# 面向神威·太湖之光的国产异构众核处理器 OpenCL 编译系统

伍明川[1),2)] 黄磊[1)] 刘颖[1)] 何先波[3)] 冯晓兵[1)]

[1)]（中国科学院计算技术研究所 计算机体系结构国家重点实验室，北京 100190）

[2)]（中国科学院大学，北京 100049）

[3)]（西华师范大学 计算机学院，四川南充 637009）

**摘 要** 近年来硬件设计呈现出异构化的趋势，如何有效开发并行程序成为制约异构系统发展的瓶颈之一，已成为业界共识。我国自主研制的“神威•太湖之光”超级计算机，采用了国产片上异构众核处理器 SW26010，为了降低程序员的编程难度、同时提高软件的移植效率，我们设计并实现了支持国产 SW26010 众核处理器的 OpenCL 编译系统。本编译系统实现了 OpenCL 平台模型、内存模型和执行模型到 SW26010 众核处理器的映射与优化机制，同时生成性能良好的可执行文件。最后通过实验验证了本编译系统的正确性和有效性，典型 OpenCL 应用经本编译系统编译后，在中小输入规模下，性能显著优于 Intel Xeon Phi，与 NVIDIA GPU 可比；在较大输入规模下，受限于局存 SPM 的容量限制，性能略低于 NVIDIA GPU。

关键词 OpenCL；异构；国产众核处理器；编译系统

**中图法分类号** TP312

# An OpenCL Compiler for the Homegrown Heterogeneous Many-core Processor on the Sunway TaihuLight Supercomputer

WU Ming-Chuan[1),2)] HUANG Lei[1)] LIU Ying[1)] HE Xian-Bo [3)] FENG Xiao-Bing[1)]

[1)](State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Science, Beijing, 100190)

[2)](University of Chinese Academy of Sciences, Beijing, 100049)

[3)](Computer School, China West Normal University, Nanchong Sichuan, 637009)

**Abstract** In recent years, with the tremendous development of the integrated circuit technology, it is possible to integrate multiple processor cores on a single chip to accomplish more complex and large computational tasks, and the processor architecture has evolved from single-core to multi-core and many-core. However, there is also a bottleneck in improving performance by means of blindly increasing the cores of same type processors. To further enhance the computing power, there has been a trend towards heterogeneous system architecture, which can provide more powerful computing power and better performance-to-power ratio. It has become the industry consensus that the programming model is one of the bottlenecks restricting the development of heterogeneous systems. The Sunway TaihuLight supercomputer is the world's first system with a peak performance greater than 100 PFlops, equipped with a homegrown heterogeneous many-core SW26010 CPU that includes both the management processing elements and computing processing elements in one chip. With 260 processing elements in one processor, a single SW26010 provides a peak performance of over 3 TFlops. On the other hand, large-scale scientific and engineering calculations such as earth, ocean, atmospheric system modeling and other critical applications are facing with the big

本课题得到国家重点研发计划项目高性能计算项目(2016YFB0200800)、国家自然科学基金重点项目（61432018）、创新研究群体项目（61521092）、南充市科技支撑项目（15A0068）、西华师范大学培育项目（13C002）、西华师范大学英才基金项目（17YC149）资助. 伍明川（通讯作者）,男，1992 年生，博士研究生，主要研究领域为异构并行编译系统，E-mail: wumingchuan@ict.ac.cn。黄磊，女，1980 年生，博士，工程师，主要研究领域为程序分析和编译优化、异构并行编程，E-mail: leihuang@ict.ac.cn。刘颖，女，1982 年生，博士，助理研究员，主要研究领域为异构并行编程、编译系统及相关工具，E-mail: liuying2007@ict.ac.cn。何先波，男，1971 年生，博士，教授，主要研究领域为嵌入式开发，E-mail: hexianbo_sctc@163.com。冯晓兵，男，1969 年生，博士，研究员，博士生导师，CCF 高级会员 ，主要研究领域为先进编译技术及相关工具环境，E-mail: fxb@ict.ac.cn。

计算机学报

performance challenge. How to fully utilize the computing power of the homegrown heterogeneous platform to achieve high performance of critical applications has important academic and practical value. In order to reduce the difficulty of programming, while improving software portability, we design and implement an OpenCL Compiler for the SW26010 processor. Based on the OpenCL programming framework and the microarchitecture of the homegrown many-core processors, the compiler provides the mapping mechanism from OpenCL platform, memory and execution model to the SW26010 many-core processor and implements thread coarsening, data layout and vectorization optimizations for the homegrown many-core processor. This paper implements the source-to-source compiler system based on Clang. The compiler translates OpenCL programs to the local low-level programming language for the processor, and calls the local compiler to generate high-performance binaries ultimately. The results of preliminary experiment have shown the correctness and effectiveness of our compiler. The performance of the typical OpenCL application which is compiled by our compilation system is significantly better than the Intel Xeon Phi and comparable with the NVIDIA GPU in the small and medium input size; it is slightly lower than the NVIDIA GPU in the large input size, for the limitations of SPM.

**Key words** OpenCL; heterogeneous system; homegrown many-core processor; compilation system

随着半导体工艺的进步，处理器完成了从单核向多核的转变，使得芯片的并行计算能力和性能得到了显著提高。但由于体系结构的复杂化和应用的多样化，核数的增加已经无法继续带来性能提升[1]。在这样的背景下，为了进一步增强计算能力，硬件设计呈现出异构化的趋势，由若干不同架构的处理器或处理器核协同工作，通用处理器与一个或多个加速设备互连组成的异构系统逐渐成为主流。在 2016 年 11 月发布的 Top500 榜单中[1]，排名第一的神威·太湖之光搭载了片上异构的众核处理器，而排名第二的天河二号则采用了 CPU 与 Intel Xeon Phi 芯片连接的异构架构。

异构架构下的程序如何编写，是异构系统需要解决的重要问题。面对复杂多样的异构架构，国际领先的研究团队纷纷开展了相关研究，提出了多种异构并行编程模型，例如 NVIDIA 提出了 CUDA[2]，微软提出了 C++AMP[2]，IBM 提出了 LIME[3]，Intel 提出了 Merge[4]，等等。然而，这些并行编程模型都只适用于特定的异构系统，出现了不同异构系统需要使用不同编程模型的问题，极大的增加了程序设计的难度和软件移植的开销。为此，Khronos Group 首次提出了通用的异构并行编程框架 OpenCL[3]，为各种不同的异构系统提供统一的并行编程接口，获得众多主流处理器厂商的支持，为异构系统上的并行计算提供了一个开放的、免费的通用标准。OpenCL 的执行模式要求底层平台支持 device 文件的动态生成、编译和加载运行，许多商业处理器厂商都对自家平台提供了 OpenCL 的运行支持，比如 Intel 发布了 Xeon 处理器和 Xeon Phi 协处理器上的 OpenCL SDK，NVIDIA 和 AMD 公司在 GPU 平台上对 OpenCL 进行了支持。

在国产异构众核处理器上，实现对 OpenCL 编程框架的支持，对于提高国产处理器的易编程性和软件通用性是大有裨益的。尤其是在地球、海洋、大气系统建模等关键应用领域[5-7]，大量的数据和密集的计算迫切需要异构加速。因此，支持国产众核异构处理器的 OpenCL 编译系统，具有重要的学术和实用价值。

异构架构下对 OpenCL 编程框架的支持，关键在于编译和运行时系统的设计、以及相关的优化技术，在这方面，学术界也展开了大量的研究。（1）OpenCL 的编译和运行时系统方面：首尔大学针对异构多核处理器提出了 SNU-SAMSUNG OpenCL Framework[8]，该框架提供了 OpenCL 到 C 的翻译器实施源到源级的优化，以及针对 OpenCL API 函数的运行时库，支持 X86、Cell BE、 ARM 和 DSP (Digital Signal Processor)等平台。此后，首尔大学进一步将此项研究工作扩展到集群系统中，提出了面向异构 CPU/GPU 集群的 SnuCL[9]编程框架。POCL[10]的核心是一个 kernel 代码的编译器，根据目标平台的硬件特征，针对性地对 kernel 代码进行线程合并、向量化、指令调度等编译优化，并最后

[1] Top500 list. https://www.top500.org/lists/2016/11/, 2016.

[2] NVIDIA CUDA toolkit. https://developer.nvidia.com/cuda-downloads,2016.

[3] KHRONOS Group. The open standard for parallel programming of heterogeneous systems, https://www.khronos.org/opencl/, 2016.

生成可执行文件。文献[11]则针对 OpenCL kernel 文件的动态执行模式需要底层平台支持的这一特点，提出了基于动态执行流的 predo 策略，可以在静态编译环境下从软件层面实现该执行模式。这里，文献[8][10]对 kernel 代码做自动变换，再由变换后的C代码或者中间表示经过编译生成目标平台的二进制码；文献[9]是面向集群的 OpenCL 运行支持；文献[11]重点论述如何支持动态执行流。如何对新型架构的异构计算机提供 OpenCL 的运行支持，这些工作并没有具体论述。若直接采用它们的编译框架支持国产异构众核处理器，则需要对国产异构众核处理器的本地编译器以及数据传输、线程管理等运行时库函数进行平台相关的修改。出于通用性的考虑，本文通过 OpenCL 编程视图向机器视图的映射，采取由 OpenCL 程序自动转成机器本地并行语言，进而编译生成本地可执行码的方法，支持 OpenCL 在国产异构超级计算机上直接运行。（2）OpenCL 静态分析和代码优化方面：文献[12]针对 OpenCL 架构提出一种 GPGPU 量化性能模型，通过静态分析 DLP(Data-Level Parallelism)应用评估其并行化后在 GPU 平台的执行性能，并设定具体的 OpenCL 执行配置。文献[13][14]分别针对直方图生成算法、图像积分图算法的 OpenCL 程序，分析不同 GPU 平台底层硬件架构特点，从访存带宽利用率、计算资源利用率、数据本地化、向量化、最优 work-group 数目选择方面进行算法优化。这些优化工作都依据了流行的商用 GPU 平台的硬件特征来进行程序优化。而对于存在硬件架构差异的国产异构计算平台而言，尤其存储体系上的不同，需要重新具体权衡优化方法。

面向国产众核芯片的新型架构和新型编程模型，本文采取 OpenCL C 向芯片本地编程语言转换的方法，并最终生成可执行文件，支持 OpenCL 代码在国产芯片上的直接运行。本文主要论述了支持国产SW26010众核处理器的OpenCL编译系统的具体设计与实现。具体内容如下：第 1 节简要介绍了 OpenCL 异构并行编程框架，第 2 节描述了国产异构众核处理器 SW26010 的架构特征，第 3 节详细阐述了国产异构众核处理器中 OpenCL 编译系统的设计依据和实现原理，第 4 节通过实验验证了该编译系统的有效性，第 5 节给出了总结和展望。

# 1. OpenCL 异构并行编程框架

在 OpenCL 编程框架中，程序员的编程视图由平台模型、内存模型和执行模型构成。其中，OpenCL 平台模型刻画了程序员眼中的硬件视图，是实际平台的高度抽象，使不同厂商的平台能在系统中共存；OpenCL 内存模型定义了程序执行时内存的结构、内容和行为，使程序员无需考虑实际的底层内存架构；OpenCL 执行模型描述了程序的执行行为，包括并行模式、线程组织、同步操作等。下面将分别介绍 OpenCL 的平台模型、内存模型和执行模型。

## 1.1 OpenCL 平台模型

OpenCL 平台模型如图 1(a)所示。在 OpenCL 平台模型中，异构系统包含一个主机(host)和多个加速设备(compute device)。运算部件包括 compute unit(CU)和 process element(PE)：每个加速设备被划分为数个对等的 CU，而每个 CU 又被划分为数个对等的 PE。程序中的计算全部执行在 PE 上，PE 是独立参与计算的最小单元，同一个 CU 中的所有 PE 可以沿完全相同或不同的控制流路径执行。OpenCL 程序通过 host 提交命令来驱动 compute device 进行并行计算。

## 1.2 OpenCL 内存模型

OpenCL 内存模型如图 1(b)所示。OpenCL 平台上的内存区域可以划分为主机端可直接访问的 host memory、以及加速设备可直接访问的 device memory，device memory 又可划分为数个内存区域：global memory、constant memory、local memory、private memory。其中 global memory 是加速设备上所有 CU 包含的所有 PE 均可访问的内存区域；constant memory 是 global memory 中的一块区域，具有只读性质；每个 CU 包含一块对内部所有 PE 都可见的 local Memory；每个 PE 具有私有的 private memory。

## 1.3 OpenCL 执行模型

OpenCL 执行模型如图 1(c)所示。OpenCL 程序分为在主机上执行的 host program 和在加速设备上执行的 kernel 两部分：host program 定义上下文，并通过上下文来管理 kernel 的执行。当 host program 提交 kernel 程序执行时，系统生成一个 N 维(N = 1、2、3)的索引空间 NDRange，kernel 按照 NDRange 的形式组织线程并发执行。NDRange 中的每个线程称为工作项(work-item)，work-item 之间以 SPMD(Same Program Multiple Data)模式并行执行，即每个 work-item 执行相同的 kernel 程序，但操作的数据不同。NDRange 还可进一步粗粒度分解成若干 work-group：其空间维度与全局空间的维度相同，每个 work-group 包含数目相同的相邻工作项。NDRange 为每个 work-item 分配一个 global ID，并

将邻近的 work-item 组织为 work-group，每个 work-group 被分配一个 group ID，work-item 在 work-group 内部具有 local ID。因此，每个 work-item 的序号都有两种表达方式，使用全局表达方式 global ID，或者使用 work-group 表达方式 group ID+local ID。全局表达方式与 work-group 表达方式之间可以按照下述关系相互转换：

以二维索引空间为例，NDRange 中的每个 work-item 的 global ID（$g_x$，$g_y$），可以通过 global ID 的偏移量（$F_x$, $F_y$）、work-group ID（$w_x$，$w_y$）、每个 work-group 的大小（$S_x$，$S_y$）以及 work-group 内的 local ID（$s_x$，$s_y$）计算得到：

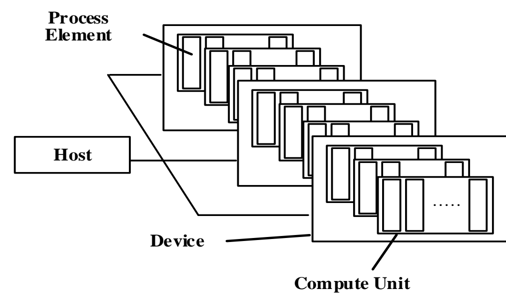
$$(g_x, g_y) = (w_x \times S_x + s_x + F_x, w_y \times S_y + s_y + F_y)$$

而 work-group ID 和 local ID，也可通过 global ID 和 work-group 大小计算得到：

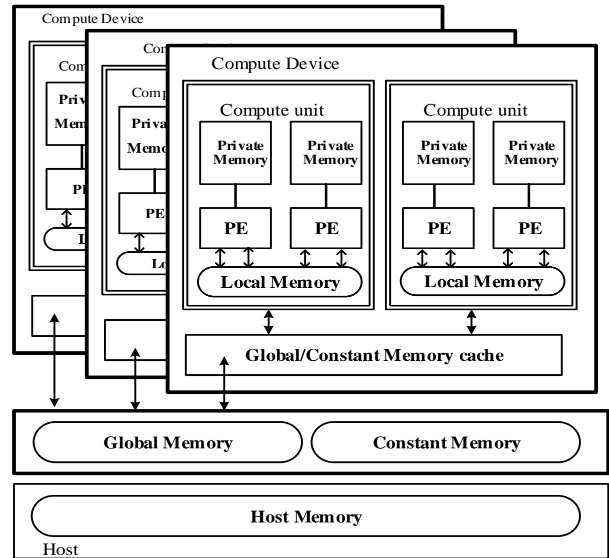
$$(w_x, w_y) = ((g_x - s_x - F_x)/S_x, (g_y - s_y - F_y)/S_y)$$

$$(s_x, s_y) = (g_x - F_x - w_x * S_x, g_y - F_y - w_y * S_y)$$

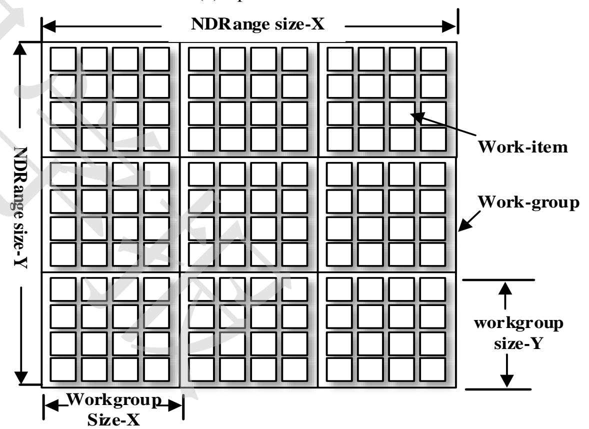
当 kernel 执行时，每个 work-item 执行在一个 PE 上，每个 work-group 执行在一个 CU 上。数据可以是 work-item 私有、work-group 内部共享、或全局共享，work-item 之间可以进行局部(work-group 内部)或全局(NDRange)的同步。




(a) OpenCL 平台模型




(b) OpenCL 内存模型




(c) OpenCL 执行模型

图 1 OpenCL 编程视图

## 2. SW26010 异构众核处理器

### 2.1 SW26010 的微体系结构

在 2016 年 6 月发布的超级计算机 Top500 排名中[1]，我国自主研制的"神威·太湖之光"超级计算机，成为全球最快的超级计算机，并在 2016 年 11 月榜单中蝉连了这个荣誉。"神威·太湖之光"超级计算机

[1] Top500 list. https://www.top500.org/lists/2016/11/, 2016.

是我国第一台全部采用国产处理器的世界第一的超级计算机，搭载的是我国自主研发的 SW26010 异构众核处理器。

SW26010 异构众核处理器的微结构如图 2 所示，它包括 4 个管理核心 MPE（management processing elements，也称为控制核）、4 个运算核心簇 CPE cluster（computing processing elements clusters）、4 个内存控制器 MC（Memory Controller）和系统接口 SI（System interface），其中运算核心簇由 8×8 阵列的 64 个运算核心组成，芯片总计有 260 个核心。其中一个 MPE、一个 CPE cluster 和一个 MC 组成一个核组（core group），核组内管理核心与运算核心簇以主、从的方式协同工作，而核组之间由片上网络（Network on Chip，NoC）连接，系统接口则用于与片外系统进行连接[15,16]。

SW26010 管理核心采用 L1 数据和指令 Cache 分离、L2 指令数据共享的两级片上存储层次。运算核心采用每个运算核心私有的 L1 指令 Cache 和一个运算核心簇共享的 L2 指令 Cache 的设计，运算核心的数据存储以 SPM（Scratch Pad Memory）的方式组织，该方式简化了传统 Cache 实现的控制开销，避免了众多运算核心之间一致性处理带来的设计复杂性和性能的降低。

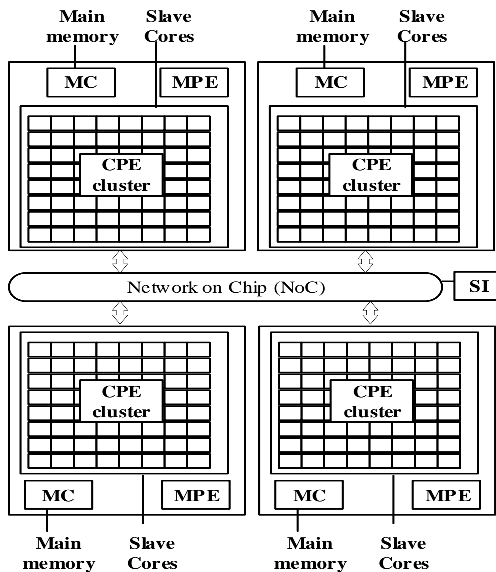


图 2.SW26010 处理器微结构

## 2.2 SW26010 的程序设计

为了能够高效利用片上的 260 个处理器核，SW26010 提供了基础编译器 SWCC 以及一套创建和管理线程的加速编程库 athread。

SWCC 可以为单个 MPE 或 CPE 生成可执行代码，支持 C、C++、和 Fortran 语言，且包含丰富的优化模块。

加速编程库 athread 可以驱动运算核心簇完成并发任务，使程序员可以通过显式调用 athread 函数管理 CPE cluster 的计算和访存，尽管这种方式极大增加了编程负担和出错概率，但是具有较高的执行效率。athread 库基于控制核与运算核心簇之间的主从协同工作模式，控制核负责创建线程、调度线程，计算核则发起 DMA 数据传输、执行核心计算。因此 athread 库分为控制核加速线程库和计算核加速线程库，主要函数如表 1 所示。控制核加速线程库主要提供控制核程序使用的 athread 接口，用于控制线程的创建回收、线程调度控制、中断异常管理、异步掩码支持等一系列操作；计算核加速线程库则提供计算核程序的接口，主要用于计算核线程的线程识别、中断发送等操作。在 SW26010 芯片架构下，每个线程绑定一个计算核心。一般地，执行模式为：（1）完成加速线程库的初始化；（2）启动核组中的所有可用计算核资源，创建线程组；（3） 显式阻塞主线程，等待该线程组运行，直至线程组终止；（4）在确定线程组所占用的计算核心无相关作业后，停滞计算核组流水线，关闭计算核组。

**表 1. athread 库的主要函数**

| 控制核加速线程库 | |
|---|---|
| 初始化线程库 | athread_init |
| 创建线程组 | athread_spawn |
| 等待线程组终止 | athread_join |
| 关闭线程组流水线 | athread_halt |
| **计算核加速线程库** | |
| DMA 数据接收 GET | athread_get |
| DMA 数据发送 PUT | athread_put |

# 3. 支持国产异构众核处理器的 OpenCL 编译系统设计与实现

## 3.1 总体设计

面向 SW26010 国产异构众核处理器，本文设计并实现了支持 OpenCL 编程框架的编译系统，采用源到源的转换方式将 OpenCL 源代码转换为 SW26010 的本地加速编程库，最后调用芯片的本地编译器 SWCC 生成可执行的目标文件，整个编译过程如图 3 所示。首先，输入的 OpenCL 源代码经过词法分析、语法分析后生成相应的抽象语法树 AST；之后通过 OpenCL-athread 转换模块，将上述对应于 OpenCL 源码的 AST，变换为对应于加速编程库的 AST；接着进入加速线程库代码生成模块，

计算机学报

将变换后的 AST 变换为调用 athread 库的源级别代码，并分解为控制核代码和计算核代码；最后调用本地编译器 SWCC 编译控制核代码与计算核代码，得到可执行文件。显然，该 OpenCL 编译系统由 OpenCL 到加速编程库的源-源编译器、以及本地编译器 SWCC 构成，本文的主要工作集中在第一段源-源编译机制的设计与实现上。

如图 3 所示，OpenCL 到 SW26010 加速编程库的源-源编译器由 3 个模块构成：词法/语法分析、OpenCL-athread 转换、athread 代码生成。本文选择基于 Clang 编译器实现上述源-源编译系统，Clang[1] 前端已实现 OpenCL 语言特性的支持，可以完成对 OpenCL 源代码的词法/语法分析，并将其转换为相应的抽象语法树，因此源-源编译器设计和实现的重点在于 OpenCL-athread 转换、athread 代码生成 2 个模块。

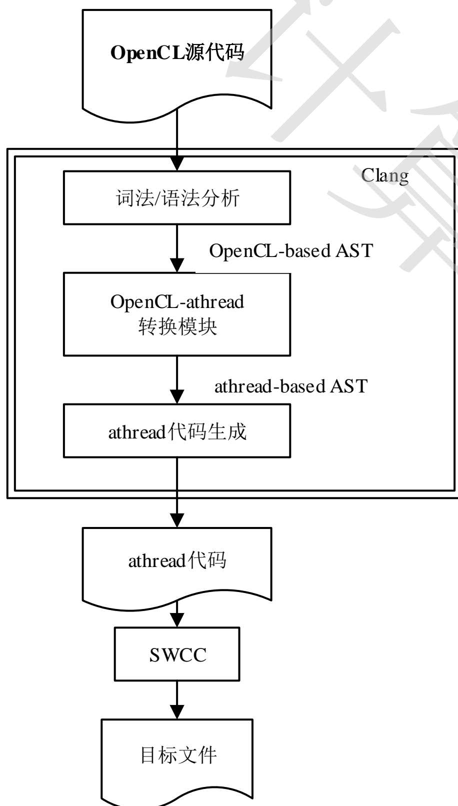



图 3.支持国产异构众核处理器的 OpenCL 编译系统设计

OpenCL-athread 转换模块的核心功能是，将 OpenCL 源码 AST 变换为使用 athread 库函数表达同样语义的 AST。如前所述，OpenCL 语义的核心是平台模型、内存模型、以及执行模型的表达，而加速编程库语义的核心是 SW26010 处理器上运算和数据的控制。因此，OpenCL-athread 转换模块设计的核心在于，OpenCL 平台、内存、执行模型到 SW26010 处理器的映射、以及通过代码变换来表达上述映射（如图 4 所示），3.2-3.4 小节将分别对这三个模型的映射机制以及相应的变换方法进行介绍。athread 代码生成模块的核心功能是，利用变换后的 AST，生成调用 athread 函数库的源级代码。因此，athread 代码生成模块的关键技术在于，扩充现有 AST 的表达，使之可以表达 athread 语义，并与具体的 athread 函数建立对应关系（如图 4 所示），该技术将在 3.5 小节进行介绍。

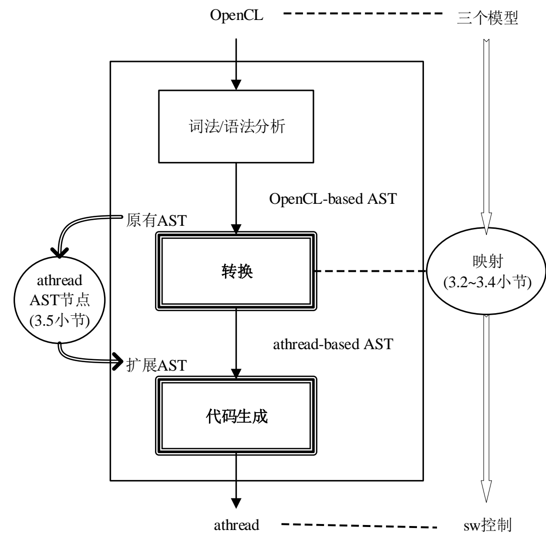
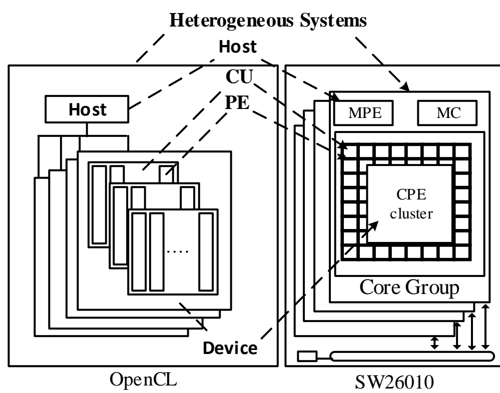


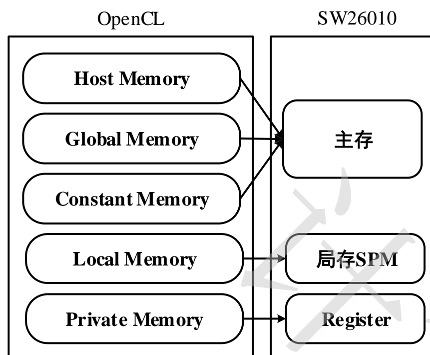

图 4. 通过代码变换表达模型映射

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

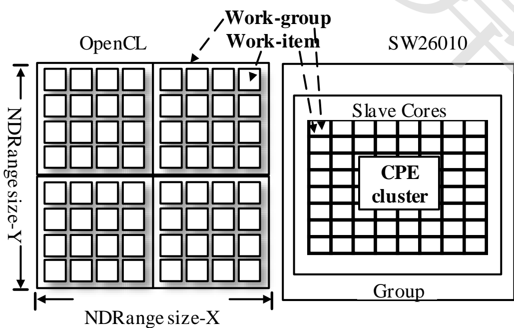
计算机学报



(a)平台模型映射



(b)内存模型映射



(c)执行模型映射

图 5.模型映射

## 3.2 OpenCL 平台模型到 SW26010 处理器的映射

如 1.1 小节所述，OpenCL 平台模型定义了 host、device、CU 和 PE，以及它们之间的控制关系。在本文设计的面向国产异构众核处理器的 OpenCL 编译系统中，我们将上述 OpenCL 平台模型映射到了 SW26010 处理器的运算部件上，如图 5(a)所示。

如 2.1 小节所述，SW26010 包含 4 个相对独立的核组，每个核组都是一个异构系统，其中集成了微结构不同的数十个处理器核，即管理核心、运算核心。因此，OpenCL 平台模型向 SW26010 处理器的映射，将以核组为单位：host 映射为每个核组的管理核心 MPE；device 映射为该核组内的运算核心簇 CPE cluster，支持 CPE 之间的线程级并行和 CPE 内部的数据级并行。进一步地，每个 CPE 只能运行用户态，不支持中断处理，且拥有私有的局部存储 SPM，无法直接访问其他 CPE 的局部存储。运算核心 CPE 的上述微结构特性，使得 CPE 之间的通信与同步将具有较大的代价。考虑到这一点，我们将 OpenCL 平台模型中的 CU 映射到 CPE 上，这是因为 OpenCL 程序在执行时，CU 之间几乎不存在通信和同步，而同属一个 CU 的 PE 之间，则存在频繁的通信与同步。在 OpenCL 平台模型中，PE 是最小的计算单元，我们需要进一步考虑 PE 如何映射。在 SW26010 处理器中，每个 CPE 都含有向量计算单元，支持数据级并行：在标量执行模式下，SW26010 处理器的最小计算单元是 CPE；而在向量执行模式下，最小的计算单元是 CPE 内部的一个向量通道（single instruction multiple data lane，SIMD lane）。因此，OpenCL 平台模型中的 PE，既可以映射为 SW26010 上的 CPE，也可以映射为 CPE 上的 SIMD lane。在不同的映射方式下，OpenCL 内存模型和执行模型的映射也随之发生变化，我们将在 3.3 和 3.4 小节讨论二者的差异。

## 3.3 OpenCL 内存模型到 SW26010 处理器的映射

如 1.2 小节所述，OpenCL 内存模型将程序执行时可以使用的内存空间划分为 host memory 和 device memory 两个部分，其中 device memory 可以进一步划分为 global memory、constant memory、local memory、private memory 四个部分。在本文设计的面向国产众核处理器的 OpenCL 编译系统中，我们将上述 OpenCL 内存模型映射到了 SW26010 处理器的内存子系统中，如图 5(b)所示。

如 2.1 小节所述，SW26010 处理器中每个核组都通过各自的 DDR3 控制器连接到主存，核组内的管理核心与运算核心簇是共享主存的关系；运算核心簇中的每个 CPE 都拥有私有的标量和向量寄存器，以及私有的局部存储 SPM。因此，我们将 OpenCL 内存模型中的 host memory 映射到 MPE 可以直接访问的主存上，device memory 则映射到 CPE cluster 可以访问的寄存器、局部存储和主存上。具体的说，device memory 中的 global memory 和 constant memory，其含义是所有 CU 都可以直接访问（读写或只读）的内存区域，因此映射到主存；local memory 是 CU 私有，因此映射到 CPE 的局部存储 SPM；private memory 是 PE 私有，考虑到我们在 OpenCL 平台模型映射时，采取了 PE 的两种映射方案，一种是在标量执行模式下将其映射为 CPE，一种是在向量执行模式下将其映射为 CPE 的一个 SIMD lane，因此 private memory 也相应的映射为 CPE 的标量寄存器、或者向量寄存器的一部分。

计算机学报

根据上述映射关系，host memory、global memory 和 constant memory 通过 malloc 向系统申请分配指定字节的主存空间；local memory 通过 athread 接口 ldm_malloc 申请分配局存空间；private memory 则由编译器分配到计算核的寄存器中，当寄存器资源不足时溢出到主存。因此，在本文提出的面向国产众核处理器的 OpenCL 编译器中，变量的地址分配和访问的实现方式是：将__global 关键字修饰的变量变换为全局变量，表示该数据对象存放在 host 和 device 程序都可见的主存空间，host memory 与 device memory 之间的数据传输，通过 host 和 device 程序同时读写该全局变量实现，即 clEnqueueWriteBuffer 和 clEnqueueReadBuffer 等 host 端发起的数据传输操作，变换为通过全局变量名直接访问，不需要数据传输语句；将__local 关键字修饰的变量变换为__thread_local 关键字修饰的变量，表示该数据对象存放在局存空间， local memory 与 global memory/constant memory 之间的数据传输，通过显式调用 athread_get 和 athread_put 函数，以 DMA 的方式实现数据在主存和局部存储之间的移动；__private 修饰的变量，在寄存器分配时具有较高的优先级，通过对 SPM 和主存直接的 load/store 操作，实现与 local memory 和 global memory 的数据传输。

### 3.3.1 SPM 数据布局优化

SW26010 处理器上 CPE 私有的局部存储（SPM），具有延迟低、带宽大的优点，SPM 的高效利用，是获得程序性能提升的关键因素之一。因此，本编译系统在内存模型的映射过程中，还实施了针对 SPM 的访存优化，即在 SPM 空间充足的情况下，将 global memory 中的数据通过 DMA 存放到 local memory 中。

首先，通过对 kernel 代码中__global 属性数据的访存行为做分析，找出访问频率高且通常被连续访问的数据，即为 DMA 传输到局存的对象数据。kernel 代码中被访问了多个元素的数组被视为访问频率高的数据，若对这些数组元素的访问方式通常是地址连续地进行，则为我们重点关注的对象数据。这类数据访存开销大，使用 DMA 传输和局存访问能有效降低此类访存开销。其次，对于一个 kernel 访问到的这种高频率访存数据，在 kernel 代码中进行如下修改：增加 DMA 语句将相应的数据从 global memory 取到 local memory，在数据使用的位置改为使用 local memory 内的相应数据，计算完成后增加 DMA 语句将计算结果从 local memroy 传回 global memory 的相应变量。如下图 6（a）所示。

当 kernel 内读/写频率高的数据量大到超过 SPM 容量时，我们采取循环迭代的方式分批地将数据存入 SPM、计算、写回结果。如下图 6（b）所示，矩阵乘法访问到三个数组 a、b、c， a 数组按行访问，b 数组全部访问，c 数组按行写，数据量超过 SPM 大小（SW26010 的一个计算核心的局存为 64KB），这里将 b 数组元素分批地读入 SPM 并进行计算。

```
#define J 64
#define I 64
__thread_local volatile unsigned long get_reply,put_reply;
__thread_local int my_id;
__thread_local float a_slave[I],b_slave[J][I],c_slave[I];
extern float a[J][I],b[J][I],c[J][I];
void func()
{                                              ──→ 源操作数读入局存中
        ...
        get_reply = 0;
athread_get(PE_MODE,&a[my_id][0],&a_slave[0],I*sizeof(float),(void*)&get_reply,0,0,0);
athread_get(PE_MODE,&b[0][0],&b_slave[0][0],J*I*sizeof(float),(void*)&get_reply,0,0,0);
        while(get_reply!=2);

        for(j=0;j<J;j++)
        {
          for(i=0;i<I;i++)
          {
                c_slave[j] += a_slave[i]*b_slave[i][j];
          }
        }
        put_reply=0;
athread_put(PE_MODE,&c_slave[0],&c[my_id][0],I*sizeof(float),(void*)&put_reply,0,0);
        while(put_reply!=1);
}                                              ──→ 结果写回global memory中
```

（a）容量需求未超过SPM大小（直接使用SPM）

```
 #define J 256
 #define I 256
 #define gap 32
 #define ls (I/gap)
 __thread_local volatile unsigned long get_reply,put_reply;
 __thread_local int my_id;
 __thread_local float a_slave[I*ls],b_slave[gap*I],c_slave[I*ls];
 extern float a[J][I],b[J][I],c[J][I];
 void func()
 {
    ...
    get_reply = 0;
athread_get(PE_MODE,&a[my_id*ls][0],&a_slave[0],I*ls*sizeof(float),(void*)&get_reply,0,0,
0);
   while(get_reply!=1);
   for(k=0;k<(I/gap);k++)                      ──→ 分批读入b数组元素
   {
       get_reply = 0;
athread_get(PE_MODE,&b[k*gap][0],&b_slave[0],I*gap*sizeof(float),(void*)&get_reply,0,0,0)
;
       while(get_reply!=1);
       for(j=0;j<J;j++)
       {
         for(ii=0;ii<gap;ii++)
         {
              i = k*gap + ii;
              for(pp=0;pp<ls;pp++) //here for is better than list.
                c_slave[I*pp+j] += a_slave[I*pp+i]*b_slave[ii*I+j];
         }
       }
   }
   put_reply=0;
athread_put(PE_MODE,&c_slave[0],&c[my_id*ls][0],I*ls*sizeof(float),(void*)&put_reply,0,0)
;
   while(put_reply!=1);
 }
```

（b）容量需求大于SPM大小（数据分批使用SPM）

图 6. SPM 数据布局优化示例

## 3.4 OpenCL 执行模型到 SW26010 处理器的映射

如 1.3 小节所述，OpenCL 执行模型定义了 work-item、work-group、NDRange，以及它们之间的组织关系。其中 work-item 是 OpenCL kernel 的最小执行单位，概念与线程类似，work-item 执行在 PE 上；work-group 由相邻的数个（典型值 $10^2$-$10^4$）work-item 构成，这些 work-item 之间通常存在频繁通信与同步，work-group 执行在 CU 上；NDRange

由数个 work-group 构成，这些 work-group 之间几乎不进行通信与同步，NDRange 执行在 OpenCL device 上。我们将上述 OpenCL 执行模型映射到了 SW26010 处理器的计算部件中，如图 5(c)所示。

在 3.2 节介绍的 OpenCL 平台模型映射中，我们将 OpenCL device 映射到 SW26010 处理器一个核组中的运算核心簇中，因此 NDRange 也将映射到 CPE cluster 上执行；CU 映射到运算核心上，因此每个 work-group 都将执行在 CPE 上；PE 的映射分为两种情形，标量执行模式下映射到 CPE 上，向量执行模式下映射到 CPE 的一个 SIMD lane 上，下面将详细介绍这两种平台映射方式对应的执行模型映射机制。

在标量执行模式下，CU 和 PE 都映射在同一个 CPE 上，因此 work-group 中一个 work-item 执行在该 CPE 上时，work-group 中的其他 work-item 将处于等待状态，也就是说，work-group 中的数个 work-item 应该以一定的顺序依次执行，如图 6(b)所示。在向量执行模式下，CU 映射在一个 CPE 上，PE 映射在 CPE 的一个 SIMD lane 上，由于每个 CPE 都支持 256bit 宽度的向量化，因此在使用 int/float/double 等常见数据类型进行计算时，SIMD 部件可划分为 4 或 8 个通道，即同时有 4 或 8 个 PE 映射到同一个 CPE 上。在这种情况下，work-group 中允许 4 或 8 个 work-item 同时执行，work-group 中的其他 work-item 处于等待状态，也就是说，work-group 中的 work-item 将以 4 或 8 个为一组分组执行，组内的 work-item 并发执行，组之间的 work-item 以一定的顺序依次执行，如图 6(c)所示。

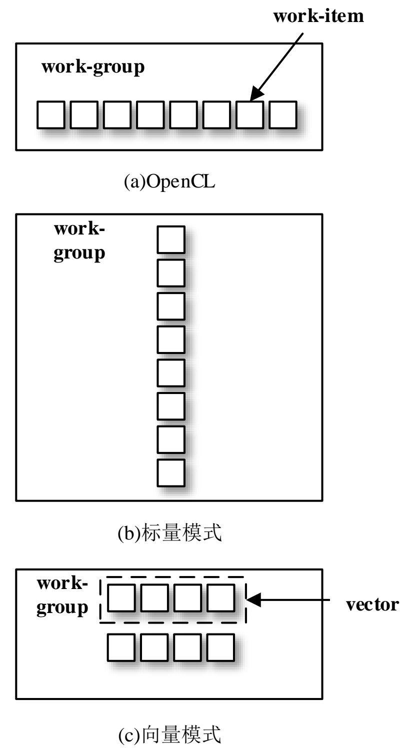



图 6. 执行模型中单个 work-group 的映射

### 3.4.1 线程合并优化

如前所述，在标量执行模式下，OpenCL 程序的每个 work-group 都执行在一个 CPE 上，work-group 内部的 work-item 依次执行在该 CPE 上。然而，SW26010 处理器中每个核组只有 64 个 CPE，考虑到 NDRange 常常包含多于 64 个 work-group，为每个 work-group 启动一个线程将造成额外的线程创建、调度和销毁等代价，本编译系统进一步实现了一个线程绑定一个 CPE、执行多个 work-group 的优化。考虑到数据局部性，我们选择将相邻的 work-group 合并执行，如图 7 所示。

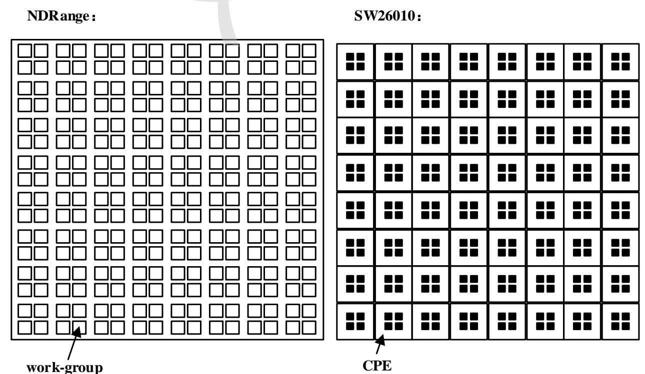



图 7. 线程组合优化示意图

work-group 的合并执行，意味着 work-item 不能依照原有序号进行数据访问，因此本优化的关键在于所含 work-item 序号的变换。如 1.3 小节 OpenCL 执行模型所述，NDRange 中的 work-item 序号可以

采用两种可以相互转换的表达方式：全局表达方式与 work-group 表达方式。对于全局表达方式，我们将其转换为 work-group 表达方式，如图 8-9 所示。对于 work-group 表达方式，我们将 group ID 映射到序号为 0~63 的 CPE 上，如图 10-11 所示。

```
__workgroup_size[0] = globalSize[0];
__workgroup_size[1] = globalSize[1];      → 计算group-size
__workgroup_size[2] = globalSize[2];

if (__workgroup_size[0] * __workgroup_size[1] * __workgroup_size[2] > N) {
  __workgroup_athread = __workgroup_size[0] * __workgroup_size[1] *
                        __workgroup_size[2] / N;
  __athread_max = N;
} else {
  __workgroup_athread = 1;
  __athread_max = __workgroup_size[0] * __workgroup_size[1] *
                  __workgroup_size[2];
}

athread_spawn(naive_kernel, 0);           → 启动计算核
athread_join();
```

图 8. 全局表达方式控制核程序映射示例

```
__thread_local int __athread_id;          → 当前线程id
extern size_t globalSize[3];
extern size_t __workgroup_size[3];

void naive_kernel() {
  ...                                                          计算group id
  for (__this_group_id = __athread_id * __workgroup_athread;
       __this_group_id < __this_group_max; __this_group_id++) {
    __group_id_x = __this_group_id % __workgroup_size[0];
    __group_id_y = ((__this_group_id - __group_id_x) / __workgroup_size[0])
                   % __workgroup_size[1];
    __group_id_z = __this_group_id / __workgroup_size[0] / __workgroup_size[1];
  ….
  }
}
```

图 9. 全局表达方式计算核程序映射示例

```
__workgroup_size[0] = globalSize[0] / localSize[0];
__workgroup_size[1] = globalSize[1] / localSize[1];   → 计算group-size
__workgroup_size[2] = globalSize[2] / localSize[2];

if (__workgroup_size[0] * __workgroup_size[1] * __workgroup_size[2] > N) {
  __workgroup_athread = __workgroup_size[0] * __workgroup_size[1] *
                        __workgroup_size[2] / N;
  __athread_max = N;
} else {
  __workgroup_athread = 1;
  __athread_max = __workgroup_size[0] * __workgroup_size[1] *
                  __workgroup_size[2];
}

athread_spawn(naive_kernel, 0);           → 启动计算核
athread_join();
```

图 10. work-group 表达方式控制核程序映射示例

```
__thread_local int __athread_id;          → 当前线程id
extern size_t globalSize[3];
extern size_t __workgroup_size[3];
extern size_t localSize[3];

void naive_kernel() {
  ...                                                          计算group id
  for (__this_group_id = __athread_id * __workgroup_athread;
       __this_group_id < __this_group_max; __this_group_id++) {
    __group_id_x = __this_group_id % __workgroup_size[0];
    __group_id_y = ((__this_group_id - __group_id_x) / __workgroup_size[0])
                   % __workgroup_size[1];
    __group_id_z = __this_group_id / __workgroup_size[0] / __workgroup_size[1];
    for (__local_id_z = 0; __local_id_z < localSize[2]; __local_id_z++)
     for (__local_id_y = 0; __local_id_y < localSize[1]; __local_id_y++)
      for (__local_id_x = 0; __local_id_x < localSize[0]; __local_id_x++) {
        global_id_x = __local_id_x + __group_id_x * localSize[0] ;
        global_id_y = __local_id_y + __group_id_y * localSize[1] ;   → 计算 global id
        global_id_z = __local_id_z + __group_id_z * localSize[2] ;

        kernel();
      }
  }
}
```

图 11. work-group 表达方式计算核程序映射示例

### 3.4.2 向量化优化

如前所述，在向量执行模式下，OpenCL 程序的每个 work-group 都执行在一个 CPE 上，该 work-group 中的 work-item 将以 4 或 8 个为一组分组执行，组内的 work-item 以 SIMD 模式并发执行，组之间的 work-item 以一定的顺序依次执行。并发执行的 work-item 可以充分利用 CPE 的向量计算部件，获得明显的性能提升。然而，这种向量模式的执行模型映射机制，需要编译器通过实施 work-item 之间的向量化优化来实现。

work-item 之间的向量化优化如图 12 所示，可以分以下两个步骤：

首先，进行 work-item 循环分割。如前所述，在标量执行模式中，work-group 中的 work-item 原本以 for、while-do 等循环结构顺序执行。在 work-item 合并过程中，编译器对 work-item 循环进行分割，将连续的 4 或 8 个循环迭代划分为一组，即相邻的 4 个或 8 个 work-item 合并为一组，并把组内 work-item 的 local ID 替换为 index×4+x(x=0~3) 或 index×8+x (x=0~7)。

接着，在一组 work-item 内部，将原有标量类型替换为 intv4/floatv4/doublev4 等向量类型；同时将变量之间的操作类型由标量替换为向量：由于 SWCC 的 SIMD 对标准 C 中的运算符进行了扩充，例如+, −,*,/等运算符都可以用于表示扩展浮点数据之间的运算，所以，该步骤将运算符修饰的标量变换为引入的向量并且添加标量与向量间的映射操

计算机学报

作函数。其中标量与向量间的映射操作函数包括：需要在 SIMD 操作之前添加的 simd_load 等装入操作函数或者 simd_set_floatv4 等赋值运算操作函数，目的是把标准类型变量映射到扩展类型变量上；在 SIMD 操作完成之后，还需要添加 simd_ store 等存储操作函数把扩展类型变量映射回标准类型变量。

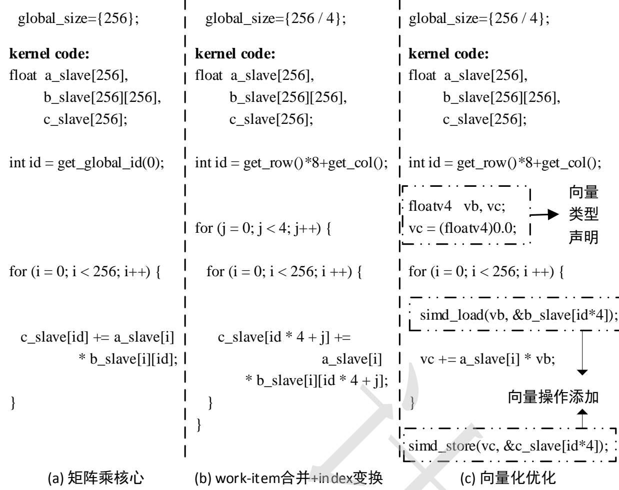



图 12. 向量化优化示例

### 3.5 语义转换模块

OpenCL-athread 语义转换模块通过遍历主机端/设备端 OpenCL 代码的 AST 节点，将 OpenCL 源码的 AST 变换为 athread 加速编程库的 AST。

首先，需要遍历 OpenCL-based AST，找到 OpenCL API 函数、OpenCL C 关键字、内建类型和内建函数。通过 TransformOpenCL 类对 AST 进行深度优先遍历，访问 Decl（declaration，声明）、Stmt（statement，语句）和 Expr（expression，表达式）等各个语法树节点，找到声明节点 DeclStmt 和函数调用节点 CallExpr。在对 CallExpr 节点的访问中，通过被调用函数的函数名匹配 OpenCL 的 API 和内建函数。在遍历 DeclStmt 节点时，对变量的数据类型以及修饰符进行匹配，找到 OpenCL C 的关键字和内建数据类型，并调用相应的转换函数进行变换。

其次，为了生成 athread-based AST，本系统对 AST 节点进行了扩展，使其可以表达 athread 语义。扩展内容如表 2 所示。针对 athread 关键字表达的存储空间属性，本系统在 VarDecl 变量节点的 StorageClass 属性中添加了__thread_local、__thread 等 athread 存储空间属性，并相应的实现了在 VarDecl 节点中设置和修改这些新属性的操作函数。针对 athread 向量类型，本系统通过继承扩展向量类型 ExtVectorType ，设计了 athread 库的向量类型 SIMDType，并提供了 ExtVectorType 与 SIMDType 的转换函数以及创建相应向量操作语句的操作函数。针对 athread 库的 API 函数，本系统新增了 CreateAthreadInit 等函数，用于构造 athread_init 等 athread 库函数的 CallExpr 节点，方便在转换模块中进行调用。

表 2. athread-based AST 扩展

| 库函数 | 关键字 | 类型 |
|---|---|---|
| athread_init | __thread | intv4 |
| athread_spawn | __thread_group | floatv4 |
| athread_join | __thread_local | doublev4 |
| athread_halt | __thread_local_fix | uintv8 |
| athread_get | | int256 |
| athread_put | | uint256 |

最后，我们利用上述 AST 节点的 athread 扩展，将 OpenCL-based AST 中 OpenCL API 函数、OpenCL C 关键字、内建函数以及内建数据类型，转换为 athread-based AST。整个过程根据 OpenCL 与 athread 加速编程库之间的转换关系进行转换，转换关系如表 3 所示。在遍历 AST 并找到 OpenCL 的关键字以及 API 函数后，通过对应的 transform 函数，对 OpenCL 关键字和 host-device 之间的数据传输、kernel 函数的参数赋值及启动执行、获取工作项索引等函数进行 AST 转换。对于 athread 库中未有表达的 get_global_id、get_local_id 等 OpenCL 内建函数，则通过添加 workItem_size /workGroup_size /global_size /dim 等数据对象完成转换。

表 3. OpenCL 到 athread 加速编程库的转换关系

| | OpenCL | athread |
|---|---|---|
| API | clBuildProgram | athread_init |
| | clEnqueueNDRangeKernel | athread_spawn 创建线程组<br>athread_join 线程执行 |
| | clFinish | athread_halt |
| 关键字 | __global/__constant | 全局变量 |
| | __local | __thread_local |
| 内建类型 | int4/float4/double4 | intv4/floatv4/doublev4 |
| 内建函数 | get_global_id/<br>get_group_id/<br>get_local_id | 对应维度的 work-group /work-item 索引变量 |
| | get_local_size/<br>get_num_groups/<br>get_global_size/<br>get_work_dim | workItem_size /<br>workGroup_size/<br>global_size/<br>dim 等数据对象 |

# 4. 实验与分析

## 4.1 实验平台与测试用例

为了验证本文 OpenCL 编译系统的有效性，我们在神威·太湖之光上进行了实验，并与 Intel CPU(Xeon CPU)、Intel MIC(Xeon Phi)和 NVIDIA GPU(Tesla K40)进行了对比。其中，Xeon CPU 和 Xeon Phi 使用 Intel SDK for OpenCL™ Applications XE 2013，Tesla K40 使用 CUDA 7.5，SW26010 使用本文 OpenCL 编译系统，SWCC 版本号为 5.421-sw-495。实验中使用的平台信息简介如表 4 所示。

表 4. 实验平台信息

| 名称 | 计算部件 | 主频（GHz） | 计算能力（TFLOPS） |
|---|---|---|---|
| SW26010 | 260 core | 1.45 | 3.06 |
| Tesla K40 | 2880 CUDA core | 0.745 | 4.29 |
| Xeon Phi | 57 core | 1.1 | 2.01 |
| Xeon CPU | 64 core | 2.13 | 1.09 |

本文实验中选用主流 OpenCL 测试集 parboil 中的测试用例 sgemm，其功能是矩阵乘法，并包含针对 NVIDIA GPU 的优化，利用 GPU shared memory 进行了分块计算。

## 4.2 实验结果与分析

实验中，我们分别验证了本 OpenCL 编译系统的正确性和性能。

在正确性方面我们以 sgemm 为例，输入 OpenCL 程序，经过本编译系统转换生成基于 athread 库的程序代码在 SW26010 众核处理器上运行结果与原 OpenCL 程序运行结果一致，说明本编译系统的正确性。图 13 给出了经本编译系统转换前后的代码对比，图 13（a）为输入的 OpenCL kernel 函数代码，图 13（b）-（d）为经本编译系统转换生成的 slave 代码，有三个版本：图 13（b）为实施了 3.4 小节的线程组合优化；图 13（c）为在(b)的基础上进行了 3.3 小节的局存优化；图 13(d)为在(c)的基础上实施了 3.4 小节的向量化优化。该实验表明 OpenCL 程序采用本编译系统能无缝运行在 SW26010 众核处理器上。

在性能方面，我们进行了两组实验，首先测试的是本 OpenCL 编译系统中各优化模块的性能（SWCC 的编译选项为-O2）。根据 3.2 小节 OpenCL 平台模型的映射规则，这里我们将 SW26010 的一个核组视为一个异构平台进行实验。我们进行了不同计算规模的试验，包括 64×64、128×128、256×256、512×512、1024×1024，依次测试了 3.4.1 小节的线程组合优化、3.3.1 小节的局存优化、3.4.2 小节的向量化优化带来的性能影响，并以未经优化、直接变换生成的 athread 代码作为基准。结果如图 14 所示，三种优化均获得了显著的性能加速，在四个计算规模下分别获得 47×、119×、469×、433×、245×的性能加速。其中：3.4.1 小节的线程组合优化获得了 1.27×、1.15×、1.06×、0.99×、0.96×的性能加速。线程组合优化的收益来源于减少了线程多次启动的开销，而随着计算规模的增大，一个 work-item 内的访存开销和计算开销增大，线程启动开销占整体运行时间的比例减少，因此优化效果递减。3.3.1 小节的局存优化在线程组合的基础上实施局存优化使得性能加速达到 34.8×、61×、288×、311×、200×。局存优化的收益来源于减少了 global memory 的访问开销，对于数据访问量大的例子有利，因此规模越大加速比越大，但我们也注意到，当规模到达 1024×1024 时，加速比的增幅缩小了，这与此时需要频繁地分批将数据换入 SPM 有关，该操作增加了 DMA 开销并减少了 SPM 内的数据重用，这导致单位时间内完成的计算量对大计算规模而言小于相对小一些的计算规模。3.4.2 小节的向量化优化在线程组合、局存优化的基础上加速比达到 47×、118.8×、468.7×、432.8×、245.1×。太湖之光上支持 floatv4 的 SIMD 指令，SIMD 优化提高了计算效率、均获得性能改善；当计算规模到达 1024×1024 时，受 SPM 容量所限，频繁地将数据分批换入 SPM，这部分开销影响到性能和 SIMD 的加速比。

第二组实验，我们将本 OpenCL 编译系统与其他平台的 OpenCL 编译系统的性能进行了最优性能的对比。同样以 64×64、128×128、256×256、512×512、1024×1024 作为例子，我们对比了几款主流平台——GPU 平台（NVIDIA Tesla K40[1]）、Intel Xeon Phi 平台（PHI3120[2]）、Intel Xeon CPU（E7- 8830[3]）平台，这些平台上执行的 OpenCL 程序也广泛地进行了共享内存优化、数据布局优化等，而 SW26010

[1] NVIDIA Tesla K40. http://www.nvidia.com/content/tesla/pdf/NVIDIA-Tesla-Kepler-Family-Datasheet.pdf, 2017

[2] Intel Xeon Phi. https://en.wikipedia.org/wiki/Xeon_Phi, 2017

[3] Intel Xeon E7- 8830. http://www.intel.com/content/dam/support/us/en/documents/processors/xeon/sb/xeon_E7-8800.pdf, 2017

计算机学报

仍然将一个核组视为一个异构平台，实验中所有平台都选取编译选项-O2/-O3 下性能最佳的版本进行对比。以 Intel Xeon CPU 的性能作为基准，其他平台的相对性能如表 5 所示，总体来说，SW26010 和 K40 是性能比较好的两个。表 5 描述的是各个平台在这个例子上的实际性能占各自峰值性能的百分比，这里以相对值表示，表中所列为各平台上占峰值性能的百分比值相对 Xeon CPU 上相应值的倍率值，SW26010 的峰值性能以一个核组的计算。在小的计算规模（64、128、256）下，SW26010 的占峰值比值优于 K40。但当计算规模为 512、1024 时，K40 的占峰值比值高于 SW26010，原因在于此时 SPM 的容量影响了 SW26010 的性能，线程访问的数据量大就需要频繁地分批将数据取入 SPM 以供数，这样既减少了 SPM 内数据的重用，同时频繁的 DMA 操作积累出的开销也不容忽视。另外，随着输入规模的增大，cache 利用率变高，Xeon CPU 和 Xeon Phi 的性能提升明显，而采用了 SPM 的 K40 和 SW26010 则由于容量限制，性能提升相对缓慢，导致表 5 结果中 512 和 1024 规模下占峰值比值的相对值降低。从实验结果可以看出，对于本 OpenCL 编译系统而言，当计算规模很大时，如何高效地使用 SPM，包括隐藏 DMA 开销、优化数据划分都是我们未来继续研究的方向。

(a)转换前
kernel关键代码

```
__kernel void simpleMultiply(
  __global float* c_slave,
  __global float* a_slave,
  __global float* b_slave )
{

  int mRow = get_global_id(1);
  int mCol = get_global_id(0);

  for (int i=0;i<64;i++){

    c_slave[mCol*J+mRow]=c_slave[mCol*J+mRow]
            + a_slave[mCol*J+i]* b_slave[i*J+mRow];

  }

}
```

(b)转换后代码
（线程组合）

```
#define J 64
#define I 64

__thread_local volatile unsigned long get_reply;
__thread_local volatile unsigned long put_reply;
__thread_local int my_id;

extern float a[J][I],b[J][I],c[J][I];

void func()
{
  int i,j;
  float s=0;

  my_id = get_row()*8+get_col();

  for(j=0;j<J;j++){

    for(i=0;i<I;i++){
      c[my_id][j] += a[my_id][i]*b[i][j];
    }
  }

}
```

(c)转换后代码
（线程组合+局存）

```
#define J 64
#define I 64

__thread_local volatile unsigned long get_reply;
__thread_local volatile unsigned long put_reply;
__thread_local int my_id;
__thread_local float a_slave[I],b_slave[J][I];
__thread_local float c_slave[I];
extern float a[J][I],b[J][I],c[J][I];

void func()
{
  int i,j;
  float s=0;

  my_id = get_row()*8+get_col();
  get_reply=0;
  athread_get(PE_MODE,&a[my_id][0],&a_slave[0],
            I*sizeof(float),(void*)&get_reply,0,0,0);
  athread_get(PE_MODE,&b[0][0],&b_slave[0][0],
            J*I*sizeof(float),(void*)&get_reply,0,0,0);
  while(get_reply!=2);

  for(j=0;j<J;j++){

    for(i=0;i<I;i++){
      c_slave[j] += a_slave[i]*b_slave[i][j];
    }
  }

  put_reply=0;
  athread_put(PE_MODE,&c_slave[0],&c[my_id][0],
            I*sizeof(float),(void*)&put_reply,0,0);
  while(put_reply!=1);
}
```

(c)转换后代码
（线程组合+局存+向量化）

```
#define J 64
#define I 64

__thread_local volatile unsigned long get_reply;
__thread_local volatile unsigned long put_reply;
__thread_local int my_id;
__thread_local float a_slave[I],b_slave[J][I];
__thread_local float c_slave[I];
extern float a[J][I],b[J][I],c[J][I];

void func()
{
  int i,j;
  float s=0;
  floatv4 va,vb,vc;
  float temp[4];
  my_id = get_row()*8+get_col();
  get_reply = 0;
  athread_get(PE_MODE,&a[my_id][0],&a_slave[0],
            I*sizeof(float),(void*)&get_reply,0,0,0);
  athread_get(PE_MODE,&b[0][0],&b_slave[0][0],
            J*I*sizeof(float),(void*)&get_reply,0,0,0);
  while(get_reply!=2);

  for(j=0;j<J;j++){
    vc = (floatv4)0.0;
    for(i=0;i<I;i+=4){
      simd_load(va, &a_slave[i]);
      vb=simd_set_floatv4(b_slave[i][j],b_slave[i+1][j],
                          b_slave[i+2][j],b_slave[i+3][j]);
      vc += va*vb;
    }
    simd_store(vc,temp);
    c_slave[j] = temp[0]+ temp[1]+temp[2]+temp[3];
  }
  put_reply=0;
  athread_put(PE_MODE,&c_slave[0],&c[my_id][0],
            I*sizeof(float),(void*)&put_reply,0,0);
  while(put_reply!=1);
}
```

图 13. OpenCL 源文件与本编译系统的输出代码

计算机学报

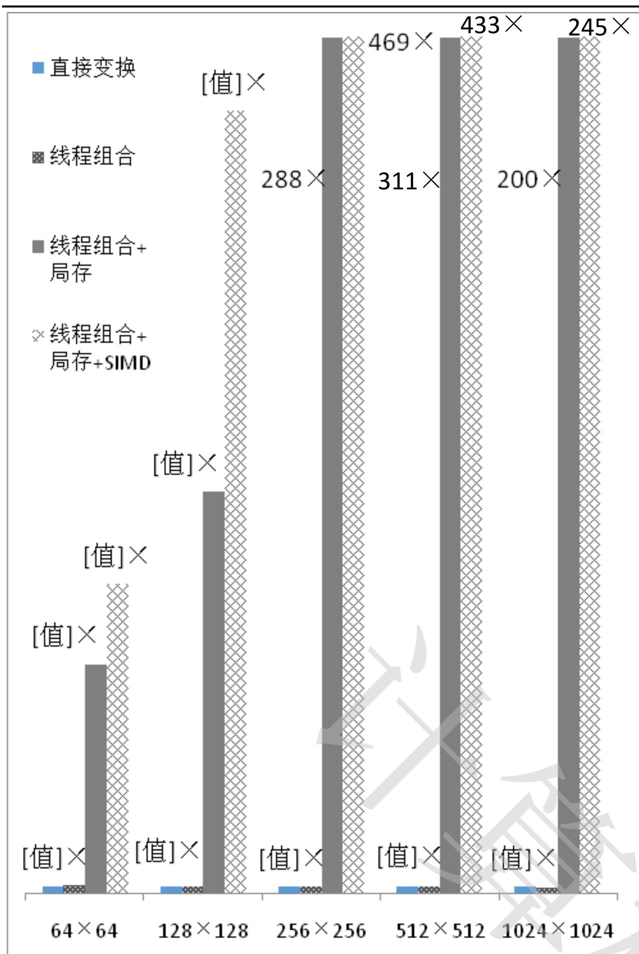



图 14. 本编译系统各优化模块性能分析

表 5. 各平台的实测性能占峰值性能的比值（相对值表示）

| 各自占峰值性能的倍比（X） | SW26010 | Tesla K40 | Xeon Phi | Xeon CPU |
|---|---|---|---|---|
| 64×64 | 6.11 | 1.35 | 0.10 | 1.00 |
| 128×128 | 5.58 | 2.19 | 0.13 | 1.00 |
| 256×256 | 7.14 | 2.64 | 0.18 | 1.00 |
| 512×512 | 1.06 | 1.77 | 0.29 | 1.00 |
| 1024×1024 | 0.29 | 0.97 | 0.41 | 1.00 |

## 5. 结束语

本文介绍了支持国产异构众核处理器的OpenCL编译系统的设计和实现。在系统设计方面，本文通过分析OpenCL编程框架的平台模型、内存模型以及执行模型的特点，结合SW26010国产众核处理器的硬件架构和所提供的程序设计方法，设计了OpenCL三个模型到SW26010处理器的映射机制，并实施了线程组合、局存优化、向量化等优化。在系统实现方面，我们基于Clang编译器实现了源源编译系统，完成了OpenCL程序向处理器本地低级编程语言的程序的转换，并调用本地编译器编译生成目标文件。

本文通过实验验证了编译系统的正确性及各优化模块的性能，并与NVIDIA K40 GPU、Intel Xeon CPU和Intel Xeon Phi的性能进行了比较。实验结果表明，本编译系统的优化模块，可获得高达百倍的性能提升，优化后的程序性能，在中小输入规模下，性能显著优于Intel Xeon Phi，与NVIDIA GPU可比，在较大输入规模下，受限于SPM的容量限制，性能略低于NVIDIA GPU。当然，要使得大量的OpenCL程序在SW26010上获得良好的性能，还需要进一步的优化和调试工作，这也是我们接下来主要的工作方向。

## 参 考 文 献

[1] Sun X, Chen Y. Reevaluating Amdahl's law in the multicore era. Journal of Parallel and Distributed Computing, 2010, 70(2): 183-188.

[2] Microsoft Corporation. C++ AMP: Language and programming model. Version 1.2, 2013

[3] Auerbach J, Bacon DF, Cheng P, Rabbah R. Lime: A Java-compatible and synthesizable language for heterogeneous architectures. Proceedings of the 2010 ACM International Conference on Object Oriented Programming Systems Languages and Applications . Reno/Tahoe, Nevada, USA 2010.89−108.

[4] Linderman MD, Collins JD, Wang H, Meng TH. Merge: A programming model for heterogeneous multi-core systems. In: Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems . Washington, USA, 2008, 287−296.

[5] YANG Haiping, SHEN Zhanfeng, LUO Jiancheng, WU Wei. Recent Development in High Performance GeoComputation for Massive Remote Sensing Data. Journal of Geo-Information Science, 2013, 15(1): 128-136 (in Chinese)
(杨海平, 沈占锋, 骆剑承, 吴炜. 海量遥感数据的高性能地学计算应用与发展分析. 地球信息科学学报, 2013, 15(1): 128-136).

[6] Wei Min, Wang Bin, Sun Jing, Gu Junxia, Hong Wendong. Analysis of the Applicability of Tianhe-1 Supercomputer in the Field of Meteorology. Advances in Meteorological Science and Technology, 2012, 02(1): 31-35 (in Chinese)
(魏敏, 王彬, 孙婧等. “天河一号”系列超级计算机系统气象领域适用性分析. 气象科技进展, 2012, 02(1): 31-35).

[7] Cai Jun, Xu Liren, Shen Xiaoying. Research on Engineering Application of Atmospheric Environment Simulation. Journal of System Simulation, 2015,01:192-196 (in Chinese)
(蔡军,许丽人,申晓莹. 大气环境仿真的工程化应用研究. 系统仿真学报,2015,01:192-196).

[8] Lee J, Kim J, Seo S, et al. An OpenCL framework for heterogeneous

multicores with local memory.Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, 2010, 193-204.

[9] Kim J, Seo S, et al. SnuCL: An OpenCL Framework for Heterogeneous CPU/GPU Clusters. Proceedings of the International Conference on Supercomputing, Venice, Italy, 2012: 341-352.

[10] Pekka J, Carlos L, Erik S, et al. pocl: A Performance-Portable OpenCL Implementation. International Journal of Parallel Programming, 2015, 43(5): 752-785.

[11] Wen Yanhua, He Wangquan, Wei Hongmei. Implementation of OpenCL Dynamic Execution Mode in Static Compiling Environment. Computer Applications and Software, 2014,31(10): 16-19 (in Chinese)
(文延华, 何王全, 尉红梅. OpenCL的动态执行模式在静态编译支持下的实现. 计算机应用与软件,2014,31(10): 16-19).

[12] ZHU Jun-feng, CHEN Gang, ZANG Ke-liang，WU Bai-feng. Quantitative GPGPU Performance Model Targeting OpenCL Architecture. Journal of Chinese Computer Systems, 2013, 34(5): 1118-1125(in Chinese)
(朱俊峰, 陈钢, 张珂良, 吴百锋. 面向OpenCL架构的GPGPU量化性能模型. 小型微型计算机系统, 2013, 34(5): 1118-1125).

[13] JIA Hai-peng, ZHANG Yun-quan, XU Jian-liang. Research on Image Integral Algorithm Optimization Based on OpenCL. Computer Science, 2013, 40(2): 1-7 (in Chinese)
(贾海鹏, 张云泉, 徐建良. 基于OpenCL的图像积分图算法优化研究. 计算机科学, 2013, 40(2): 1-7).

[14] AN Xiao-jing, ZHANG Yun-quan, JIA Hai-peng. Research on Histogram Generation Algorithm Optimization Based on OpenCL. Computer Science, 2015, 42(11): 31-36 (in Chinese)
(安小景, 张云泉, 贾海鹏. 基于OpenCL的直方图生成算法优化方法研究. 计算机科学, 2015, 42(11): 31-36).

[15] Fu H H, Liao J F, Yang J Z, et al. The Sunway TaihuLight supercomputer: system and applications. SCIENCE CHINA Information Sciences, 2016, 59(7): 1-16.

[16] ZHENG Fang, XU Yong, LI HongLiang, XIE XiangHui, CHEN ZuoNing. A homegrown many-core processor architecture for high-performance computing. Science China: Information Sciences, 2015, 45(4): 523–534 (in Chinese)
(郑方, 许勇, 李宏亮, 等. 一种面向高性能计算的自主众核处理器结构. 中国科学: 信息科学, 2015, 45(4): 523–534).

计算机学报

**WU Ming-Chuan**, born in 1992, Ph.D. candidate. His research include heterogeneous parallel compilation system.

**HUANG Lei**, born in 1980, Ph.D., engineer. Her research include program analysis and compiling optimization, heterogeneous parallel program.

**LIU Ying**, born in 1982, Ph.D., assistant researcher. Her research include heterogeneous parallel program, compilation system and related tools.

**HE Xian-Bo**, born in 1971, Ph.D., professor. His research include embedded system development.

**FENG Xiao-Bing**, born in 1969, Ph.D., professor, Ph.D. supervisor. His research include advance compiling technology and related tools.

**Background**:

The Sunway TaihuLight supercomputer is developed by the National Supercomputing Center in Wuxi, the first supercomputer which provide a peak performance over 100 PFlops in the world, ranking the first in the Top500 supercomputer in June this year. To achieve the autonomy of the core processor, the Sunway TaihuLight supercomputer equipped with the homegrown heterogeneous many-core processor SW26010, which was developed independently by China. How to reduce the programmers' difficulty of programming and improve the portability of the software has always been one of the main challenges in high-performance computing. Some researches provides OpenCL to C translator which implement source-to-source optimization, as well as runtime libraries for OpenCL API functions. Some other researches designs the kernel code compiler, which based on hardware features of the target platform, to do thread merging, vectorization and other compilation optimizations to kernel codes. But there are still some flaws within these system, such as how to provide OpenCL support for heterogeneous computers with new architectures. If we use these compiling framework to support homegrown heterogeneous many-core processors, we have to change the local compiler on the domestic heterogeneous many-core processors and runtime libraries for platform's specific.

To address those problems, this paper designs and implements an OpenCL Compiler for the SW26010 heterogeneous many-core processor. The compiler provides the mapping and optimization mechanism from OpenCL platform, memory and execution model to the SW26010 many-core processor, and ultimately generates executables with excellent performance. The results of preliminary experiment have shown the correctness and effectiveness of our compiler. The performance of the typical OpenCL application which is compiled by our compilation system is significantly better than the Intel Xeon Phi and comparable with the NVIDIA GPU in the small and medium input size; it is slightly lower than the NVIDIA GPU in the larger input size, because of the limitations of SPM.

This work is supported by the National Major Research High Performance Computing Program of China (Grant No.2016YFB0200800), the National Natural Science Foundation of China (No.61432018) and the Innovative Research Groups Project (No.61521092).

计算机学报