高性能处理器体系结构
可重构计算专题

读书报告

杨晓晖(BA07011001)yangxhcs@mail.ustc.edu.cn

冯晓静(SA07011002)bangyan@mail.ustc.edu.cn

赵　琼(SA07011013)qiongz@mail.ustc.edu.cn

裴建国(SA07011083)ustcowen@mail.ustc.edu.cn

俞华铭(SA07011053)yhm2007@mail.ustc.edu.cn

王　仁(SA07011089)wangren@mail.ustc.edu.cn

张志雄(SA07011090)zzxiong@mail.ustc.edu.cn

中国科学技术大学计算机科学技术系

2007年12月

目　录

1 可重构计算概述

1.1 引言

传统的计算方式主要有两种[1][8]。一种是专用集成电路（Application Specific Integrated Circuit，ASIC），以硬件方式执行特定的操作。由于ASIC专为某一特定计算任务设计，因此可以快速高效地执行所针对的计算任务。然而，ASIC一旦生产出来后就再也无法改变其功能。即使只要修改其中一小部分功能，也不得不投入巨资重新进行设计和生产，显然在生产量达不到一个很大的规模时，这个代价是很昂贵的。另外一种是通用微处理器（General Purpose Processor，GPP），执行一系列指令来完成特定的计算任务。如果计算任务发生变化，通过改写程序即可改变软件中所包含的指令序列，GPP通过执行不同的指令序列即可完成不同的计算任务，而硬件无需做出任何改动。因此GPP非常具有灵活性，但是这种灵活性是以牺牲性能为代价换取的。处理器为了完成一个计算任务，需要从存储器中逐条取得指令和数据然后加以译码和执行，每次操作都需要很大的执行开销。所以GPP的性能远远落后于ASIC。

可重构计算（Reconfigurable Computing，RC）正是为了填补硬件实现和软件实现之间的空白而提出的，它试图在达到比软件实现更高性能的同时，还能提供一定的硬件实现灵活性[1]。

可重构计算的定义多种多样。Bondalapati和Prasanna将可重构计算定义为“通过后生产方式（post-fabrication）和处理单元的时空编程连接（spatially and temporally programmed connection）实现的计算”[5]。DeHon和Wawrzynek认为可重构计算是“一种具有两个重要特征的计算机组织类型：在生产后可以被定制用来解决任何问题；能够在很大程度上利用空间定制的计算来完成计算任务”[4]。Hartenstein则把可重构计算资源称为“活件（Morphware）”、“配件（Configware）”和“流件（Flowware）”[7]。

如图1-1所示，可重构计算本质上可以看作是“在（微处理器的）灵活性和（ASIC的）性能之间的一个折衷，时间计算和空间计算的一种结合”[7]。同时，可重构计算还可以看作是“在灵活性和面积/功耗之间的一个折衷”[2]，如图1-2所示，基于FPGA实现的可重构处理器或者嵌入式可重构逻辑可以使用比ASIP或者GPP更低的面积和功耗，来提供比ASIC更好的灵活性。图1-3则进一步揭示了可重构计算“空间计算和时间计算相结合”的本质[4]。

可重构计算具有如下特征[5]：

- 空间计算：数据处理由空间分布的计算来实现；
- 可配置的数据通路：通过配置机制，互联网络和计算单元的功能可以在运行时动态变化（以适应不同的计算任务需求）；
- 分布式控制：计算单元基于本地配置来进行数据处理；
- 分布的资源：计算所需的资源，如计算单元和存储器，都是分布的。

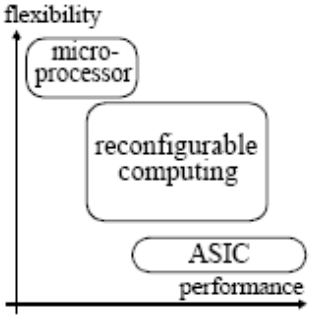


图1-1 可重构计算填补了ASIC与微处理器之间的空白[7]

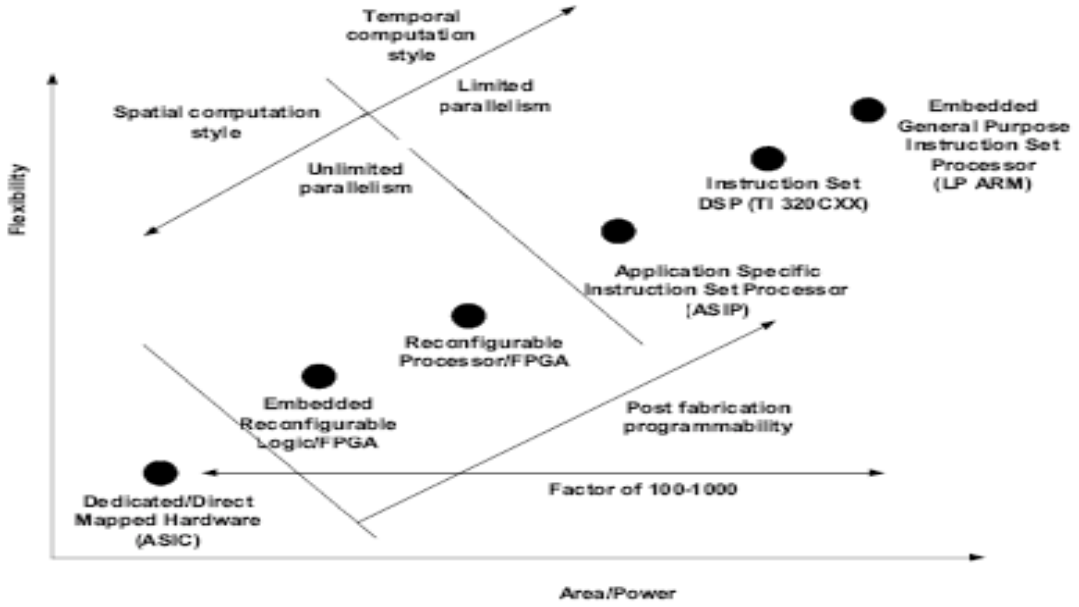


图1-2 可重构计算是在灵活性和面积/功耗之间的一个折衷[2]

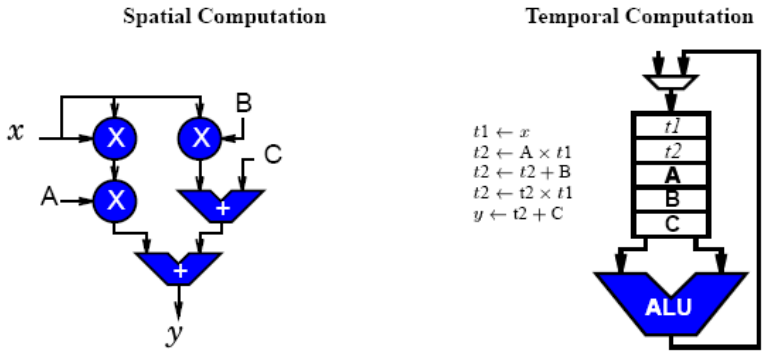


Figure 1: Spatial versus Temporal Computation for the expression $y = Ax^2 + Bx + C$

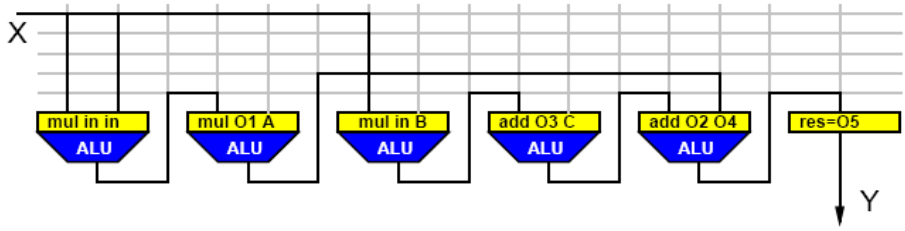


Figure 2: Spatially Configurable Implementation of expression $y = Ax^2 + Bx + C$

图1-3 可重构计算是空间计算和时间计算的一种结合[4]

1.2 可重构计算分类

可重构计算总的来讲可以按照逻辑块粒度、可重构单元耦合程度和重构方式来进行分类。

1.2.1 按照逻辑块粒度分类

大多数可重构硬件都是一个基于多个基本功能单元组合而成的阵列，这个基本功能单元就是逻辑块。所谓逻辑块的粒度是指由映射工具综合而成可重构硬件的最小功能单元的大小和复

杂度[8]。按照逻辑块的粒度，可重构计算可以分为如下五类[2][3][5]：

- 甚细粒度体系结构：逻辑块能够实现1-3输入的处理能力，如Xilinx6200系列FPGA；
- 细粒度体系结构：逻辑块一般实现4输入的基本处理功能，如Altera FLEX10K；
- 中粒度体系结构：逻辑块可以实现更加复杂的4输入的处理功能，如Garp；
- 粗粒度体系结构：逻辑块能够实现字宽度的复杂运算，如RaPid；
- 甚粗粒度体系结构：每个逻辑块就相当于一个微处理器，如RAW。

逻辑块粒度越细，单个逻辑块的功能就越简单，可重构硬件的功能就越灵活，但是电路集成度就越低，逻辑块之间的连线就越多，所需要的面积和功耗也会越大；反之，逻辑块粒度越粗，单个逻辑块的功能就越复杂，电路集成度就越高，逻辑块之间的连线就越少，所需的面积和功耗也较低，但是可重构硬件的灵活性会变差[3][5]。

1.2.2 按照可重构单元耦合程度分类

如图1-4和图1-5所示，可重构计算按照可重构处理单元（Reconfigurable Processing Unit，RPU）与主处理器之间的耦合程度或连接方式，可以分为如下四类[2][3]：

- 连接I/O总线的RPU，其作用相当于一个外部的独立处理单元（Standalone Processing Unit）；
- 连接局部总线的RPU，其作用相当于一个附加的处理单元（Attached Processing Unit）；
- 与主处理器直接连接的RPU，其作用相当于一个协处理器（CoProcessor）；
- 集成到主处理器内部的RPU，其作用相当于主处理器的一个功能单元（Function Unit）。

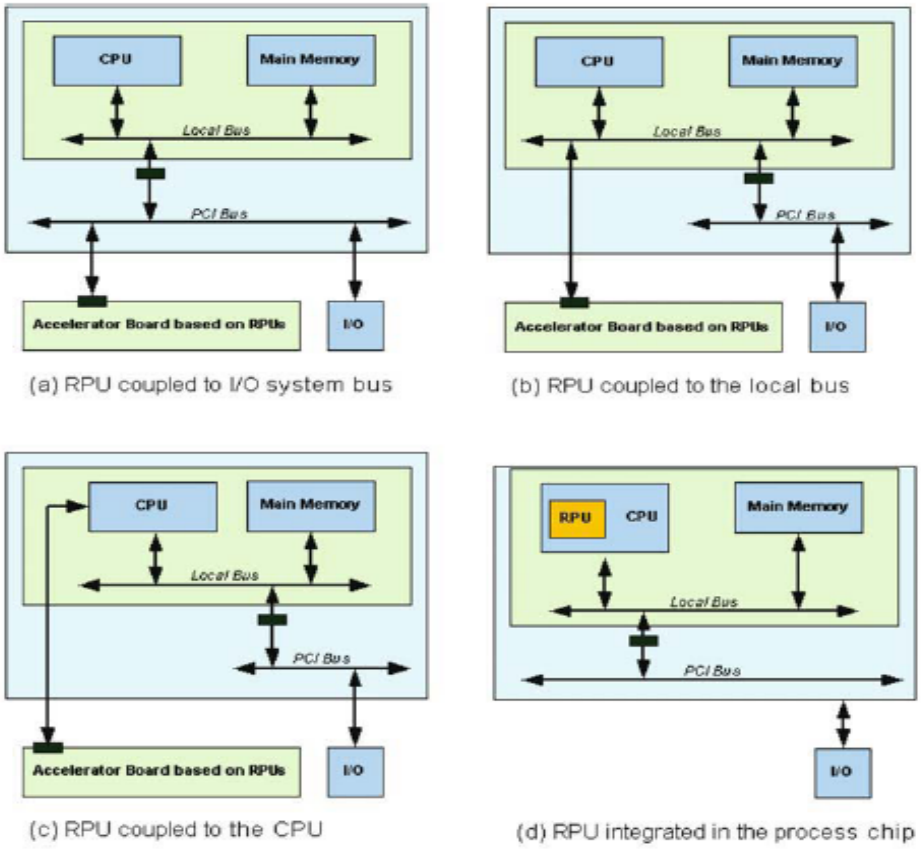


图1-4 RPU与主处理器之间的连接方式[2]

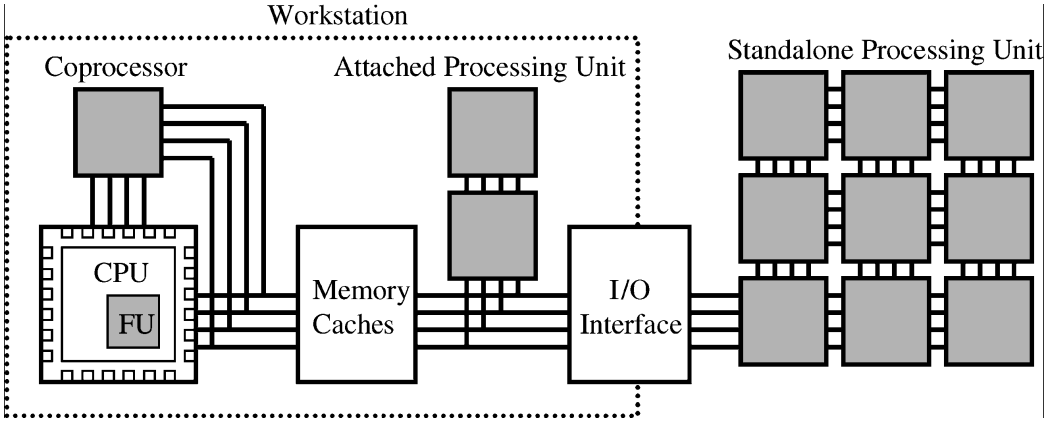


图1-5 可重构系统的耦合程度[3]

1.2.3 按照重构方式分类

简单地，重构方式分为静态重构（static reconfiguration）和动态重构（dynamic reconfiguration）两大类[1]。

静态重构又称编译时重构（compile time reconfiguration），是最简单、最常见的一种重构方式[2]。如图1-6所示，静态重构在系统编译时完成配置信息的生成，在执行前进行逻辑块的重构，系统在投入运行后逻辑块的功能不能改变，除非系统终止运行[1][2]。

动态重构又称运行时重构（run-time reconfiguration）[2]。如图1-7所示，系统可以在运行时收集所需的状态信息，然后据此生成新的配置信息，并在系统运行过程中完成逻辑块的重构，从而改变其功能[1]。

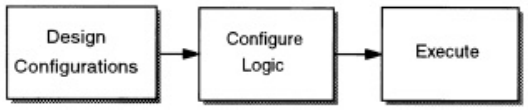


图1-6 静态重构[1]

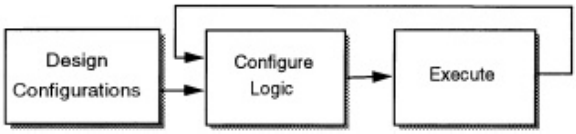


图1-7 动态重构[1]

动态重构又可以进一步分为单上下文可重构（single context reconfigurable）、多上下文可重构（multi-context reconfigurable）、部分可重构（partially reconfigurable）[2]，以及流水线可重构（pipeline reconfigurable）[3]，分别如图1-8、图1-9所示。

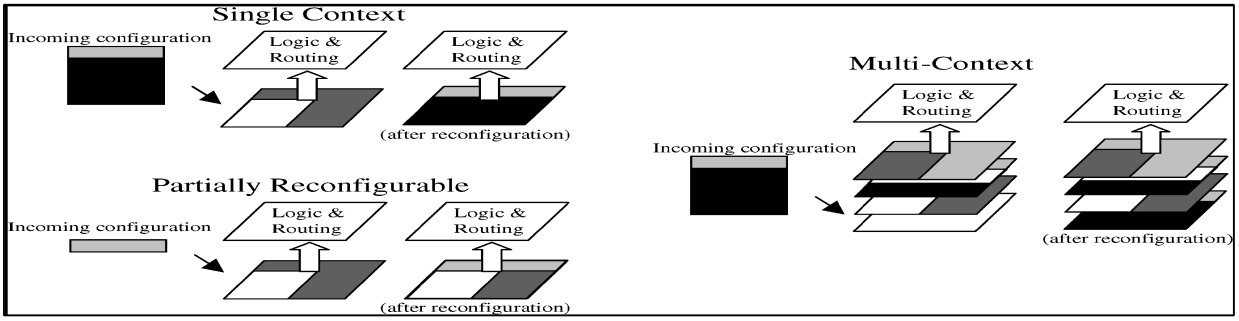


图1-8 动态可重构方式[2]

在单上下文可重构体系架构中，配置信息以串行流（serial stream）的形式存储在配置存储器中。当发生上下文切换时，即改变系统的配置时，需要重新装入新的配置信息，才能完成对可重构部件的配置，因此配置信息在串行流中的分割会对重构延迟产生较大影响。Xilinx4000系列就属于这种架构。

在多上下文可重构体系结构中，每个配置位对应着配置存储器中的多个存储位，但是任一时刻只有一个存储位处于激活状态。当需要上下文切换时，只需要改变配置位和存储位的对应关系即可实现系统的快速重构。多上下文重构相当于多个单上下文重构的堆叠实现。典型产品有Chameleon公司的CS2000系列。

部分可重构体系结构针对配置只需要使用全部可重构硬件的一部分，或者只有一部分配置需要改变的情况。部分可重构硬件可以在一部分硬件运行的同时改变另外一部分硬件的配置，这是通过给每个可配置位编址访问来实现的。部分可重构硬件体系结构可以进一步减少重构时间，缩短重构延迟。Xilinx6200系列和Virtex系列都属于可部分重构的体系结构。

流水线可重构体系结构可以看作是部分可重构体系结构的一个特殊的例子，它可以实现流水线各段的部分重构[3]。如图1-9所示，重构过程是逐段进行的，这意味着配置信息要先于数据信息到达流水线的各段。完成硬件配置的流水段可以进行相应的运算。在整个流水线部件中，配置和运算是在不同的流水段中交叉进行的。

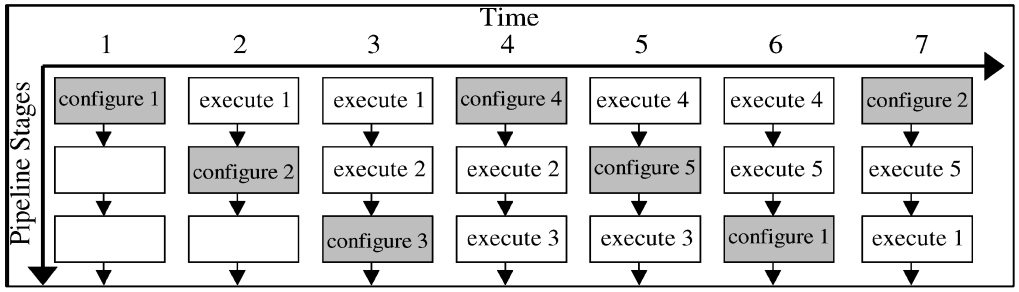


图1-9 流水线可重构体系结构[3]

1.3 可重构计算体系结构

1.3.1 设计综合方法

利用可重构体系结构来实现更高的计算性能，需要开发高效的配置设计技术。但是现有的设计方法都是基于ASIC设计工具的、传统的逻辑综合方法，无法充分发挥可重构硬件的加速潜能，其最大问题就在于设计过程[1]。如图1-10的左图所示，逻辑综合方法利用行为模型进行设计，但是算法的语义和本质却在软硬件的映射过程中丢失了。

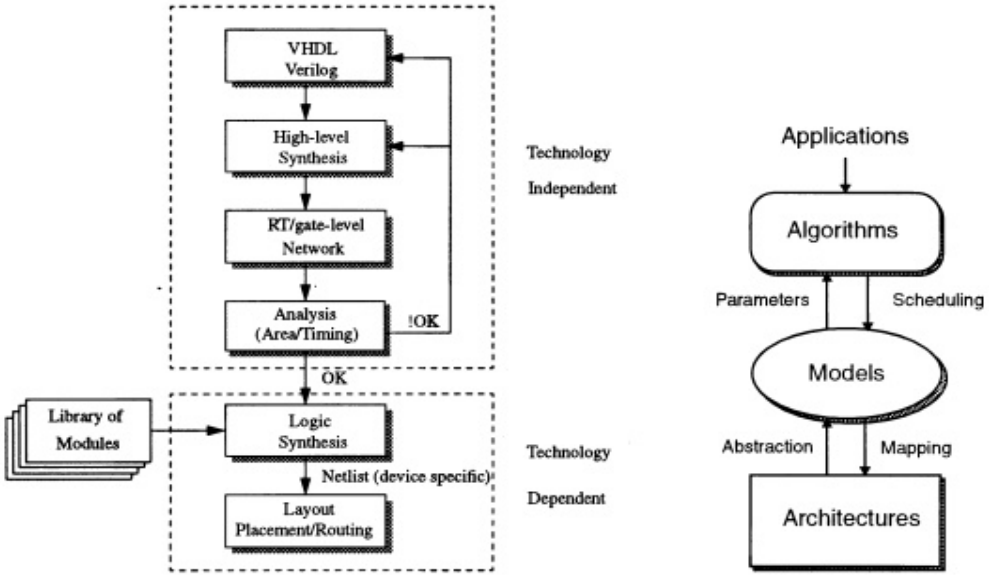


图1-10 逻辑综合方法与算法综合方法[1]

算法特定且实例相关的配置是实现高加速比的一个关键所在。基于算法的综合方法[1]，如图1-10的右图所示，提出了一个“算法-模型-体系结构”的综合设计方法。算法经过调度成为模型，模型进而映射为体系结构；反之，体系结构抽象为模型，模型再参数化为算法。这种局部优化的设计方法可以获得更好的面积与时延性能。

1.3.2 可重构计算体系结构

可重构体系结构从现场可编程门阵列（Field Programmable Gate Array，FPGA）发展而来，现在已经有很多种类的商业化FPGA产品面市，各种各样的可重构计算系统也随之被构建起来。这些系统从体系结构上可以分为FPGA体系架构和混合体系结构两大类[1]。

基于FPGA体系结构的可重构计算系统由一个组合逻辑块阵列以及一个与之相连的互连网络组成。逻辑块作为FPGA的基本组成单元，由查找表（Look Up Table，LUT）、D触发器和配置信息存储器组成。通过改变反熔丝元件的开关状态或者SRAM存储器中存储的比特信息，逻辑块与互连网络都可以实现重新配置。如图1-11所示的是典型的多FPGA板级可重构体系结构，可以用来构造更大规模的可重构计算系统。在这种多FPGA体系结构中，每个FPGA都拥有局部存储器，彼此之间通过互连网络连接从而可以实现更加灵活的重构。这种多FPGA体系架构的实验产品有DECPeRLe、SPLASH-2和Teramac，商业化产品则有Annapolis公司的WILD系列产品。

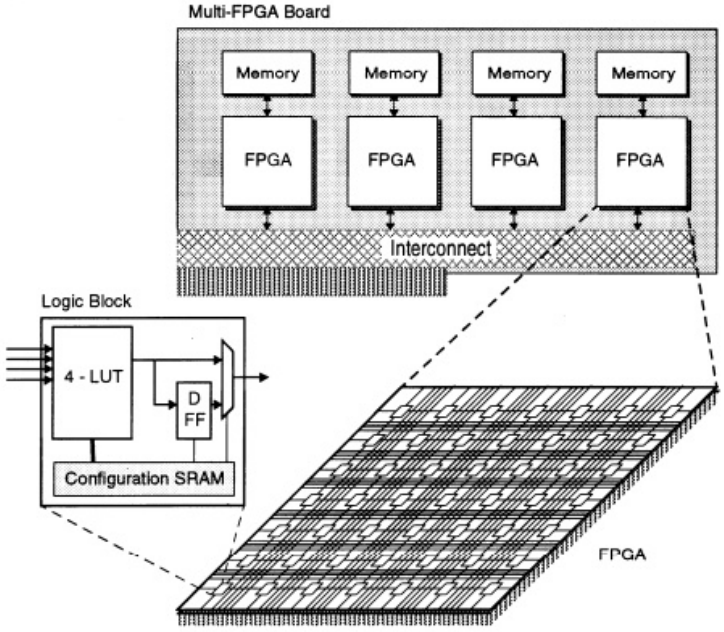


图1-11 典型的FPGA板级可重构体系结构[1]

混合体系结构的可重构系统，面向特定应用把宿主机系统（host system）与可重构计算部件连接到一起，可以进一步提高硬件加速性能，减少通信开销。集成的方式是多种多样的：

- 片上系统（Systems-on-chip，SoC）
- 可配置片上系统（Configurable system-on-chip，CSoC）
- 可重构片上系统（Reconfigurable systems-on-chip，RSoC）
- 可编程片上系统（Systems on programmable chip，SoPC）

现在混合体系结构的可重构系统已经开始尝试将可重构逻辑集成到通用处理器、通用控制器或者专用集成电路芯片内部，如Berkeley的Garp，Triscend的E5，以及Chameleon系统，后两个已经实现了商业化。

1.4 可重构计算模型

计算模型是对体系结构的高级抽象，可以用于开发算法技术，以实现从应用到体系结构上的映射。可重构计算与传统的冯·诺依曼式计算有所区别，因此可重构计算模型也与传统的计算模型不尽相同。对应可重构计算体系结构，有两种可重构计算模型：可重构网状模型和混合系统结构模型（Hybrid System Architecture Model，HySAM）[1]。

可重构网状模型是采用可重构总线体系结构的多个处理器组成的超大规模集成电路阵列的一个理论模型。在可重构总线体系结构中，多个处理单元（Processing Element，PE）构成的多维阵列通过一定数量的I/O端口连接到总线上。作为可重构网的基本计算单元，PE由开关、本地存储空间和ALU组成，可以被设置为特定的连接模式以实现特定的处理功能。如图1-12所示，通过改变总线上开关的状态，可以实现处理单元的功能重构。可重构网状模型可以由多个参数来刻画，如PE的数据宽度，信号传输延迟，共享/互斥总线访问，开关连接模式等。

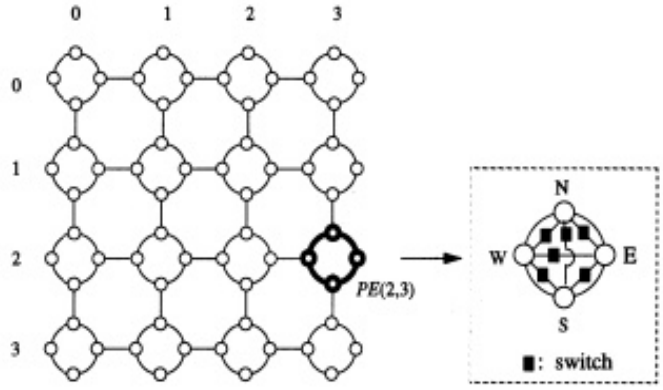


图1-12 可重构网[1]

混合系统结构模型是一个参数化模型，表示的是一个把可重构逻辑附加到传统的微处理器上构成的可重构计算系统。HySAM模型把硬件的能力（capacity）与实现（implement）明确分开，同时以清晰的接口展示给用户。如图1-13所示，HySAM模型由传统的微处理器、标准的存储器、可配置逻辑单元（Configurable Logic Unit，CLU）、可配置存储器以及与互连网络通信的数据缓冲区组成。右图为模型的一个具体实例，说明了各部分之间的连接方式。

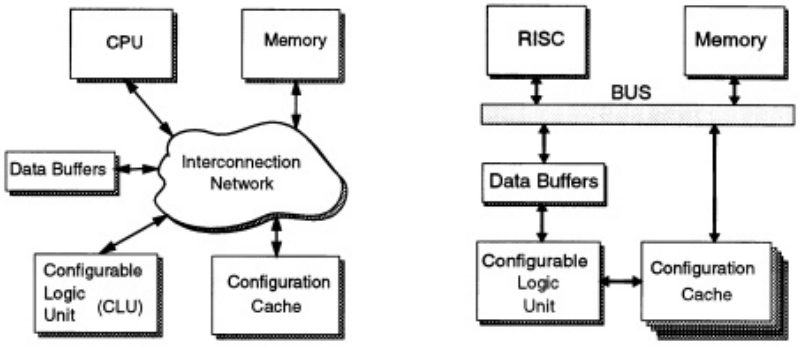


图1-13 混合系统结构模型及其实例[1]

对于一个可重构系统来说，不同配置的硬件实现具有不同的面积、时间、精度、功耗等特征。作为一个参数化的模型，HySAM提供了一系列的参数来描述和衡量这些特征：

- F：可在CLU上执行的功能F_1、F_2、……、F_n的集合，表示硬件的能力；
- C：CLU上可以实施的配置C_1、C_2、……、C_n的集合，表示硬件的实现；

- A_{ij}：使用配置C_j实现功能F_i时的属性的集合；
- R_{ij}：从配置C_i切换到配置C_j所需要的重构开销；
- B：互连网络的通信带宽，以字节/周期计；
- k_d，K_d：分别从片上存储器或者外部存储器中访问数据的开销，以周期/字节计。

利用上述这些参数，可以给各种不同类型的混合可重构系统进行建模，并可以根据应用的实际需求选择合适的体系结构来加以映射实现。

可重构系统总的执行时间等于各个配置的执行时间与配置切换时的重构时间之和。可以利用动态和部分重构技术，结合配置缓存（configuration caching）、配置预取（configuration prefetching）、配置压缩（configuration compression）等技术来减少重构时间[1][3]。

1.5 可重构计算算法

将应用映射到硬件上并随后执行的设计流程如图1-14所示[1]。应用及特定问题实例被计算模型用来生成执行该应用的计算体系结构，然后利用体系结构特征和基于模型的分析结果生成硬件执行所需的配置，最后这些配置被用来重构硬件和执行应用。还有可能利用计算的中间结果可以更新配置和重构硬件。

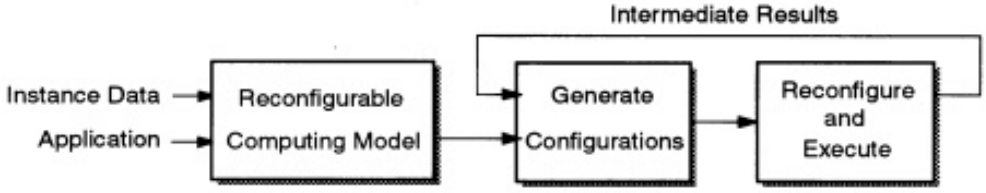


图1-14 在基于模型的方法中的配置与重构[1]

接下来介绍两个算法的可重构实现，一个是基于可重构网状模型的算法实现，一个是基于HySAM模型的动态精度管理算法实现。

基于可重构网状模型可以实现算法中常见的基本运算如与、或、异或、加、乘等，通过改变互连网络中相应配置比特位的值，就可以实现计算功能的重构，可以广泛应用于图像处理、计算几何、图论等领域。如图1-15所示是EXOR运算的一个可重构硬件实现[1]。

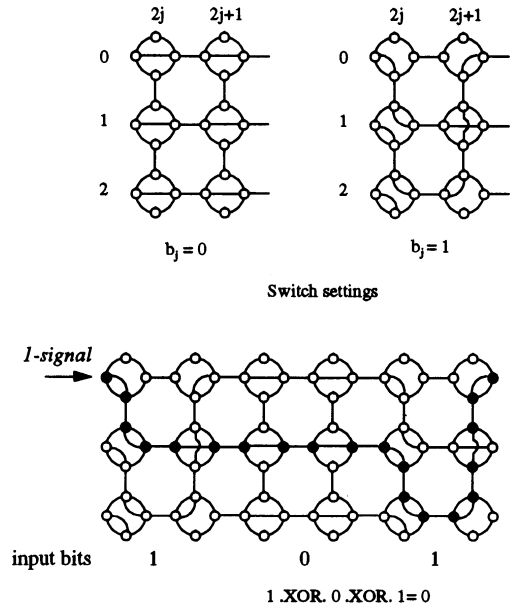


图1-15 EXOR运算的可重构硬件实现[1]

排序算法也可以基于可重构网状模型的硬件计算来实现。可以将排序过程看作子序列排序及子序列间数据移动的过程。这方面的例子是利用可重构网状硬件实现Leighton八段列排序算法，其核心思想是利用可重构硬件实现不同阶段的排序、交换和归并。

可重构硬件比ASIC拥有更大的灵活性，因而可以被用来实现更加多样化的计算集。可重构硬件的一大优势就在于其具有执行不同精度算术运算的能力。细粒度实现的可重构硬件可以构造不同大小的计算模块以适应运算精度的变化，使部件执行时间更短、所需硬件资源更少、重构速度更快，从而实现更高的计算性能。这种性能加速在循环中体现的尤为明显。在循环中动态选择精度并重构硬件的策略包括静态重构、用贪心算法重构、用动态精度管理算法（Dynamic Precision Management Algorithm，DPMA)重构、用运行时DPMA算法重构等[1][6]。以如下循环程序为例，其中标号为20的程序行中MAXQ * SCALE(I)这一乘法运算可以进行硬件加速实现。

```
      DO 20 I=1,N
            DO 10 J=1,N
                  RSQ(J) = RSQ(J)+XDIFF(I,J)*YDIFF(I,J)
10                IF (MAXQ.LT.RSQ(J)) THEN
                        MAXQ = RSQ(J)
            POVERR = POVERR / MAXQ
20          VIRTXY = VIRTXY + MAXQ * SCALE(I)
```

采用各种重构策略的算法硬件实现的执行时间如图1-16所示。由测试结果可以看出，采取运行时DPMA算法的硬件实现总的执行时间最短。

Table 1. Execution times using different approaches

Algorithm	Execution time (ns)	Reconfiguration time (ns)	Total (ns)
Standard	655360	20480	675840
Static	532480	17920	550400
Greedy	468010	56320	524330
DPMA	471160	33280	504440
DPMA-run	409600	15360	424960

图1-16 采用不同重构策略实现动态精度运算的执行时间[6]

1.6 问题讨论

问题一：可重构计算的适用场合

可重构计算的研究迅速兴起，很大程度上是因为它在基本不损失硬件性能的前提下提供类似软件的灵活性。那么是不是可以说，可重构计算的适用场合正是那些使用硬件实现可以明显加速同时还需要一定灵活性的应用呢？

问题二：可重构计算的价值衡量体系

当一个应用采取可重构计算加以实现时，如何衡量可重构计算实现的价值？单纯使用总执行时间来计算加速比？还是应该结合硬件实现以及重构的代价（包括价格以及时间方面的代价）全面衡量？全面衡量时指标体系如何构建？是否加权？如何加权？

问题三：可重构计算的通用编程模型

目前可重构计算的实现大多与具体的软硬件平台紧密相关且针对特定应用。能否建立一个通过的可重构计算编程模型，以统一的或者一致的视图来描述目标系统的软硬件划分以及功能模块的软硬件实现？

本章小结

可重构计算是为了填补硬件实现和软件实现之间的空白而提出的，它试图在达到比软件实现更高性能的同时，还能提供一定的硬件实现灵活性，本质上可以看作是在微处理器的灵活性和ASIC的性能之间的一个折衷，同时也是时间计算和空间计算的一种结合。

可重构计算可以按照逻辑块粒度、可重构单元耦合程度和重构方式来进行分类：

- 按照逻辑块粒度，分为甚细粒度，细粒度，中粒度，粗粒度和甚粗粒度体系结构；
- 按照可重构单元耦合程度，分为嵌入主处理器的功能单元，协处理器，附加处理单元和外部的独立处理单元；
- 按照重构方式，分为静态重构和动态重构，后者又进一步分为单上下文重构，多上下文重构，部分重构和流水线重构。

传统的逻辑综合方法，无法充分发挥可重构硬件的加速潜能。基于算法的综合方法提出了“算法-模型-体系结构”的设计方法。算法经过调度成为模型，模型进而映射为体系结构；反之，体系结构抽象为模型，模型再参数化为算法。这种局部优化的设计方法可以获得更好的性能。基于此，可重构计算体系结构分为FPGA体系结构和混合体系结构两大类。目前的趋势是将可重构处理单元集成到微处理器、微控制器或者ASIC内部以降低面积、功耗和通信开销。

计算模型是对体系结构的高级抽象，可以用于开发算法技术，以实现从应用到体系结构上的映射。可重构计算模型可以分为可重构网状模型和混合系统结构模型HySAM。其中HySAM是一个参数化模型，能够把硬件的能力与实现明确分开，同时以清晰的接口展示给用户。利用F、C、A_{ij}、R_{ij}、B、k_d、K_d等参数，可以给各种不同类型的混合可重构系统进行建模，并可以根据应用的实际需求选择合适的体系结构来加以映射实现。

基于可重构网状模型可以实现算法中常见的基本运算，通过改变互连网络中相应配置比特位的值，就可以实现计算功能的重构。而基于混合系统结构模型实现的动态精度管理算法则具有明显的性能加速。

参考文献

[1] Kiran Bondalapati, Viktor K. Prasanna. Reconfigurable Computing: Architectures, Models and Algorighms[J]. *Current Science, Special Section on Computational Science,* 2000, 78(7):828-837.

[2] Konstantinos Masselos, Nikolaos S. Voros. Introduction to Reconfigurable Hardware, Chapter One of *System Level Design of Reconfigurable System-on-Chip*[M]. Springer, 2005, pp.15-26.

[3] Katherine Compton, Scott Hauck. Reconfigurable Computing: A Survey of Systems and Software[J]. *ACM Computing Surveys*, 2002, 34(2):171–210.

[4] Andr´e DeHon, John Wawrzynek. Reconfigurable Computing: What, Why, and Implications for Design Automation[C]. *Proc. of DAC'99*, June 1999, pp.610-615

[5] Kiran Bondalapati, Viktor K. Prasanna. Reconfigurable Computing Systems[J]. Invited Paper, *Proc. of the IEEE*, 2002, 90(7):1201-1217

[6] Kiran Bondalapati, Viktor K. Prasanna. Dynamic Precision Management for Loop Computations on Reconfigurable Architectures[C].*Proc. of FCCM'99*, April 1999, pp.249-258

[7] Reiner Hartenstein. A Decade of Reconfigurable Computing: a Visionary Retrospective[C], Invited Paper, *Proc. of DATE2001*, pp.642-649

[8] Katherine Compton, Scott Hauck. An Introduction to Reconfigurable Computing. *Internal Report,* April 2000

2 案例分析之一：可重构超级计算

2.1 引言

可重构超级计算的三个重要组成部分是高性能微处理器，可重构逻辑部件，以及高带宽低延迟的网络互连。目前可重构逻辑部件(FPGA)为很多的应用带来灵活性和性能上的提升，如何将可重构技术应用于超级计算中，成为可重构超级计算要解决的主要问题[1]。

2.2 论文工作

目前，可重构超级计算系统有多种体系结构。SGI RASC的结构如图2-1所示，CPU和FPGA都是对等体，对等体之间通过NUMAlink互连。SRC MAPStation则是将FPGA放置在主板上DIMM槽的位置，通过SNAP接口与1.4GB/s的速度与CPU相连，其结构如图2-2所示。

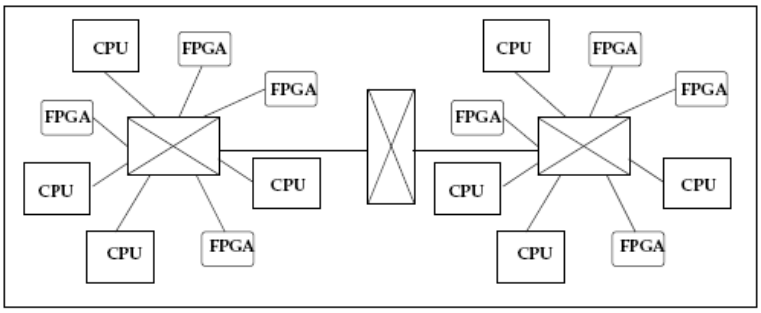


图2-1 SGI RASC的结构

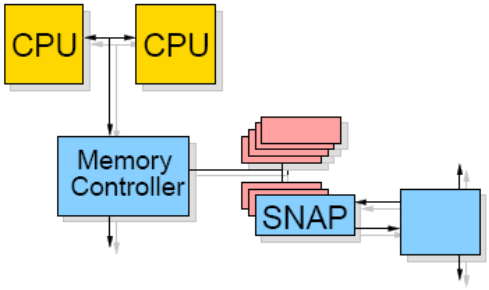


图2-2 SRC MAPStation的结构

当前的可重构器件（FPGA）有很好的性能，可以带来量级上的性能提升，但是如何将可重构技术方便有效地利用于超级计算系统呢？

代数函数库方法是一种解决方案。这种方法基于的是大部分科学计算中,代数函数库的计算部分在可重构超级计算系统中所占的比例很大这个事实.因此,将一些常用的代数函数用可重构硬件加速,并作为库例程封装起来,提供给应用来调用。目前已经在可重构部件（FPGA）实现了线性代数函数。科学计算通常是通过高级语言来表示的，如果使用自动化工具进行线路设计，得到的结果通常比手工设计结果要差，这是因为用自动化工具很难实现优化设计。如果可以将优化过的设计以线性函数库的形式保存起来，这样既能得到很好的性能，又能简化设计过程。

虽然从理论上分析，线性函数库在应用上应该能体现出很大的性能提升，然而事实并非如此。一些应用实例显示，可以用线性函数库进行硬件加速的计算部分在可重构超级计算系统中所占的比例很小，大概在10%左右。那么根据Amdahl定律，即使硬件加速的加速比为无穷大，所得到的性能提升也才仅仅只有11%。从而，又出现了新的问题，那就是如何提高硬件加速部分在整体计算任务中的比例。

要想提高可用硬件加速部分在整个计算任务中的比重，需要仔细地对应用进行分析，合理地划分软硬件功能。文献[1]中列举了两个应用实例，一个是波特兰地区道路交通系统的仿真，一个是分子动力学方面的一个应用。在这两个应用实例中，通过合理划分软硬件功能，得到了较高的性能加速比。

在城市道路交通系统中，道路系统被建模成节点和连接，节点用来模拟交叉路口，因为这部分的活动比较复杂，所以用软件实现；连线用来表示一段单向道路，这部分都可以用硬件来实现。这样理论上有80%的计算可以用硬件加速。在实现过程中，为了克服可重构资源不足的困难，Los Alamos National Laboratory使用了一种流处理的方法，最终在基于Cray XD1的平台上得到了20%的性能加速比。

在动力学仿真实例中,通过对应用的分析,发现计算分子间nonbonded Forces在整个计算时间中的比例是75%左右。因此将计算nonbonded Forces的工作交给硬件来完成，FPGA从存储器中读取计算需要的数据，在完成计算后，将数据写回到存储器中。在实现过程中，通过改写计算nonbonded Forces的算法和write_back技术，增加了指令间的并行度。最终的实验是在MAPstation这个平台上完成的,可以得到2倍的加速比。

2.3 问题及讨论

在分组讨论的过程中，我们发现了一些问题，通过扩展阅读相关论文并通过组员之间的讨论，这些问题都已经解决。问题和讨论结果如下：

1) Cray XD1的体系结构

Cray XD1由多块底盘组成，每块底盘有6个SMP处理器模块，每个SMP模块中包含两块皓龙200系列处理器，这样每个底盘中有12块处理器。每个SMP模块中有一块Virtex-II pro系列的FPGA，每块FPGA唯一地对应一块处理器。底盘的示意图如图2-3所示[1]。

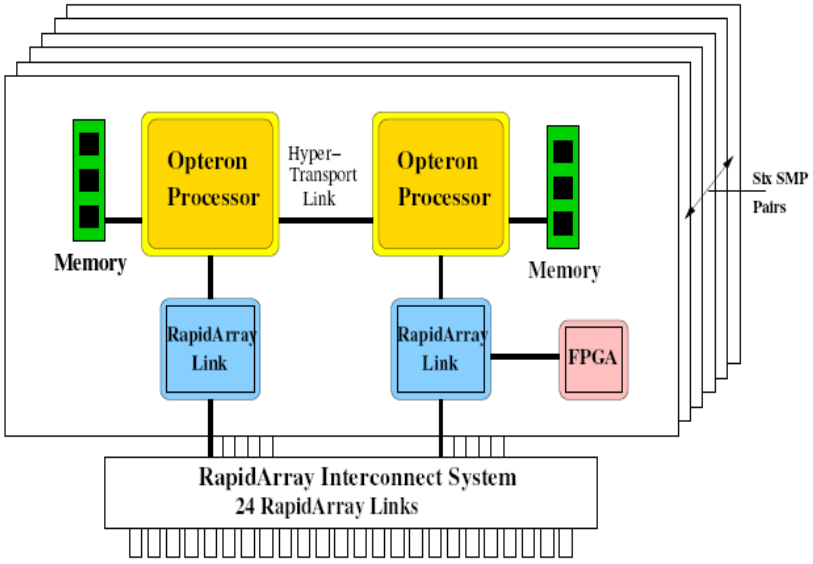


图2-3 Cray XD1处理器模块结构

处理器与FPGA的互连方式如图2-4所示，每块处理器有一3.2GB/s的通道与同一SMP模块中的另一块处理器相连，另外还有两条2GB/s的通道连往两个相邻的SMP模块。每块FPGA有一条3.2GB/s的通道连向与它对应的CPU，相邻SMP模块中的FPGA也有2GB/s的通道相互连接。至于为什么FPGA之间也需要专用通道连接呢？这是因为，某些比较复杂的逻辑功能需要用可重构逻辑来实现，然而一块FPGA所含的可重构逻辑资源可能不够，在这种情况下，就需要靠这些通道

将FPGA聚集在一起，共同实现一个较复杂的功能模块[2]。

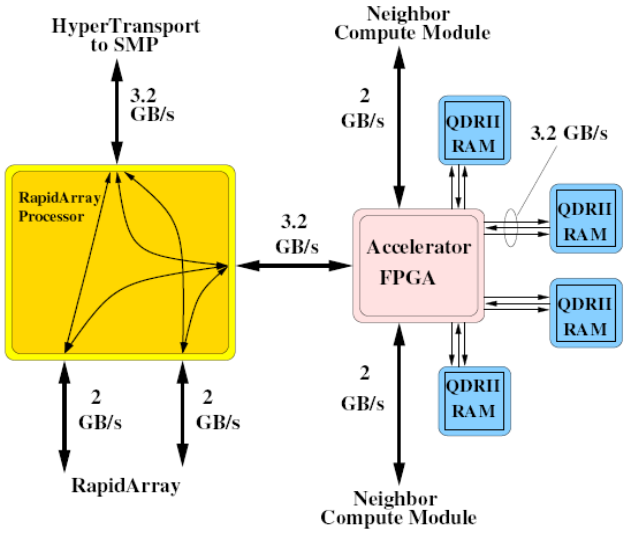


图2-4 Cray XD1的互连

2) 道路交通系统实现过程中使用的流处理方法

如果直接将波特兰市的道路节点一一映射到FPGA，那么就需要用FPGA模拟一个中型城市，由于可重构逻辑资源的限制，这种做法是不可行的。通过仔细分析应用需求，在实现过程中使用了流处理方法，对软硬件功能进行了合理的划分。

流处理方法如图2-5所示，它的思想是，道路相关信息并不保存在可重构器件上，而是保存在主存储器中，仅用可重构器件实现道路节点的处理单元PE（Processing Engine），它从存储器读入道路节点的信息，进行处理，并将处理完的道路信息写回存储器[2]。

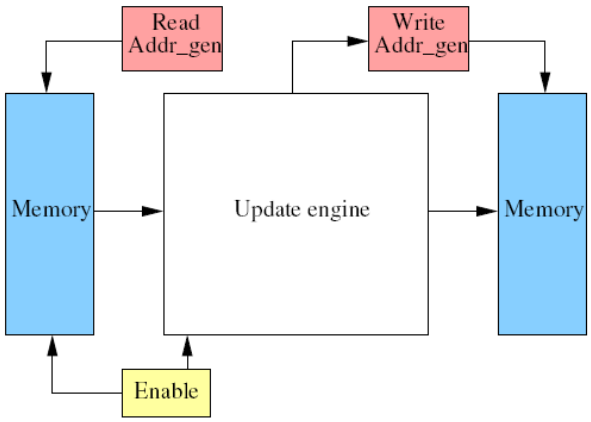


图2-5 流处理方法的结构

3) 关于线性代数函数库的问题

在动力学仿真的实现过程中，用可重构部件（FPGA）实现了向量乘积,矩阵和向量乘积以及矩阵和矩阵乘积这三个线性函数的方法。在实现矩阵乘积时，采用多个 PE 级连构成流水线。之所以在标题上注明“design tradeoffs”,是因为运算中的浮点型数据操作对片上存储和带宽的要求很大,而目前 FPGA 上片上存储以及带宽满足不了要求,因此在设计中作了一些调整,限制了流水线的级数，从而影响了加速比，但最终的结果仍然是让人满意的[4]。我们可以认为随着可重

构技术的不断发展，片上存储和互连带宽的增大，将使线性代数函数库给应用带来的性能提升更大。

2.4 对论文的思考

由于可重构资源不足，城市道路交通系统最终用可重构硬件实现的PE仅仅占PE总量的61%。可重构资源不足是超级可重构计算实现过程中普遍存在问题。由于动态可重构技术可以降低对可重构资源的需求量[3]，因此我们设想改用动态可重构技术来实现城市交通系统。

1)基于动态全局可重构技术的应用实现

将波特兰市的道路交通系统分为南北或东西两块，每次仅重构50%的道路交通系统。如图2-6所示，首先利用可重构硬件实现某时刻东城区的PE，完成计算任务之后，再重构同一时刻的西城区PE；之后不必重构，计算下一时刻西城区的道路交通情况，继而再重构该时刻东城区的PE，这样周而复始。

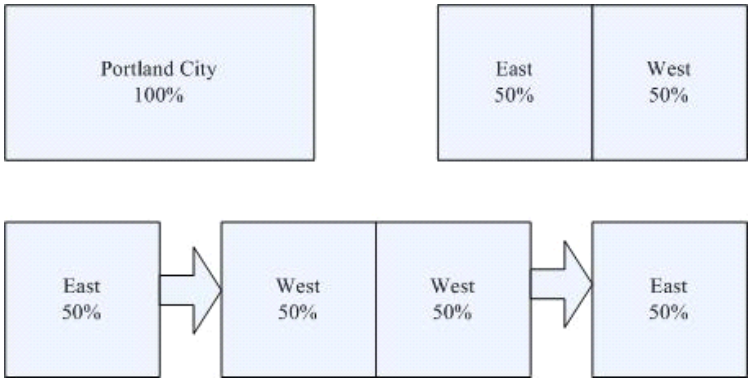


图2-6 基于动态全局可重构的实现

动态全局可重构带来的好处是，所有PE都可以用硬件来实现，解决了可重构资源不足的问题。同时，全局重构也引入了额外的重构开销。如果硬件加速带来的好处大于重构开销，那么用这种方法来实现道路交通系统可以带来整体的性能提升。

2)基于动态部分可重构技术的应用实现

如图2-7所示，将波特兰市分为三部分。东西部分各占40%，中间部分占20%，如图所示。东西部为可重构模块；中部为固定模块，不需要重构。

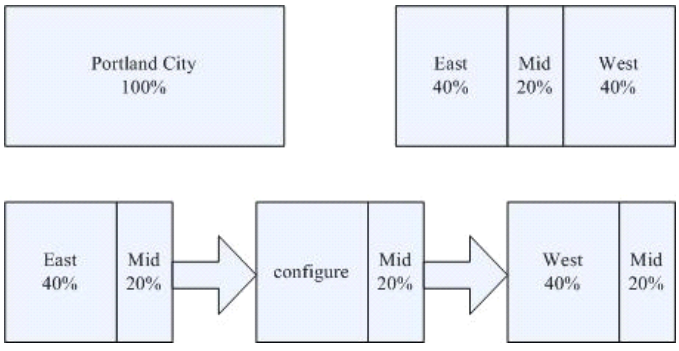


图2-7 基于动态部分可重构的实现

这样不但可以用硬件对所有的计算单元PE进行加速，而且与全局重构方法相比，每次重构过程可以减少20%的重构开销。

3)基于流水线可重构技术的应用实现

在阅读论文的过程中，我们发现PE的实现分为5个流水步骤。因此可以利用流水线可重构技

术，每次仅重构流水线的某一阶段。

这种实现方式固然可以节省可重构资源，但是会破坏流水线的连续执行。

上面的几种基于动态重构技术的应用实现方式可以带来不同程度上的硬件加速，但是，同时也引入了额外的重构开销。重构开销有可能大于硬件加速带来的好处，但是不管怎样，在超级可重构计算系统里引入动态可重构技术这种方法是值得一试的。

本章小结

超级可重构计算技术正在发展，目前已有不少超级可重构计算系统担负起沉重的科学计算任务。增加可重构资源是提高超级可重构系统的有效方法；利用现有的系统，如果希望用硬件加速尽可能多的计算部分，那么需要仔细地对应用实例进行分析、合理地划分软硬件功能；除此之外，为了解决可重构资源不足的问题，可以尝试将动态重构技术应用于超级可重构计算系统。

参考文献

[1] Maya Gokhale, Christopher Rickett, Justin L. Tripp, Chung Hsu, Ronald Scrofano : *Promises and Pitfalls of Reconfigurable Supercomputing*. ERSA 2006 : 11-20

[2] Justin L. Tripp, Henning S. Mortveit, Anders A. Hansson, Maya Gokhale: *Metropolitan Road Traffic Simulation on FPGAs*. FCCM 2005: 117-126

[3] KATHERINE COMPTON: Reconfigurable computing: *A survey of systems and software*. ACM Computing Surveys:2002 / 34 / 02 P 171-210

[4] Ling Zhuo, Viktor K. Prasanna: *Design Tradeoffs for BLAS Operations on Reconfigurable Hardware*. ICPP 2005: 78-86

3 案例分析之二：可重构计算在数据挖掘中的应用

3.1 引言

数据挖掘中有很多算法都在研究数据的可用性：即如何计算数据的支持度。其计算的公式为：数据支持度=使用这个数据的事务数/数据库中事务总数。支持度是反映数据有用性的一个量化指标。计算支持度的目的是：从给定的一个候选集文件，计算出所有的候选集的支持度，按照预定的裁剪标准，挖掘出比较有用的候选集数据。

apriori算法是计算支持度的著名算法，其算法思想如下：

Apriori 算法的基本思想可以用一段伪代码描述如下[24]：

```
(1)L1={large1-itemsets};          //频繁 1-项集 主FPGA做的，初始化
(2)for(k=2; Lk-1 ≠∅; k++){        //Lk 表示频繁 k-项集
(3)   Ck=apriori-gen(Lk-1);  //candidate generation and pruning
(4)   for all transactions t∈D{
(5)      Ct=subset(Ck, t);        //事务 t 中包含的候选集 计数，支持 Ck 的事务数
(6)    for all candidates c ∈ Ct  support calculation
(7)         c.count++;            硬件化的部分，是静态重构技术
(8) }
(9) Lk={c∈Ck|c.count≥minsup}
(10)}
(11)Answer=∪kLk;
```

Apriori 算法有三个嵌套循环，复杂度很大。在这可以把整个算法硬件实现，但是时间性能依然很差，所以可以引入可重构技术，使用静态重构技术直接提高第三层循环的执行效率。

3.2 算法实例

算法统计下面表中使用每个数据的事务个数，最后计算出每个数据的支持度。其中算法输入数据如下表所示：

表 3.1 源数据库

事务标识号（TID）	项目集
T1	A, B, E
T2	B, D
T3	B, C
T4	A, B, D
T5	A, C
T6	B, C
T7	A, C
T8	A, B, C, E
T9	A, B, C

经过算法的第一次迭代，对事务数据库进行一次扫描，得到候选项集集合 C_1，如表 3.2 所示。

计算出的支持度如下：

表 3.2 候选 1-项集

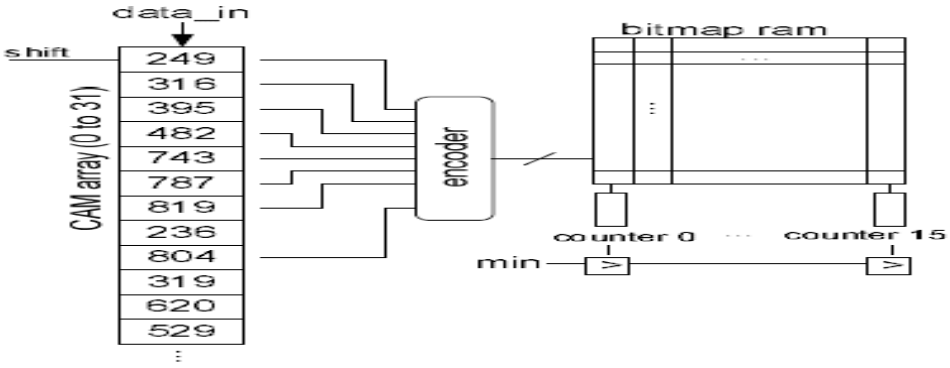
项集	支持度计数
A	0．67
B	0．78
C	0．67
D	0．22
E	0．22

项集的支持度=使用此项集的事务数/事务总数

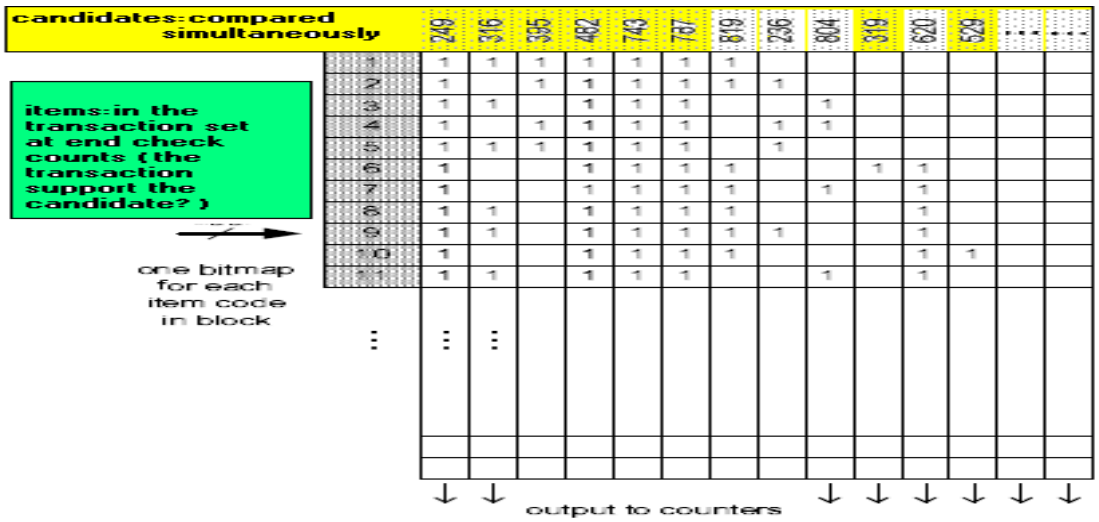
假定最小事务支持计数为 2(即 minsup=2/9=0.22)。通过比较可以得到频繁 1-项集的集合 L_1，如表 3.3 所示。

3.3 主要贡献

在研究这个算法的文献[2]中就提出使用硬件来完成支持度的计算问题。但是那篇文章并没有引入可重构的思想，而文献[1]提出并发地计算数据支持度，即同时计算几个数据的支持度，进一步提高效率。文献[1]基于 SRC MAPstation 来完成实验，并验证这个的思想。硬件执行的过程主要是利用 Bitmapped CAM 结构来完成：



其中 bitmap 上方是需要计算支持度的数据：候选集（candidates）；CAM arrary 中每个条目都是 bitmap 上方数据中的一个独一无二的元素：候选项（item），事务的操作使用数据是从 data_in 进入 CAM ，依次和 CAM 中 items 进行比较，如果匹配，则通过解码器，知道其在 bitmap 中对应项的位置，并置 1，表示这个事务使用了这个数据：候选集。如下图所示：



例如 data_in 输入下面的事务数据，系统通过 count 计数可以知道支持每个数据的事务数多大，最后可以很快得出每个数据的支持度。其中每一个准备被计算的候选集中独一无二的元素（候选项），都会在加到 CAM array 中，CAM array 被装满或需要同时计算支持度的候选集数已经达到最大值。如何确定 CAM array 的大小和需要同时计算支持度的数据个数的大小是一个大问题。文献[1]通过一个实际的数据库运行得出 CAM array32，candidates：16 这个结果。实验结果如下：

证明CAM entrys 32， candidates： 16， 性能最优

作者认为：CAM:32 candidates：16的reason

Generation Number	Max CAM elements	Num blocks above 32 max	Max cand. in 31 CAM entries
1	16	0	–
2	18	0	–
3	22	0	–
4	31	0	–
5	41	1	12
6	28	0	–
7	41	2	9
8	29	0	–
9	31	0	–
10	41	1	15
11	25	0	–

Table 1: Detailed CAM usage information for each generation for $|S_{cam}|$=32 and $|c|$=16 for the T40I10D100K dataset at 0.01 support

Max cand： 同时比较的candidates最大值

上面的工作都是在系统的从 FPGA 中做的，从 FPGA 得出 count 数，传回主 FPGA ，供其计算支持度。这里的主 FPGA 传 bitmap 结构的硬件程序用于从 FPGA 的重构，可以知道这是静态重构，主 FPGA 还传输事务数据给从 FPGA 用于 count 的计数。从 FPGA 把计数的结果回传给主 FPGA。这里虽然只有一个从 FPGA，但是实际上是很多从 FPGA，也就是说上面讲的是一个从 FPGA 中的数据，现实中如需要计算很多数据的支持度，就可以同时放在很多从 FPGA 中计算。整个系统的体系结构如下：

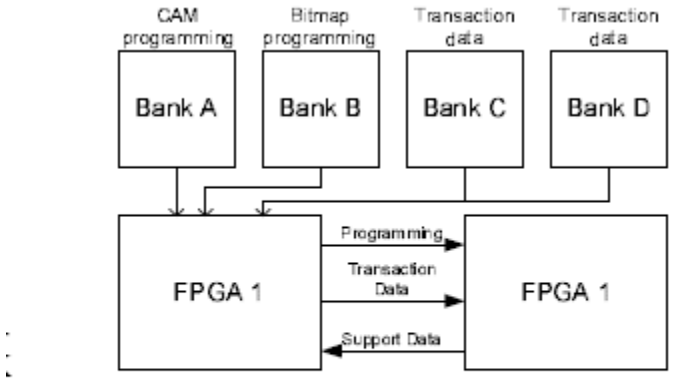
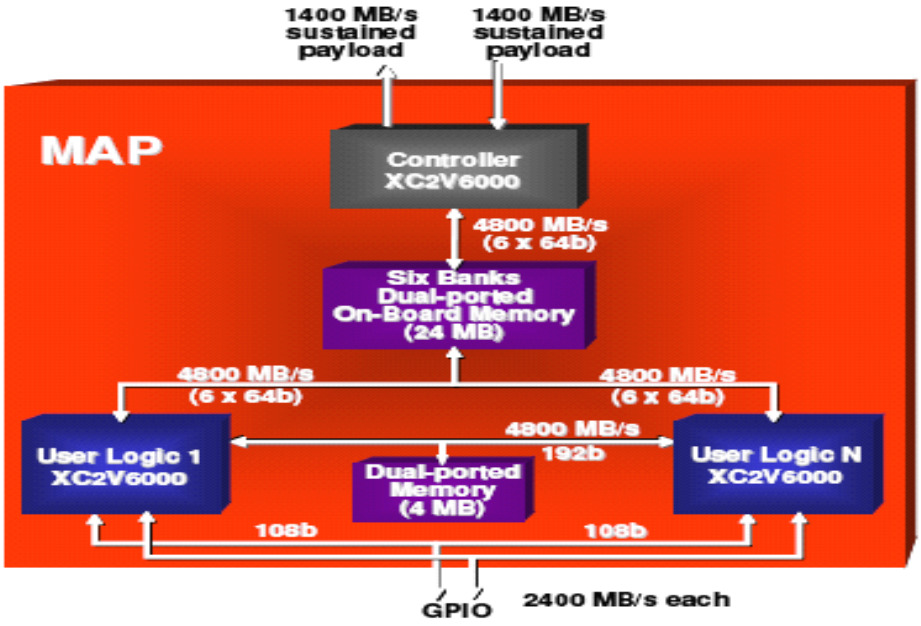


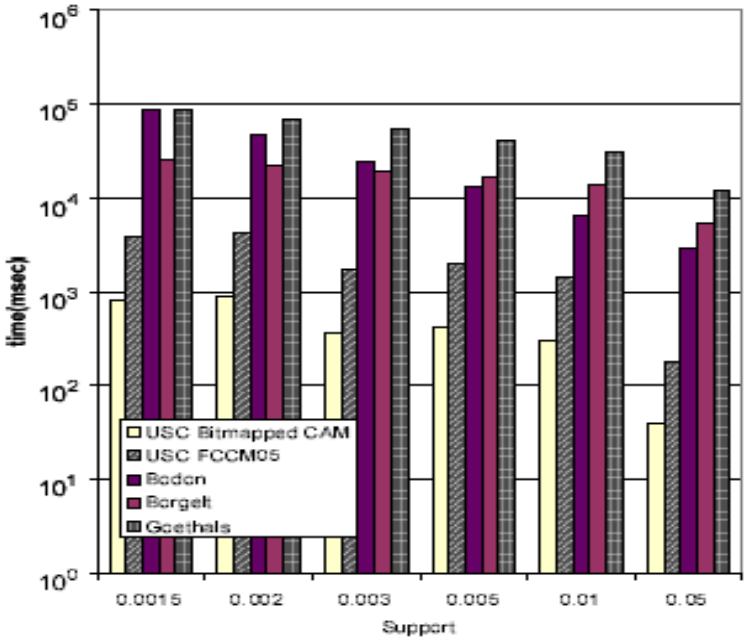
Figure 8: Architecture of bitmapped CAM implemented on reconfigurable computer

主 FPGA 做了算法的整个硬件实现过程，并把算法中第三层循环交给从 FPGA 去执行。从 FPGA 把执行结果返回给主 FPGA，例如第三层循环要做 160 次比较，主 FPGA 可以把 count 计数的工作，交给十个从 FPGA 来做，时间性能会有很大提高。主 FPGA 在把计数工作交给从 FPGA 之前，先对从 FPGA 做静态重构，使从 FPGA 具有完成计数的硬件功能。

文献[1]在 SRC MAPstation 实验平台上做了纵向和横向比较实验，实验平台结构图如下：



文献[1]做个实验作了横向的比较，下面是横向比较实验结果：



从上面的图中可以看出使用 bitmapped CAM 的方式，在各种默认支持度下，时间消耗都是最小，因此这种方法比其他方法要好。文献[1]也做了纵向的比较，并统计这个系统的各部件时间消耗。实验统计结果如下：

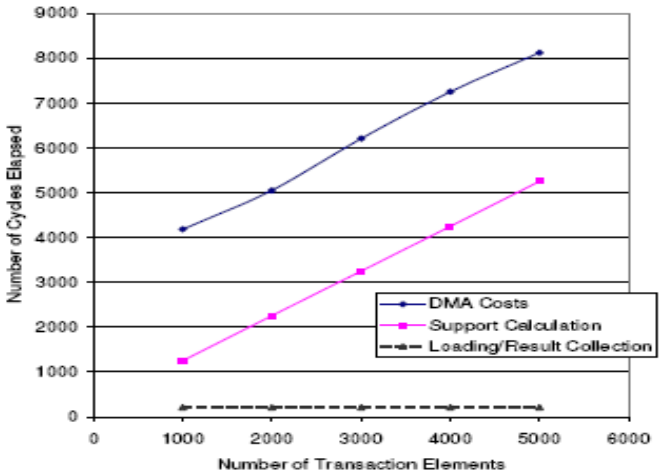


Figure 9: Throughput and pipeline latency for support operation

3.4 问题讨论

问题一：数据传输的改进

从上面的纵向实验结果可以看出数据传输的时间消耗是最大的，而支持度计算的时间消耗相对来说是小的，因此可以改进系统，使数据传输量最小。

传输时间=传输数据总量/传输速度（传输速度和带宽成正比）

可能改进方法：减少传输数据总量/提高传输速度。因为减少数据的传输量就是减少事务的数据传输量，会导致系统计算的结果不准确，所以只有提高传输速度这条路。

这里将介绍四种解决方法：

（1）加大事务传输通路的带宽法：加大主从 FPGA 之间事务传输通路的带宽。

优点：可以减少 DMA 传输时间，提高系统整体效率。

缺点：这种方法需要特别的改进硬件体系结构，所以主从 FPGA 的硬件设计复杂度都会上升，而且受 FPGA 的引脚数限制，即：受生产工艺的的限制。另外，我们需要保证重构配置信息的传输速度，所以不可以给事务数据传世过多的带宽。

（2）共享数据线法：事务数据，重构信息数据和写回数据共享数据线。

优点：无须改进硬件体系结构，就可以显著提高传输速度。因为重构信息数据，事务数据和写回数据的传输过程是分时进行的，首先要重构好从 FPGA，才可以传事务数据供其计算，直到事务数据传输完，计算完毕，写回数据才可以传输。

难点：是从 FPGA 如何知道当前接收的数据是重构信息数据，还是事务数据。这需要主 FPGA 和从 FPGA 有个约定。

这个约定可以是：

1）主 FPGA 要使用某从 FPGA 时，可以先读从 FPGA 的状态寄存器（每个从 FPGA 在主 FPGA 都有一个一一对应的状态寄存器），知道当前从 FPGA 的状态，如是忙，即：正在计算，或正在重构，或正在写回，则不可以使用，仅当从 FPGA 空闲时才可以使用。

2）当主 FPGA 使用该从 FPGA 时，先把状态寄存器置为 busy，在传输数据时，是有讲究的，主 FPGA 先传输重构信息数据给从 FPGA, 传输完毕，等待几个周期，再传输事务数据，让从片知道现在传输的是事务数据，从片开始计算，最后传输完毕，且计算完毕，从片写回数据，

既：计算结果，传输完毕之后，告诉主片传输完毕，主 FPGA 再次把相应的状态寄存器置为空闲。

缺点：主 FPGA 做的较复杂，性能提高有限。

（3）分组计算法：一组从 FPGA 合作完成计算。

只传输部分事务数据（总事务数据/组大小）给每一个从 FPGA，每个从 FPGA 只统计算部分事务数据中的数据使用情况，最后主 FPGA 把这小组的数据相加得出最后的计数结果，并计算出支持度。

优点：传输时间可以显著提高。

数据传输时间=数据传输总量/（分组组大小*单片传输速度）

难点：主 FPGA 的工作量加大，复杂性上升，但是与节约的时间相比，有的时候，还是值得的，具体可以在两者之间找一个折中的方法，也就是得出一个分组大小，时间性能和主 FPGA 复杂性代价之间的一个计算公式。用户根据这个公式决定自己的系统是怎么分组，时间性能是多大。

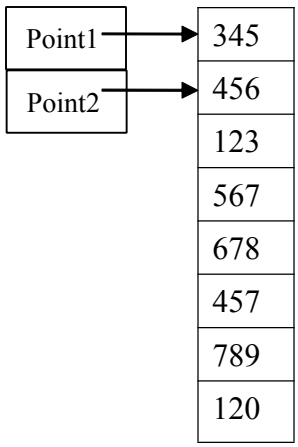
（4）混合法：分组计算法中的主 FPGA 和组中的每一个从 FPGA 可以使用共享数据线法的方式进行数据传输，可以进一步提高性能，但是主片设计的复杂性也进一步加大了。

缺点：主 FPGA 的设计复杂性提高，会引起容易出错，测试困难，可靠性下降。

问题二：算法在硬件上的改进

当数据量大的时候，对数据进行连接和裁减的过程用算法执行也很麻烦，需要的时间也是相当大的。所以可以在这方面也进行可重构计算来执行更快地算法。

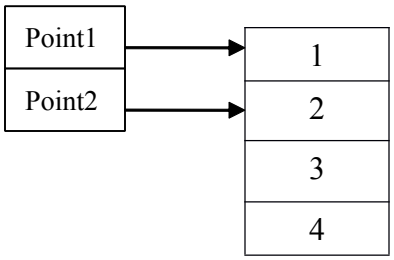
首先进行数据的连接，如下图：



上图中右侧的是候选集的 array（类似于 CAM），point1 和 point2 可以看作是指针，当然在硬件中是数据线，首先 point1 指向第一个候选项和 point2 指向的候选项求并集，然后输出。每比较一次 point2 向下移动一个位置，等整个都比较完后，point1 向下移动一个位置从新开始求并集。这样直道 point 指向最后一个候选项的时候连接就算结束了。

接下来是裁减的过程。现在假设我们得到了 1，2，3，4 这个连接后的项目。首先我们看第一个候选项，这里再一次使用 array（CAM）。如下图。

同上面的原理，首先并联个数，point1 并上 point2，然后 point2 往下移动（第一次移动），当 point2 移动到最后的时候说明本次并已经结束，然后 point2 回到原来的位置，这次是并 3 个数的，当 point2 往下移动两个后并且并上这两个后，point2 又重新指向原来 point2 的位置的下一个.....这个是以 1 位首元素的并集，依次类推，可以求出以每个元素开头的并集。



接下来要解决的问题是看这个项目是否可行，这就需要上一个步骤所剩下来的并集，以 1，2，3，4 这个项目为例，剩下来的是 1　2　3　4　12　13 14　23　24　34　123　134　234　1234 这些个并集，然后再把这些元素放到 CAM 中去，把上一代的候选集来进行比较 ，如果发现这个 CAM 中的任何一个元素不能和上一代的候选项进行匹配的话则说明这一代的这个候选项是不可行，于是要把它从这个候选集中裁减掉。

本章小结

现在数据库中计算数据支持度非常重要，是一个研究热点，此前有人提出一个经典的计算支持度算法apriori。这个算法的时间复杂度很大，实际应用中，数据会很多，因此这个算法的时间性能让人不可以忍受。文章作者首先对这个算法硬件化，最后又引入重构技术进一步提高算法的时间性能。作者做了横向比较实验，发现这个方法比其他改进思路要好。我们提出改进数据传输速度的几种方案，进一步提高算法执行的时间性能。另外看到静态重构技术带来的好处很多，所以又进一步改进了算法，对数据的链接和裁剪过程使用了可重构技术进一步提高算法执行的时间性能。

参考文献

[1] Z. K. Baker and V. K. Prasanna.　Efficient Hardware Data Mining with the Apriori Algorithm on FPGAs. In *Proceedings of the Thirteenth Annual IEEE Symposium on Field Programmable Custom Computing Machines 2005 (FCCM '05)*, 2005.

[2] F. Bodon. A Fast Apriori Implementation. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, 2003.

[3] SRC Computers, Inc. http://www.srccomputers.com.

[4] C. Borgelt and R. Kruse. Induction of Association Rules: Apriori Implementation. In *Proceedings of the 15th Conference on Computational Statistics*, 2002.

4 案例分析之三：可重构计算在分子动力学仿真中的应用

4.1 引言

分子动力学仿真是一种模拟分子运动的方法，该方法主要是依靠牛顿力学体系通过计算分子的受力情况来模拟分子的运动，然后在由分子体系的不同状态构成的系统中抽取样本，从而计算体系的构型积分，并以构型积分的结果为基础进一步计算体系的热力学量和其他宏观性质。

分子动力学仿真的主要任务包括力的计算、位置和速度的更新，以及其他一些相关任务。在传统意义上，这些任务都是用单精度或者双精度算法编制软件，然后在通用处理器上执行。但是由于 FPGA 的发展和可重构硬件的发展，可以把仿真中的一些计算密集型的任务（如力的计算）放到可重构硬件上实现。这样可以提高计算的速度，即提高相应的计算加速比。文献[1]就是在这种需求下做的一个仿真研究。

4.2 仿真算法分析

在分子动力学仿真中，使用的仿真对象数量很多，每个分子的初始位置，初始速度和整个系统的温度、能量都会给定。仿真需要确定在一段时间里，每个分子的运动轨迹。用软件实现分子动力学仿真使用的是 velocity Verlet 算法，这个算法的流程为：

1、 使用 t 时刻的 v_i（t）（速度）和 r_i（t）（位置）计算 t＋△t/2 时刻的速度 v_i（t＋△t/2），其中△t 为时间步，一般为 femtosecond（飞秒）量级；

2、 用 v_i（t＋△t/2）计算 r_i（t＋△t）；

3、 对所有的 j ≠ i，基于 r_i（t＋△t）和 r_j（t＋△t）计算 a_i（t＋△t）（加速度）；

4、 由 a_i（t＋△t）计算 v_i（t＋△t）；

这里加速度的更新是计算密集型的，也是最费时的，因为这里涉及到了力的计算。分子动力学中的力可以分为 bonded force（健合力）和 nonbonded force（非健合力）。其中健合力指的是相邻的分子之间的作用力；非健合力指的是任意两个分子之间的作用力。在这里只考虑 Lennard-Jones 力和 Coulomb 力（库仑力），其中 Lennard-Jones 力是非健合力，Coulomb 力是健合力[1]。根据统计非健合力占全部计算时间的四分之三左右（如表 4-1 所示），故可以将该任务单独拿到可重构硬件上实现，而其他的部分将继续留在通用处理机上执行。具体的划分方法如图 4-1 所示。

表 4-1 软件实现中各任务所占的比例

Task	% of Computation Time	
	Palmitic Acid	CheY Protein
Nonbonded Forces	75.15	74.09
Building Neighbor List	20.75	22.76
Dihedral Torsion Forces	2.14	1.51
Other	1.96	1.64

从图中可以看到，通用处理器（GPP）处理的任务分为初始化、构建邻接表、计算健合力、速度更新、位置更新和加速度更新，而可重构硬件（RH）的处理的任务是计算非健合力。GPP 做完初始化和构建邻接表的工作之后，把分子位置和分子的邻接表给 RH，RH 就根据这些信息来计算非健合力。计算结束后把计算结果再传送给 GPP。就这样可以实现 GPP 和 RH 的协同工作。从图中我们还可以看出可重构计算机仿真 MD 的每一步所需的时间为 $T_{SWtasks}$、T_{comm} 和 T_{RH} 三者之和。其中 $T_{SWtasks}$ 指的是软件上的执行时间，这个时间等于仅由软件实现的那部分时间（即除去非健合力的计算时间）；T_{comm} 指的是通用处理器和可重构硬件之间数据的传送时间； T_{RH}

指的是可重构硬件部分的执行时间。为了减少 T_{RH}，实现时使用了并行技术和流水线技术。并行技术是指数据从通用处理器部分传送到可重构硬件部分时，使用 DMA 来控制数据的传送，这样在传送了一部分数据后，数据传送和可重构硬件可以并发执行。流水线技术是指在可重构硬件上采用多重流水，实现指令的并发执行。

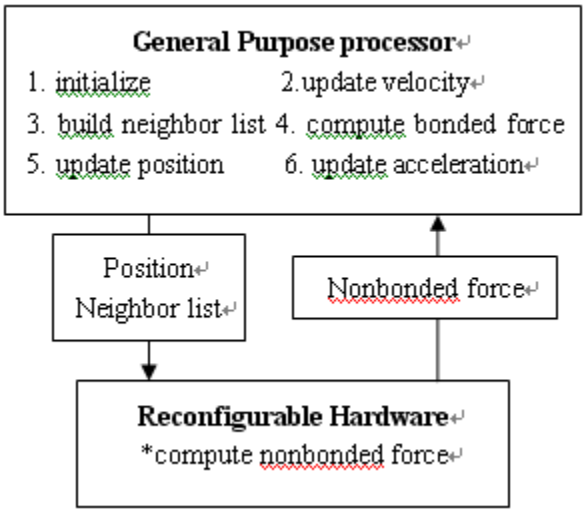


图 4-1 软硬件划分及相应任务分配

为了得到更好地仿真效果，在可重构硬件部分，文献[1]的作者提出了三种实现方法，并对每种方法的执行时间进行了预测。经过各种综合考虑，作者选择了一种叫做 write-back 的设计。这种设计在一定程度上消除了基本方法的读后写相关，并且使用了一种由 FIFO 连接的两段独立的流水线技术。在具体硬件实现之前，作者预测这种设计方法比用纯软件实现的方法将可以获得 2 倍左右的加速。

在可重构硬件实现时，使用的是单精度的运算器。实验结果表明在使用了可重构硬件技术后，无论对于 palmitic acid 分子还是 CheY protein 分子，都获得了 2 倍左右的加速，结果与预测值基本相同。

4.3 与相关工作的比较

在文献[2][3]中，讲述的是使用可重构硬件来实现分子动力学的仿真。在文献[3]中，作者首先介绍了通用计算机、ASIC 和可重构计算机的区别，在此基础上提出了一个假设：FPGA 在高性能处理方面将有很大的生存空间。为了测试这个假设，作者选择了分子动力学仿真实验。这个仿真实验只考虑了 Lennard-Jones 力，并且这个力是使用查找表和插值的方法来计算的。所有的设计均在硬件平台上实现，使用的是定点的运算器。实验结果表明这种仿真方法可以获得 20 倍以上的加速。值得注意的是，这篇文章中提到了可以使用单精度的运算器和只将部分仿真放在可重构硬件上实现的思想，而这正是文献[1]中所使用的方法。文献[2]使用的方法与第一篇类似，但是它用数据来说明仅使用定点的运算器也是可以获得很高的精度的，因此可以不用单精度（双精度）的运算器来仿真。文献[2]考虑了 Lennard-Jones 力和 Coulomb 力，Lennard-Jones 力也是使用查找表和插值的方法来计算，使用的同样也是定点运算器。它的仿真也全部是在硬件平台上实现的，实验最后获得了 80 多倍的加速。

文献[1]的工作只有计算密集的部分才在可重构硬件上实现，大部分都是在通用处理器上实现，并且使用的是单精度的运算器，所以获得的实验结果没有文献[2][3]的好。但是，这种设计方法的优点一是它每移动一步都需要通过复杂的力的计算从而得到分子移动后的位置，这样的

计算比使用查找表和插值的方法要精确；二是这种方法的灵活性好，对于不同的分子，它在可重构硬件部分的实现完全不用修改，这样可以大大的减小设计的复杂性；三是这种设计中使用了并行和流水线技术，使得仿真速度也得到了相应的提高。

4.4 问题讨论及解决

问题一：文章中计算的时间哪一部分是属于可重构硬件的时间？

从文章的几个时间计算来看，这里谈到的仿真计算时间实际上并没有包含有可重构硬件的时间。而从文献[4]中可以看到，它也没有考虑可重构硬件所花费的时间。因此，可以认为对于仿真来说，可重构硬件耗费的时间与仿真实现时的时间计算是没有关系的。

问题二：图 4-2 右上角标识了三种图示，为什么其中的 Comm. Time 在柱状图上没有表示出来？

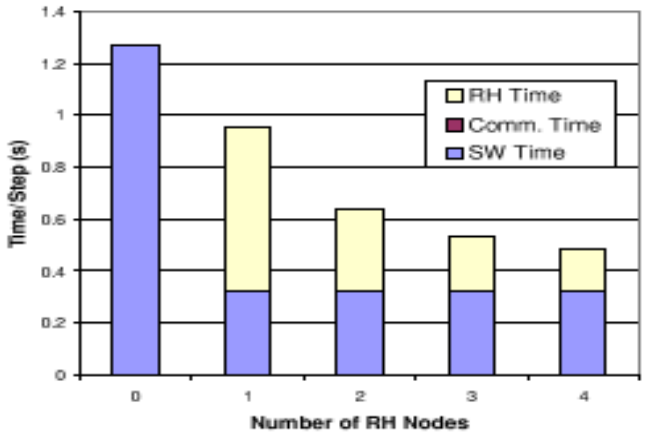


图 4-2 估计设计的性能

在论文的工作过程描述中提到，由 DMA 控制通用处理器和可重构硬件之间的通信，因此通信开销实际上是非常小的，以致于在图上没有表示出来。

问题三：26 个 cell 的来历

文中提到 cutoff 距离时，使用了一个分区（cell）的办法，文中直接给出每个 cell 只与其相邻的 26 个 cell 有联系。我们的困惑是这个“26”是怎么得到的。实际上以原子所在的 cell 为中心，作一个正立方体，与这个立方体相邻的同样的立方体刚好有 26 个（即 3^3-1=26）。

4.5 对论文的思考

在文献[3]中提到，进行分子动力学仿真主要是为了得到一些物质的性质属性。为了达到这个目的，仿真的结果应该是越精确越好。所以，如果条件允许，仿真过程使用的运算器精度应该使用双精度的比较好。但是，随着运算器精度的增加，计算的复杂性也必然会相应增加，这样使用可重构硬件来实现的话成本增加，所花费的时间也增多。所以在实验精度可以达到预期水平的时候，对仿真的过程还需要综合考虑，否则可能得不到预期的加速效果。

在讨论的问题中提到，这个仿真实验实际上并没有把可重构硬件的时间计算在内，而对其他很多方面的应用来说，似乎也忽略了这样一个计算。对于分子动力学的仿真，分子每步的计算都是相同的，这表明可重构硬件部分在仿真开始时只需要一次重构就足够了，这样不考虑可重构硬件的时间还是可以的。但是对于很多应用来说，可重构硬件部分要随着仿真的进行而不断进行重构，并且可重构硬件花费的时间很大，这样的话就不能忽略可重构硬件的时间。

在总结这篇文章工作的基础上，我们考虑了几个可能的研究方向：

1）使用更高精度的可重构硬件。有关分子动力学仿真的几篇文章都提到了仿真误差在 5%左右，我们通过使用更高精度的硬件，是不是可以达到 1%甚至更低的误差呢？这样我们就可

以得到更好的仿真结果，从而推断出更准确的分子性质。

2）是否有更好的软硬件协同涉及方法可以使仿真加速？文章中将计算复杂度为 O（n^2）的部分用可重构硬件来实现，而其他仿真都由通用处理器来实现。通过改变软硬件部分的划分，是否可以使通用处理器部分和可重构硬件部分实现并行工作呢？

3）文献[1]的工作使用单个通用处理器和单个 FPGA 实现仿真。可以考虑使用多个 FPGA 的协同工作来达到提高仿真速度的目的，当然这里还需要考虑各个 FPGA 的通信开销。

本章小结

随着可重构硬件的快速发展，特别是 FPGA 的发展，使得运用可重构硬件来加速诸如科学计算之类的复杂应用成为可能。这几篇文献主要就是通过一个使用软/硬件结合的方法实现分子动力学的仿真这个超级计算，验证了使用可重构硬件实现一个任务的计算密集型部分是可以获得很好的整体加速效果的。

通过对这些文献的学习，了解了要很好地使用可重构硬件实现各种应用，需要做到几点。首先，在认真分析具体应用的基础上，找到良好的软/硬件划分方法。对于大多数的应用，如果全部使用软件实现，则速度可能会太慢；如果全部使用硬件实现，则成本太高，因此找到一个好的软/硬件划分是实现应用加速的关键。其次，找到好的软/硬件划分方法后，要考虑软硬件实现的细节，挖掘潜在的并行能力和流水线处理能力。如果只是简单地将应用划分为软硬件实现部分，则这两者之间的通信开销就有可能抵销硬件的加速效果，反而得不偿失。最后也是很重要的一点，就是要学习相关工作。一篇论文只涉及了很有限的内容，通过对相关工作的调研学习，举一反三，才能更好地理解文章的精华所在。

参考文献

[1] Ronald Scrofano, Maya Cokhale, Frans Trouw and Viktor K Prasanna. A Hardware/Software Approach to Molecular Dynamics on Reconfigurable Computers, In *Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 2006.

[2] Y.Cu, T.VanCourt and M.C.Herbordt. Accelerating molecular dynamics simulations with gonfigurable circuits, In *Proceedings of the 2005 International Conference on Field Programmable Logic and Applications*, August 2005.

[3] Navid Azizi, lan Kuon, Aaron Egier, Ahmad Darabiha and Paul Chow. Reconfigurable Molecular Dynamics Simulator, In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 197-206, April 2004.

[4] Zachary K. Baker and Viktor K. Prasanna. An Architecture for Efficient Hardware Data Mining using Reconfigurable Computing Systems,

[5] Katherine Compton, Scott Hauck. Reconfigurable Computing: A Survey of Systems and Software[J]. ACM Computing Surveys, 2002, 34(2):171–210.

5 结论与展望

可重构计算技术利用可重构逻辑部件（如FPGA）的可重构特性，通过不同的配置信息来改变部件所实现的功能，从而能够以近乎全硬件实现的性能灵活实现多种应用。可重构计算技术避免了微处理器计算模式因为取指、译码等步骤导致的性能损失，同时也消除了专用集成电路ASIC计算模式因为前期设计制造的复杂过程带来的高代价和不可重用等缺陷。

总结起来，可重构计算技术的优势主要体现在三个方面。第一，可使设计者用更为简单的硬件来实现更多的功能。由于所有逻辑不需要同时出现在FPGA内，因此，支持额外特性所需的成本降低到存放配置信息所需的成本。第二，可降低系统的成本。对于生产量较小的应用而言，可节省ASIC设计与制造所带来的成本；而对于大批量生产的产品而言，可重构计算技术的采用使产品可现场升级，延长了产品的生命周期，因此而节省的成本更为可观。另一方面，FPGA可通过重构多个软核在单片上实现多个指令集的处理器。根据现场计算任务的划分，实时实现不同的处理器功能，达到一次芯片设计，多个功能实现，从而大幅度降低芯片设计和制造的一次性工程费用成本。第三，可缩短产品周期。由于不再使用ASIC，可节省大量的芯片设计和验证所需的时间，此外，可重构特性增加了设计的灵活性，设计者在设计之初无需将所有功能加入产品中，新的功能可在产品上市后逐步添加。

可重构计算技术已经广泛应用在科学计算、国防军事、航空航天等诸多领域，用于实现如目标匹配、大数值运算、数据挖掘、模型仿真等功能，都取得了非常好的效果。目前可重构计算技术的应用也在逐渐向民用领域扩展，在汽车电子、网络设备等领域已经有产品出现。

对可重构计算技术的研究这些年也是方兴未艾。从近年来可重构计算技术方面的国际会议（如ReConFig、ERSA、RAW、ARC等）的主题可以看出，对可重构计算技术的研究主要集中在可重构计算体系结构、可重构计算应用、可重构计算工具、可重构计算教育、可重构计算性能测试等方面。例如，即将在2008年举办的第15届可重构体系结构会议（RAW2008）的主题包括模型与体系结构（Models&Architectures)、算法与应用（Algorithms&Applications)、工具与技术（Tools&Technologies）等，同年举办的第4届可重构计算应用会议（ARC2008）的主题则包括方法与工具（Methods&Tools)、体系结构（Architectures)、应用（Applications)、教育（Teaching)、综述与未来趋势（Surveys&Future Trends)、性能测试（Benchmarks）等。

业界著名的Makimoto曲线揭示了半导体主流应用从1957年开始在标准化和定制化之间每隔十年切换一次，68k的总设计师Tredennick把这个规律进一步阐述为算法和资源逐渐从全是固定的向全是可变的发展趋势，IEEE“Top10 Fellow”之一的Hartenstein早在2001年便预言2007年将开始Makimoto曲线的“最后一波”，那就是可重构计算。

展望未来，集高性能、高灵活性、低功耗、低成本等优点于一身的可重构计算必将获得更大的发展和更广的应用。