(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 111124999 A
(43)申请公布日 2020.05.08

(21)申请号 201911258025.0

(22)申请日 2019.12.10

(71)申请人 合肥工业大学
地址 230000 安徽省合肥市屯溪路193号

(72)发明人 张章 曾剑敏 魏亚东 解光军

(74)专利代理机构 合肥方舟知识产权代理事务所(普通合伙) 34158
代理人 刘跃

(51)Int.Cl.
G06F 15/78(2006.01)

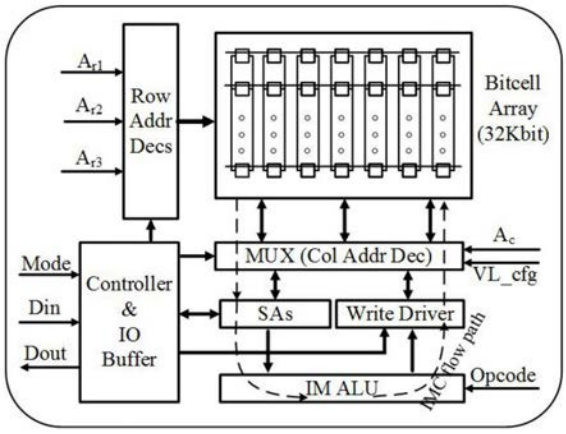
权利要求书1页 说明书4页 附图3页

(54)发明名称

一种支持存储内计算的双模计算机架构

(57)摘要

本发明涉及计算机架构技术领域，具体涉及一种支持存储内计算的双模计算机架构，包括一处理器核、一指令存储器、一存储计算协处理器、若干计算型的SRAM宏单元。本发明的有益效果：提出的架构可以工作在非冯诺依曼体系结构下，也保留了传统的冯诺依曼结构。这样做的目的是最大限度地利用现有的编译工具链和编程模型。通过仿真表明，根据使用IMC-SRAM宏单元数量的不同，与基准对比，本发明架构可以对特定的应用提供十几至上百倍的加速。在采用一块宏单元的情况下，可以对规模为32*32的二值神经网络和512位字符输入的哈希算法分别加速6.7和12.2倍。与基准对比，本架构平均节省能耗为3倍。



CN 111124999 A

1.一种支持存储内计算的双模计算机架构,其特征在于,包括一处理器核、一指令存储器、一存储计算协处理器、若干计算型的SRAM宏单元,所述处理器核采用精简指令集核心,具有六级流水线,在取指级和译码级插入了预译码级流水,用于对存储计算专用指令进行译码,所述指令存储器采用存储编译前生成,所述存储计算协处理器用于对处理器核传送过来的存储计算专用指令进行译码、切换系统的工作模式、控制SRAM宏单元的读写和存储计算,所述SRAM宏单元具有三组行译码器以及一组列译码器,所述SRAM宏单元中还包括一IM-ALU单元,所述IM-ALU单元内含有少量逻辑门和若干选择器,所述逻辑门用以完成在SA之外的存储计算操作,所述选择器用以选择计算后的结果写回存储器阵列中,所述存储计算协处理器控制计算机的架构模式选择并且由CPU所取指令类型控制,在普通模式中,所述存储计算协处理器处理数据遵循“从存储器读取数据-处理数据-将数据写回存储器”的范式,在IMC模式下,架构等效为非冯诺依曼体系结构,CPU将存储计算指令发送至存储器的SRAM宏单元中,直接在存储器中进行数据的运算,并直接写回Bitcell。

2.根据权利要求1所述的一种支持存储内计算的双模计算机架构,其特征在于,所述列译码器由存储计算协处理器进行配置。

一种支持存储内计算的双模计算机架构

技术领域

[0001] 本发明涉及计算机架构技术领域，具体涉及一种支持存储内计算的双模计算机架构。

背景技术

[0002] 当前大多数计算机都采用了传统的冯诺依曼结构，即计算单元和存储单元分离。这样的计算机工作时遵循“(从存储器中)加载-(在CPU中)计算-写回(存储器)”这样的范式。一方面，由于芯片或者IP引脚受限，计算和存储分离使得计算单元和存储器之间的带宽也受到限制，不能充分利用存储模块的带宽资源。另一方面，计算存储分离使得CPU在处理数据时，需要不停地对远处的存储器进行读写，这样就不可避免地带来巨大的功耗浪费。例如，一个64位的双精度浮点运算大概需要消耗50pJ能量，CPU核从20mm外的缓存读取数据大概消耗500pJ能量，而从片外的DRAM读取相关数据则需要大约6000pJ的能力。而随着大数据、人工智能和物联网的发展，解决CPU和存储器之间的带宽瓶颈、提高系统的能量利用效率显得日益迫切。处理器架构的设计面临着更大的挑战。例如，如何满足人工智能芯片中计算单元对存储模块的带宽需求，如何使得物联网终端(多为MCU，即微型中央处理器)在提高处理性能的同时保持甚至降低功耗。

[0003] 对于这些问题，其中的一个解决方案是在存算一体化(PIM，Processing-In-Memory，或者IMC，In-Memory Computing)，其设计理念是尽可能使得计算单元和存储模块靠近，甚至能够在融为一体，即在存储器中进行计算。

[0004] 其实，存算一体概念早在上世纪七十年代就被提出来了，由于CPU对存储器的带宽需求或者功耗不是计算机技术发展的主要矛盾，这个想法没有受到足够的重视。直到九十年代，大量的存算一体计算机涌现，设计思路可以概括为将计算单元和存储单元放到一块芯片中，用逻辑工艺或者DRAM工艺生产。然而，由于这样的设计生产耗费极大，而且设计出来的处理器在可编程性，可升级性方面都比较差，最后并没有得到推广。

[0005] 近年来，随着人工智能的发展，基于静态随机存储器(SRAM，Static Random Access Memory)的存储计算技术得到广泛关注。该技术的基本原理是同时对一对位线上的多个SRAM存储单元(Bitcell)进行读取，那么位线上的电压降行为表现为逻辑与的关系，如果在位线末端使用双输入双输出的灵敏放大器(SA，Sense Amplifier)，那么便可以在SA端得到多种逻辑结果。图1描述了基于SRAM的存算一体技术原理。

[0006] 然而目前基于SRAM的存算一体设计中，绝大多数都是针对人工智能加速，还没有一款是面向通用目的计算。

发明内容

[0007] 本发明的目的在于克服现有技术中存在的问题，提供一种支持存储内计算的双模计算机架构，它可以实现至少一定程度上解决现有技术中的问题。

[0008] 为实现上述技术目的，达到上述技术效果，本发明是通过以下技术方案实现的：

[0009] 一种支持存储内计算的双模计算机架构，包括一处理器核、一指令存储器、一存储计算协处理器、若干计算型的SRAM宏单元，所述处理器核采用精简指令集核心，具有六级流水线，在取指级和译码级插入了预译码级流水，用于对存储计算专用指令进行译码，所述指令存储器采用存储编译前生成，所述存储计算协处理器用于对处理器核传送过来的存储计算专用指令进行译码、切换系统的工作模式、控制SRAM宏单元的读写和存储计算，所述SRAM宏单元具有三组行译码器以及一组列译码器，所述SRAM宏单元中还包括一IM-ALU单元，所述IM-ALU单元内含有少量逻辑门和若干选择器，所述逻辑门用以完成在SA之外的存储计算操作，所述选择器用以选择计算后的结果写回存储器阵列中，所述存储计算协处理器控制计算机的架构模式选择并且由CPU所取指令类型控制，在普通模式中，所述存储计算协处理器处理数据遵循“从存储器读取数据-处理数据-将数据写回存储器”的范式，在IMC模式下，架构等效为非冯诺依曼体系结构，CPU将存储计算指令发送至存储器的SRAM宏单元中，直接在存储器中进行数据的运算，并直接写回Bitcell。

[0010] 进一步地，所述列译码器由存储计算协处理器进行配置。

[0011] 本发明的有益效果：与传统冯诺依曼体系结构相比，本发明具有双模体系结构的特点，即本发明提出的架构可以工作在非冯诺依曼体系结构下，也保留了传统的冯诺依曼结构。这样做的目的是最大限度地利用现有的编译工具链和编程模型。通过仿真表明，根据使用IMC-SRAM宏单元数量的不同，与基准(没有存储内计算功能，其他与本架构一样)对比，本发明架构可以对特定的应用提供十几至上百倍的加速。在采用一块宏单元的情况下，可以对规模为32*32的二值神经网络(BNN)和512位字符输入的哈希(Hash)算法分别加速6.7和12.2倍。与基准对比，本架构平均节省能耗为3倍。

附图说明

[0012] 为了更清楚地说明本发明实施例的技术方案，下面将对实施例描述所需要使用的附图作简单地介绍，显而易见地，下面描述中的附图仅仅是本发明的一些实施例，对于本领域普通技术人员来讲，在不付出创造性劳动的前提下，还可以根据这些附图获得其他的附图。

[0013] 图1为现有技术中基于SRAM的存储内计算原理图；

[0014] 图2为本发明提出的支持存储内计算的SRAM架构图；

[0015] 图3为本发明提出的双模计算机架构；

[0016] 图4为传统指令和IMC指令的对比示例；

[0017] 图5为本发明提出的在SRAM内进行自适应并行向量运算的示意图。

具体实施方式

[0018] 为了使本发明实现的技术手段、创作特征、达成目的与功效易于明白了解，下面将结合本发明实施例中的附图，对本发明实施例中的技术方案进行清楚、完整地描述，显然，所描述的实施例仅仅是本发明一部分实施例，而不是全部的实施例。基于本发明中的实施例，本领域普通技术人员在没有作出创造性劳动前提下所获得的所有其它实施例，都属于本发明保护的范围。

[0019] 图1给出了基于SRAM位线放电进行计算的原理。其中左图为在一列Bitcell上做逻

辑运算的原理图,右图为传统的6管单元(Bitcell)。将需要做逻辑运算的数据写入Bitcell中(以图中的Bitcell j和Bitcell k为例),之后将位线BL和BLB预充电至高电平。之后同时打开字线WLj和WLk,两边位线会根据Bitcell j和Bitcell k所存储的数据而进行不同情况的放电。对于某个Bitcell而言,若其存储的是0(即图1右图中的Q为0),BL就会通过M5和M3放电至地,相反,则是BLB通过M6和M4进行放电。可以得知,对于某根位线而已,只要该侧有bitcell存储的是0,位线则会放电,通过SA比较放大后,在SA输出逻辑0,自然地形成了一种逻辑与的关系,同理可以得到与非NAND、或OR和或非NOR等逻辑。在SA后增加一个与非门,则可以得到异或逻辑。再增加几个逻辑门,则可以得到加法等运算结果。这就是基于SRAM进行存内计算的基本原理。

[0020] 图2是本发明根据图1所述原理提出的计算型SRAM宏单元,叫做IMC-SRAM。与传统的SRAM不同的是,为了能够进行存内计算,IMC-SRAM采用了三套行译码器和一套可配置的列译码器。此外,IMC-SRAM还包含有一个轻量级的算术逻辑单元(IM-ALU),主要包含一些逻辑门,用以完成SA之后的部分运算。

[0021] IMC-SRAM工作模式是可以配置的,当系统工作与普通模式时,IMC-SRAM等同于传统的SRAM,只有一套行译码器和列译码器起作用,读写行为均和普通SRAM无异。当系统工作于IMC模式时,IMC-SRAM也会切换到该模式。此时,IMC-SRAM直接读取数据进行计算,并且将结果写回存储阵列中。该模式下,两个行译码器Ar1和Ar2用于两个源操作数的读取,Ar3用于将计算结果写回阵列。由于进行计算的两个操作数需要行对齐存储,所以操作数可以共用一套列译码器。当在IMC模式下进行两个操作数的读操作是,可以通过VL_cfg配置需要读取的列。Mode、VC_cfg和行列地址等数据均来源于IMC-CP。

[0022] 图3是本发明提出的双模计算机架构总览图。可以看出,架构包含一个精简指令集CPU核,一个指令集存储器、一个存储计算协处理器IMC-CP和若干计算型存储宏单元。该架构与传统的冯诺依曼体系结构的主要区别是采用了计算型存储器IMC-SRAM,CPU不仅可以根据地址对存储器进行存取,还可以发送指令给存储器,让数据在存储器中进行运算,并直接将运算结果写回存储器。传统指令和存储计算指令位宽相等,可以混合存储于同一个指令存储器,CPU对取回来的指令进行译码,如果是传统类型指令,则继续执行流水线操作;如果是存储计算型指令,则将指令发送至IMC-CP处理。此时,如果CPU中下一条指令与访存无关,则继续执行,否则,流水线将暂停,直到存储计算完成。接收到存储计算指令的IMC-CP将模式切换到IMC模式。存储计算指令中包含有存储计算的操作码、源操作数地址、目的操作数地址和计算的向量长度。IMC-CP将操作数地址、操作码等信息传送到IMC-SRAM,并控制IMC-SRAM进行存储运算,直到整个存储计算完成,发送一个Finish信号给CPU,表示存储计算已经完成。CPU可以继续运行。同时,IMC-CP将系统切换到普通模式,IMC-SRAM随机也被切换到该模式。

[0023] 图4是本发明架构采用的CPU流水线示意图。其中的IF、PID、ID、EX、MEM和WB分别表示取指、预译码、译码、执行、访存和写回几个流水级,其中,在IF和ID直接插入的PID是本发明提出的预译码级,用于对指令存储器中的混合指令进行译码,用以判断一条指令是属于传统指令还是存储计算指令。

[0024] 图5给出了一个本发明提出的自适应的存内向量计算实例。左图是使用传统方式和存内计算方式的编程对比图。A和B是分别是具有20个分量的向量,A和B的各对应向量分

别相加，结果存于C向量中。图中分别给出了使用传统范式和存储计算指令编程的对比。右图是存储计算指令在IMC-SRAM中的映射关系。一个bank的IMC-SRAM每行有8个字(word)，使用2个bank的IMC-SRAM，则A和B的前16个分量加法可以在一个周期内计算完，剩余的四个分量的加法在第二个周期计算。此设计使得IMC-SRAM可以在一个周期内进行大量的向量并行计算。如果采用3个bank的IMC-SRAM，则每个周期最多可以计算14个向量分量，那么A和B的向量加法便可以在一个周期内完成。对于一个特定长度的存内向量运算，一共需要花费多少个周期才能完成，这个是由IMC-CP根据接入的IMC-SRAM块数自动计算，这就是本发明架构的自适应存内向量计算。

[0025] 以上公开的本发明优选实施例只是用于帮助阐述本发明。优选实施例并没有详尽叙述所有的细节，也不限制该发明仅为所述的具体实施方式。显然，根据本说明书的内容，可作很多的修改和变化。本说明书选取并具体描述这些实施例，是为了更好地解释本发明的原理和实际应用，从而使所属技术领域技术人员能很好地理解和利用本发明。本发明仅受权利要求书及其全部范围和等效物的限制。

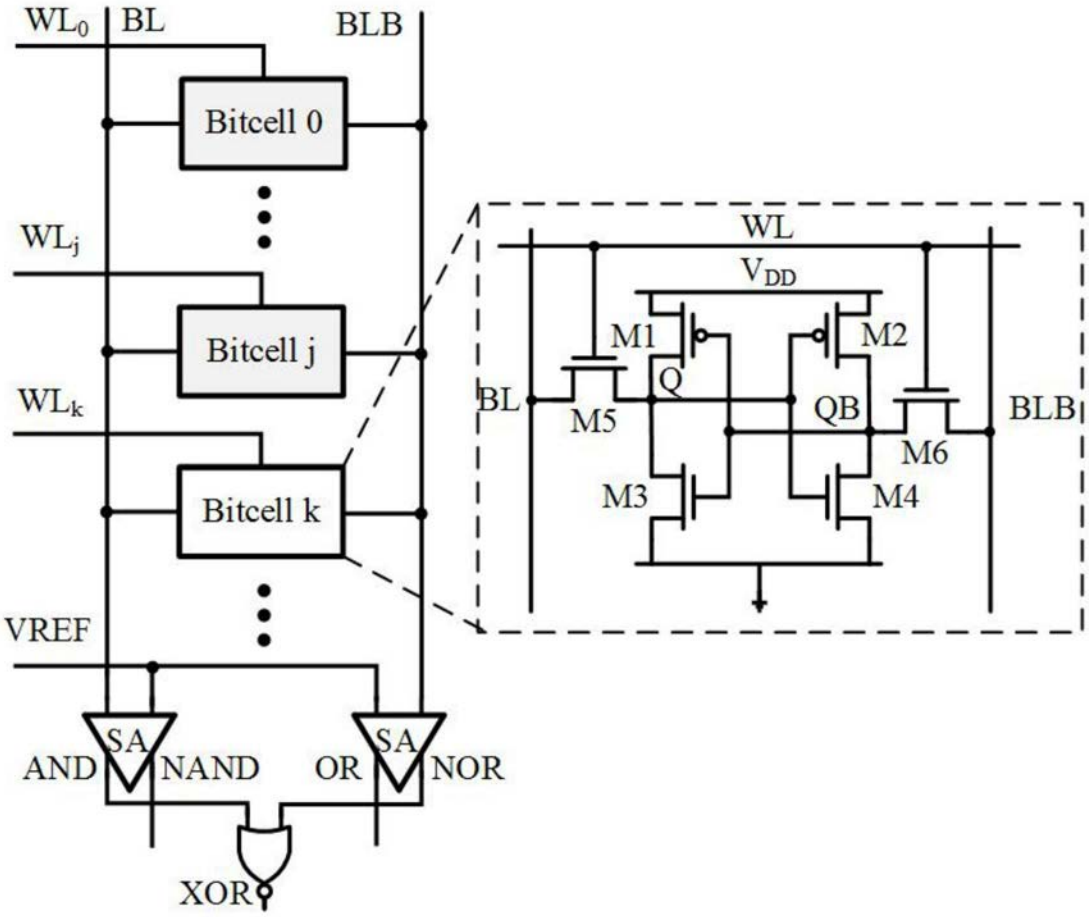

WL0
BL
BLB
Bitcell 0
WLj
Bitcell j
WLk
Bitcell k
VREF
SA
AND
NAND
OR
SA
NOR
XOR
WL
VDD
M1
M2
Q
BL
M5
QB
BLB
M6
M3
M4

图1

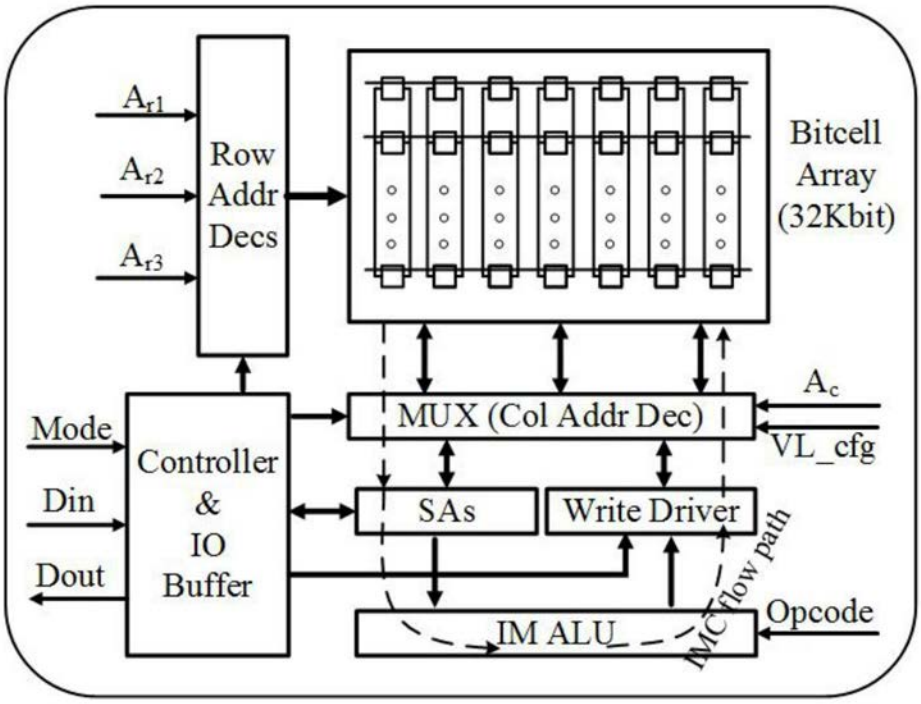

A_{r1}
A_{r2}
A_{r3}
Row Addr Decs
Bitcell Array (32Kbit)
A_c
MUX (Col Addr Dec)
VL_cfg
Mode
Din
Dout
Controller & IO Buffer
SAs
Write Driver
IMC flow path
IM ALU
Opcode

图2

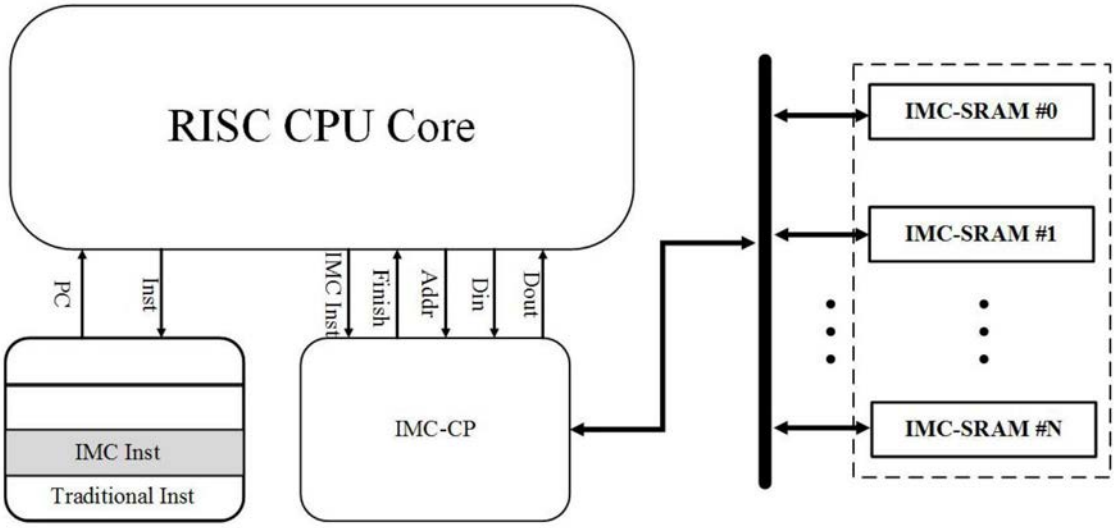

RISC CPU Core
PC
Inst
IMC Inst
Finish
Addr
Din
Dout
IMC Inst
Traditional Inst
IMC-CP
IMC-SRAM #0
IMC-SRAM #1
IMC-SRAM #N

图3

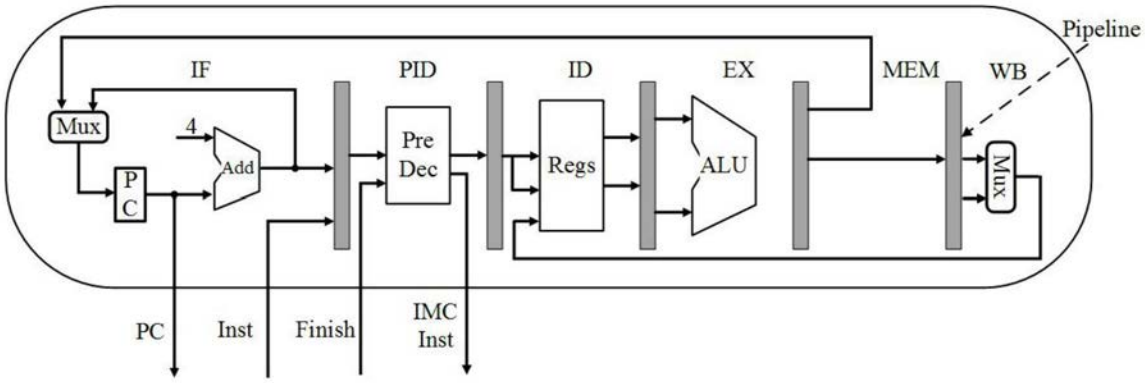

Pipeline
IF
PID
ID
EX
MEM
WB
Mux
4
Add
P
C
Pre
Dec
Regs
ALU
Mux
PC
Inst
Finish
IMC
Inst

图4

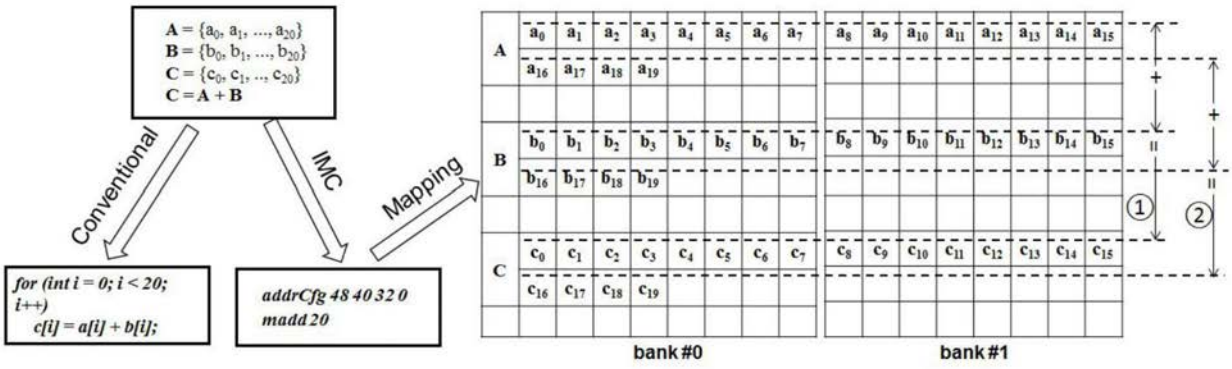

A = {a0, a1, ..., a20}
B = {b0, b1, ..., b20}
C = {c0, c1, .., c20}
C = A + B
Conventional
IMC
Mapping
for (int i = 0; i < 20; i++)
c[i] = a[i] + b[i];
addrCfg 48 40 32 0
madd 20
A
B
C
bank #0
bank #1
①
②

图5