Modeling and Simulation 建模与仿真, 2023, 12(3), 2318-2329
Published Online May 2023 in Hans. https://www.hanspub.org/journal/mos
https://doi.org/10.12677/mos.2023.123213

Hans汉斯

基于ARM + DSP的异构平台优化加速算法

宋奇伟，晋高成，李丕丁

上海理工大学健康科学与工程学院，上海

收稿日期：2023年3月17日；录用日期：2023年5月15日；发布日期：2023年5月22日

摘　要

OpenCL编程模型应用于ARM + DSP异构多核平台存在核心利用率低、开发效率低等问题。本文基于AM5728异构开发平台，对OpenCL异构编程模型进行研究，提出了异构多核计算动态优化加速算法。分析了动态优化加速算法中的最优分配比例算法和数据划分原则，动态算法会根据运行情况动态调整相应参数。完成了测试系统的设计，对异构计算加速算法的相关参数进行测量，展示了Sobel算法、奇异值分解算法(SVD)分别采用计算加速驱动的结果和OpenCL异构编程模型的结果，分析两种不同方式下算法完成时间情况。测试结果表明优化加速算法使得Sobel算法执行时间降低至原执行时间的72.2%，SVD算法执行时间降低至原执行时间的80.2%。

关键词

OpenCL，异构多核，动态优化，数据划分

Optimization and Acceleration Algorithm for Heterogeneous Platforms Based on ARM + DSP

Qiwei Song, Gaocheng Jin, Piding Li

School of Health Science and Engineering, University of Shanghai for Science and Technology, Shanghai

Received: Mar. 17th, 2023; accepted: May 15th, 2023; published: May 22nd, 2023

Abstract

When OpenCL programming model is applied to ARM + DSP heterogeneous multi-core platform, there are still some problems such as low core utilization and low development efficiency. Based on AM5728 heterogeneous development platform, this paper studies the OpenCL heterogeneous

文章引用: 宋奇伟, 晋高成, 李丕丁. 基于 ARM + DSP 的异构平台优化加速算法[J]. 建模与仿真, 2023, 12(3): 2318-2329.
DOI: 10.12677/mos.2023.123213

programming model and proposes a dynamic optimization algorithm for heterogeneous multi-core computing. Analyze the calculation method of the optimal data segmentation ratio, determine the principle of data segmentation, and the dynamic algorithm will dynamically adjust the corresponding parameters according to the operation. The design of the test system is completed, and the results of Sobel algorithm and Singular Value Decomposition (SVD) algorithm driven by computing acceleration and OpenCL heterogeneous programming model are shown. The completion time of the algorithm under two different modes is analyzed. The test results show that the optimization algorithm reduces the execution time of Sobel algorithm to 72.2% of the original execution time, and the SVD algorithm to 80.2% of the original execution time.

Keywords

OpenCL, Heterogeneous Multi-Core, Dynamic Optimization, Data Division

1. 引言

随着对生物医学信号处理需求的提升，在相当多的应用场景，如远程医疗、实时监护等方面，算法的实时性要求较高。此时，在硬件平台受限情况下，部分算法的实时性已经不能满足应用场景的需求。算法实时性不足的问题主要由硬件平台的计算能力欠缺导致，因此计算能力限制了医疗电子设备的进一步发展。多核技术的发展引起了许多研究学者和企业的关注[1]，将多个架构功能不同的核心集成在一个处理器上的方法被广泛应用于各种嵌入式设备使用场景，用于解决计算量和实时性问题[2] [3]。

ARM + FPGA 异构多核平台被广泛的运用于图像加速，如 Xilinx 公司的 ZYNQ 系列开发平台，利用 FPGA 出色的并行处理能力与 ARM 处理器相结合，对图像算法加速有着显著的效果[4]。在人工智能领域采用 CPU + GPU 的异构多核架构，其中使用较多的为 NVIDIA 公司的 GPU，这种架构的异构多核平台利用 GPU 强大的并行计算能力，常用来解决计算量较大的问题[5]。在应用程序编写方面，商业公司都有应用于本公司产品的异构并行编程模型，AMD 公司的 Rrook + 编程模型、NVIDIA 公司的 CUDA 编程模型等[6]。OpenCL 是适用于多个平台的具有较强兼容性的并行编程模型被广泛使用[7]。通过改进任务调度算法可以提升核心利用率，相关静态启发式表调度算法对任务排序和任务调度进行改进，有效的解决了传统调度算法在异构多核平台上的问题[8] [9]。在应用领域，基于 ARM 和 DSP 的双核嵌入式视频监控系统，提高了设备的计算响应速度和传输稳定性[10]；有学者设计了一款基于 ARM + DSP 的腹腔镜语音自动定位系统，ARM 负责系统控制和界面操作，DSP 负责模型的训练和语音识别，进一步提高了系统的准确率和实时性[11]。

针对 ARM + DSP 核心的程序编写，OpenCL 异构编程模型被广泛使用，但采用该模型不能充分利用异构多核平台的全部性能[12]。OpenCL 异构编程模型在编写并行程序时需进行复杂的代码编写，需在主机上实现上下文的创建、设备创建等操作；并且在从机执行计算时，易出现主机循环等待从机计算结果的情况，产生了资源的浪费，降低了核心利用率。

2. OpenCL 异构编程模型简介

2.1. OpenCL 平台模型

开放语言 OpenCL 拥有很强的跨平台性，广泛支持多种架构的处理器，如 CPU、GPU、FPGA、DSP

等[13]。OpenCL 异构编程机制主要由四大模型：平台模型、执行模型、内存模型、编程模型组成，OpenCL 平台模型由主机和设备组成。一个平台模型包含有一个主机以及一个或者多个 OpenCL 设备。OpenCL 设备由一个或者多个计算单元组成，由计算单元来处理和计算接收到的元素。

2.2. 基于 ARM + DSP 平台 OpenCL 模型分析

根据上述的 OpenCL 模型，得出在 ARM + DSP 平台上采用 OpenCL 编程模型的程序的执行流程为：构建 OpenCL 设备的上下文；ARM 端准备需进行计算的数据；将数据传输给 OpenCL 设备；构建 OpenCL 设备的命令队列通知设备运行准备好的内核函数。执行流程如图 1 所示。

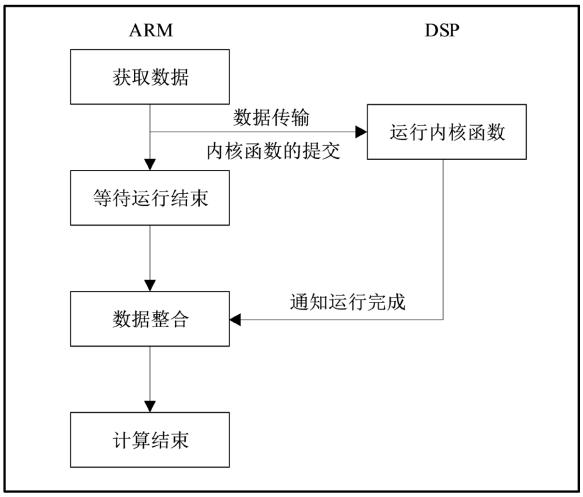


Figure 1. OpenCL execution process
图 1. OpenCL 执行流程

3. 异构平台计算动态优化算法设计

3.1. 基于异构平台的动态优化算法

基于 OpenCL 实现的应用，在 DSP 端执行计算时，易出现 ARM 端循环等待 DSP 端计算结果的情况，造成了资源的浪费，降低了 ARM 核心利用率。本文基于 OpenCL 编程模型提出了并行计算的方法，将计算任务动态地分配给异构处理器上所有的计算核心。影响该动态分配方案的因素有目标数据的大小、各个核心的占用率、所涉及的计算类型、各个核心的计算能力、核心间通信的时间开销。

本文在 ARM + DSP 异构平台上设计的并行计算动态优化算法如图 2 所示，整个算法的关键步骤为计算任务的划分，可根据 OpenCL 内核提交和执行时间、单次计算在各个核心上的时间开销、单个数据在核心间数传输时间开销、ARM 核心占用率、数据大小、计算类型计算任务分配比。按照计算所得出的动态比例，对目标数据进行分割，并在对应处理器核心上执行计算任务。

OpenCL 内核提交和执行时间指，ARM 核心通知 DSP 核心执行内核函数到 DSP 核心开始执行的延时，以及 DSP 核心结束计算到 ARM 端接收到 DSP 端计算完成的延时，该时间主要与核心间通信以及 OpenCL 内核函数部署的时长有关，以上与数据分割比例计算相关的参数中，OpenCL 内核提交和执行时间、单次计算在各个核心上的时间开销、单个数据在核心间数传输时间开销，这三个值的大小不会随着程序的运行而改变，在算法实现之前需对其进行测量；ARM 核心占用率，会随着操作系统的运行不断改变；数据大小和计算类型会随着需求的改变不断改变。该算法会根据这些变化量，动态改变计算分配比例，并根据数据的计算类型，进行相应的数据划分。

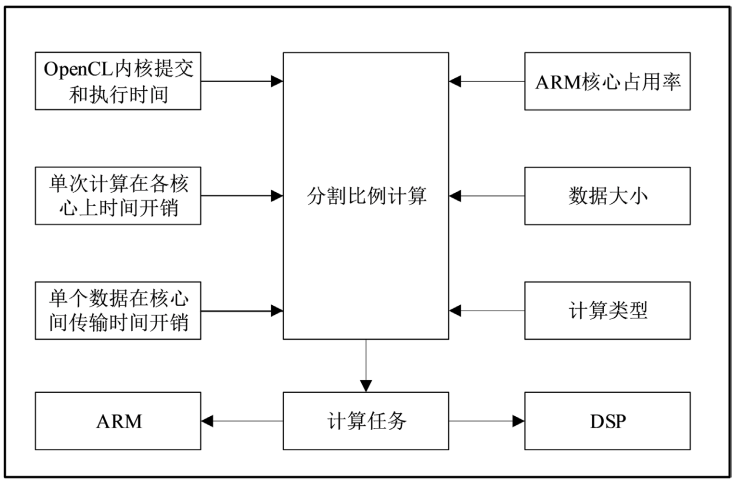


Figure 2. Design diagram of dynamic optimization algorithm for parallel computing
图 2. 并行计算动态优化算法设计图

3.2. 动态优化算法中的最优分配比例算法

异构平台并行计算动态优化算法中，关键问题在于计算任务的划分问题，若计算任务的划分合理，可以降低计算时间，反之，计算时间增长。为了得出最优分配比例，首先需对各个核心运算时间进行分析，将运行时间分为计算消耗时间、数据传输时间和 OpenCL 设备核心提交执行时间。ARM 核心和 DSP 核心所消耗时间的公式如式(1)和式(2)所示：

$$tarm = \frac{1}{1-a} * p * t\_coma * l + t\_send \tag{1}$$

$$tdsp = (1-p) * t\_comd * l + t\_build \tag{2}$$

式(1)中和式(2)中，*tarm* 为在 ARM 核心上的计算任务消耗的总时间；*a* 为 ARM 核心被其他进程占用的比例；*p* 为 ARM 核心的计算任务占总任务的比例，*t_coma* 为在 ARM 核心单次计算消耗的时间，*l* 为计算任务的计算量大小，*t_send* 为异构核心间数据迁移消耗的总时间。式(2)中 *tdsp* 为在 DSP 核心上的计算任务消耗的总时间，*t_comd* 为 DSP 核心单次计算消耗的时间，*t_build* 为从 ARM 端发送计算执行指令到 DSP 端开始计算的延时。

计算任务总耗时 T 与 *tarm* 和 *tdsp* 之间的关系示意如图 3 所示，T 与 *tarm* 和 *tdsp* 中的最大值相等，即为计算任务总耗时各个核心的最晚任务完成时间。

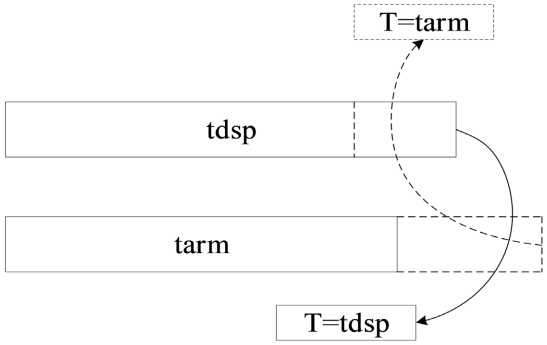


Figure 3. Calculation completion time diagram
图 3. 计算完成时间关系图

为了计算任务总耗时 T，必须使得 *tarm* 和 *tdsp* 中的最大值尽可能的小。由式(1)和式(2)可以看出，通过调整 p 可改变 *tarm* 和 *tdsp* 大小。当 *tarm* 和 *tdsp* 相等时，不会存在核心等待的情况，从而使得核心利用率最大，此时分配比例 p 为最优值，即可由式(3)计算最优分配比例 p。

$$p=\frac{t\_build-t\_send+t\_comd*l}{\left(\frac{1}{1-a}\right)*t\_coma*l+t\_comd*l} \tag{3}$$

其中 l 和 a 两个参数会随着程序的执行动态变化，需在程序运行时进行动态获取。*t_coma* 和 *t_comd* 与实际进行的运算相关，如加法在 ARM 核心上消耗的时间为 1 个指令周期，乘法消耗的时间为 2~5 个指令周期，而乘法和加法运算在 DSP 核心上都消耗的时间为 1 个指令周期，故不同的运算类型在相同的核心上消耗的时间可能不同，相同的运算类型在不同的核心上消耗的时间可能不相同。由于现代处理器架构中普遍使用流水线的方式读取并执行指令，且运算时还需其它相关指令如赋值、寻址等，所以在实际运行时，计算消耗的时间不能直接按照消耗的指令周期进行计算，需对每种运算在不同的核心上的实际运行的消耗时间进行测量。*t_build* 与平台相关，为固定值。*t_send* 与数据的计算类型相关，不同的计算类型需要不同的数据分割方式，导致 *t_send* 的计算不同。

3.3. 动态优化算法中的数据划分原则

目标数据的划分需考虑到数据的维度和计算类型，包括一维数据的乘法、加法以及二维数据的矩阵乘法、卷积。对于一维数据的加法和乘法，数据划分方法如图 4 所示，p 为 ARM 核心的计算任务占总任务的比例，l 为计算任务的计算量大小，左侧的数据在 ARM 端上计算，右侧的数据在 DSP 端进行计算。此时，*t_send* 的计算如式(4)所示，其中 *t_data* 为传输一个数据消耗的时间，d 为总数据量。实际运行时，仅需将目标计算数据传输给 OpenCL 设备即可。

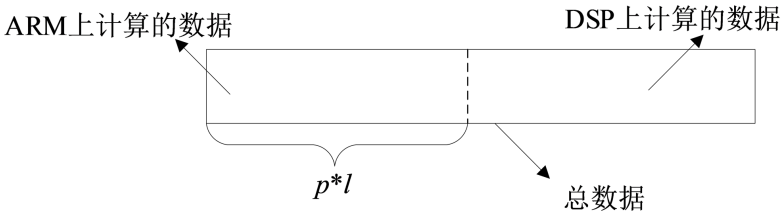


Figure 4. One-dimensional data calculation
图 4. 一维数据计算

$$t\_send=(1-p)*t\_data*d,\ l=d/2 \tag{4}$$

二维数据的矩阵计算如图 5 所示，包含在 ARM 上计算和在 DSP 上计算的部分，传输给 OpenCL 设备的数据不再是一维数据比例关系，而是所运算的部分对应的行向量和列向量。为了避免数据缺失，在对数据进行划分时需将计算所需的数据一次性传输至 DSP 核心上。

为计算方便将二维矩阵抽象成以一维数组，抽象图如图 6 所示，将参与计算的较小的矩阵全部传输至 DSP 端，较大的矩阵按照行向量进行比例分配，并延伸至全部矩阵。此时 *t_send* 的计算如式(5)所示，任务量大小 l 如式(6)所示。其中 m_1 和 n_1、m_2 和 n_2 分别为参与计算的矩阵的行数和列数。

$$t\_send=t\_data*(1-p)*m_1*n_1+t\_data*\ m_2*n_2 \tag{5}$$

$$l=n_2(2*n_1-1)m_1 \tag{6}$$

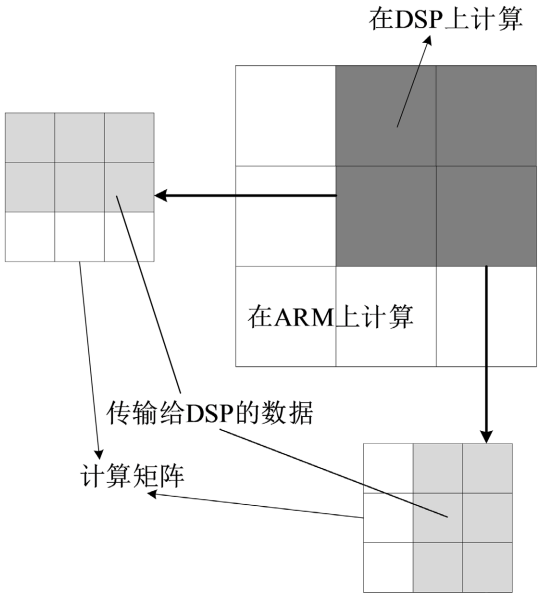


Figure 5. Calculation of two-dimensional data
图 5. 二维数据的计算

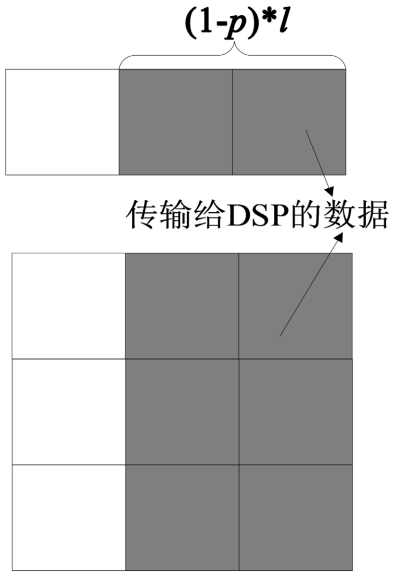


Figure 6. Two-dimensional matrix abstraction
图 6. 二维矩阵抽象

医学图像处理算法中的卷积运算，相对于上述一维和二维运算都有差别。医学图像处理算法中的卷积运算为算子和图像之间的运算，由于算子数据量一般较小，且只需传输一次，所以在数据划分的时候可不用考虑算子的划分，只需要考虑图像数据的划分。将卷积运算抽象成一维的计算，即可减少分配的复杂度，抽象如图 7 所示。此时 *t_send* 的计算如式(7)所示，任务量大小 *l* 如式(8)所示。*d* 为矩阵的数据量，d_1 为算子的数据量。

$$t\_send = (1-p)*t\_data*d \tag{7}$$

$$l = (2*d_1 - 1)d \tag{8}$$

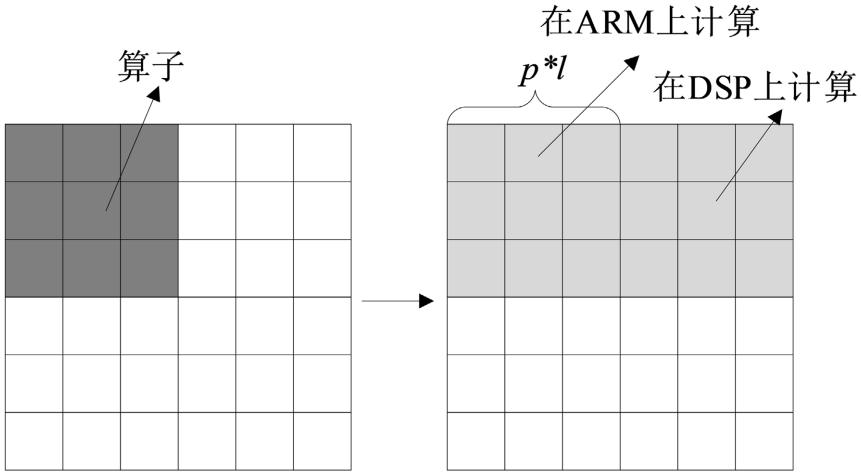


Figure 7. Convolution assignment diagram
图 7. 卷积运算分配图

在计算所有的参数之后，可以得到一个最优的分配比例 p，该比例随着计算的不同以及处理器的占用比会实时的变换，根据 p 将计算任务分配给各个核心进行处理，通过调度各个核心，并行执行计算任务，会极大降低整个计算的总消耗时间。

4. 异构计算优化算法中的参数测量

4.1. OpenCL 内核提交和执行时间测量

本文设计实验来计算出 OpenCL 设备内核提交和执行所消耗的时间，该实验步骤如下：首先编写一个空的计算内核函数，ARM 端将对内核函数进行编译，并通过 OpenCL 将该内核函数和执行指令发送给 DSP，记录下此时 ARM 端的系统时间 T1；当 DSP 端收到指令后，执行空的内核函数，此函数将不会执行任何操作，立即返回，此时 DSP 端将会立刻结束内核函数的执行，ARM 端在 DSP 端返回后记录下当前的系统时间 T2，T2-T1 即为 OpenCL 设备内核提交与执行时间。该实验的步骤如图 8 所示。

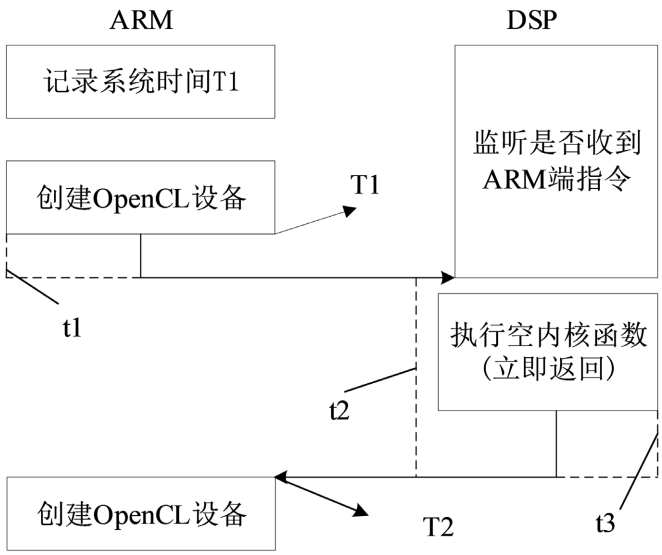


Figure 8. Flow chart of OpenCL device establishment time measurement procedure
图 8. OpenCL 设备建立时间测量程序流程图

图 8 中 t1 为设备开始建立到从机开始执行的时间，t2 为内核函数执行的时间，t3 为程序返回到主机得知从机运行完成的时间。为了计算内核函数部署至 DSP 内核上消耗的时间以及 ARM 和 DSP 通讯的时间总和(t1 + t3)，需使得 t2 为 0，此时 T2-T1 = t1 + t3，可测得总共的延时时间 T (T2-T1)。当内核函数为空时，可达到 t2 = 0 的效果。

经过多次运行该程序，获得程序首次核心提交和执行时间，程序非首次核心提交和执行时间。每个程序第一次建立的时间在 722 us 左右波动，之后内核提交和执行的时间稳定在 200 us，在程序中第一次提交与执行内核消耗的时间远远大于之后的提交与执行时间。消耗在队列的提交和开始的时间相对于核心执行时间较少。由于 OpenCL 在程序中第一次加载到硬件上时会很慢，而之后的提交与执行内核将会从设备的缓存中直接获取，因此时间相对减少。此测试程序执行内核函数会直接返回，除第一次执行外其余的核心提交和执行时间较短，在实际生物医学信号算法进行计算时，两次运算间隔较短，因此采用 200 us 表示 OpenCL 核心提交和执行时间。

4.2. ARM 和 DSP 间数据传输时间测量

为了计算 ARM 核心和 DSP 核心间单个数据传输消耗时间，实验在建立好 OpenCL 设备并且准备数据之后，记录下系统时间，将数据传输给 OpenCL 设备，之后记录数据传输完成时间，两个时间作差即可得到数据传输总时间，与数据量相除，即可得到 ARM 核心和 DSP 核心间单个数据传输消耗时间。由于该实验涉及到内核函数的运行，所以在最终得到的结果中还需减去 OpenCL 内核提交和执行的时间。图 9 展示了不同数据大小的数据量传输时间。

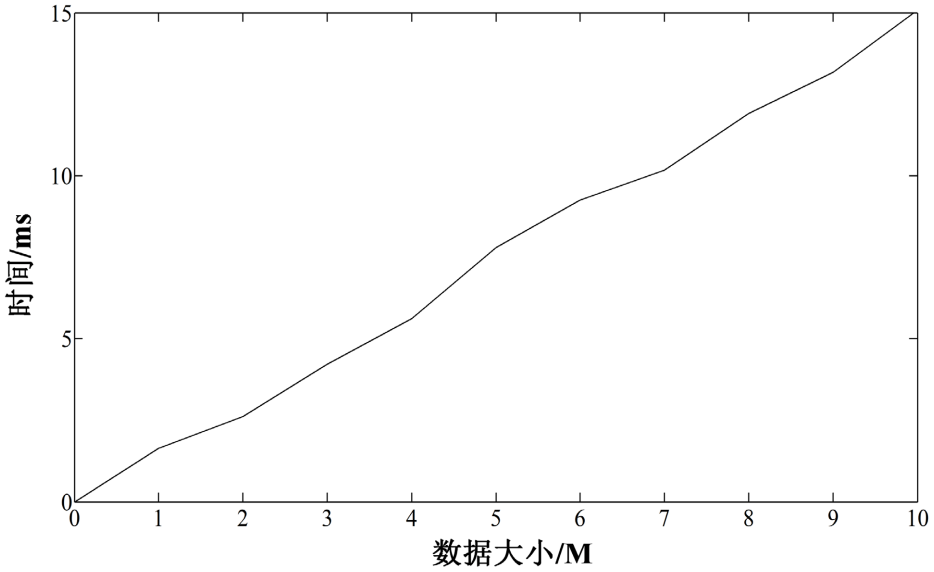


Figure 9. Data transmission time of different data sizes
图 9. 不同数据大小的数据量传输时间

由图 9 的结果可知，在传输时间上消耗的时间远远大于内核提交和执行的时间，并且随着数据量的变化呈现出一定的变化关系，为了找出数据量的大小与传输消耗之间的关系，需对各个数据量的数据传输时间进行多次测量。由测试结果可知，数据传输消耗时间与传输的数据量之间存在一定的线性关系，因此对其进行线性拟合，得出消耗时间 T (单位 ms)与传输数据大小 D (单位 M)之间存在如下的关系：

$$T = 1.5 * D \tag{9}$$

4.3. ARM 和 DSP 核心对不同计算消耗时间测量

对于 ARM 端和 DSP 端，相同计算所消耗的时间不相同，且在同一核心上，不同计算消耗的时间也可能不相同。分配计算比例需考虑每个核心对于该运算的运算能力，才能计算出最优分配比例。各个核心对不同计算的指令周期不同导致了如上差异，但是仅仅从计算的指令周期来计算各个计算消耗的时间，缺少了对其他硬件因素的考虑，如流水线，硬件加速等。因此本文通过实际的计算来得到各个计算在不同核心的计算消耗时间，用于之后的分配比例计算。由于本文采用的医学信号处理算法主要以乘法和加法为主，因此实验仅考虑乘法和加法的情况。将两个大小为 1 M 的数组分别在 ARM 端和 DSP 端执行加法和乘法运算，并在每个计算之后执行赋值语句，将赋值语句执行的时间一并加入乘法和加法的运算时间内，计算总共计算所消耗的时间。

处理器对每条指令的执行时间与处理器的主频有着较大关系，本实验平台采用的 ARM 核心为 1500 MHz，DSP 核心为 750 MHz，但 DSP 核心较 ARM 核心更擅长数据计算，因此会出现计算消耗时间相接近的情况。为了避免数据偶然性，经过多次测量，得到的结果如图 10 所示。

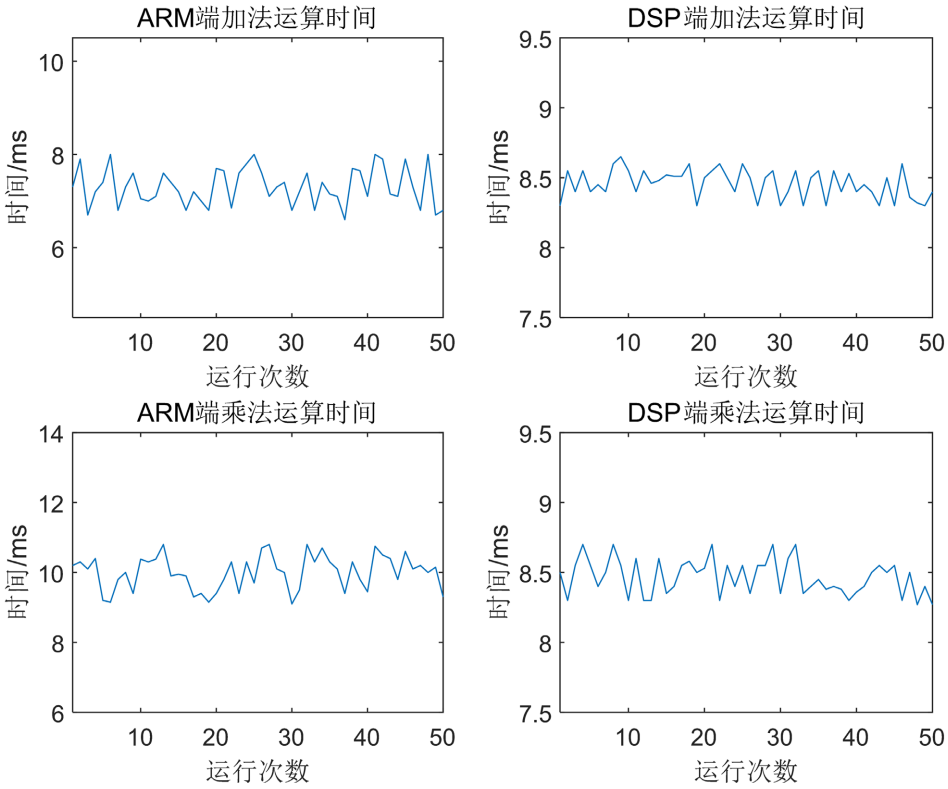


Figure 10. Addition and multiplication operation results at ARM and DSP
图 10. ARM 端和 DSP 端加法和乘法运行结果

由图 10 的测试结果可知，在乘法运算上，DSP 核心表现较好，ARM 核心较差，而在加法运算上则相反。计算量为 2 M 时，ARM 端乘法和加法计算消耗时间波动较大，DSP 端乘法和加法消耗时间波动较小，产生此现象的原因主要为 ARM 核心上的 Linux 操作系统本身的延时和任务调度导致，因此将取上述测量结果的平均值来表示计算消耗时间。为了方便后续计算保留一位小数，本文 ARM 端加法运行时间平均值为 7.6 ms，乘法运行时间平均值为 10.2 ms，DSP 端加法和乘法都是 8.5 ms。本文后续测试所用数据为 float 型数据，所以仅测量了 float 型数据运行时间。

5. 异构计算优化算法中的参数测量

本文采用异构多核平台如图 11 所示，基于 TL5728 平台进行设计、实现、验证。TL5728 搭载 AM5728 处理器，该处理器集成了双核 ARM 处理器以及双核 DSP 处理器。

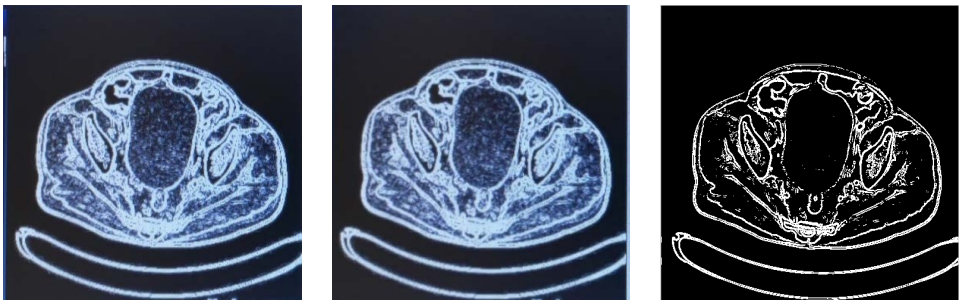
Figure 11. TL5728 physical image
图 11. TL5728 实物图

将设计的优化加速算法封装为驱动，系统分别采用 OpenCL 异构编程模型和计算加速驱动对选取的两种生物信号处理算法进行加速优化。测试算法 1 为 Sobel 算法，测试算法 2 为 SVD 算法。

5.1. 测试 Sobel 算法运行结果

Sobel 算法的整个过程为图像的读取、灰度图像的转换、sobel 算子 x、y 方向的卷积计算、梯度计算四个部分[14]，其中采用 OpenCL 模型和加速驱动的计算部分为卷积计算和梯度计算部分。

直接采用 OpenCL 并行编程模型的结果如图 12(a)所示，采用计算加速驱动的 Sobel 算法结果如图 12(b)所示，在 PC 端 Matlab 运行结果如图 12(c)所示。采用不用方法实现的算法在处理结果上并无明显差别，这也证明了计算加速驱动的计算结果具有可靠性。



(a) OpenCL 结果　(b) 计算加速驱动结果　(c) Matlab 结果

Figure 12. Sobel algorithm operation results
图 12. Sobel 算法运行结果

表 1 显示了各个模块在运行测试图像集后的平均时间，加速比为计算加速驱动下算法运行时间与 OpenCL 下算法运行时间之比。从表中可以看出图像读取、灰度图像转换与加速方式无关的部分运行速度相同，在卷积，梯度计算等与加速模型相关的部分在运行时间上采用异构计算加速模型时间有着显著减少，总运行时间减少为采用 OpenCL 编程模型的 72.2%。

Table 1. Comparison of Sobel algorithm acceleration time by different methods
表 1. 不同方法对 Sobel 算法加速时间对比

加速方式	OpenCL	优化算法	加速比
图像读取	1.4 ms	1.4 ms	1.0
灰度图像转换	3.2 ms	3.2 ms	1.0
X 方向卷积	56.1 ms	39.6 ms	0.706
Y 方向卷积	56.0 ms	39.6 ms	0.707
梯度计算	18.9 ms	14.1 ms	0.746
总计	135.6 ms	97.9 ms	0.722

由表 1 结果可知，因为加速驱动在 OpenCL 模型的基础上使得 ARM 参与运算，速度得到较大提升，但是却增加了数据传输的时间，所以并未达到理论上 50%的效果。其中梯度计算的计算量较卷积计算的小，所以计算加速驱动对卷积计算的加速效果更好。

5.2. 测试 SVD 算法运行结果

实验采用 SVD 算法对心电波形进行滤波，消除其他噪声干扰，获取干净的心电波形，采用的数据为 MIT-BIH 心电数据库中的数据。SVD 算法实现主要分为心电波形的提取和矩阵构造、奇异值分解计算、选取奇异值后的矩阵构建几个部分[15]。

表 2 为多次在心电数据库中提取不同心电信号运行 50 次的平均值，展示了 OpenCL 编程模型和优化算法对 SVD 算法加速时间对比，主要列出了心电数据读取、心电矩阵构建、奇异值分解、奇异值排序及选取、心电矩阵的重构。其中心电数据的获取和矩阵的构建以及奇异值的排序都是在 ARM 端完成，因此两种方式消耗的时间相同。奇异值分解和心电矩阵的重构为异构加速运算，其中奇异值分解计算采用迭代，将大量的矩阵计算拆分为向量的计算，在右奇异矩阵计算时，由于数据量太小，并未使用驱动进行加速。心电矩阵的重构主要为矩阵计算，采用计算加速驱动对整个计算过程进行加速，所以相对奇异值分解，心电矩阵重构计算的加速效果更好，总运行时间减少为采用 OpenCL 编程模型的 80.2%。

Table 2. Comparison of acceleration time of SVD algorithm by different methods
表 2. 不同方法对 SVD 算法加速时间对比

加速方式	OpenCL	优化算法	时间比
心电数据读取	1.6 ms	1.6 ms	1.0
心电矩阵构建	0.2 ms	0.2 ms	1.0
SVD 分解	80.2 ms	65.2 ms	0.813
SVD 排序及选取	0.2 ms	0.2 ms	1.0
心电矩阵重构	17.4 ms	12.7 ms	0.731
总计	99.6 ms	79.9 ms	0.802

6. 结论

对 OpenCL 编程模型应用于该平台的不足，本文提出异构计算动态优化算法，该算法的核心是将计算任务根据动态比例进行划分，分配至各个核心，到达并行执行计算的目的，解决了核心利用率不足的

问题。采用本文设计的优化加速算法以及 OpenCL 编程模型分别对其进行实现，分析了两种方式下的算法执行时间。对于 OpenCL 编程模型，采用优化算法进行异构计算加速驱动的方式，使得 Sobel 算法的时间降低至 72.2%，SVD 算法的时间降低 80.2%。经验证，算法提升了异构核心利用率，降低了代码编写的复杂性，减少了开发周期。

参考文献

[1] 李志勇, 卢松升. 基于多核 DSP 的 3 通道偏振图像 FMT 配准方法[J]. 电子测量技术, 2022, 45(19): 155-160.

[2] 王新玥. 面向图像算法的异构多核平台并行加速技术研究[D]: [硕士学位论文]. 济南: 山东大学, 2021.

[3] 江仲鸣, 杨全胜. 基于异构多核 SoC 的 LT 码编码硬件化技术研究[J]. 计算机工程与科学, 2020, 42(12): 2125-2132.

[4] 王威. 基于 ARM-FPGA 异构多核平台上图像处理算法的加速研究[D]: [硕士学位论文]. 西安: 西安电子科技大学, 2019.

[5] Baji, T. (2018) Evolution of the GPU Device Widely Used in Ai and Massive Parallel Processing. 2018 *IEEE* 2*nd Electron Devices Technology and Manufacturing Conference* (*EDTM*), Kobe, J13-16 March 2018, 7-9. https://doi.org/10.1109/EDTM.2018.8421507

[6] Memeti, S., Li, L., Pllana, S., Kołodziej, J. and Kessler, C. (2017) Benchmarking OpenCL, OpenACC, OpenMP, and CUDA: Programming Productivity, Performance, and Energy Consumption. *Proceedings of the* 2017 *Workshop on Adaptive Resource Management and Scheduling for Cloud Computing*, Washington DC, 28 July 2017, 1-6. https://doi.org/10.1145/3110355.3110356

[7] 汪思彤. 基于异构嵌入式环境的 OpenCL 主机端架构的设计与实现[D]: [硕士学位论文]. 南京: 南京大学, 2021.

[8] Atef, A., Hagras, T., Mahdy, Y.B. and Janeček, J. (2018) Lower-Bound Complexity and High Performance Mechanism for Scheduling Dependent-Tasks on Heterogeneous Grids. *Proceedings of* 2018 *International Conference on Innovative Trends in Computer Engineering* (*ITCE*), Aswan, 19-21 February 2018, 1-7. https://doi.org/10.1109/ITCE.2018.8316591

[9] 刘林东, 邬依林. 基于 HEFT 和 CPOP 的相关任务表调度算法[J]. 计算机系统应用, 2019, 28(3): 118-125.

[10] 张磊, 卢刚, 彭力. 基于 ARM 和 DSP 的双核嵌入式视频监控系统[J]. 计算机测量与控制, 2017, 25(6): 49-52+67.

[11] 赵鑫. 基于 ARM + DSP 双核的腹腔镜语音自动定位系统[D]: [硕士学位论文]. 天津: 天津大学, 2017.

[12] 吴树森, 董小社, 王宇菲, 等. UPPA: 面向异构众核系统的统一并行编程架构[J]. 计算机学报, 2020, 43(6): 990-1009.

[13] 李安民, 计卫星, 廖心怡, 等. 一种面向异构计算的结构化并行编程框架[J]. 计算机工程与科学, 2019, 41(3): 424-432.

[14] 朱寒, 林丽, 陈德全, 陈健. 基于多方向改进 Sobel 算子的 PCB 图像定位校正方法[J]. 电子测量与仪器学报, 2019, 33(9): 121-128.

[15] 章司怡, 陈熙源. 运动约束辅助的基于 SVD-CKF 的组合导航方法[J]. 电子测量与仪器学报, 2022, 36(4): 82-89.