

利用数据流语言 DFC 实现矩阵乘法

赵文杰

摘要

在过去，控制流计算模型与体系结构主导了大部分计算机体系结构的设计方向。但是，近些年来处理器制造技术面临瓶颈，控制流计算体系结构的短板日益浮现；随着机器学习，科学计算等对计算资源敏感的计算任务规模增长，控制流计算模型逐渐开始转向数据流计算模型。DFC 数据流语言由深圳大学 SKT 团队开发。本文复现了基于 C 语言的语法扩展的数据流语言 DFC (DataFlow C) 与 DFC 运行环境。并利用 DFC 语言实现了数据流式矩阵乘法运算。最终实验证明 DFC 能利用数据流机制完成进程通信并正确运行计算结果，编译时还能同时生成数据流 DAG。

关键词：并行计算；数据流计算；矩阵乘法

1 引言

目前，日益增长的数据量、日益复杂的数据关系，都向计算机的处理性能提出挑战。学术界与工业界为此在执行模型、访存模式、并行算法、编程接口等多方面寻求突破，其中执行模式方向就有数据流作为最有希望的技术之一。数据流执行的概念在二十世纪 70 年代提出，而在上世纪末期陷入低谷。近十年，随着基于控制流执行模型的冯诺依·曼架构在并行运算、存储访问等方面的问题日益严重，数据流执行再度回归到研究人员的视野中。

传统的并行编程工作易于出错、难于调试，编程者需要显式地处理并发、同步等问题，多线程编程的缺陷和不足在文献业界已有充分讨论。数据流计算克服了冯氏架构中的程序计数器 PC 和全局共享内存的问题，更适合大规模并行计算。数据流编程语言在过去几十年里得到很大的发展，特别是在可视化编程语言领域更加活跃。近期的发展方向，与在上世纪七八十年代的工作的明显不同在于：当前更趋向于混合数据流和控制流的方向（或称混合冯氏-数据流）、采用粗粒度的形式。面对早期的数据流处理器硬件，系统软件开发发现难以将传统命令式语言编译成可在数据流硬件上运行的程序。但是通过对命令式语言进行限制，例如赋值方面的约束，则可以构建出相符的语言，我们称这类语言为数据流编程语言。

2 相关工作

此部分对课题内容相关的工作进行简要的分类概括与描述，二级标题中的内容为示意，可按照行文内容进行增删与更改，若二级标题无法描述内容进行概括，可自行增加三级标题，后面内容同样如此，引文的 bib 文件统一粘贴到 Ref 文件夹下的 Collection.bib 中并采用如下引用方式^[1]。对

2.1 早期的数据流语言

早期伴随着数据流硬件的发展，如果在这样的硬件上进行编程就成为一个新的问题。由于计算是通过数据依赖关系来调度的，因此编程语言必须暴露这些依赖关系。然后在各种语言中数据依赖的表达程度差别很大，并行性的表达也很不相同（隐式或显式）。因此需要寻找一个新的数据流计算机“编程范式”以及“能产生计算流图的编译器”。通过分析命令式、函数式、逻辑式的编程，最终确定函

数式语言（functional language）作为数据流编程语言。虽然可以用数据流图来表示程序，但实际上计算流图的产生是需要人工或编译器来完成，除非使用可视化编程 DVPL 才能让程序员直接产生数据流图表示的程序。

2.1.1 函数式语言方向

TDFL 是属于最早一批的文本式的数据流语言，编译后能构建数据流图并检测死锁。TDFL 程序包括一系列模块，类似于普通语言的过程，内部有一串赋值语句（满足单赋值约束）、条件语句或调用其他模块。并不提供循环，但是模块允许递归调用。LAU 语言是法国 ONERA-CERT 计算架构组于 1976 年在 LAU 静态数据流架构上的开发的。它包含了条件分支和循环，并通过使用 old 关键字来保证单次赋值。属于一小类通过 expand 关键字显式地完成并行赋值。其次还有类似面向对象的编程、可以对数据和操作进行封装。Lucid 是一种用于形式化验证的函数式语言。它通过引入两个非数学特征（变换和赋值）是实现迭代，但是认为递归（recursion）对于 loop 来说过于严格。Lucid 数学上可表示地通过 next 关键词和单次赋值约束实现迭代。Id 用于操作系统开发，设计成没有顺序控制和内存单元的——也就是否定冯氏模型。该语言是单次赋值、基于块和表达式。为了表示数据结构，引入了 I-structures（advance language 在 6.3）。LAPSE 基于 Pascal 语言用于 Manchester 数据流机，包括单次赋值语法、函数、条件评估、用户自定义类型。没有使用专用关键字来表示迭代，编译器默认旧值在表达式右边而新值在表达式左边。类似 LAU，LAPSE 提供了单独一个结构，for all 用于并行数组赋值。VAL 程序由一系列函数构成，每一个都能返回多个数值。Loop 循环采用 Lucid 技术，也有 for all 并行赋值结构，由于应用领域不需要递归（recursion）故没有提供。缺点包括没有通用 IO、无法表达非确定性程序。Cajole 是一个函数式语言可以编译成 DAG, 但不允许递归。DL1 用于混合数据流架构的函数式语言，可以编译成底层的数据流图，而且比其他语言更加明晰——具有子图 subgraph 关键字。提供递归和条件执行。SISAL 类似 Lucid，是结构化函数式语言，提供条件评估、满足地磁复制的迭代。虽然提供数据结构，但是被当作数值而无法项 I-structure 那样可以重写。唯一的并行结构就是 parallel loop。Valid 是一个函数式语言，提供递归，也提供采用 Lucid 方法的函数式 loop，以及一个并行 loop 结构。

2.1.2 传统命令式语言方向

依赖于编译器，可以将命令式语句来实现数据流计算。这些语言可以是 Fortran、Pascal、以及 C 语言的变种。文献^[2]指出在数据流机上用命令式语言编成时，产出高质量的代码将是一个极其艰难的任务。同时也指出，非函数式的因素，例如全局变量，将会降低并发性。基于命令式语言的原因是因为已经有大量的命令式语言所编写的软件以及熟悉它们的程序员。而新语言在产业界是否成功很大程度上取决于现有开发人员的编程习惯和迁移意愿。在文献^[3]提到在相同复杂度的编译器前提下，数据流语言在几个方面优于命令式语言。也提到使用更加复杂的编译器技术和显式的并行结构，命令式语言也可以达到和数据语言相同的性能。虽然不否认语法上的优势，但是也不足以支撑必须要一个全新的数据流语言。在数据流语言加入显式并行的想法被研究人员所抵制，这可能是由于隐式并行才最具有吸引力。文献^[3]提到，只有证明搞糟数据流语言是为了在通过利用并行性来提升性能时，才能否定创

造特定的数据流编程语言。但是，实际上数据流语言在对编程者而言具有若干个优势。可视化数据流编程语句则提供了更多的证据，优势存在于软件工程领域。这就说明数据流语言本身的必要性

2.2 近期的数据流语言

过去十多年里，数据流语言在可视化编程领域中有了较多的发展，而传统概念上的数据流的进步并不明显，主要在于编译器和硬件上的改进。由于细粒度问题已经很明确，因此大家都是基于对冯氏架构加以约束或者在多核冯氏机器上以多线程方式运行数据流程序。这就导致了将数据流程序编译成一组串行线程，但是线程间按照数据流激活（fire）方式来执行。在这种情况下，数据流语言的底层语义仍保持不变，不影响语言本身。数据流可视化编程语言虽然已经存在很久，但是在上世纪 90 年代廉价的图形显卡普及后才蓬勃发展。它也导致了一些性的挑战和问题，包括可视化语言自身的问题、数据流语言自身的问题等。数据流机上的“机器”语言是数据流图，数据流文本语言必须经过编译生成计算流图才能在数据流机上调度执行。数据流图的方式存在优势，便于创新思想的交流、更高效的开发。可视化语言的动态语法和可视化等优势已经被一些商业产品所证实。它还让开发者以自然的方式思考程序设计问题。

2.2.1 可视化数据流语言

最早的 DDN（Data Driven Nets）数据流可视化语言程序是由周期性的数据流图构成，带有类型的数据以 FIFO 形式在边（arcs）上流动。DDN 证明无需文本，也能表示迭代、过程调用、条件执行等。后来的 GPL（Graphical Programming Language）本质上是在高层次的 DDN，允许利用子图概念进行自顶向下的设计，子图可以递归声明。边是带有类型的，而且整个环境支持调试、可视化和文本编程。FGL（Function Graph Language）基于结构模型而不是基于令牌的模型，其他方面和 GPL 很像。Grunch 系统是由 Cajole 文本数据流语言开发者开发的，作为 Cajole 的图形表示——最终转换成 Cajole。在信号和图像处理领域，特别适合数据流计算，因此也出现了相应的数据流语言工具。Labview 就是被广大读者所知道的数据流语言工具，用于实验室进行数据分析的虚拟仪器开发。相对于使用 C 语言开发，Labview 的效率和舒适度极高。ProGraph 相对于 Labview 而言更加通用，是基于面向对象编程的。类似于 Labview，可以重复构建、也允许将一个图压缩成一个节点。NL 通过引入 block 和 guard 节点来提供控制流特性，循环也是有相似的办法实现的。

2.2.2 协调性数据流语言

协调性语言的概念，程序由两部分构成，一个是子任务的计算，另一个子任务间的协调。理由是分布式和异构计算的普及，使得协调性语言成为必要——数据流语言正好适合。例如 Vipers DfVPL 语言可以完成协调任务，内部节点却使用 Tcl 语言来描述。另一个例子是 Granular Lucid（GLU），它建立在 Lucid 基础之上，但是功能由外部语言定义。

3 本文方法

3.1 DFC 语言简介

数据流通用 C 语言 (Dataflow C language, DFC) 是由深圳大学高性能计算研究所的系统技术组 SKT (System Kernel Technology Group) 开发, 旨在提供通用的数据流编程语言。DFC 语言需要对数据流程序进行解释。DFC 描述数据流图时, 需要区分输入、输出数据, 以及数据流图中的函数节点对数据的依赖关系。以下为新增数据流特性的概念:

1.DFC 主动数据 (Active Data, AD) AD 数据与传统数据流执行模型中 actor 触发局部控制事件所依赖的数据概念相似, 我们将 DF 函数所依赖的输入数据称为主动数据。

2.DFC 源主动数据 (Source Active Data, 源 AD) 按照设计思想, 按照设计思想, 数据流图入口函数没有数据流执行条件, 它们负责为数据流图内的 DF 函数提供主动数据; 主动数据由源数据流函数提供。

3.DFC 源数据流函数 (Source Dataflow function, 源 DF 函数) 当一个数据流函数的主动数据列表为空, 我们称之为源数据流函数。

4.DFC 非源数据流函数 (Non Source Dataflow function, 非源 DF 函数) 当一个数据流函数的主动数据个数不为 0, 我们称之为非源数据流函数, 非源数据流函数可以不断产生主动数据向外传递

3.2 本文方法概述

3.2.1 DFC 语言编译器与编译环境

在 DFC 语言系统中, 需要存在 DFC 源代码、DFC 编译器、C 语言编译器、DFC runtime 运行库。它们构成的关系如图 1所示, DFC 源代码经过专用的 DFC 编译器生成符合标准 C 语法的程序, 然后再经过 C 编译链接器生成可执行程序就可以运行在通用平台上。

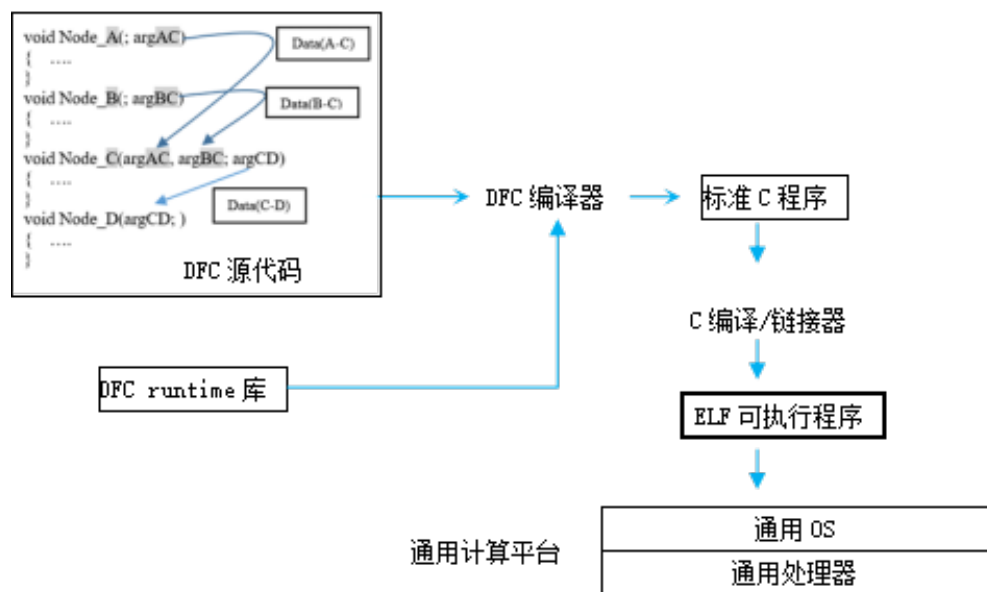


图 1: DFC 数据流计算系统构成^[4]

DFC 代码必须转换为正常的 C 代码, 以便将其编译为 GCC 的可执行程序, 与 DFC 运行时库一起运行。为了实现这一点用户可以使用命令 “dfcc-k-v-o Myfile Myfile.dfc”, 其中 dfcc 是 DFC 编译器。DFC 编译器的工作流程如图 2所示。DFC 编译器执行以下操作: 预处理、词汇分析、解析、语义分析

(语法导向翻译)，将输入程序转换为中间程序表示以及代码生成。

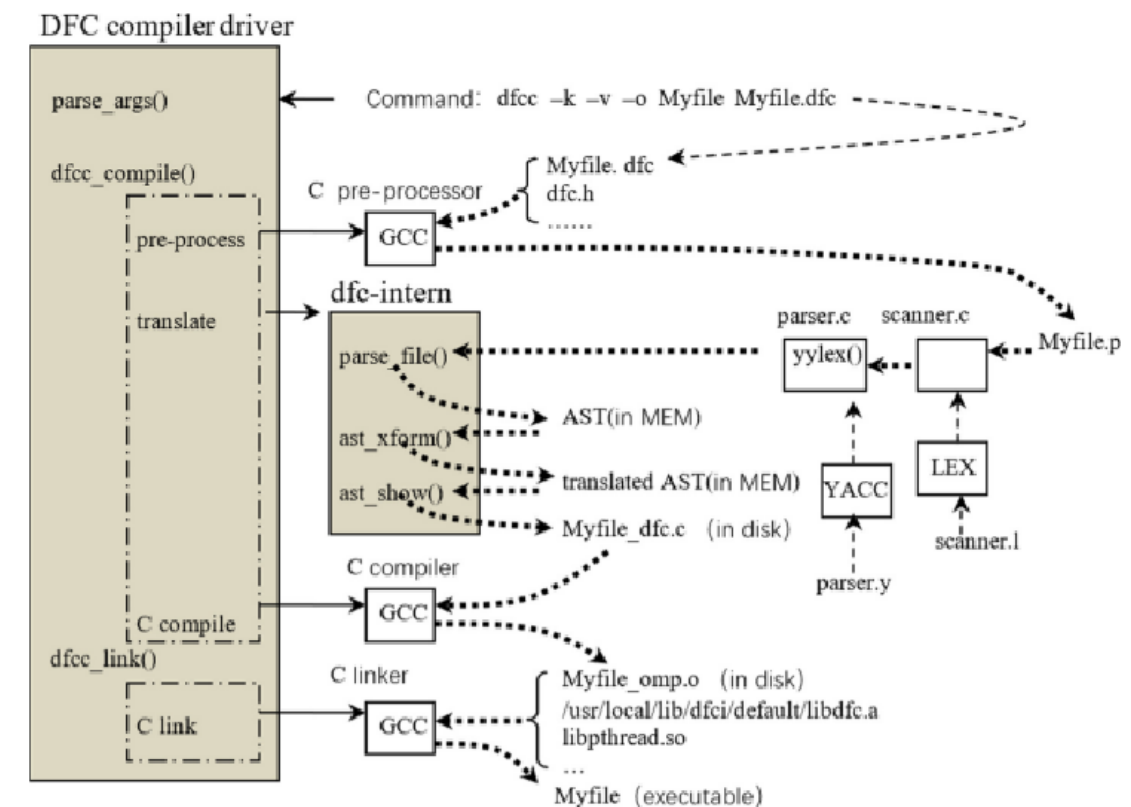


图 2: DFC 编译器示意图^[5]

首先，在预处理期间，三行代码将插入到源 DFC 代码（Myfile.DFC，如图 2 所示）的开头，包括必要的头文件。然后 GCC 的 C 预处理器将 Myfile.dfc 扩展到 Myfile.p。然后，词法分析器/扫描器和解析器将使用以描述 DFC 词汇和语法规则的“scanner.l”和“parser.y”来进行进一步处理。DFC 编译器采用 AST（抽象语法树）作为中间表示分析器输出，可以保持源的层次语法结构，从中可以很容易地获得相反的代码。建立 DFC 的 AST 后，DFC 编译器将 DFC 的 AST 转换为标准 C 的 AST，以便可以导出在 C 代码中实现的转换程序。一旦 C 代码版本该程序可用，可执行程序的生成的工作就交给 GCC。如前所述，DFC 编译器是一个预编译器或源代码转换器，仅负责将 DFC 代码转换为标准 C 代码。DFC 编译器重用了开源 OMPi 编译器源代码的一部分，用于词汇/语法分析和 AST 框架。

3.2.2 DFC 语言矩阵乘法编程

矩阵乘法的并行计算是一个并行计算领域的一个经典问题。在 DFC 语言中，利用数据流的序列触发特性，能以一种更简单的方法实现矩阵乘法的并行。一个矩阵乘法的数据流图可以被简单的编写成两个 DF 函数，如图 3 所示，其中两个 DF 函数就足以表达整个计算流。DF 函数 F_SRC 是数据源节点，它向下一个节点发送驱动数据。由于矩阵的全部子矩阵数据 $A_{00} A_{11}$ 与 $B_{00} B_{11}$ 是可被全部线程访问的的全局共享数据，因此 F_SRC 只需要发送子分块矩阵的行列坐标索引即可。DF 函数 F_MM 对应结果矩阵元素 C_{ij} 的生成。

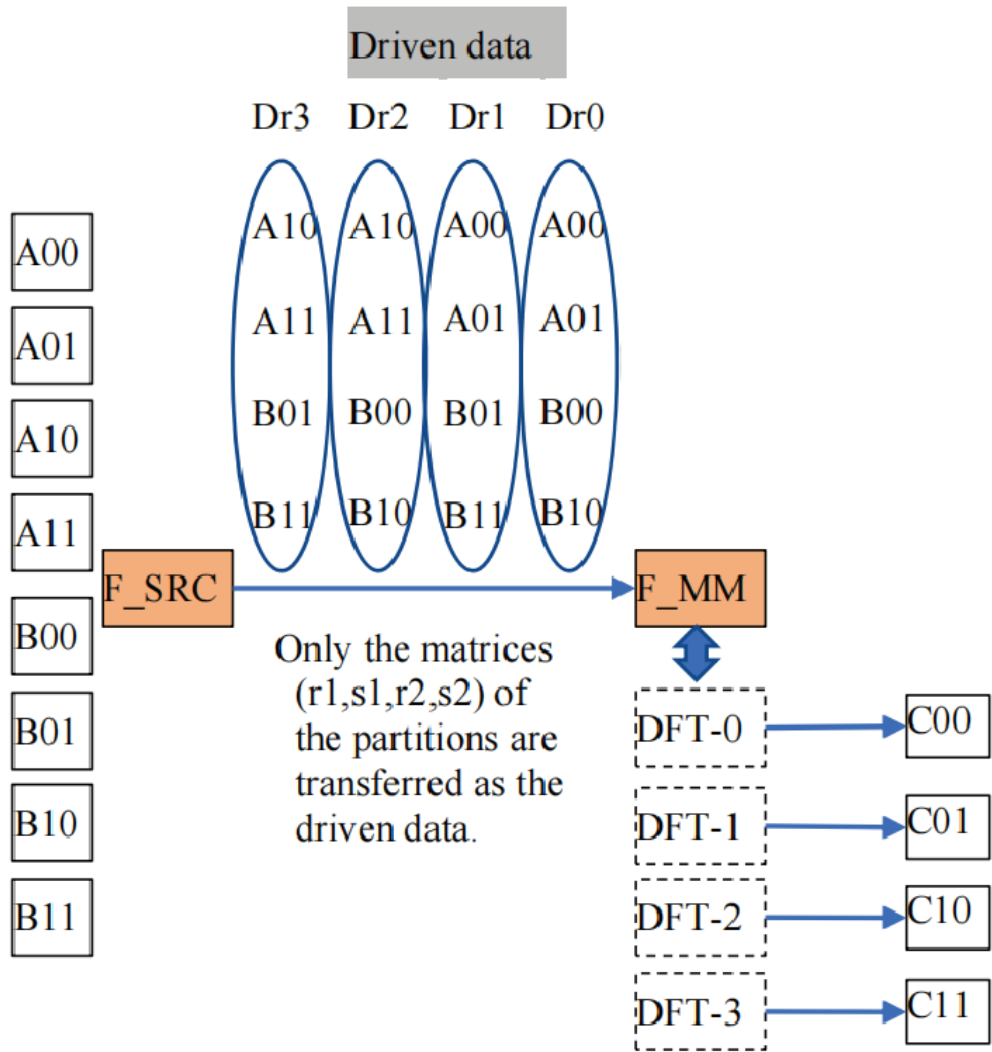


图 3: DFC 矩阵乘法示意图^[6]

4 复现细节

4.1 与已有开源代码对比

论文中没有提供任何利用 DFC 语言进行矩阵乘法实现的代码，只有多年前课题组在 github 上遗留下来的不完善的 DFC 语言编译器。本工作对 DFC 编译器的 BUG 进行了修复，复现了整体，并利用其完成了矩阵乘法运算

4.2 实验环境搭建

本次实验在 ubuntu 系统上开展，DFC 数据流计算系统由编译器与运行环境构成，我们需要分别安装配置这两个组件。首先安装编译器，具体命令如图 4 所示，进入编译器路径后先设定安装环境变量，然后执行 make 与 make install 操作即可完成安装。然后按照 4 编译的环境配置，将运行环境的三个组件解压到对应位置，整个 DFC 编译与运行环境就搭建完成了。

```

2 ## 使用方法
3 ### DFC编译器安装
4 ```makefile
5 cd compiler/ompi
6 ./configure --prefix="安装路径"
7 make
8 make install
9 ```
10 如果configure和gen_version文件没有权限, 则运行chmod +x configure/gen_version赋予运行权限
11
12 ### 利用DFC编译器将DFC代码转换为标准C代码
13 ```
14 安装路径/bin/ompicc -k -v -s -g -I"path-to-dfc" -I"path-to-threadpool" "program-in-dfc"
15 ```

```

图 4: 编译器的安装

5 实验结果分析

首先依照论文理论编写 DFC 矩阵乘法程序, 以 4×4 的矩阵乘法为例。输入 `ompicc -k -v -s -g -I/home/sktgroup/runtime/src/ -I/home/sktgroup/runtime/threadpool/ test.c` 命令进行 DFC 编译。得到处理好的 C 语言程序 `test_dfc.c`, 对其进行查看, 发现编译过程中出现问题如图 5 所示, 导致生成的 C 语言程序出错, 将该语句注释掉即可。

```

test > C test_dfc.c > [e] program_start
61
62 FILE *fp_sche = NULL;
63
64 pthread_mutex_t sched_info_mutex;
65
66 pid_t gettid() { return syscall(SYS_gettid); }
67

```

图 5: DFC 编译产生的错误

查看生成的数据流图文件如图 6, 检查结果没有发现问题, 图的表现正常。

```

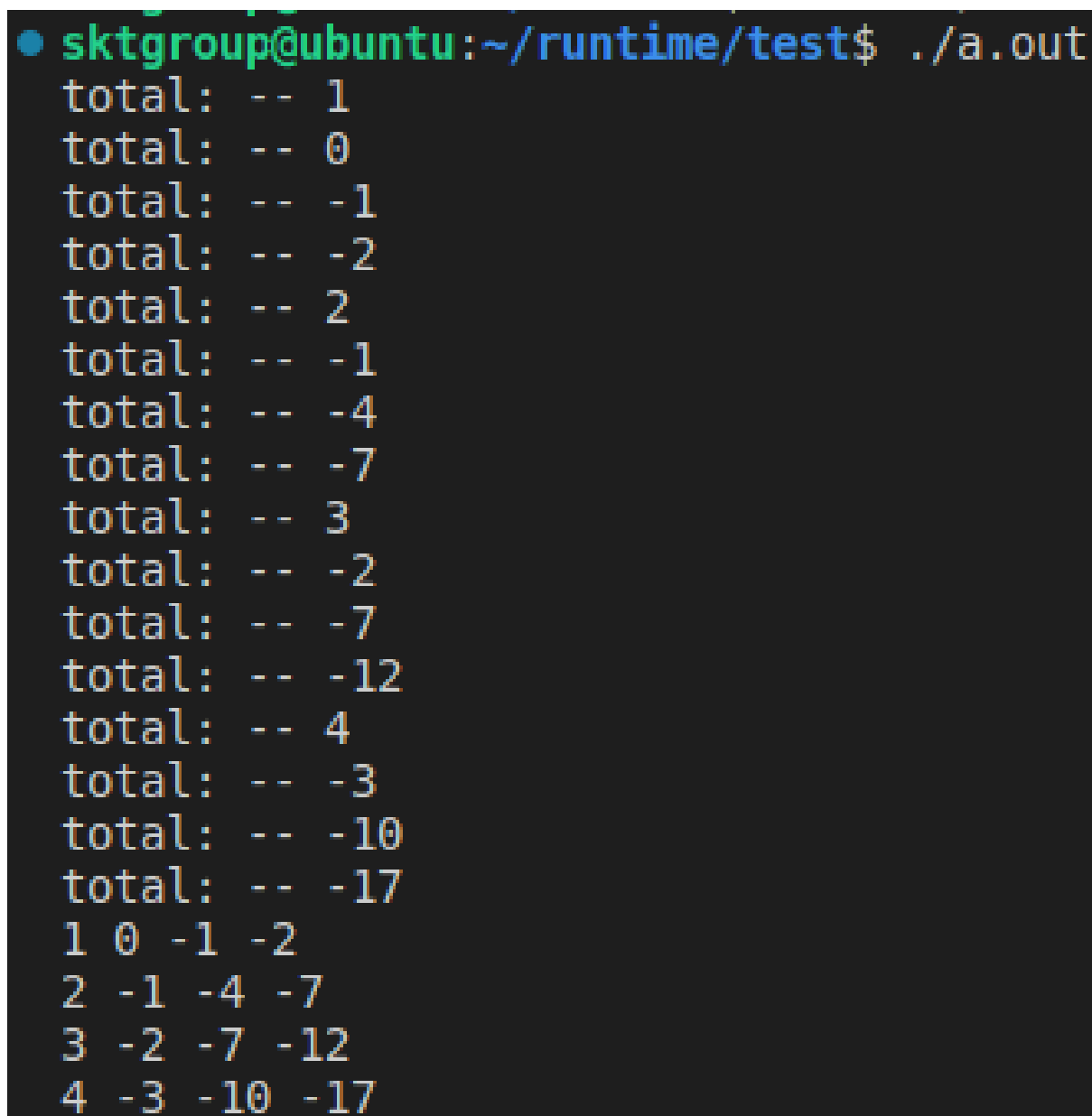
test > ≡ digraph.gv
1 digraph
2 {
3     FUNCS -> TERMINATE [label="DF_output_A"];
4     SOURCED1 -> FUNCS [label="DF_Source_D1"];
5     SOURCEC1 -> FUNCS [label="DF_Source_C1"];
6     SOURCED -> FUNCS [label="DF_Source_D"];
7     SOURCEC -> FUNCS [label="DF_Source_C"];
8     SOURCEB1 -> FUNCS [label="DF_Source_B1"];
9     SOURCEA1 -> FUNCS [label="DF_Source_A1"];
10    SOURCEB -> FUNCS [label="DF_Source_B"];
11    SOURCEA -> FUNCS [label="DF_Source_A"];
12 }

```

图 6: 数据流图文件

然后输入 `gcc -pthread -I/home/sktgroup/runtime/src/ -I/home/sktgroup/runtime/threadpool/src/ -I/home/sktgroup/test_df.c ./src/dfc.c ./threadpool/src/threadpool.c ./c_printf/sources/c_printf.c` 命令进行处理好的 C 语言程序的编译，这里要把运行环境中所依赖的库全部连接上，否则无法正常编译。

如下图 7 为一个 4×4 的矩阵乘法运行过程与结果，可以看到一共花费 16 个数据流执行周期，其中每次全部数据就绪触发合并操作与各源节点从数组 A 或 B 中取数值是并行完成的，整个数据流程序就像一条流水线一样，源节点运行周期不断地去公有变量去取数据，合并节点不断地对到来的数据进行计算，前后数据流节点间通过数据流机制保持同步互斥，互不干扰，减少了传统互斥锁的开销。



```
● sktgroup@ubuntu:~/runtime/test$ ./a.out
total: -- 1
total: -- 0
total: -- -1
total: -- -2
total: -- 2
total: -- -1
total: -- -4
total: -- -7
total: -- 3
total: -- -2
total: -- -7
total: -- -12
total: -- 4
total: -- -3
total: -- -10
total: -- -17
1 0 -1 -2
2 -1 -4 -7
3 -2 -7 -12
4 -3 -10 -17
```

图 7: 实验结果示意

最终的计算结果正确，且符合预期，说明该矩阵乘法程序的编写是正确有效的。值得注意的是 DFC 语言在没有用户同步互斥机制的保护下，依然正确完成的计算，值得令人惊叹。

5.1 创新点

DFC 语言的开发与应用即为本文的创新点，作为本土开发的数据流编程语言，DFC 语言需要支持与鼓励。本文通过复现 DFC 数据流语言的编译环境，执行环境，并依照原理自己编写了 DFC 矩阵乘法程序并成功运行得到正确结果。

5.2 使用说明

- 1.compiler 文件夹为编译器源码，按 4.2 节配置即可完成安装。
- 2.runtime/c_printf,runtime/src,runtime/threadpool 为运行环境组件，按照 4.2 节配置即可完成安装。
- 3.runtime/test 为矩阵乘法 DFC 程序路径，其中 test.c 为 DFC 程序，test_df.c 为 DFC 编译后的 C 程序，a.out 为可执行程序，digraph 为数据流 DAG 文件。

6 总结与展望

本文复现了 DFC 数据流语言的编译环境，执行环境，并依照原理编写 DFC 矩阵乘法程序，并成功运行的到预期结果。DFC 语言不仅能正确的描述一个数据流程序，还能在编译的过程中产生数据流程序的数据流图。DFC 语言依靠数据流执行的原生机制，而不靠线程进程的同步互斥机制，完成线程进程间的通信，正确的完成计算。DFC 语言的潜力还有待进一步开发。

参考文献

- [1] YU J. Research on Runge Phenomenon[J]. Advances in Applied Mathematics, 2019, 08(08): 1500-1510.
- [2] WAIL S F, ABRAMSON D. Can Dataflow Machines be Programmed with an Imperative Language?[J]. IEEE Computer Society Press, 1995, CA(229-265).
- [3] GAJSKI D, PADUA D J, D. A.and KUCLE, ANDKUHL R. A second opinion on data-flow machines and languages.[J]. IEEE Compute, 1982, 15, 2(58-69).
- [4] ZHANG J, LI J, DU Z, et al. The Dataflow Runtime Environment of DFC[J]. Parallel and Distributed Computing, Applications and Technologies, 2021: 57-69. DOI: 10.1007/978-3-030-69244-5_5.
- [5] DU Z, ZHANG J, LI J, et al. The Compiler of DFC: A Source Code Converter that Transform the Dataflow Code to the Multi-threaded C Code[J]. Parallel and Distributed Computing, Applications and Technologies, 2021: 185-197. DOI: 10.1007/978-3-030-69244-5_16.
- [6] DU Z, ZHANG J, SHA S, et al. Implementing the Matrix Multiplication with DFC on Kunlun Small Scale Computer[J]. 2019 20th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2019: 115-120. DOI: 10.1109/PDCAT46702.2019.00032.