一种数据流驱动的混合粒度可重构阵列结构设计

与仿真实现

摘 要

人工智能时代的来临，使得以数据为中心的计算需求变得越来越大，传统的计算平台已经不能满足这类需求，这就需要有一种全新的计算形式来支持这些数据量大、计算复杂的应用场景。同时，计算需求的快速变化对计算平台的灵活性提出了更高的要求。而可重构计算平台充分利用了空间计算的特性，可以满足数据密集型的计算场景，同时，由于可重构平台相对于其他计算平台的可重构特性，使其在人工智能时代可以发挥较大的作用。

基于高效字级操作的粗粒度可重构阵列已经被设计出来了，但是粗粒度的阵列结构不能完美解决计算任务中控制部分，使得其适用范围变小。本文首先对计算密集型算法核心代码进行分析，拆分核心代码中的数据流图(DFG)和控制流图(CFG),得出了粗粒度可重构阵列适合计算，即实现代码中的数据流图；细粒度可重构阵列适合控制，即实现代码中的控制流图。基于这种思想，本文从数据流驱动原理出发，提出了一种可重配置的混合粒度可重构阵列架构，将细粒度可重构技术和粗粒度可重构技术相结合，实现了对控制密集型算法的支持，整个结构支持多发射循环迭代技术和空间展开循环迭代技术。最后，对这种结构进行仿真器建模，通过C/C++语言实现仿真器，通过对算法采取不同的映射方法和优化技术，相对于静态数据流执行情况获得了较大的性能提升。

关键词：数据流驱动，混合粒度可重构阵列，多发射

DESIGN AND SIMULATION IMPLEMENTATION

OF A HYBRID-GRAINED RECONFIGURABLE ARCHITETURE BASED ON DATA STREAM

ABSTRACT

The advent of the era of artificial intelligence has made data-centric computing demands more and more large, and traditional computing platforms have been unable to meet such needs.

KEY WORDS: data stream drive,hybrid-grained reconfigurable array,multi-issue

1. 绪 论
   1. 课题研究背景及意义（为什么要研究该课题）

集成电路行业诞生以来的六十多年里，CPU、GPU、FPGA等计算平台在各个领域发挥了及其重要的作用，一步一步的引领者人类进入到了信息化的社会，有着不可磨灭的贡献，但是最近几年这些计算平台不能提供各种场景需求的计算算力。从外部环境看，随着人工智能时代的到来，为了适应各种复杂的场景，很多机器学习算法、神经网络算法、深度学习算法开始出现，这些算法的出现对计算平台提出了更高的要求，要求计算平台能提供较大的算力，要求计算平台能提供较大的灵活性，根据计算需求快速变换计算架构。

从内部环境看，各个计算平台都是在特定的历史条件下产生的，具有一定的局限性，CPU的出现是为了实现复杂的工业控制，所以适合于那些控制密集型的算法和应用，其提供的计算力是有限的；GPU的出现是为了解决CPU在图形处理能力不足的情况下协助CPU加速图形图像处理的问题，所以GPU擅长的领域是图形图像处理，但是其灵活性不够；FPGA的出现是为了实现硬件可编程，是可重构计算平台的一种，其提供了灵活的可配置性，提供了丰富的计算资源，在过去几十年里被芯片设计人员广泛选择用来作为设计和验证平台，取得了较大的成功。但是这种可配置是在配置每一比特位，配置量较大，不适合较大规模的电路；这些通用计算平台都是基于通用处理器架构和冯·诺伊曼结构，这种架构的特点是有较大的适用范围，适合于通用计算，但是一种计算平台的计算能力和平台的架构有很大的关系，基于这些情况，新型的可重构计算架构的研究才显得尤为重要。

在这些内外环境的条件下，需要有一种既能提供较大的计算能力同时还能具有灵活性的新型芯片架构，可重构处理器提供了一种更好的选择，可重构处理器充分利用硬件的空间资源，属于空间计算范畴，实现了在特定资源限制下计算能力的极大提升；同时可重构处理器根据应用的不同，自由灵活的配置信息，以较小的时间代价实现了不同硬件电路的切换。

可重构计算结合了通用计算的灵活性和专用计算的高效性，是一种比较新型的计算方式。 传统CPU以顺序执行机器指令的方式完成计算任务，优势体现在对复杂场景的控制和对各种算法的适应性上，但是计算能力有限。专用集成电路（ASIC）是面向特定算法和应用定制出来的硬件电路，可以充分发掘算法中的并行性，具有较高的计算效率，但是灵活性低，研发周期长。可重构计算结合了两者的优点，有广泛的应用前景。如表1所示，展现的是三种不同计算平台的特性。

表1可重构系统、通用计算系统、专用计算系统特性比较表

|  |  |  |  |
| --- | --- | --- | --- |
|  | 通用处理器 | 专用处理器（ASIC） | 可重构处理器 |
| 速度 | 慢 | 快 | 较快 |
| 开发难度 | 易，周期短 | 难，周期长 | 易，周期介于两者之间 |
| 资源消耗 | 少 | 多 | 一般 |
| 并行性 | 没有，指令级并行 | 有，较大的空间并行 | 有，指令级并行和空间并行 |
| 可升级性 | 有 | 没有 | 有 |

* 1. 国内外研究现状

在20世纪60年代可重构的概念就被Gerald Estrin[1] 提出，但是那时候对通用处理器的研究正如火如荼，很少有研究学者会把目光转移到可重构计算上来；另一方面，局限于那个时候制造工艺不是很先进，所以可重构计算并没有进入到很多研究学者的研究范围。

FPGA：随着硬件电路中可编程开关的实现，使得某些电路的功能可以被改变，这是可重构系统发展起来的先决条件。通过在芯片内部集成一些存储单元，在其中记录一些信息，这些信息被用于改变电路的某些结构。在这些信息被固化前，芯片电路的功能是不确定的，通过将配置文件固化，这些芯片就具有了特定的功能，这就是早期的可重构的雏形。而这些可编程的存储器主要包括：可编程只读存储器(PROM-Programmable Read-Only Memory)、电可擦除只读存储器(E2PROM-Eletrically Erasable Programmable Read-Only Memory)、紫外线可擦除只读存储器(EPROM-Erasable Programmable Rean-Only Memory)。后来随着计算机科学的进步，出现了结构稍微复杂的可编程逻辑器件(PLD-Programmable Logic Device)

Grap

Morphsys

XPP

wavescaler

* 1. 主要研究思路（自己的研究和别人的有什么不同）

本文研究的混合粒度可重构阵列是由粗粒度处理单元CPE(Coarse Processing Element）、细粒度处理单元FPE（Fine Processing Element）、互连网络和控制单元组成，如图1。阵列的基本工作方式是针对不同的算法和应用，提取出算法中的控制数据流图CDFG(Control-Data Flow Graph）,结合阵列上处理单元算子资源分布，生成配置信息。这些配置信息分布式的存储在每一个处理单元的配置寄存器中，决定每一个处理单元的功能、处理单元之间的数据交互方式以及处理单元内部的数据通路的形式。通过改变每一个计算单元内部配置寄存器的内容，重新配置处理单元及其之间的互连方式，从而实现了可重构的能力。

本文提出的混合粒度可重构阵列结构采用数据流驱动的执行方式，通过对循环级流水和代码空间展开的支持，大大提高了HGRA(Hybrid-Grained Reconfigurable Array)结构的计算吞吐率。同时，数据流驱动的机制实现了阵列执行任务的自主性，每一个计算单元需要的数据准备好之后就可以激活该数据对绑定的操作，相比于集中式指令调度减少了操作时间。结构还具有以下主要特点：

1) 细粒度节点灵活的控制能力。FPE支持bit级的操作，实现代码中循环控制，分支跳转控制。增加了整个阵列的灵活性。

2) 本地Memory存储空间。粗粒度阵列频繁的和外部内存交换数据显然降低了计算数据的存取效率，通过设计本地存储Memory，实现局部数据的复用，减少传输数据的开销。

3) 固定配置模式。在任务执行期间，配置信息是固定的，每一个计算单元都固定执行一种功能，固定连接一条通路，减少了配置信息量。同时配置包分布式的存储在各个处理单元中，减小切换配置的代价。

* 1. 论文组织结构

本论文提出了一种基于数据流原理的混合粒度可重构阵列处理器的架构，并面向具体的应用探讨了整个阵列结构的实现细节，同时基于该阵列结构，设计并实现了该硬件结构的软件仿真器，用于评估本论文所提出的阵列结构的功能和性能指标。本文设计的阵列结构是一种全新的计算架构，将数据流计算技术和空间计算技术相结合，实现了一款处理器芯片既能提供较大的计算能力，又能提供一定的灵活性，其软件可配置的特性使得其在不同的应用场景之间进行切换，减少了芯片设计和制造过程中的时间成本。

本论文的章节安排如下：

第一章介绍了本论文研究课题所面向的研究背景，确定了做混合粒度可重构计算阵列的必要性和重要性。

第二章这种介绍了混合粒度可重构阵列结构设计过程中设计到的主要技术，包括数据流技术和可重构技术，为本研究提供了强有力的理论支撑，提现了研究的创新性。

第三章详细介绍了基于特定应用场景下混合粒度可重构计算阵列的设计细节，包括粗粒度阵列处理单元的结构，执行机制，数据流动原理，细粒度阵列处理单元的功能等实现细节，同时介绍了该阵列结构支持的两种应用映射优化技术；最后，展现了图像卷积操作算法的映射实例，和两种优化技术的对比分析。

第四章介绍了基于该阵列架构的仿真器的实现细节。介绍了仿真器的原理，和该结构的仿真器的基本框架，最后使用仿真器对比分析该结构的性能瓶颈。

第五章对论文进行了总结，分析其中的不足，并对该研究领域进行了展望。

1. 数据流驱动及可重构技术
   1. 数据流驱动技术
      1. 数据流计算机概述

电子计算机诞生以来，计算机的体系结构都是基于冯·诺伊曼结构及其变种来实现的，这种结构是基于控制流实现的，程序被编译成一条条的指令存储到指令寄存器中去，计算机通过执行指令来实现代码中指定的功能。基于控制流实现的计算机是一种通用计算平台，其主要的特点就是通用性强，可以适用于各种应用场景，但是冯·诺伊曼结构在并行计算场景上有着天生的困难。

数据流驱动计算机能较好的支持并行计算，它突破了以指令为基础的数据流动，它已数据来驱动相应操作的执行，当相应的操作所需要的数据都准备就绪之后就可以激活该操作进行相应的动作，在数据流里面成为点火(fire)行为，其操作结束之后产生的数据可以作为下一步的操作需要的源操作数。由于硬件结构中没有程序计数器，点火行为可以在空间硬件资源上同时的进行，所以这种隐含的并行性使得数据流计算机很适合于并行计算。

* + 1. 数据流驱动执行机制

数据流处理机中的指令主要是由数据令牌和操作码构成，数据令牌是由两部分组成，分为数据和数据的目的地址，每一条数据令牌都包含这两条信息。操作码被分配到相应的硬件资源上并具有一个地址，当操作码点火操作需要的数据令牌到达该操作码指定的硬件资源地址时，该操作码指定的操作被启动，

如图1所示，表示的是快速傅里叶变换中的蝶式变换，通过这张图可以知道数据流计算中，数据流图包含两种不同的节点，分别称为输入弧（link）和动作（actor）。由图可知，两个动作之间必有一输入弧，两输入弧之间必有一动作，如图所示，其中圆圈表示动作，小圆圈表示输入弧，每一动作表示进行一步运算，其结果由输入弧传送到下一动作。

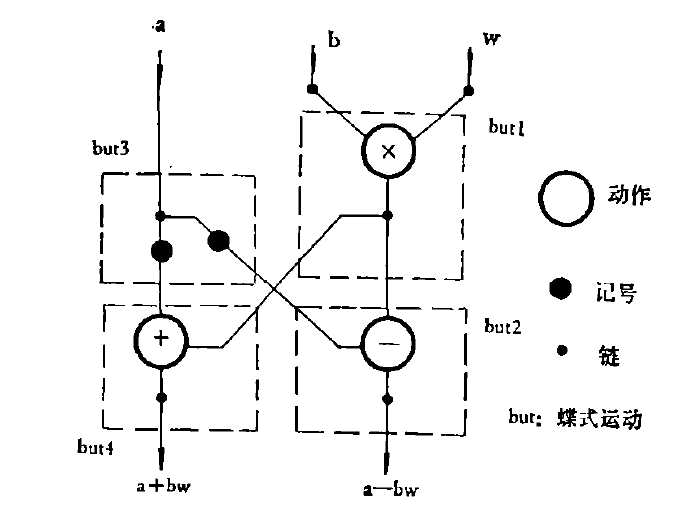


图1蝶式变换的数据流图

数据流计算机可以分为静态的数据流计算机和动态的数据流计算机。对于静态数据流处理机，在两个动作之间的输入弧上只能存在一个数据令牌，也就是说该条输入弧上的数据令牌是没有标号(tag)的，如图1，任意时刻每一个动作节点只会接受一个来自于输入弧上的数据来激活自己的操作。由于数据令牌没有标记，所以如果在任意时刻任一条输入弧上同时出现了两条以上的数据令牌的话，节点对于传输到自己的数据令牌就没有能力区分它们中哪些是属于同一批的操作数，节点会任意选择其中某一个数据令牌中的数据进行操作，从而会出现错误。由于这种限制静态数据流处理机只能用于处理一般的循环，循环迭代之间不能进行流水，在计算性能上有较大的损失。

而动态数据流处理机让每一个数据令牌带上记号(tag)，这样可以在两个节点之间的输入弧上出现多个数据令牌，由于记号的存在，每一个动作节点可以匹配相同批次的数据进行点火动作。所以动态数据流处理机不需要像静态数据流处理机那样通过控制输入弧上的数据令牌个数来保证程序的正确执行。如图2所示，这样每一个节点可以不间断的进行点火操作以达到最高的执行效率。所以，动态数据流处理机更适合于做循环代码的展开操作，可以充分提高程序中的并行性。



图2动态数据流情况下的输入弧

* 1. 可重构技术

可重构计算即通过软件字段来配置硬件电路使其形成特定结构的专用电路，并通过外围的辅助软硬件来运行一定的计算任务。其最主要的优点就是灵活的可配置性。它有区别与专用计算电路灵活性不够的特点，又区别于通用计算系统计算效率低的特点，可重构计算结合了两者的长处，既能实现计算效率高还能灵活可配置，正成为研究的热点。如图 3，是可重构计算在时间和空间利用率上和其他两种计算平台的比较。



图 3三种计算平台在灵活性和计算效率上的对比

* + 1. 可重构计算技术概述

可重构计算属于一种新型的计算方式，有其特别的计算架构，它的架构介于通用处理器架构和专用处理器架构之间。传统的通用处理器架构的实现是通过一套固定的硬件资源通过在时间维度上去调用不同的资源实现对一个任务的处理，其通过将代码编译成一条条时间上连续的指令，在通过指令执行期间去调度数据，从而完成数据的流动；专用处理器架构则是通过特定的硬件结构如乘法器、加法器等，通过将软件算法之间映射成硬件电路，通过利用空间维度上的优化实现计算任务，算法要实现的功能都能在硬件资源上体现， 原始数据顺序地流过相应的处理单元，数据就会被处理，从而完成计算。而可重构计算结合了两者的长处，其具体的执行流程如下：

首先，针对任务A，将任务A对应的代码编译成相应的配置信息，这一步相当于通用计算平台上将代码编译成一条条的指令。然后将配置信息导入到可配置硬件电路中去，可配置的硬件电路被固化成针对A任务的ASIC电路，这一步类似于ASIC电路开发中的直接算法映射。通过这样的处理，相当于任务A同时被在时间维度和空间维度上展开，兼顾了灵活性和高性能。如图 4，是可重构计算系统、通用计算系统和专用计算系统的架构对比图。

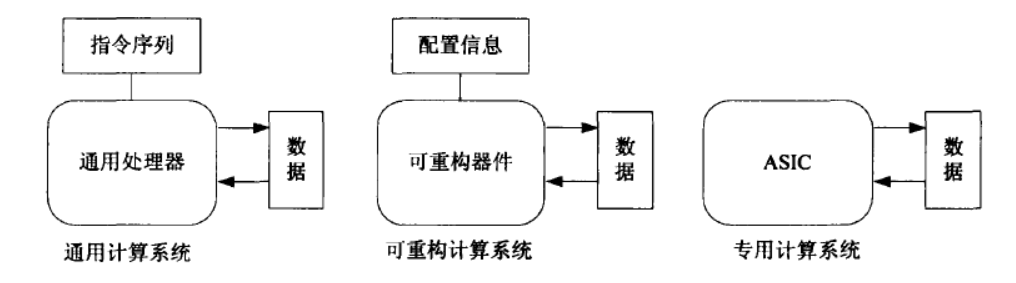


图 4三种计算平台系统架构对比图

一般来讲，可重构计算系统包含可重构计算阵列、任务控制器和存储器及控制部分组成，如图 5，可重构计算阵列是可重构系统的核心，它负责数据处理和数据的传递，是可配置性的实现主体；任务控制器负责对系统进行控制，负责对运行在计算阵列上的任务进行控制，实现数据的输入和输出，它时刻监控着计算阵列和整个系统的运行情况，进行整个系统的任务调度，是整个系统能成功运转的核心；存储器及控制部分负责阵列的数据存取，由于可重构计算阵列和主处理器构成一个异构计算平台，所以可重构计算阵列需要有自己的本地存储器，这一部分主要实现阵列数据的存取以及和主处理器进行数据交换。

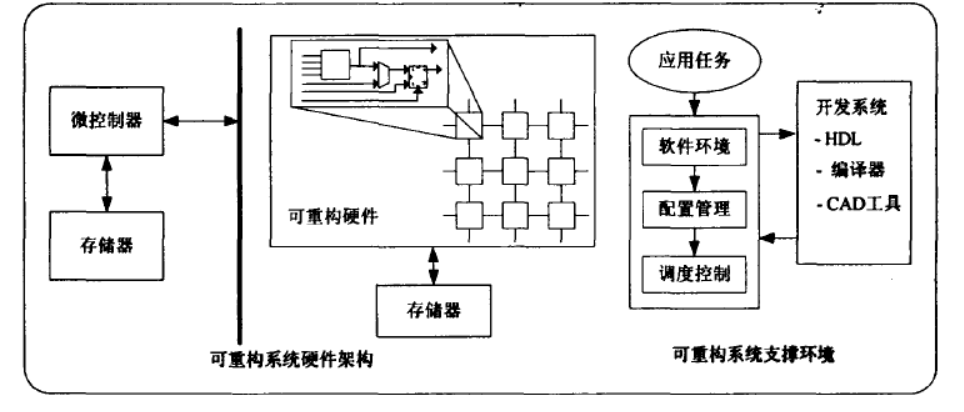


图 5可重构计算系统的组成

可重构处理器系统是充分利用了时间特性和空间特性的一种计算系统，如果时间特性占主要性时，就是通用计算平台；如果空间特性占主要性时，就是专用计算平台。下面通过举一个例子来说明可重构计算在时空利用程度上与通用计算系统和专用计算系统的不同。假设我们要完成一个多项式的操作，如下：(改成for循环)

三种计算系统的时空利用情况如图，



图 6三种计算系统在某一应用上的时空利用图

第一种实现方式是通过专用计算系统实现的，其通过将算法中所用出现的算子全部在空间上展开，多项式中有三个乘法操作，两个加法操作，所以通过设计三个乘法器两个加法器，让数据进入到相应的处理单元中进行处理，得出计算结果，几个处理单元同时进行，在几个时钟周期就可以计算出需要的结果，这种计算架构根据算法本身进行定制，具有计算效率高的特点。

第二种实现方式是通过通用计算系统实现的，通过将多项式的内容编译成一条条时间维度上的指令，指令中包含需要执行的操作和操作需要的数据来源寄存器地址，按照时间顺序进入到统一的计算单元中进行计算，并将计算结果存储到相应的寄存器中，准备好给下一条指令使用。通过这种时间维度上的推移和空间维度上的复用，完成了一个计算任务。如图所示，这种计算结构只需要包含一些寄存器和算术计算单元，这种结构具体的执行流程是这样的：所给定的表达式被编译器编译成5条指令，这些指令依次会出现在指令寄存器中，程序计数器会在每个时钟周期从指令寄存器中取出一条指令送到译码单元，译码之后每一条指令需要执行的操作和数据来源寄存器是哪一个就知道了，之后算数逻辑单元会从指令指定的寄存器中取出数据进行计算，当该程序对应的所有的指令被执行完毕，就得到了相应的计算结果。这种计算模式的优势是所需要的硬件资源是固定的，适用范围较广，但是由于指令是在时间维度上展开的，这种结构面临着计算效率低的困难。具体表现在：首先，这种体系结构依赖于指令集，指令集限制了具体硬件电路的适用范围，也就是说硬件电路支持的操作种类完全依赖于指令集中规定的指令数，由于指令个数的限制，在遇到复杂的代码的时候需要将复杂的功能拆分成几条简单的指令来实现(这也是RISC指令集的主要特点和优势)，但是这种做法很明显降低了计算效率；其次，通用处理器的设计初衷就是为了实现通用计算，必然要实现指令集中规定的指令操作，这就有可能面临着资源浪费的可能，假设设计了一个64位的通用平台处理器，但是只用于处理16位的程序，那么处理器中很多的系统资源都是浪费的。所以，这种时间域上的计算平台满足了通用性却也牺牲了计算效率。

第三种方式是通过可重构计算系统来实现，可重构计算系统结合了以上两种计算系统的优点，充分利用了时间域特性和空间域特性，从整体来看时间域的体现是任务级的，不在是指令级的，也就是说针对不同的任务，可重构计算系统都是支持的，空间域特性体现在针对每一个具体的任务，可重构计算系统看起来是一个专用计算系统，每一个任务提供一套配置包，类似于通用计算中的指令，在任务中电路的存在形式是一个ASIC电路，这两者的结合使得可重构计算既可以具备灵活性又可以具备较高的计算效率。具体到上面的例子，针对可重构计算系统它的流程是这样的：首先，整个硬件电路是由一系列的可配置计算单元构成的，多项式被分解成3个乘法和2个加法，所以会被生成两个加法配置包和3个乘法配置包，将配置信息固化到配置阵列上，整个硬件电路就形成了一块专用电路，注入原始数据就可以执行相应的操作，最终得出相应的计算结果。可重构计算允许改变电路计算功能来适应不同的算法和应用，是一种动态的体系结构，这种动态变化保证了阵列的灵活性，在阵列被固化之后形成的专用硬件电路保证了计算性能。所以，在计算密集型需求场景中，可重构计算是一种较为合适的解决方案。

* + 1. 可重构计算关键技术

**协同编译器技术**：基于冯诺依曼结构的通用处理器从诞生以来发展到现在，基于通用处理器的编译器经过了这么多年的研究，其在编译技术上已经相当成熟了。通用处理器上的编译器是和处理器平台和高级编程语言紧密结合下产生的。能完全实现全自动化。可重构系统的编译技术只是在起步过程中，现阶段的可重构系统和通用处理器系统协同的工作方式都依赖于手工分配任务来实现的。也就是说通用处理器上的代码和可重构阵列上的代码时分别使用不同的编译技术实现的，通用平台上的代码被编译成一条条指令通过通用CPU来执行，可重构系统的代码被编译成一段配置信息通过可重构阵列中的计算单元和互连网络来固化，现阶段对任务的划分没有一套完整的编译系统来支持。

针对于像可重构阵列和通用处理器组合成异构的计算平台来讲，协同编译技术的实现显得尤为关键，这种编译技术需要实现从源程序中自动划分出主处理器执行的代码和可重构阵列处理器执行的代码，再分别调用两种计算平台的编译器实现代码的编译，同时还要有队整个代码任务进行调度的能力，根据硬件资源和当前任务执行情况动态的进行任务的调度和调优。

**粒度**：可重构计算最早的产物就是现场可编程门阵列（FPGA）,FPGA的配置单元的最小位宽是1bit，所以FPGA是粒度为1的可重构计算平台，也就是细粒度可重构阵列。通常来讲，我们把配置单元的最小位宽小于等于4bit的可重构阵列成为细粒度可重构阵列，大于4bit的称为粗粒度的可重构阵列。

一般来讲，在进行可重构系统设计时粒度的选择需要考虑很多方面的因素，粗粒度阵列和细粒度阵列在阵列结构、组成单元、配置方式、执行机制等方面有着很大的区别。细粒度阵列是由查找表、逻辑门等构成，执行机制是进行比特级的操作；而在粗粒度阵列中，处理的都是字级的数据，内部也是需要包含完整的功能单元，需要有集成很多算子的算术逻辑单元。另一方面，选择不同的粒度会导致整个可重构系统具有不同的配置信息量，具有不同的配置时间和切换配置的功耗，最主要的影响是两种不同的粒度的计算效率是不同的，所以在进行可重构系统设计时要充分考虑到这些因素。

**静态和动态重构**：可重构系统中可重构的实现来自于两方面，首先是计算单元的可重构，也就是说可以改变某一个计算单元要实现的功能，另一方面来自于计算单元之间的连接关系可重构，可以根据具体的任务来改变互连形式，从而达到可重构的目的。所以可重构系统在时间上就有两种表现形式，重构的系统在某一段时间内其结构不发生变化，这是静态的可重构系统。早期的FPGA是这种系统的典型，优点是配置切换的次数少，功耗低，但是缺点是必须等到一个任务已经执行完成并已经保存现场之后才能开启配置的切换，通过终端程序执行的方式来完成任务的切换。

另一种是重构系统在某一段时间内会根据配置信息的具体内容来实现计算单元和互连的动态切换，每一条配置都会激活一次配置额切换，这种技术把程序执行的过程糅合在配置的切换过程中，其优点是配置的切换是在程序的执行过程中进行的，可以减少配置的时间开销，缺点是需要的配置信息比较多，整个可重构系统的功耗比较大。

**互连网络：**可重构系统的实现不仅依赖于每一个计算单元的功能可以根据需要变化，同时依赖于可以改变的互连网络。如图 7所示，互连网络可配置的原理是通过多选一的MUX电路实现，通过选择信号将两个处理单元连接在一起实现数据通路，其中的选择信号就是配置信息中的某一位或几位。

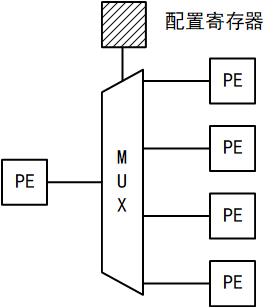


图 7互连网络可配置原理图

可重构系统的互连网络是复杂的，通常来说有两种方案可供选择。第一种是全连接的方式，即处于阵列中的每一个计算单元都可以和出自己以外的任何一个计算单元连接，如图 8所示，假设阵列中有9个处理单元，处理单元A要实现全连接的功能，针对单元A的某一个输入端口就需要设计一个8选1的MUX电路，这种情况下，随着阵列硬件资源的增多，这种选择电路的规模是成倍的增长的，很显然对于规模较大的可重构系统的实现，这种方案不是最好的选择。第二种方案是处于可重构阵列中的某一个计算单元只可以和自己周边的上下左右相邻的计算单元连接，如图 9，单元A的输入端口只能和单元B,C,D,E连接，对于单元A的某一个端口都是固定的4选1的MUX电路就可以实现，实现电路比较简单固定，易于实现。

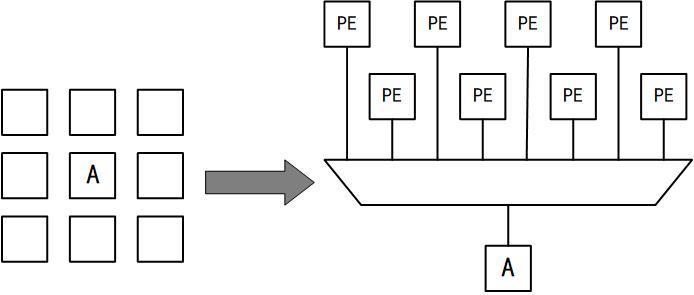


图 8互连网路全连接

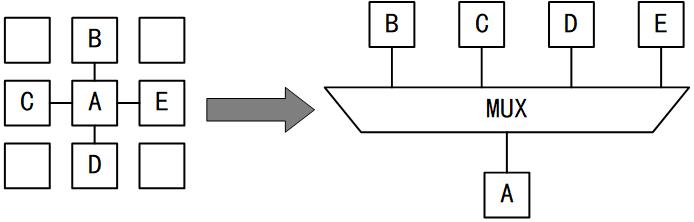


图 9互连网路局部连接

比较这两种方案，各有自己的优缺点，第一种可以提供较大的灵活性，布局布线算法可以设计的较为简单，但是实现的代价是需要消耗较多的硬件资源，布线网络也变得很复杂。对于规模较小的可重构系统可以选取这种方案。第二种方案的优点就是互连方式较为固定，对于可重构系统的布线器实现较为简单，同时硬件电路实现复杂度较低，但是灵活性不够，需要强大的布局算法的支撑。

**存储系统：** 基于冯诺依曼结构的通用处理器平台的存储系统是分层次的，从速度快到慢分类看，分别是基于SRAM技术的cache、基于DRAM技术的内存以及存储器，这种层次的存储系统是在处理器速度比较快而存储器速度比较慢的情况下提出来的，使得处理器系统和存储器系统能更协调的工作。在可重构阵列中，存储系统越来越成为其计算性能的瓶颈。存储系统设计中有两个这要问题，首先，可重构处理器和通用处理器组合成的一个系统，需要协同工作完成某一个具体的任务，从外部看来，这是一个多核系统，需要考虑存储系统的一致性的问题。现在通用的做法是，需要DMA在一个任务开始的时候将阵列运算所需要的数据全部搬运到可重构阵列的存储器上，在整个阵列运行期间阵列的运算数据都是和自己的存储系统进行交互，只有在任务结束之后才会将整体的数据再搬运到这处理器的存储系统上。

第二个问题是可重构计算属于空间计算范畴，其较大的并发性对存储器的压力是很大的，所以可重构处理器的存储系统要采用分层次、分布式的存储系统。如图所示，根据作用范围的不同，需要有global memory,share memory和local memory，global memory是整个阵列的全局存储器，整列运行的早期数据存储上面。Share memory是整个阵列中各个计算单元数据交互的媒介，可以通过share memory进行数据的分享和输送，local memory是每一个处理单元所拥有的的存储单元，

* + 1. 可重构计算技术分类

1. 按照可重构阵列和主核处理器间的耦合关系

根据文献x所述，可重构系统通常需要有一个以上的主处理器和一个以上的可重构阵列处理器构成，所以这两者之间的位置关系是值得研究的问题。

如图a所示，可重构阵列是作为独立的外设设备和主核处理器组成一个可重构计算系统，可重构阵列和主核处理器之间的数据传输是通过输入输出总线实现的，很显然数据传输的速度会受到很大的影响，这种结构不适合处理那种需要和主核处理器进行频繁数据交换的应用。但是这种结构设计简单，主核处理器的选择更多，所以也被广泛使用。

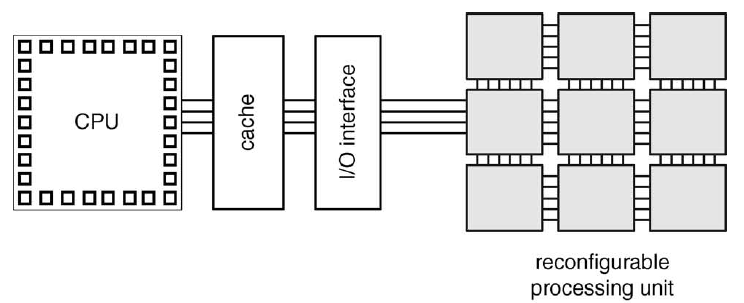


图 10可重构处理单元与CPU耦合关系一

如图b所示，可重构阵列作为片外设备挂在主核处理器上，可重构阵列和主核处理器之间的书籍传输是通过片上网络实现的，这种实现结构在数据传输速度上要明显好于如图a所示的情况。但是在主核处理器和可重构阵列之间有cache的存在，使得数据一致性的问题变得很关键，这个问题的解决将成为这种结构能否被正确使用的关键。

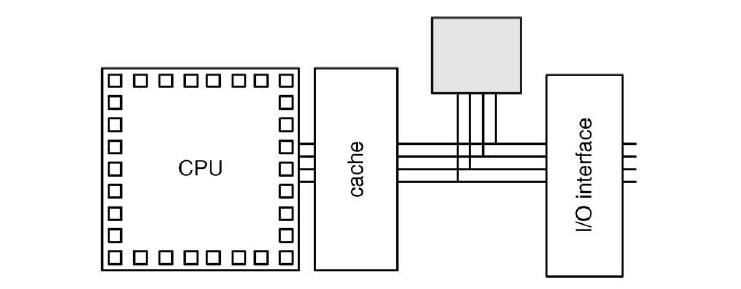


图 11可重构处理单元与CPU耦合关系二

如图c所示，可重构阵列处理器处于和主核处理器同等的地位上，可重构阵列处理器作为主核处理器的协处理器，他们组成了一个多核异构计算系统，使用同一套的存储系统，包含cache系统，I/O处理系统，存储系统等，这是一种较为常见的耦合方式，本论文研究的混合粒度可重构阵列结构的设计也是采用这种耦合方式来实现的。这种结构下主核处理器将配置阵列的配置信息和原始数据发送大可重构阵列上，阵列在配置完成之后自己开始激活计算，主核处理器较少的参与到阵列的计算中去，相较于前面几种耦合方式，这种耦合方式可以大大减少可重构阵列和主核处理区之间通信代价。是一种使用相当广泛的可重构系统。

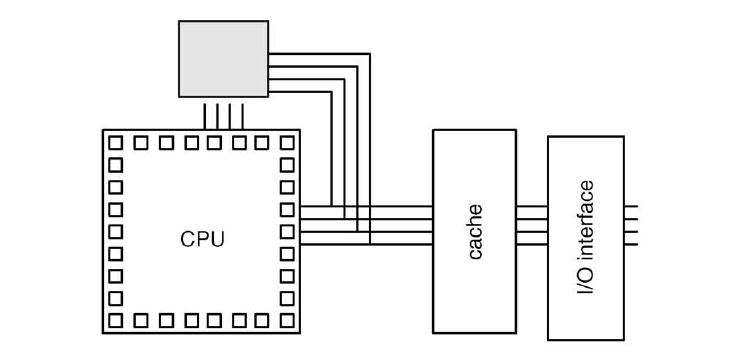


图 12可重构处理单元与CPU耦合关系三

如图d所示，是一种更为紧密的耦合方式，可重构阵列作为一个硬件模块嵌入到主核处理器中，在这种结构下，可重构阵列模块具有较高的权限，主核处理器的资源对可重构阵列模块都是可见的。

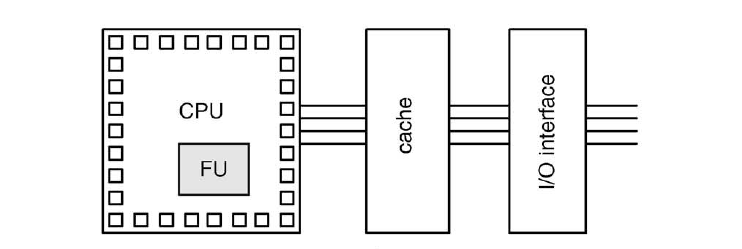


图 13可重构处理单元与CPU耦合关系四

如图e所示，主核处理器嵌入到可重构阵列中去，主核处理器负责对整个可重构阵列进行控制，例如Xilinx最新的FPGA就是这种结构，通过嵌入一块通用处理器核，使得整个可重构系统变得更加的灵活多变。同时在可重构阵列中嵌入一个通用处理器核的好处不仅如此，粗粒度的可重构阵列适合于做重复的大量计算的任务，但是其对于任务中出现的控制代码几乎没有处理能力，所以嵌入的这一个通用处理器核可以协助可重构阵列解决其在控制能力上不足的问题。两者的结合组成了一个高效的系统。

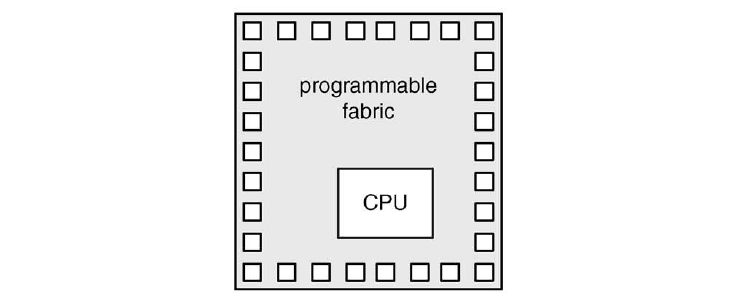
、

图 14可重构处理单元与CPU耦合关系五

1. 按照阵列的空间维度划分

线性阵列处理机：(LAP)

方形阵列处理机：（SAP）

1. 按照可重构阵列的粒度划分

可重构系统中粒度是指配置信息操作的最小位宽，根据这个位宽的不同可以将可重构阵列划分为粗粒度可重构阵列、细粒度可重构阵列和表达式粒度可重构阵列，不同粒度的可重构阵列适用于不同的应用场景，也具有不同的计算效率和功耗等其他特性。

细粒度可重构阵列[2]：(FGRA)细粒度可重构阵列是指配置的最小位宽为1bit的可重构阵列，其典型的代表就是FPGA，它是由可配置的逻辑单元、触发器、查找表等组成，由于其可配置的逻辑单元是比特级的，所以原理上可以实现任意逻辑表达式的可配置操作,为了实现用户指定的逻辑功能，FPGA内部的可编程逻辑单元是由多输入的查找表（LUT）、寄存器等组成，通过丰富的互联资源实现逻辑块之间的重构，从而实现逻辑功能。

但是，从上面的介绍中也可以看出，这种比特级可配置功能的实现的代价就是配置信息的增多，比如说要实现一个多比特的复杂的电路，就需要较多的配置信息，通常可以达到几兆到几十兆[3]；配置时间较长，由于大量的配置信息需要固化到大量的逻辑可配置单元上，这是配置时间通常可以达到几十毫秒；这些会导致整个配置功能单元的配置时间变长、配置功耗变大，同时配置单元之间的互连也变得尤为复杂。还有一个尤为关键的问题是，FPGA中的可配置是静态的，一次配置完成之后其具体的电路形式已经确定，不能再运行期间更改电路形式。所以，在整个FPGA中加入通用计算平台芯片来协助可配置电路完工一些功能，减少可配置电路负载的做法得到了应用。比如Xinix的ZYNQ 7020中嵌入了基于ARM的通用处理器。

粗粒度可重构阵列：(CGRA)它是指可重构单元的最小位宽大于4bit的可重构电路。和细粒度可重构阵列中最小的逻辑单元是查找表不同，粗粒度可重构阵列中最小的逻辑单元是算数逻辑单元(ALU),其中集成了各种通用的算子，比如说加法、乘法、乘累加等。同时，由于处理的数据位宽不同，CGRA和FGRA所面向的应用场景是不一样的，细粒度的可重构阵列具有较高的灵活性，只要硬件资源够多，理论上可以生成任意的电路结构，但是缺点也不少；而粗粒度可重构阵列由于其处理单元和互连网络都是字级的，所以其应用场景可以面向密集计算型的场景，比如图像处理、视频处理等。但是其是面向特定领域的，所以在灵活性上有所欠缺。

相比较于FPGA来说，CGRA最大的优势是其配置的对象是多比特的，将多个比特捆绑在一起进行配置的方式大大减少了配置量，这解决了FPGA中配置信息过多的问题。由于配置信息的减少，其在重配置期间的配置时间也大大减少。支持字级操作的另一个好处就是逻辑单元之间的互连变得简单多了，不要要复杂的互连算法就能实现电路的完美配置。同时在进行这种粗粒度的重构时，多套配置之间的切换具有更小的切换代价，这些都是粗粒度可重构阵列的优势。



图 15粗粒度处理器(CGRA)和细粒度处理器(FPGA)特性对比

* 1. 理论优势

本文研究的阵列加速器是将数据流驱动的先进技术和可重构先进技术结合在一起，支持动态数据流技术和任务级可配置技术。在充分保证循环代码没有数据依赖性的前提下，可以实现较快的循环流水。

1. 硬件实现的软流水，不需要软件控制硬件自动控制流水进行

图所示的是矩阵相乘的示例代码，从图中可以看出要完成这两个矩阵的相乘，要进行L\*J\*K次循环，矩阵规模越大消耗的时间越多。常规的处理方式是一次一次的展开循环，每一层的循环串行的被执行。通用处理器执行这段代码也是在某一层的循环执行结束之后再返回到循环头处判断条件，再确定要不要执行接下来的代码。

for(int i =0;i < L;i++)

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

for(int k=0;k<K;k++)

{

C[i][j] += A[i][k]\*B[k][j];

}

在可重构计算上可以通过消耗大量的空间计算资源来实现循环层之间的流水，

1. 流水深度较大，PE被设计成有buffer，可以存储更多的循环层数
2. 动态数据流，
3. 混合粒度可重构阵列结构设计
   1. 系统模块化划分

整个结构的设计是分模块的，如图1所示，阵列的中间部分是粗粒度处理单元阵列CPE，左右两侧分布的是细粒度处理单元阵列FPE，阵列上、下分别是Load和Store单元，再加上控制器和全局存储器、本地存储器，构成一个完整的阵列结构。



整个阵列的执行过程是这样的：当主核执行到加速代码段时，主动生成跳转指令，将其发送给可重构阵列中的任务控制器，任务控制器解码指令后，发送控制信号给配置控制器，配置器把该任务的所有配置从全局的存储器中取出，按照处理单元ID分别存储到处理单元的配置寄存器中，各个计算单元的配置工作完成；同时，任务控制器不断检测配置信息完成情况，一旦检测到有配置信息配置完成，即发送执行信号给整个阵列，启动信号注入成功，相应配置包被激活，各个处理单元的功能和互连关系被确定,每个计算单元独立计算，基于数据流驱动的原则，FPE实现对循环的控制，CPE实现循环体的计算，LE/SE实现对数据的load和store操作，当循环结束的时候，阵列发送结束信号给任务控制器，任务控制器接着激活下一套配置包，直到所以的配置包信息都被执行完毕之后，任务控制器反馈任务结束信号给主核，整个任务执行完成.

* 1. 粗粒度阵列结构设计
     1. 输入输出buffer
     2. 可配置ALU单元
     3. 本地寄存器
     4. 本地配置器

粗粒度处理单元是计算的实际载体，对CPE的结构设计关系着整个结构的计算效率。如图2所示， 粗粒度处理单元是由输入buffer、输出buffer、可配置功能的ALU、LR(local reg)、配置寄存器和可配置MUX构成。

配置寄存器的深度可以达到4层，可以同时存储4x32bit的配置信息。任务控制器激活某一配置包，配置信息会体现在CPE内部的配置MUX上，通过MUX的选通功能，CPE内部的数据通路被建立起来，当所需数据准备好之后，便可以激活相应功能的计算.

输入buffer是CPE中的关键部件，它能够实现对输入数据的缓存并且对其进行匹配输出。当输入的两个数据带有相同的tag标记时，被认为是相同批次的数据，可以发射进入ALU运算单元，再结合ALU中的配置码实现对该数据对的操作,完成配置中规定的操作。输入buffer可以提供存储空间，使其成为未匹配数据的中转站，基于此，该结构支持循环迭代多发射功能，多层循环间的数据被分布式的存储在各个不同的CPE的输入buffer中，tag匹配机制保证了不同循环层数之间的数据不会冲突，有力支撑了循环动态展开。



LR(local reg)是CPE中的另一个重要部件，既可以作为常量寄存器，还可以作为通用寄存器。LR输入输出端口的可配置功能，使得该寄存器可以灵活使用，方便快捷的构建CPE内部的数据通路。当ALU的输出连接到LR的输入，LR的输出连接到ALU时，就可以实现累加等操作。

* 1. 细粒度阵列结构设计
     1. 循环头组合节点
     2. 细粒度控制算子

针对计算密集或数据密集类算法，其代码的主要内容是实现计算，没有太多的循环控制和分支跳转，粗粒度阵列对这种类型的算法有很好的支持，但是对于控制密集型的算法粗粒度阵列结构不能完美的支持。正是由于这种限制我们提出了基于细粒度阵列实现代码控制流图（CFG），粗粒度阵列实现代码数据流图(DFG)的思想，提出让细粒度节点作为代码中循环控制和分支跳转等功能的实现载体，让粗粒度节点专注于计算。



如图3(a)所示,是一段循环结构的C语言代码，图3(b)是这段代码展开一层循环的CDFG图，从(b)图可知，循环是从a节点开始的，也中止于a节点，a节点的功能就是每次有一个结束信号输入的同时就产生一个输出激活信号，由于这些信号属于控制信号，位宽不超过两位，适合于使用细粒度的处理单元进行处理。通过细粒度节点a可以实现循环步进和循环边界的控制.图中b节点是一个选择通路的节点，对应于代码中就是if-else结构，也是一种细粒度功能的节点。

如图4所示，细粒度节点的输入输出的位宽是2bit,输入数据经过入口控制逻辑后，进入逻辑块，逻辑块内部包含可配置的各种算子，通过配置信息来实现特定的细粒度节点功能。从而实现对代码中循环控制和分支跳转的支持。



* 1. 混合粒度阵列的优化技术
     1. 多发射循环迭代技术

在传统处理器上循环程序被编译成顺序执行的指令，新的循环迭代必须要等到上一次循环迭代结束之后才开始执行。如图3(b)所示，一般的执行流程是store节点结束之后，a节点才开始激活下一次的循环迭代。这种执行方式显然不适合HGRA结构，本文提出了一种多发射循环迭代的技术，实现了循环体在阵列上流水。如图5B所示，矩形表示的是每一次的循环迭代，多发射技术实现的是在循环迭代1还没有结束的时候循环开始控制节点就发射循环控制变量，激活循环块2，再激活循环块3，以此类推，按照这种方式实现了循环间的流水，最优情况下细粒度节点可以每一个时钟节拍发射一个循环控制变量，直至循环边界，大大提高了计算效率。



* + 1. 空间展开循环迭代技术

上面介绍的多发射循环迭代技术是发掘程序中的时间可重叠性，由于HGRA结构中在某一固定时间窗口内CPE执行的功能固定，这为将更多的运算需求分配到其他闲置的计算单元上提供了可能。空间展开循环迭代技术充分挖掘了程序中的空间可并行性，如图5C所示，循环迭代1,2,3在空间上展开，属于不同循环块的计算单元可自主执行计算任务，不相互干扰，性能获得较大提升。如图6是两个4x4矩阵相乘的例子，使用空间展开技术，一个时钟节拍可以完成第i行与第j列所有元素的相乘，即阶段1完成的操作，通过使用4个CPE同时执行将代码中最内层的循环展开到空间阵列上，整个代码从3层循环减低到2层循环，降低了循环的维度.



通常针对某一种算法，可以同时使用两种优化技术，如图6所示，四个乘法的CPE是空间展开循环迭代技术的应用，该技术的应用可以降低循环层数，减少配置复杂度。同时多发射循环迭代技术的应用，使得阶段1到阶段3以流水线的方式执行计算,充分挖掘了时间可重用性.两种技术的应用为高效率计算创造了条件。

* 1. 阵列反馈机制
     1. 数据流和循环间依赖
     2. 多级反馈机制
  2. 应用映射实例
     1. 矩阵乘应用直接映射
     2. 矩阵乘应用的优化实现

1. 混合粒度可重构阵列的仿真实现
   1. 仿真原理概述

随着电路设计的规模越来越大，电路架构变化的频度越来越快，传统的在电路设计阶段进行整个电路的仿真验证的方法已经不能适应这种快速变化的场景，使得电路架构设计人员变得被动，不能快速调整设计方案，整个电路的研发周期因此变得很长，容错成本极高。所以在整个电路架构设计阶段有一款能快速验证架构设计人员所设计的电路架构是否满足设计要求的工具变得极为重要。一般的电路设计中，这个阶段通常是缺失的，主要依靠架构设计人员丰富的专业技能来保证。

在架构设计初期，需要确定整个电路的功能和性能，在这一阶段设计的电路是一种较高层次的行为级电路，所以比较适合对整个电路进行软件上的建模，通过软件来描述所要设计电路的具体的功能模块，通过软件的方式实现每一个模块中的功能，这就是软件仿真器需要实现的功能。

设仿真器是一套软件工具，在芯片架构设计中扮演着重要的作用。通常来讲，我们要运行一段程序在计算机系统上需要有以下几个必要条件，

操作系统----对程序任务进行划分和调度

编译器----将代码编译生成硬件电路能识别的指令

硬件电路----执行具体的指令

在以上三种条件中，缺少任何一项程序都不能被完整的运行，以上三个条件的关系如图所示，层次关系是越往下越重要。现在我们面对的情况是最底层的硬件电路是我们需要设计的，那么其他任何基于硬件电路上的工具都没有办法使用。而软件仿真器的实现原理是通过将需要设计的电路的软件模型构建出来，运行在通用的处理器平台上，如图，通过已经存在的一套环境来构造一套软件工具，软件仿真器试运行在确定的操作系统之上，但是这是对所要设计电路的一种抽象，运行在通用平台的软件仿真器从外部来看就是需要设计的硬件电路的模拟。



图 16程序运行在硬件平台上



图 17程序运行在仿真器平台上

* 1. 仿真器框架

本论文所设计的混合粒度可重构阵列仿真器是基于X86平台，Windows10系统下，基于visual studio开发。如图18所示，仿真器需要描述出阵列上的处理单元、执行控制器、任务调度控制器、主核接口和存储器系统等模块部分。仿真器本身提供的是一个框架，会根据读进来的原始配置文件和原始输入数据重构模拟的电路结构。整个仿真系统分成以下几个部分：

组件系统：该部分主要是由静态的模块组成，通过软件代码描述硬件底层的各个组件，将其作为一种使用的实体存在。这些模块是独立的，静态的，是一个仿真系统必不可少的部分。

数据流控制系统：这部分是在读入配置信息后，模拟硬件行为，将配置信息分别存储到每一个可重构单元的配置寄存器中去，再通过配置寄存器配置处理单元内部的配置MUX，使其形成配置信息指定的电路，形成配置电路后仿真器从原始输入读取数据，数据流开始进入到阵列，控制系统的职责就是要在此时对数据流的流动进行协调和控制，控制的规则就是架构设计时预定的规则，包括处理单元外部数据流动规则和处理单元内部数据流动的规则。有了这些规则的保证和控制系统对这些规则的执行，使得仿真系统能能真实的模拟出这些规则是否是正确的。

任务控制系统：在可重构阵列和通用CPU组成的异构计算平台中，任务的划分和调度至关重要，而任务控制系统在这里面扮演者重要的作用，控制系统需要在主核协调下，对需要加速任务进行划分，同时协调任务段需要被执行的硬件资源地址，对整个计算阵列进行监控，在某一硬件执行资源执行完毕之后要对阵列资源进行回收和初始化，同时将新的任务段安排到该执行块上去。所以，任务控制系统是进行任务级调度的枢纽，关系着任务执行的效率。

存储系统：任何一种计算平台的存储系统都是复杂的，在前面介绍过可重构阵列系统的存储系统是分层次的。由于可重构阵列是和主核处理器组成一个多核系统，在进行仿真器设计的时候需要实现多层次的存储系统，包括为仿真器提供原始数据的外部存储器、一定容量和速度的内存以及cache缓存。这些组件作为外挂模块连接在仿真器上，在整个仿真器运行期间提供访存支持。

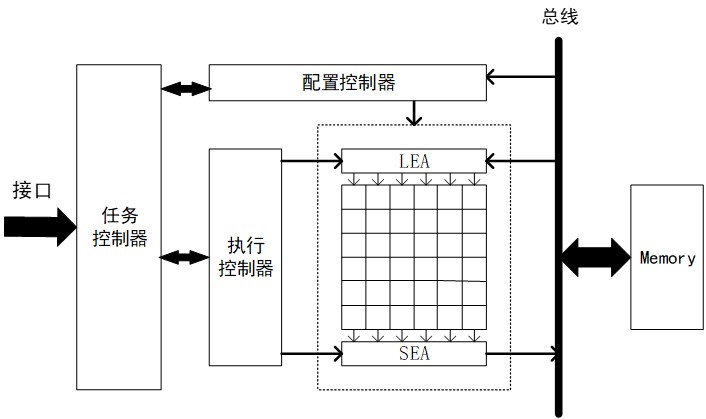


图18仿真器实现的模块框架

* 1. 仿真器实现及特性

时钟周期是怎么实现的？

* 1. 仿真效果对比与分析

1. 总结和展望
   1. 论文总结
   2. 展望

参考文献

1. Estrin G, Bussell B, Turn R, et al. Parallel processing in a restructurable computer system[J]. IEEE Transactions on Electronic Computers, 1963 (6): 747-755.
2. 段然, 樊晓桠, 高德远, 等. 可重构计算技术及其发展趋势[D]. , 2004.
3. 陶文卿, 毛志刚, 何卫锋. 面向媒体处理的可重构阵列的结构设计与研究[D]. 上海: 上海交通大学, 2010.
4. 袁开坚. 基于粗粒度可重构处理器的任务映射技术研究[D]. 战略支援部队信息工程大学, 2018.

致谢