) 求中值数操作:其输入为 *pEdge* 矩阵,该计算过程就是计算图像上所有像素点的和,然后再求其平均值,主要是归约计算.

c) 计算初步锐化矩阵:利用步骤 b)计算到的中值数再结合其他用户参数,计算出图像亮度的强度值.然后,再用该强度值调整 *pEdge* 矩阵值来控制最终锐化后图像的锐化等级.最后,根据 *pEdge* 矩阵、*pError* 矩阵和上采样后的矩阵 *Up* 进行计算,得到初步的锐化后矩阵.

d) 过冲控制操作:该操作用来去掉不需要的效果,例如放大后的噪声.该过程以初步锐化后的矩阵作为输入,以最终计算得到锐化后的矩阵 *pSharpness* 作为输出.

3.1.3 图像模糊算法

图像模糊算法主要分为两个步骤,分别在水平和垂直方向上进行连续的 stencil 计算,在计算过程中分别用形状为 3×1 和 1×3 的核心在水平方向和垂直方向上进行计算.

3.1.4 反锐化掩膜算法

如图 12 所示,反锐化掩膜算法共分为 5 个步骤,分别是生成灰色图、水平方向上的模糊处理、垂直方向上的模糊处理、图像差值和图像锐化操作.

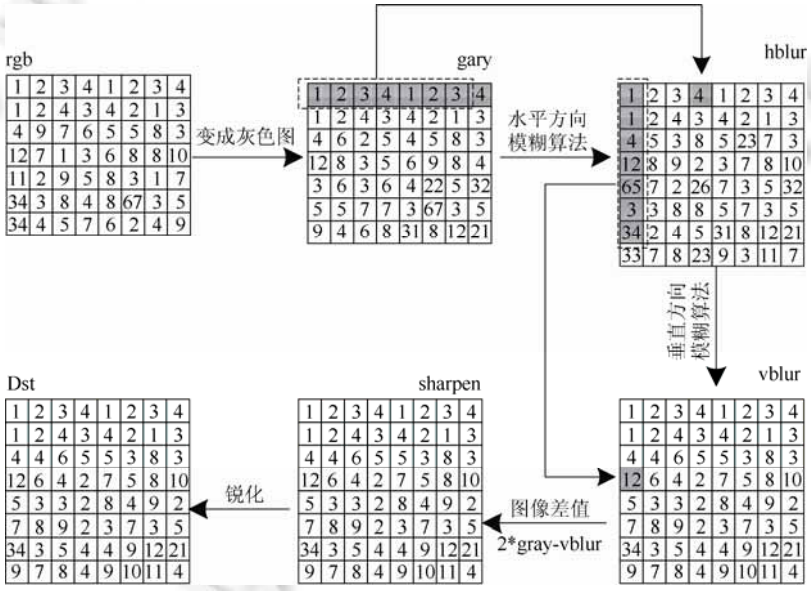

Fig.12 The workflow of unsharp mask algorithm

图 12 反锐化掩膜算法流程图

1) 获得灰色图:将原图像从 RGB 格式转换为灰色图格式,并在边界按照重复图像边缘模式对原图像进行

填充处理,获得灰色图像 gray.

2) 水平方向模糊处理:对于图像 gray,对每行上的像素进行长度为 7 的 stencil 计算,以求得模糊后图像 hblur.

3) 垂直方向模糊处理:对于图像 hblur,对每列上的像素进行长度为 7 的 stencil 计算,以求得模糊后图像 vblur.

4) 计算图像差值:对于模糊处理后的图像 vblur,与原图像进行差值计算得到图像 sharpen,该过程过滤掉低频部分,得到原图像的高频部分.

5) 图像锐化操作:利用图像 sharpen 和 gray 计算出修正因子 ratio,然后再将该修正因子与原图像相乘得到锐化后的图像 Dst.

3.2 应用程序优化分析

点算法:拉普拉斯金字塔算法中的步骤 5 和图像锐化算法中的步骤 3 都属于点算法.每个并行任务只需要源图像的单个像素点,同时,相邻并行任务间不存在数据重用,所以,ParaC 编译器未对这类算法进行优化.

局部块算法:滤波算法、采样算法和索贝尔算子都属于局部块算法.首先,单个任务需要访问多个数据,因此,在任务内采用向量化访存可能会提升访存效率.对于上采样或者下采样算法,存在二维数据重用,所以,并发任务可以在两个维度上同时读取数据,此时,为了保证每行的数据对齐,需要进行自动填充优化以保证每行上的数据都是对齐的.图 13 给出了自动填充优化以及向量化访存带来的程序加速,获得了相对于优化前 2.6%~57.2%的性能收益.

其次,不同并发任务间存在数据重用,向量化并行能够减少所有任务的总访存数量,从而降低带宽需求.图 14 展示了向量化并行对程序的加速情况,获得了相对于优化前程序执行时间 9.5%~22.3%的性能收益.另外,在索贝尔算子中,编译器还进行了循环展开优化,该优化相对于源程序能够减少 24.6%的时间开销.

全局算法:如图 11 所示,图像锐化算法步骤 S4.2 所表示的求中值数操作属于全局算法,在该算法中,ParaC 编译器在代码生成过程中采用了树形归约优化,增加了程序的并行性,同时在代码生成过程中,利用硬件平台特征消除部分同步操作.

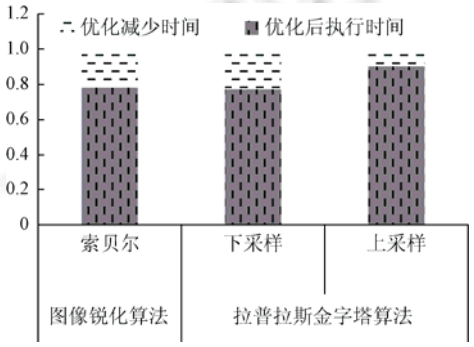

Fig.13 The performance benefit of vectorized access after automatic padding

图 13 自动填充后向量化访存优化获取的性能收益

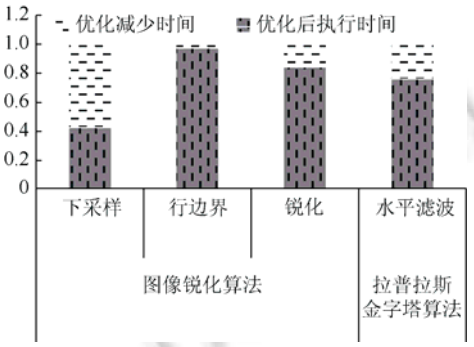

Fig.14 The performance benefit of vector parallelization

图 14 向量并行化优化后的性能收益

3.3 性能与产能分析

3.3.1 ParaC 与人工优化的效率对比

图 15 所示的实验结果表明,ParaC 编译器自动生成的优化程序 OpenCL 程序在不同的输入规模上都具有较好的扩展性.对于图像锐化中的下采样算法模块,ParaC 版本相对于专家手工优化版本在输入规模大于 1 024 时,加速比达到了 18 倍以上,通过分析两个版本的 OpenCL 核心函数发现,手工优化版本使用了 *ALLOC_HOST_PTR* 方式为变量 *pad_yPlane0Buffer* 分配内存空间,导致核心函数访问该变量的开销有所增加.当将其修改为 *CL_MEM_READ_WRITE* 模式时,ParaC 的加速比分别为在输入为 1024 和 2048 时的 1.086 和 2.535.

在优化拉普拉斯算法中的下采样垂直滤波模块时,ParaC 编译器通过搜索比较不同优化方案的性能,最终

采用了水平方向上的向量化并行优化方案,相较于人工优化版本采用的垂直方向上的向量化并行,前者具有更好的访存效率,而且随着输入规模的扩大,其性能优势更加明显,最高获得了相对于后者 1.76 倍的加速比.

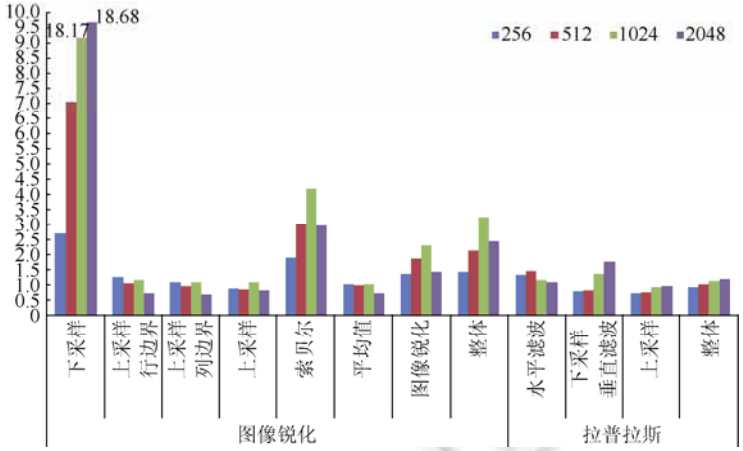

Fig.15 The scalability under different input size of ParaC

图 15 分析 ParaC 算法在不同规模输入下的性能

图 15 中横轴表示不同的算法模块及整体执行时间,纵坐标轴表示各算法核心函数的 ParaC 版本相对于专家优化版本的加速比(专家优化版本的执行时间/ParaC 版本的执行时间),其中,整体执行时间是指两个程序的所有算法模块核心函数执行时间之和.作为基准版本的 OpenCL 程序均由专家手工优化获得,其中图像锐化算法手工优化版本来自于公开发表的文献[13],拉普拉斯算法同样由文献[13]的作者通过手工优化获得,且两者的性能作为合作项目的输出成果都得到某业内领先通信与移动设备制造公司的认可.

对于图像锐化算法中的上采样行边界、上采样列边界和拉普拉斯算法中的上采样算法中存在很多的控制流分支,手工优化版本对这些分支作了更细的优化,例如通过合并分支条件消除不必要的分支,所以性能略好于 ParaC 自动生成版本.

通过搜索算法优化任务的组织形式,ParaC 版本的代码在拉普拉斯的水平滤波、上采样和图像锐化算法中的上采样列边界模块有较大的性能收益,相对于默认的任务组织形式,优化后的版本分别获取 8.51%、4.34%和 20.24%的加速比.

从代码行数角度来评估 ParaC 的产能,表 1 中的数据是拉普拉斯金子塔算法和图像锐化算法的 ParaC 版本与专家优化版本在 GPGPU 平台上的代码行数,从中看到,专家优化版本的代码行数是 ParaC 版本的 3.44 倍~82 倍.这是因为,ParaC 提供了针对领域特征的算子运算符,例如 *parac_matrix_sum* 运算,手工优化版本的代码行数会是 ParaC 的近百倍.另外,ParaC 编程环境能够自动制定程序的优化策略、完成代码变换和输出标准的 OpenCL 程序,所以相较于手工开发优化的 OpenCL 程序,ParaC 能够显著提高用户的开发效率.

Table 1 Comparing the number of code lines between ParaC and OpenCL on Laplacian and sharpness

表 1 比较相应算法模块的 ParaC 版本与专家优化版本的代码行数

	拉普拉斯算法			图像锐化算法						
	水平滤波	下采样与垂直滤波	上采样	下采样	上采样列边界	上采样行边界	上采样	索贝尔	平均值	锐化
专家优化	106	199	110	75	68	72	73	93	82	149
ParaC	16	11	32	5	28	15	5	8	1	34
专家优化/ParaC	6.63	18.09	3.44	15	2.52	4.8	14.6	11.63	82	4.38

为了分析 ParaC 优化算法在不同厂商硬件平台上的适用性,在 NVIDIA GPU 平台上评估了 ParaC 版本和手工优化版本的性能是,实验数据如图 16 所示.从实验结果来看,在部分热点函数上手工优化版本的性能优于 ParaC 版本;但是对于整体性能而言,在多数情况下,ParaC 版本略优于手工优化版本.

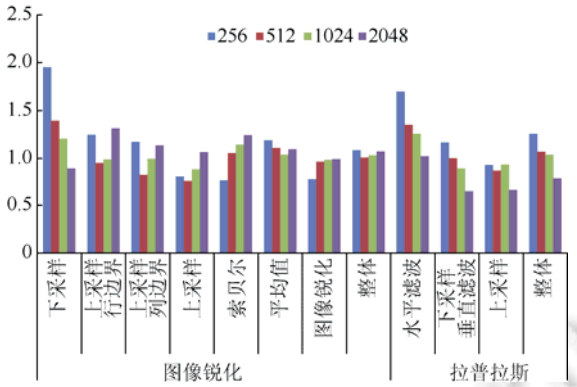

Fig.16 Comparing the performance between ParaC and manual optimization version on NVIDIA K40m GPU platform

图 16 比较 ParaC 版本和手工优化版本在英伟达 GPU 平台上的性能

图 16 中,横轴表示不同的算法模块及整体执行时间,纵坐标轴表示各算法核心函数的 ParaC 版本相对于专家优化版本的加速比(专家优化版本的执行时间/ParaC 版本的执行时间),其中,整体执行时间是指两个程序的所有算法模块核心函数执行时间之和.

3.3.2 ParaC 与 Halide 系统的对比

从代码行数来评估对比 ParaC 与 Halide 的产能,见表 2,其中,OpenCL 版本为由 ParaC 编译器自动生成的 OpenCL 代码,ParaC 版本只包含其核心算法代码行数,Halide 版本代码除核心算法外还包含其对应的调度策略代码.ParaC 和 Halide 都是针对图像领域的高层抽象语言,所以在表达核心算法本身的能力方面,两者非常接近,在反锐化掩膜应用程序中,Halide 版本由于需要设置较为复杂的调度策略,所以其代码行数略高于 ParaC.

Table 2 Comparing the number of code lines between ParaC and Halide, OpenCL on blur and unsharp mask applications

表 2 比较相应算法模块的 ParaC 版本与 Halide 版本、OpenCL 版本的代码行数

	图像模糊应用程序			反锐化掩膜应用程序	
	水平 stencil	垂直 stencil	水平模糊算法	垂直模糊算法	图像锐化
OpenCL	149	206	157	146	166
ParaC	3	3	3	3	8
Halide	3	3	3	3	9
Halide/ParaC	1	1	1	1	1.125

图 17 所示的实验结果表明,在当前的测试用例中,ParaC 获得了相对于 Halide 较优的性能.在图像模糊算法中,Halide 采用了两种优化调度设置,第 2 种调度策略是由 Halide 作者建议的最佳优化方案,是在第 1 种方案的基础上做了内核函数合并,最终只有 1 个内核函数.实验结果表明,ParaC 相对于第 1 种调度策略的 Halide 版本有大于 1.49 倍的加速比,其原因是 ParaC 除了采用与 Halide 相似的优化方案,还针对程序具有数据重用的特征,根据优化模型完成了对内核函数的向量化并行,所以性能优于 Halide.Halide 的第 2 个版本通过内核函数合并提高了性能,但是 ParaC 仍然具有 1.22 倍的加速比.在反锐化掩膜算法中,Halide 调度策略进行了激进的内核函数合并优化,将所有核心计算合并成统一的内核,该优化能够显著减少全局存储空间访问次数,其性能优于 ParaC 版本.综合两个应用程序来看,ParaC

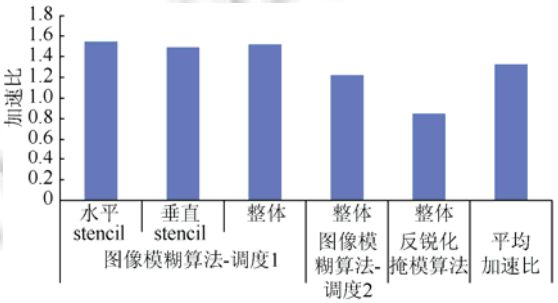

Fig.17 The speedup of ParaC to Halide

图 17 ParaC 版本相对于 Halide 的加速比

相对于 Halide 系统,在当前的测试用例下获得了较优的性能.

4 相关工作

通用编程语言:CUDA[2]和 OpenCL[1]都是支持 GPGPU 加速器平台的通用编程语言,前者只支持英伟达公司的硬件平台,后者是开放的具有平台可移植性的异构平台编程语言规范.CUDA 和 OpenCL 都属于底层编程语言,支持用户针对具体的硬件特征进行编程,其上的优化变换也完全由程序员负责,所以具有很高的编程复杂度且门槛较高.OpenACC[4]和 OpenMP[3]都是支持 GPGPU 平台的基于 pragma 的通用编程语言,能够支持 GPGPU 和 SIMD 等加速部件,两者都能够根据用户的制导自动完成相关的优化变换和代码生成,减轻了程序员的编程负担,但是仍然依赖于程序员给出具体的优化策略.

领域编程语言:领域编程语言包含了领域应用的高层抽象特征,使应用算法的描述比较简洁,同时能够隐藏底层硬件特征,同一份代码可以运行在不同的异构平台.因此,一些针对图像算法的领域编程语言被提出来用于降低 GPGPU 等异构加速平台的编程复杂度.Gordon 等人[14]提出一个支持 CUDA 平台的针对视觉特效的领域编程模型,该模型利用程序员提供的 Indexer Metadata 信息在编译时和运行时进行优化以获取尽可能高的性能.Howes[15]提出了支持 Cell 平台的图像领域编程方法,该方法利用程序员提供的数据访问信息和迭代执行顺序自动地完成数据访问的优化实现.Halide[5]从计算特点的角度将图像处理算法看成是流计算和 stencil 计算的综合,任务图像处理应用程序的优化过程需要处理并行度、局部性和冗余计算之间的冲突,是一个复杂困难的过程.为了简化编程,Halide 提出了将算法描述和调度分离的编程框架,用户给出调度空间的描述,编译器通过机器学习方法自动地搜索出最佳的实现方案.但是,Halide 在优化过程中未考虑领域算法特点,而且将边界处理当成独立的函数作用在每个像素点上,会带来额外的开销.HIPA^{cc}[8]根据图像处理领域的算法特征基于 C++的类定义了一组 DSL 组件,用来描述图像的存储类型、数据访问操作、边界处理、滑动窗口以及操作符,通过这些组件,程序员能够非常简洁地写出基于 HIPA^{cc} 的图像处理程序.但是,HIPA^{cc} 的边界处理不能描述多个点计算一个的过程,例如 Imag[−1][0]=(Imag[0][0]+Imag[1][0])的情况.另外,HIPA^{cc} 在进行高层抽象时,只从算法角度进行考虑,未考虑对编译优化的影响.

流程序的编译优化:在图像的实际处理过程中,同一个图像会经过不同的算法处理阶段,所以图像处理程序属于流式应用程序的一种.StreamIT[14,16]只考虑了 1D 的流处理,而在图像处理过程中,通常是至少为 2D 的流.另外,在实现过程中未针对冗余计算进行优化.Li[17]利用编译指导的方式进行共享内存优化.Li[18]设计了两层任务调度来实现非规则算法在 GPU 加速器平台上的优化.

5 结 论

本文中,我们引入了面向图像处理的领域编程语言 ParaC,并设计和实现了支持 ParaC 的编译器环境,该编译器利用先验知识驱动的编译优化机制,结合软件的程序特征和硬件特征,生成高性能的 OpenCL 程序.ParaC 语言提供了对典型图像操作的高层简化编程接口,例如 stencil 计算、滤波操作和归约操作等;对于其他一般性的算法,ParaC 也提供了迭代器等语言扩展支持程序开发.

在本文所评估应用程序上的实验结果表明,在性能方面,ParaC 编译器自动生成 OpenCL 程序,在性能上超过了由专家手工优化得到的对应程序,与 Halide 系统相比,在总体平均性能上也有一定的优势.同时,在不同输入规模上的实验结果说明,ParaC 编译器自动生成的 GPGPU 程序具有可扩展性.另外,通过对比 ParaC 程序和手工优化版本程序、Halide 版本程序的代码行数和程序员承担的开发责任,也表明 ParaC 具有较高的产能.

ParaC 的优点是通过在语言扩展中引入的高层语言抽象和基于先验知识驱动的优化模型能够在生成高性能代码的同时提高异构平台上的开发效率.其不足和未来需要进一步开展的工作是,进一步加强编译优化能力,例如支持更好的核心函数合并优化;在更多的应用程序和硬件平台上评估分析 ParaC 的能力.

References:

[1] Khronos OpenCL Working Group. The OpenCL specification. Version 1.2, 2012.

[2] NVIDIA. CUDA toolkit documentation. Version 8.0.61, 2017.

[3] OpenMP ARB. OpenMP application program interface. Version 4.0, 2013.

[4] OpenACC.Org. The OpenACC application programming interface. Version 2.5, 2015.

[5] Jonathan RK, Connelly B, Andrew A, Sylvain P, Frédo D, Saman A. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In: Proc. of the 34th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI 2013). New York: ACM, 2013. 519−530. http://dx.doi.org/10.1145/2491956.2462176

[6] Jonathan RK, Andrew A, Sylvain P, Marc L, Saman A, Frédo D. Decoupling algorithms from schedules for easy optimization of image processing pipelines. ACM Trans. on Graphics (TOG), 2012,31(4):1−12. [doi: 10.1145/2185520.2185528]

[7] Membarth R, Hannig F, Teich J, Körner M, Eckert W. Generating device-specific GPU code for local operators in medical imaging. In: Proc. of the 26th IEEE Int'l Conf. on Parallel & Distributed Processing Symposium (IPDPS). Shanghai, 2012. 569−581. [doi: 10.1109/IPDPS.2012.59]

[8] Membarth R, Reiche O, Hannig F, Teich J, Körner M, Eckert W. HIPA^{cc}: A domain-specific language and compiler for image processing. IEEE Trans. on Parallel and Distributed Systems, 2016,27(1):210−224. [doi: 10.1109/TPDS.2015.2394802]

[9] Bankman IN. Handbook of Medical Image Processing and Analysis. 2th ed., New York: Academic Press, 2008. 3−18.

[10] Russ JC. The Image Processing Handbook. 6th ed., Boca Raton: CRC Press, 2006. 288−355.

[11] Klette R, Zamperoni P. Handbook of Image Processing Operators, Vol. 1. Hoboken: Wiley, 1996. 38−42.

[12] Clang: A C language family frontend for LLVM. http://clang.llvm.org

[13] Fan M, Jia H, Zhang Y, An X, Cao T. Optimizing image sharpening algorithm on GPU. In: Proc. of the 44th Int'l Conf. on Parallel Processing (ICPP). Beijing, 2015. 230−239. [doi: 10.1109/ICPP.2015.32]

[14] Gordon MI, Thies W, Karczmarek M, Lin J, Meli AS, Lamb AA, Leger C, Wong J, Hoffmann H, Maze D, Amarasinghe S. A stream compiler for communication-exposed architectures. ACM SIGPLAN Notices, 2002,37(10):291−303. http://dx.doi.org/10.1145/605432.605428

[15] Howes L, Lokhmotov A, Donaldson A, Kelly PHJ. Deriving efficient data movement from decoupled access/execute specifications. In: Proc. of the 4th Int'l Conf. High-Perform. Embedded Archit. Compilers. 2009. 168−182. [doi: 10.1007/978-3-540-92990-1_14]

[16] Thies W, Karczmarek M, Amarasinghe S. StreamIt: A language for streaming applications. In: Compiler Construction. Berlin, Heidelberg: Springer-Verlag, 2002. [doi: 10.1007/3-540-45937-5_14]

[17] Li J, Liu L, Wu Y, Liu XH, Gao Y, Feng XB, Wu CY. Pragma directed shared memory centric optimizations on GPUs. Journal of Computer Science and Technology, 2016,31:235. [doi: 10.1007/s11390-016-1624-8]

[18] Li J, Liu L, Wu Y, Feng XB, Wu CY. Two-Level task scheduling for irregular applications on GPU platform. Int'l Journal of Parallel Programming, 2015. [doi: 10.1007/s10766-015-0387-0]

卢兴敬(1983－),男,山东临沂人,博士生,工程师,CCF 专业会员,主要研究领域为并行编程,程序并行优化及分析.

刘雷(1980－),男,博士,助理研究员,CCF 专业会员,主要研究领域为编程语言,编译优化.

贾海鹏(1983－),男,博士,助理研究员,CCF 专业会员,主要研究领域为高性能计算,面向多核/众核的编程方法与优化技术.

冯晓兵(1969－),男,博士,研究员,博士生导师,CCF 杰出会员,主要研究领域为编程模型,编译优化.

武成岗(1969－),男,博士,正高级工程师,博士生导师,CCF 高级会员,主要研究领域为计算机系统结构,编译技术.