PanguLU 26

用户手册

C API 使用与参考

版本：26.44

适用平台：NVIDIA CUDA

修订日期：2026 年 10 月 26 日

目录

1 简介 **1**
1.1 支持范围 . . . 1
1.2 文档约定 . . . 1

2 安装与快速开始 **2**
2.1 运行与编译环境 . . . 2
2.2 解压下载文件 . . . 2
2.3 配置运行路径 . . . 3
2.4 配置 License . . . 3
2.5 编译与运行示例 . . . 4
2.6 应用链接 . . . 5
2.7 CMake 接入 . . . 6

3 通用说明 **7**
3.1 句柄与状态 . . . 7
3.2 执行阶段 . . . 7
3.3 内存所有权与生命周期 . . . 7
3.4 设备数据与结果 . . . 7
3.5 并发使用 . . . 8
3.6 重排与复用配置 . . . 8
3.7 复用条件 . . . 8

4 数据类型与存储格式 **9**
4.1 不透明对象 . . . 9
4.2 pangulu_value_type_t . . . 9
4.3 pangulu_index_type_t . . . 9
4.4 pangulu_memory_space_t . . . 10
4.5 稀疏存储格式 . . . 10
4.6 稠密右端项与解 . . . 11
4.7 显存容量 . . . 12

5 函数参考 **13**
5.1 pangulu_create() . . . 13
5.2 pangulu_config_set() . . . 14
5.3 pangulu_matrix_create() . . . 15
5.4 pangulu_vector_create() . . . 16
5.5 pangulu_multiple_vector_create() . . . 17
5.6 pangulu_execute() . . . 18
5.7 pangulu_matrix_destroy() . . . 18
5.8 pangulu_vector_destroy() . . . 18
5.9 pangulu_destroy() . . . 19

6 使用示例 **20**
6.1 完整 C 示例 . . . 20
6.2 一次提交多个 RHS . . . 21
6.3 同一矩阵，连续提交多组 RHS . . . 22

6.4 相同稀疏结构，矩阵数值改变 22
6.5 运行完整的多 RHS 示例 22

7 错误处理与结果验证 23
7.1 错误状态 23
7.2 常见集成问题 24
7.3 数值精度与残差 24
7.4 性能测量 25
7.5 问题报告 26

附录 A 示例输入文件 27
A.1 矩阵：Matrix Market 坐标格式 27
A.2 右端项：不带尺寸头的纯文本 27

附录 B 文档与支持 28
B.1 交付文件 28
B.2 技术支持 28

附录 C 版本日志 29
PanguLU 26.44 29
PanguLU 26.42 29

1 简介

PanguLU 是面向 GPU 的稀疏线性系统直接解法器，通过 C 接口求解以下形式的线性系统：

$$AX = B, \qquad A \in \mathbb{R}^{n\times n} \text{ 或 } \mathbb{C}^{n\times n}.$$

其中，A 为 n 阶稀疏系数矩阵，B 为已知右端项，X 为待求的解。B 和 X 可以包含一列或多列；多列右端项对应同一个系数矩阵，共享其 LU 因子。

计算依次经过分析、数值分解和求解三个阶段。对于重复求解的问题，可根据系数矩阵是否变化，选择复用因子或复用分析结果。具体流程与适用条件见第 3 章。

1.1 支持范围

本手册适用于 PanguLU 26 的对象式 C API。集成前，请根据下表确认产品能力是否满足应用需要。

项目	支持范围
分解方法	支持稀疏 LU 分解，目前仅支持完整矩阵输入。
数值类型	支持单精度、双精度的实数与复数。
稀疏存储	支持从 0 开始编号的 CSC、CSR、BSC 和 BSR 格式。
右端项与解	使用 GPU 上的列主序稠密存储，支持单个及多个右端项（RHS）。
设备使用	每个线性系统仅由单个 GPU 完成分解与求解；多个 GPU 可以同时分别处理独立矩阵。
内存模式	仅支持显存内（in-core）模式，LU 因子和计算工作区均驻留在 GPU 显存中。

1.2 文档约定

等宽字体用于表示函数、类型、枚举值、参数、路径和命令。示例路径需在执行前替换为实际位置。首次使用时，建议先按第 2 章运行随包示例，再参照数据规范和函数参考接入应用。

文中的 ANALYSIS、NUMERIC 和 SOLVE 分别表示分析、数值分解和求解阶段。代码中请使用完整枚举名称，例如数值分解对应 `PANGULU_PHASE_NUMERIC_FACTORISATION`。

PanguLU 26 保持与 26.42 的既有接口二进制兼容，并新增多 RHS 创建函数。单 RHS 与多 RHS 的函数选择及升级说明，见第 5.4 节和第 5.5 节。

安装要求、交付包选择与配置步骤见第 2 章。

2 安装与快速开始

本章说明首次部署的操作顺序：解压交付包、配置路径与 License、编译示例、检查运行结果。开始前，请准备与目标系统和 GPU 匹配的交付包，以及适用于该设备的有效 License。示例直接链接随包运行库，无需重新构建 PanguLU。

2.1 运行与编译环境

本版交付包适用于 x86_64 Linux。CUDA 12.1 包要求 glibc 2.17 或更高版本；CUDA 12.4、12.9 和 13.1 包要求 glibc 2.34 或更高版本，并需提供含 `GLIBCXX_3.4.29` 的 C++ 运行库。所有包均需提供支持 `GOMP_4.5` 的 OpenMP 运行库 `libgomp.so.1`。

编译示例需使用支持 C11 的编译器及 GNU Make；使用 CMake 时，版本需为 3.18 或更高。

运行时需安装与下载包对应的 CUDA 运行库、cuBLAS、cuSOLVER 和 cuSPARSE。以下列出四种交付包的 CUDA 版本及建议使用的 Linux 驱动基线；使用该版本或更新且支持目标 GPU 的驱动。

交付包	建议驱动基线	包内原生 GPU 计算能力
CUDA 12.1	530.30.02	7.0、7.2、7.5、8.0、8.6、8.7、8.9、9.0。
CUDA 12.4	550.54.15	7.0、7.2、7.5、8.0、8.6、8.7、8.9、9.0。
CUDA 12.9	575.51.03	7.0、7.2、7.5、8.0、8.6、8.7、8.9、9.0、10.0、10.1、10.3、12.0、12.1。
CUDA 13.1	590.44.01	7.5、8.0、8.6、8.7、8.8、8.9、9.0、10.0、10.3、11.0、12.0、12.1。

驱动基线采用 NVIDIA 对相应 CUDA 版本发布的驱动要求。[1] A100 与 A800 的计算能力为 8.0，可选上述任一包；RTX 5090 的计算能力为 12.0，应选择 CUDA 12.9 或 CUDA 13.1 包。通过 `nvidia-smi` 查看 GPU 和驱动，通过 `ldd --version` 查看系统 glibc。

2.2 解压下载文件

以下以 CUDA 12.4、x86_64、glibc 2.34 安装包为例。假设下载文件保存在当前用户的 `Downloads/` 目录，执行下列命令，在用户主目录下创建安装目录，并在解压时去除压缩包最外层目录。文件位于其他位置时，修改压缩包路径。

```
cd "$HOME"
mkdir -p pangulu-26.44-CUDA-12.4-x86_64-glibc2.34
tar -xzf "$HOME/Downloads/pangulu-26.44-CUDA-12.4-x86_64-glibc2.34.tar.gz" \
  -C pangulu-26.44-CUDA-12.4-x86_64-glibc2.34 --strip-components=1
```

请完整保留交付包，尤其是 `lib/` 中的配套运行库。主要文件如下：

```
pangulu-26.44-CUDA-12.4-x86_64-glibc2.34/
  include/pangulu.h     # Public C header
  lib/                  # Runtime libraries
  examples/             # Examples and sample data
  licenses/             # Third-party notices
  SHA256SUMS            # File checksums
```

[1] NVIDIA CUDA 发布说明：12.1、12.4、12.9、13.1。

2.3 配置运行路径

打开 Bash 兼容的终端，设置安装目录和 CUDA 开发工具链目录。下例使用前面创建的目录；请将 CUDA 路径替换为实际位置，并保留双引号，以便处理含空格的路径。

```
export PANGULU_ROOT="$HOME/pangulu-26.44-CUDA-12.4-x86_64-glibc2.34"
export CUDA_ROOT="/usr/local/cuda-12.4"
export LD_LIBRARY_PATH="$PANGULU_ROOT/lib:$CUDA_ROOT/lib64${LD_LIBRARY_PATH:+:
    $LD_LIBRARY_PATH}"
```

PANGULU_ROOT 指向解压后的 PanguLU 安装目录，CUDA_ROOT 指向开发工具链。动态库搜索路径同时加入两个运行库目录，并保留已有设置。CUDA 运行库位于其他目录时，将命令中的 `lib64/` 改为实际位置。

这些设置仅对当前终端及其启动的程序生效。打开新终端后需重新执行；通过作业脚本或其他方式启动应用时，应在对应的启动环境中配置。

2.4 配置 License

将原始 License 文件保存在具有读取权限的目录，使用绝对路径设置文件位置：

```
export PANGULU_LICENSE_PATHS="/absolute/path/to/license.key"
```

将 `/absolute/path/to/license.key` 替换为实际文件路径。例如，文件保存在 `/opt/pangulu/license.key` 时，直接填写该路径。请保持文件内容完整。授权检查在本地完成，计算可离线运行。

2.4.1 授权生命周期

调用 `pangulu_create()` 时，软件读取 License 和签名撤销列表（CRL），检查产品、设备、版本范围、使用权限、有效期限及构建完整性。计算阶段继续校验授权期限，到期后将阻止后续计算或结果提交。

更新 License 或 CRL 后，请先结束当前计算，销毁旧句柄，再创建新句柄。新句柄会重新读取更新后的文件，现有句柄不会自动重新加载。

> **说明：设备选择与多个授权文件**
>
> `pangulu_create()` 的 gpu_index 为已授权 GPU 列表中从 0 开始的序号，与 CUDA 设备编号分别管理。PANGULU_LICENSE_PATHS 使用逗号分隔多个授权文件的绝对路径；PANGULU_LICENSE_PATH_LIST 指定文本列表文件，每行一条路径，忽略空白行及以 # 开头的注释行。列表文件中的条目先于逗号列表条目。应用数据应分配在句柄对应的 GPU 上。

2.5 编译与运行示例

编译随包示例需要 C 编译器、make 和 CUDA 开发工具链中的头文件及运行库。完成前述运行路径与 License 配置后，依次执行以下命令。

```
cd "$PANGULU_ROOT/examples"
make CUDA_ROOT="$CUDA_ROOT"
./example_01_double data/matrix_real.mtx data/rhs_real_1.vec
```

cd 将工作目录切换到安装目录的示例目录，make 编译示例。编译成功后，执行最后一条命令运行双精度实数示例。`./` 表示当前目录，两个 `data/` 路径也相对于该目录。

`CUDA_ROOT` 指定开发工具链的位置。编译器或库路径有变化时，修改示例目录中的 `Makefile`。

2.5.1 检查运行结果

运行后，确认句柄创建和三个计算阶段均成功，并保留阶段耗时及相对残差。若出现错误码、NaN、Inf 或异常大的残差，按第 7 章排查；数值结果是否可用，还需结合应用精度要求判断。

2.5.2 随包示例

下表文件均位于安装目录的 `examples/` 目录。基础示例运行成功后，可按数据类型和复用方式选择其他示例。

示例文件	说明
`example_01_double.c`	演示双精度实数系统的基本调用流程。
`example_02_float.c`	演示单精度实数数据的使用。
`example_03_dcomplex.c`	演示双精度复数数据的使用。
`example_04_fcomplex.c`	演示单精度复数数据的使用。
`example_05_csr.c`	演示 CSR 格式的矩阵输入。
`example_06_cpu.c`	演示主机内存中的矩阵输入；计算及右端项、解仍使用 GPU。
`example_07_l0.c`	演示将重排方法显式设为 L0。
`example_08_multi_rhs.c`	演示一次分解后，通过一次 SOLVE 求解多个 RHS。
`example_09_refactor.c`	演示稀疏结构不变时的数值重分解。

> **说明：示例程序与运行库的分工**
> 文件读取、格式转换、数据上传与下载、残差计算由示例或应用负责，公共 API 接收转换后的数据描述符。使用自有文件时，按附录 A 核对格式，再按第 4 章检查数组。

2.6 应用链接

在源码中包含公开头文件 pangulu.h，并将安装目录的 include/ 加入头文件搜索路径。该头文件提供 C 链接声明，可直接用于 C++ 工程，无需手工复制函数声明。

链接时使用 libpangulu.so 和 libpangulu_platform.so；若应用调用 GPU 内存分配或传输函数，还需链接相应的 GPU 运行库。工程配置可参考本包 examples/Makefile。

2.6.1 直接编译示例

以下命令用于编译手册源码包中的 quick_start.c，该文件位于 examples/ 目录。请先进入文件所在目录，并在当前终端配置 PANGULU_ROOT 和 CUDA_ROOT。

```
gcc quick_start.c -o pangulu_demo -std=c11 -O3 \
  -I"$PANGULU_ROOT/include" -I"$CUDA_ROOT/include" \
  -L"$PANGULU_ROOT/lib" -L"$CUDA_ROOT/lib64" \
  -Wl,-rpath,"$PANGULU_ROOT/lib" \
  -Wl,-rpath,"$CUDA_ROOT/lib64" \
  -Wl,--no-as-needed -Wl,--allow-shlib-undefined \
  -lpangulu -lpangulu_platform -lcudart -ldl -lm
./pangulu_demo
```

编译成功后，第二条命令运行生成的 pangulu_demo。如果开发工具链的运行库不在 lib64/，请将命令中的对应路径改为实际位置。

--allow-shlib-undefined 沿用交付示例的链接配置，允许部分共享库符号在运行时解析，但不会补齐缺失的运行库。部署检查方法见第 7 章。

2.6.2 编译时路径与运行时路径

编译、链接与程序启动使用不同的搜索路径，需分别配置：

设置	用途
-I	在编译时指定头文件搜索目录，包括 PanguLU 和应用使用的 GPU 头文件。
-L	在链接时指定库文件搜索目录；该设置本身不配置运行时搜索路径。
RPATH / RUNPATH	将动态库搜索路径写入可执行文件，供程序运行时使用。
LD_LIBRARY_PATH	在启动环境中指定动态库搜索路径，可通过终端或启动脚本设置。

说明：随应用一起部署运行库

若应用采用并列的 bin/ 和 lib/ 目录，可使用 '-Wl,-rpath,$ORIGIN/../lib'，使运行库搜索位置相对于可执行文件。这里的单引号用于阻止 shell 提前展开 $ORIGIN。目标机器仍需满足 GPU 驱动及厂商运行库的安装要求。

2.7 CMake 接入

手册源码包提供以下 CMakeLists.txt。将它与 quick_start.c 放在同一目录，即可使用安装目录中的运行库与 CUDA 开发工具链构建示例。

```
cmake_minimum_required(VERSION 3.18)
project(pangulu_demo LANGUAGES C)
set(PANGULU_ROOT "" CACHE PATH "PanguLU installation directory")
find_package(CUDAToolkit REQUIRED)
find_path(PANGULU_INCLUDE pangulu.h
  PATHS "${PANGULU_ROOT}/include" NO_DEFAULT_PATH REQUIRED)
find_library(PANGULU_CORE pangulu
  PATHS "${PANGULU_ROOT}/lib" NO_DEFAULT_PATH REQUIRED)
find_library(PANGULU_PLATFORM pangulu_platform
  PATHS "${PANGULU_ROOT}/lib" NO_DEFAULT_PATH REQUIRED)
add_executable(pangulu_demo quick_start.c)
target_compile_features(pangulu_demo PRIVATE c_std_11)
target_include_directories(pangulu_demo PRIVATE "${PANGULU_INCLUDE}")
target_link_options(pangulu_demo PRIVATE
  -Wl,--no-as-needed -Wl,--allow-shlib-undefined)
target_link_libraries(pangulu_demo PRIVATE
  "${PANGULU_CORE}" "${PANGULU_PLATFORM}" CUDA::cudart
  ${CMAKE_DL_LIBS} m)
set_target_properties(pangulu_demo PROPERTIES
  BUILD_RPATH "${PANGULU_ROOT}/lib;${CUDAToolkit_LIBRARY_DIR}"
  INSTALL_RPATH "$ORIGIN/../lib")
```

将路径占位符替换为实际位置后，执行：

```
cmake -S . -B build -DPANGULU_ROOT="$PANGULU_ROOT" \
  -DCUDAToolkit_ROOT="$CUDA_ROOT"
cmake --build build
./build/pangulu_demo
```

第一条命令读取当前目录的工程文件，在 build/ 中生成构建配置；第二条命令编译，第三条命令运行。配置失败时，先检查两个安装路径和开发工具链，再重新执行第一条命令。

3 通用说明

本章说明句柄状态、阶段依赖和资源管理要求，并给出单次求解、因子复用及数值重分解的适用条件。

3.1 句柄与状态

pangulu_handle_t 保存配置、分析结果、LU 因子和工作资源，并绑定到创建时选定的 GPU。使用前通过 pangulu_create() 获取句柄，结束后通过 pangulu_destroy() 释放内部资源。

完成配置和数据描述符创建后，依次执行 ANALYSIS、NUMERIC 和 SOLVE。若不需要分别控制或计时各阶段，也可用 ALL 一次完成整个计算流程。

3.2 执行阶段

pangulu_execute() 通过 pangulu_phase_t 选择执行阶段。下表使用枚举后缀，调用时需加上 PANGULU_PHASE_ 前缀。

阶段后缀	功能与调用条件
ANALYSIS	对 A 进行重排、符号分解和预处理。此阶段不使用向量，x 与 b 均传 NULL。
NUMERIC_FACTORISATION	在分析完成后，为 A 建立数值因子。x 与 b 均传 NULL。
SOLVE	使用已建立的因子求解 GPU 上的右端项 b，并将结果写入 GPU 上的 x。
REFACTORISATION	初次数值分解完成、且复用级别为 L2 时，为稀疏结构相同、数值更新后的矩阵重建因子。x 与 b 均传 NULL。
ALL	按顺序完成分析、数值分解和求解。调用时须同时提供 A 以及 GPU 上的 x、b 描述符。

阶段执行依赖前一步建立的状态。只有 ANALYSIS 成功后才能进行 NUMERIC，只有因子建立成功后才能执行 SOLVE；任一步失败时，应停止依赖该状态的后续计算。

3.3 内存所有权与生命周期

矩阵和向量描述符记录已有数组的地址、类型与布局，不接管数据所有权。应用负责分配和释放数组，并在描述符存活期间保持其地址和存储空间有效。

计算期间，不得修改、移动或释放正在使用的输入。结束后，先销毁向量描述符、矩阵描述符和句柄，再释放应用持有的缓冲区。创建失败时只清理已经成功创建的对象；任何情况下，都应先确认数据不再被访问，再释放其存储空间。

3.4 设备数据与结果

SOLVE 中的右端项重排、缩放、求解和结果排回均在 GPU 上完成，解保存在应用提供的 GPU 缓冲区中。后续 GPU 计算可直接使用该结果；需要在 CPU 上读取或校验时，由应用自行下载。

3.5 并发使用

同一句柄仅支持串行调用。应用需协调其使用顺序，避免上一调用尚未结束便再次执行。并行处理独立线性系统时，应分别管理句柄及其数据依赖。

3.6 重排与复用配置

重排方法与复用级别相互独立，均应在首次计算前设置。虽然二者都使用 L0、L1 等取值，但前者选择主元预处理方式，后者控制已有结果的复用。

3.6.1 重排方法

通过 PANGULU_REORDERING_METHOD 选择重排方法。例如，以下调用启用 L1 重排：

```
pangulu_config_set(h, PANGULU_REORDERING_METHOD, L1);
```

值	重排方式
L0	无主元模式，默认配置。
L1	静态主元模式，在分解前确定排列与缩放。

重排会影响数值行为、因子填充和显存占用，L1 并非对所有矩阵都更快或更准确。比较 L0 与 L1 时，应保持输入、精度和运行条件一致，同时记录耗时及原方程残差。

3.6.2 复用级别

通过 PANGULU_REUSE_LEVEL 选择下表中的复用方式。系数矩阵不变、需要重复求解新的 RHS 时，设置为 L1：

```
pangulu_config_set(h, PANGULU_REUSE_LEVEL, L1);
```

值	适用场景	执行流程
L0	默认方式，完成一次分解和求解。	依次执行 ANALYSIS、NUMERIC、SOLVE。
L1	矩阵不变，连续求解新的 RHS。	执行一次 ANALYSIS 和 NUMERIC，随后重复 SOLVE。
L2	稀疏结构不变，仅矩阵数值改变。	首次执行 ANALYSIS 和 NUMERIC，随后按需重复 REFACTORISATION 和 SOLVE。

标准调用无需设置算法调优环境变量。变更配置不会自动重建既有分析结果或因子；需要改变求解方案时，应重新创建句柄。

3.7 复用条件

L1 复用要求系数矩阵保持不变；矩阵数值更新后，须按 L2 流程重建数值因子，不能直接使用旧因子求解。

L2 复用要求新旧矩阵的压缩指针和行列索引描述相同的稀疏模式，尺寸、布局、类型及存储约定保持一致。仅有相同阶数和 nnz 不足以确认可复用；新增或删除存储项、改变索引模式后，需重新创建句柄并分析。调用示例见第 6 章。

4 数据类型与存储格式

本章规定稀疏矩阵及稠密右端项、解的存储方式。创建描述符前，应用需确认数组内容合法、容量足够，且类型与尺寸符合声明。

4.1 不透明对象

公开接口使用以下三种对象类型：

```
typedef struct pangulu_handle *pangulu_handle_t;
typedef struct pangulu_matrix *pangulu_matrix_t;
typedef struct pangulu_vector *pangulu_vector_t;
```

三种对象分别表示计算句柄、稀疏矩阵和稠密 RHS/解。多 RHS 仍使用 pangulu_vector_t。应用只能通过公开函数管理对象，不得访问内部字段、按字节复制，或使用已销毁的对象。

4.2 pangulu_value_type_t

value_type 声明数值元素的精度及实数、复数类型。矩阵、右端项和解应使用相同的数值类型，并与下表的实际存储布局一致。

精度与类型	枚举值	一个元素的存储方式
单精度实数	PANGULU_R_32F	一个 float。
双精度实数	PANGULU_R_64F	一个 double。
单精度复数	PANGULU_C_32F	相邻两个 float，依次为实部和虚部。
双精度复数	PANGULU_C_64F	相邻两个 double，依次为实部和虚部。

复数问题应选择复数类型；仅提取实部会改变原始线性系统，不能代替复数求解。

4.3 pangulu_index_type_t

PANGULU_R_32I 对应 int32_t，PANGULU_R_64I 对应 int64_t。压缩指针和行列索引的整数宽度可分别指定。例如，64 位压缩指针配合 32 位行索引时，设置 ptr_type=PANGULU_R_64I 和 index_type=PANGULU_R_32I。

输入采用 64 位整数，不表示内部计算支持任意大的矩阵或索引。接近容量边界时，需在目标交付包和 GPU 上验证实际支持范围。

4.4 pangulu_memory_space_t

存储位置枚举声明数组位于主机内存还是 GPU 显存。该声明不会自动分配或迁移数据；创建描述符前，应先在对应位置准备好数组。

对象	可用存储位置	要求
矩阵 A	PANGULU_MEMORY_HOST 或 PANGULU_MEMORY_DEVICE	ptr、index、values 三个数组均位于所声明的存储空间。
右端项 B	PANGULU_MEMORY_DEVICE	形状为 $n \times$ nrhs，数据位于句柄对应的 GPU。
解 X	PANGULU_MEMORY_DEVICE	形状和数值类型与 B 一致，并使用独立的 GPU 缓冲区。

4.5 稀疏存储格式

系数矩阵须为方阵，满足 m=n>0 和 operation=PANGULU_LU。使用 CSC、CSR、BSC 或 BSR 保存完整矩阵；对称或 Hermitian 文件若只含一个三角部分，需先展开镜像项。

CSC 按列压缩，CSR 按行压缩。ptr[k] 至 ptr[k+1]-1 给出第 k 列或行的存储范围，k 从 0 开始。数组长度及索引含义如下：

布局	ptr 长度	index 含义	values 长度
CSC	$n+1$	每个存储项的行号	nnz
CSR	$m+1$	每个存储项的列号	nnz

BSC 按块列压缩，BSR 按块行压缩。block_size 为每个稠密方块的阶数 b；矩阵尺寸仍按标量行列数填写，n 应能被 b 整除，nnz 表示存储块数。压缩指针长度为 $n/b+1$，索引记录块行号或块列号，数值数组包含 nnz b^2 个元素。每块内部采用列主序，元素 (i,j) 位于块内偏移 $jb+i$。

4.5.1 同一矩阵的两种表示

下面的矩阵有 6 个存储项，可分别表示为以下 CSC 和 CSR 数组：

$$A = \begin{bmatrix} 4 & 1 & 0 \\ 2 & 5 & 3 \\ 0 & 0 & 6 \end{bmatrix}, \qquad n = 3, \quad \text{nnz} = 6.$$

```
CSC (compressed sparse column)
ptr    = [0, 2, 4, 6]
index  = [0, 1, 0, 1, 1, 2]
values = [4, 2, 1, 5, 3, 6]

CSR (compressed sparse row)
ptr    = [0, 2, 5, 6]
index  = [0, 1, 0, 1, 2, 2]
values = [4, 1, 2, 5, 3, 6]
```

以 CSC 第一列为例，ptr[0]=0、ptr[1]=2 表示读取前两个索引和值，即第 0 行的 4 和第 1 行的 2。其余片段按相同方法读取。

4.5.2 创建描述符前的检查

调用 pangulu_matrix_create() 前，请检查：

1. ptr[0] 为 0，ptr 各项单调不减，最后一项等于 nnz；数组容量满足所选压缩格式的要求。
2. CSC/CSR 的索引对应标量行列，BSC/BSR 的索引对应块行列；按所选格式检查索引范围及数组长度。
3. ptr_type 和 index_type 分别匹配实际整数数组，不能仅修改枚举而不转换数组。
4. value_type 匹配实际数值数组。请检查实数值及复数的实部、虚部，避免输入 NaN 或 Inf。
5. 建议对每个压缩片段内的索引排序并合并重复项。若原始数据为 COO 三元组，应先转换为 CSC 或 CSR。

> **说明：文件索引与接口索引**
> Matrix Market 坐标文件通常使用从 1 开始的索引，公共 API 则要求从 0 开始。随包读取器负责转换；直接集成时，该转换及压缩格式构建由应用完成。

4.6 稠密右端项与解

4.6.1 长度、列数与列间距

多 RHS 描述符使用三个尺寸参数：length 为每列长度，等于矩阵阶数 n；nrhs 为列数，至少为 1；ld 为相邻列起点的间距，满足 ld>=length，单位是数值元素。

$$\text{元素地址} = \mathsf{data}[\mathsf{row} + \mathsf{column} \times \mathsf{ld}], \quad 0 \le \mathsf{row} < n, \quad 0 \le \mathsf{column} < \mathsf{nrhs}.$$

目前仅支持列主序，各列按上述地址公式排列。复数类型的 ld 按完整复数元素计数，不按实数分量数或字节数计数。

4.6.2 两列 RHS 与列间填充示例

对本章的 3×3 矩阵，令 length=3，nrhs=2，ld=4。下面的两列右端项分别对应全 1 和全 2 的解向量：

$$B = \begin{bmatrix} 5 & 10 \\ 10 & 20 \\ 6 & 12 \end{bmatrix}, \qquad X = \begin{bmatrix} 1 & 2 \\ 1 & 2 \\ 1 & 2 \end{bmatrix}.$$

column	0				1			
offset	0	1	2	3	4	5	6	7
B data	5	10	6	pad	10	20	12	pad
X data	1	1	1	pad	2	2	2	pad

每列前 3 个元素参与求解，第 4 个位置为不参与计算的填充区 pad。按每列保留完整跨度分配时，本例需要 ld*nrhs=8 个元素；字节容量的计算见第 4.7 节。

4.6.3 输入与输出的对应关系

B 与 X 应具有相同的长度、列数和数值类型，并位于句柄对应的 GPU。两者使用独立缓冲区，ld 可不同，但均须满足各自的长度下界。

说明：选择单 RHS 或多 RHS 创建函数

连续单向量：pangulu_vector_create()。
多列或显式列间距：pangulu_multiple_vector_create()，也支持 nrhs=1。
完整声明见第 5 章，两列示例见第 6 章。

4.7 显存容量

设压缩指针、索引和数值的单元素字节数分别为 s_p、s_i、s_v，矩阵阶数为 n，存储项数为 $z = \mathrm{nnz}$。CSC/CSR 输入与两个稠密缓冲区均驻留 GPU 时，所需字节容量可按下式计算：

$$B_A = (n+1)s_p + zs_i + zs_v, \qquad B_{B,X} = (\mathrm{ld}_B + \mathrm{ld}_X)\,\mathrm{nrhs}\,s_v.$$

B_A 对应 CSC/CSR 的三个输入数组，$B_{B,X}$ 对应右端项和解，并为每列保留完整跨度。紧凑列主序取 $\mathrm{ld}_B = \mathrm{ld}_X = n$；双精度实数取 $s_v = 8$，双精度复数取 $s_v = 16$。BSC/BSR 输入的矩阵容量为 $B_A = (n/b+1)s_p + zs_i + zb^2 s_v$。主机输入与 GPU 驻留数据分别统计。

注意：分解阶段的额外显存

完整峰值显存还需计入分析临时数据、LU 因子及工作区。显存规划见第 7 章。

多 RHS 的 B/X 可以在 ANALYSIS 和 NUMERIC 完成后再分配，使其不与分析阶段临时工作区同时占用显存。SOLVE 前准备好输入数据及两个描述符即可。

分配前，检查元素计数及字节数乘积是否超出整数范围。索引转换为较窄类型时，还需确认每个值都可表示，并同步更新描述符中的类型声明。

5 函数参考

本章列出公开头文件 include/pangulu.h 中的核心函数。参数方向“输入”表示由应用提供，“输出”表示接收创建的对象或计算结果。描述符与数据缓冲区的生命周期约定见第 3 章。

函数	用途
pangulu_create()	创建计算句柄并初始化授权与设备上下文。
pangulu_config_set()	为句柄设置重排方法或复用级别。
pangulu_matrix_create()	为稀疏系数矩阵创建描述符。
pangulu_vector_create()	创建单个 RHS 或解向量的描述符，保留 26.42 的调用约定。
pangulu_multiple_vector_create()	创建列主序多 RHS 或解的稠密描述符。
pangulu_execute()	按指定阶段执行计算。
pangulu_matrix_destroy()	销毁矩阵描述符，保留应用持有的数组。
pangulu_vector_destroy()	销毁稠密描述符，保留应用持有的缓冲区。
pangulu_destroy()	销毁句柄并释放库内部的工作资源。

所有函数返回 pangulu_status_t，成功值为 PANGULU_STATUS_SUCCESS。每次调用后均应检查状态；失败时停止依赖该操作的计算，并按第 7 章排查，不得将输出缓冲区中的旧值视为本次结果。

5.1 pangulu_create()

创建计算句柄。调用前，需按第 2 章配置运行路径和 License。

```
pangulu_status_t pangulu_create(
    int32_t gpu_index,
    pangulu_handle_t *handle);
```

参数

名称	方向	说明
gpu_index	输入	已授权 GPU 有序列表中的下标，从 0 开始；该值不是 CUDA 设备编号。
handle	输出	指向句柄变量的地址，成功时接收新句柄。建议先将变量初始化为 NULL，再传入其地址。

行为与返回

函数完成授权、设备和构建完整性检查后，返回绑定到所选 GPU 的句柄。后续配置和计算使用该句柄，设备数据也应分配在对应 GPU 上。

5.2 pangulu_config_set()

为句柄设置重排方法或复用级别，应在首次执行依赖该配置的计算前调用。

```
pangulu_status_t pangulu_config_set(
    pangulu_handle_t handle,
    pangulu_config_option_t option,
    pangulu_config_value_t value);
```

参数

名称	方向	说明
handle	输入	由 pangulu_create() 成功创建且尚未销毁的句柄。
option	输入	配置项，选择 PANGULU_REORDERING_METHOD 或 PANGULU_REUSE_LEVEL。
value	输入	该配置项支持的取值。重排支持 L0、L1；复用支持 L0、L1、L2。含义见第 3 章。

行为与返回

本次调用只设置 option 指定的配置项，不会自动重建既有分析结果或因子。各取值的含义及配置变更要求见第 3 章。

5.3 pangulu_matrix_create()

为第 4 章规定的稀疏数组创建矩阵描述符。

```
pangulu_status_t pangulu_matrix_create(
    pangulu_matrix_t *matrix,
    pangulu_matrix_layout_t layout,
    int64_t m, int64_t n, int64_t nnz,
    int64_t block_size,
    void *ptr, void *index, void *values,
    pangulu_index_type_t ptr_type,
    pangulu_index_type_t index_type,
    pangulu_value_type_t value_type,
    pangulu_operation_t operation,
    pangulu_memory_space_t memory_space);
```

参数

名称	方向	说明
matrix	输出	指向矩阵描述符变量的地址，成功时接收新对象。
layout	输入	稀疏格式，选择 PANGULU_MATRIX_LAYOUT_CSC、PANGULU_MATRIX_LAYOUT_CSR、PANGULU_MATRIX_LAYOUT_BSC 或 PANGULU_MATRIX_LAYOUT_BSR。
m,n,nnz	输入	分别为标量行数、标量列数和存储项数；BSC/BSR 的存储项数按块计。要求 $m = n > 0$，nnz 与数组内容一致。
block_size	输入	块结构矩阵输入的块阶数。
ptr,index,values	输入	压缩指针、标量或块索引、数值数组的地址，索引从 0 开始；布局见第 4 章。
ptr_type,index_type	输入	分别声明压缩指针数组和索引数组的实际整数宽度。
value_type	输入	声明 values 数组的精度及实数、复数类型。
operation	输入	设置为 PANGULU_LU。
memory_space	输入	选择 PANGULU_MEMORY_HOST 或 PANGULU_MEMORY_DEVICE；三个输入数组均须位于该存储空间。

行为与返回

本函数只创建描述符，不执行分析或分解。调用前应完成数组合法性检查。主机矩阵作为输入，分解与求解使用 GPU 显存。

5.4 pangulu_vector_create()

为 GPU 上的单个右端项或解向量创建描述符。该函数保留 26.42 的五参数声明与二进制调用约定；旧程序可以直接替换为匹配目标环境的 26.44 动态库，无需重新编译。

```
pangulu_status_t pangulu_vector_create(
    pangulu_vector_t *vector,
    int64_t size,
    void *data,
    pangulu_value_type_t valueType,
    pangulu_memory_space_t memorySpace);
```

参数

名称	方向	说明
vector	输出	指向向量描述符变量的地址，成功时接收新对象。
size	输入	向量元素个数，应等于系数矩阵阶数，且大于 0。
data	输入	连续 GPU 缓冲区的地址，用于保存右端项或接收解。
valueType	输入	与系数矩阵一致的数值类型。
memorySpace	输入	当前仅支持 PANGULU_MEMORY_DEVICE。

行为与返回

描述符采用紧凑单列布局，等价于多 RHS 接口的 length=size,nrhs=1,ld=size。本函数不分配或复制 GPU 数据；输入与输出应使用独立缓冲区。

5.5 pangulu_multiple_vector_create()

创建 GPU 列主序稠密 RHS 或解描述符。该函数在 26.44 新增，支持单列及多列数据；计算仍使用 pangulu_execute()，销毁仍使用 pangulu_vector_destroy()。

```
pangulu_status_t pangulu_multiple_vector_create(
    pangulu_vector_t *vector,
    int64_t length,
    int64_t nrhs,
    int64_t ld,
    void *data,
    pangulu_value_type_t valueType,
    pangulu_memory_space_t memorySpace);
```

参数

名称	方向	说明
vector	输出	指向稠密描述符变量的地址，成功时接收新对象。
length	输入	每个 RHS 的元素个数，即矩阵阶数，必须大于 0。
nrhs	输入	右端项的列数，至少为 1。解描述符也使用相同列数。
ld	输入	相邻两列起点之间的间距，以数值元素计数，要求 ld>=length。
data	输入	GPU 缓冲区地址。用于 B 时保存输入数据，用于 X 时供求解写入结果。
valueType	输入	与系数矩阵相同的数值类型。
memorySpace	输入	当前仅支持 PANGULU_MEMORY_DEVICE。

行为与返回

元素按 data[row+column*ld] 寻址，填充区不参与计算；复数的 ld 按完整复数元素计数。B/X 的长度、列数和类型必须相同，列间距可以不同，缓冲区必须独立。本函数不分配或上传数据，布局和容量示例见第 4 章。

> **说明：旧程序升级与新功能接入**
> 使用新增函数时，需采用新头文件编译，并链接 26.44 或支持该符号的后续运行库。不要向旧函数名传入七个参数，也不要将引用新增符号的程序与 26.42 库混用。完整示例见第 6 章。

5.6 pangulu_execute()

对指定矩阵执行一个计算阶段，前置状态和阶段顺序见第 3 章。

```
pangulu_status_t pangulu_execute(
    pangulu_handle_t handle,
    pangulu_phase_t phase,
    pangulu_matrix_t A,
    pangulu_vector_t x,
    pangulu_vector_t b);
```

参数

名称	方向	说明
handle	输入/输出	有效句柄，用于读取既有状态并保存本阶段产生的状态。
phase	输入	本次执行的阶段枚举，如 PANGULU_PHASE_ANALYSIS。
A	输入	系数矩阵的有效描述符。
x	输出数据	解缓冲区的描述符。分析、数值分解及重分解阶段传 NULL。
b	输入数据	右端项缓冲区的描述符。上述三个阶段传 NULL。

行为与返回

末尾参数顺序为 A,x,b，即解在右端项之前。SOLVE 和 ALL 需要尺寸、类型及设备匹配的 B/X；SOLVE 使用既有因子处理全部 RHS 列。连续求解或重分解的配置见第 6 章。

求解成功后，结果保存在 x 的 GPU 缓冲区中。原方程残差的计算与质量判定见第 7 章。

5.7 pangulu_matrix_destroy()

销毁稀疏矩阵描述符。调用前应结束所有使用该对象的计算。

```
pangulu_status_t pangulu_matrix_destroy(
    pangulu_matrix_t matrix);
```

参数与行为

matrix 为待销毁的描述符。函数不释放 ptr、index 和 values 数组，它们仍由应用管理。销毁后不得继续使用或再次销毁该对象；可将保存它的变量设为 NULL，避免误用。

5.8 pangulu_vector_destroy()

销毁右端项或解描述符，适用于单 RHS 和多 RHS 创建函数生成的对象。

```
pangulu_status_t pangulu_vector_destroy(
    pangulu_vector_t vector);
```

参数与行为

vector 为待销毁的稠密描述符。结束对该对象的使用后调用，函数不会释放对应的 GPU 缓冲区。销毁后，该描述符失效。

5.9 pangulu_destroy()

销毁句柄并释放其持有的 LU 因子、工作区及其他内部资源。

```
pangulu_status_t pangulu_destroy(
    pangulu_handle_t handle);
```

参数与行为

handle 为待销毁的句柄。应在相关计算结束后调用；即使 License 到期，仍可清理资源。应用提供的矩阵和向量缓冲区不由该函数释放。销毁后不得继续执行或再次销毁同一句柄。

6 使用示例

本章提供单 RHS 的完整程序，以及多 RHS、因子复用和数值重分解的调用片段。运行环境按第 2 章配置。除完整程序外，片段省略了错误处理和资源清理，实际集成时仍需逐次检查返回值。

6.1 完整 C 示例

程序求解第 4 章的 3×3 系统，右端项为 $b = [5, 10, 6]^{\mathsf{T}}$，参考解为 $x = [1, 1, 1]^{\mathsf{T}}$。矩阵使用主机数组，右端项和解使用 GPU 缓冲区。求解完成后，将解下载到主机并计算原方程相对残差。

源码位于手册源码包的 `examples/quick_start.c`，编译方法见第 2.6 节。PTRY 和 CTRY 分别检查 PanguLU 与 CUDA 调用，失败时跳转到 `cleanup` 清理已创建的资源。

```c
#include <stdio.h>
#include <math.h>
#include <stdint.h>
#include <cuda_runtime_api.h>
#include "pangulu.h"

#define PTRY(call) do { \
    pangulu_status_t s_ = (call); \
    if (s_ != PANGULU_STATUS_SUCCESS) { \
        fprintf(stderr, "%s: status=%d\n", #call, (int)s_); \
        goto cleanup; \
    } \
} while (0)
#define CTRY(call) do { \
    cudaError_t e_ = (call); \
    if (e_ != cudaSuccess) { \
        fprintf(stderr, "%s: %s\n", #call, cudaGetErrorString(e_)); \
        goto cleanup; \
    } \
} while (0)

int main(void)
{
    int result = 1;
    int64_t ptr[4] = {0, 2, 4, 6};
    int32_t idx[6] = {0, 1, 0, 1, 1, 2};
    double val[6] = {4, 2, 1, 5, 3, 6};
    double b[3] = {5, 10, 6}, x[3] = {0};
    double *d_b = NULL, *d_x = NULL;
    pangulu_handle_t h = NULL;
    pangulu_matrix_t A = NULL;
    pangulu_vector_t B = NULL, X = NULL;

    PTRY(pangulu_create(0, &h));
    CTRY(cudaMalloc((void **)&d_b, sizeof(b)));
    CTRY(cudaMalloc((void **)&d_x, sizeof(x)));
    CTRY(cudaMemcpy(d_b, b, sizeof(b), cudaMemcpyHostToDevice));
    PTRY(pangulu_matrix_create(
        &A, PANGULU_MATRIX_LAYOUT_CSC, 3, 3, 6, 1,
        ptr, idx, val, PANGULU_R_64I, PANGULU_R_32I,
        PANGULU_R_64F, PANGULU_LU, PANGULU_MEMORY_HOST));
    PTRY(pangulu_vector_create(&B, 3, d_b, PANGULU_R_64F,
                               PANGULU_MEMORY_DEVICE));
    PTRY(pangulu_vector_create(&X, 3, d_x, PANGULU_R_64F,
                               PANGULU_MEMORY_DEVICE));

    PTRY(pangulu_execute(h, PANGULU_PHASE_ANALYSIS, A, NULL, NULL));
```

```
    PTRY(pangulu_execute(h, PANGULU_PHASE_NUMERIC_FACTORISATION,
                         A, NULL, NULL));
    PTRY(pangulu_execute(h, PANGULU_PHASE_SOLVE, A, X, B));
    CTRY(cudaMemcpy(x, d_x, sizeof(x), cudaMemcpyDeviceToHost));
    {
        double r0 = 4*x[0] + x[1] - b[0];
        double r1 = 2*x[0] + 5*x[1] + 3*x[2] - b[1];
        double r2 = 6*x[2] - b[2];
        double nr = hypot(hypot(r0, r1), r2);
        double nb = hypot(hypot(b[0], b[1]), b[2]);
        double rel = nr / nb;
        printf("x = [%.9g, %.9g, %.9g]\n", x[0], x[1], x[2]);
        if (!isfinite(x[0]) || !isfinite(x[1]) || !isfinite(x[2]) ||
            !isfinite(nr) || !isfinite(nb) || !isfinite(rel))
            printf("residual: non-finite (NaN or Inf)\n");
        else
            printf("relative residual = %.6e\n", rel);
    }
    result = 0;  /* API completion; quality is reported separately. */
cleanup:
    if (X) pangulu_vector_destroy(X);
    if (B) pangulu_vector_destroy(B);
    if (A) pangulu_matrix_destroy(A);
    if (h) pangulu_destroy(h);
    cudaFree(d_x);
    cudaFree(d_b);
    return result;
}
```

6.2 一次提交多个 RHS

先分配独立的 GPU 缓冲区 d_b 和 d_x，按列主序准备所有 RHS，再创建两个描述符。下面的调用只分析和分解一次，随后一次 SOLVE 处理全部列，共享同一组 LU 因子。

```
/* Device buffers: column-major, nrhs columns, ld >= n. */
pangulu_multiple_vector_create(&B, n, nrhs, ld_b, d_b,
                      PANGULU_R_64F, PANGULU_MEMORY_DEVICE);
pangulu_multiple_vector_create(&X, n, nrhs, ld_x, d_x,
                      PANGULU_R_64F, PANGULU_MEMORY_DEVICE);
pangulu_execute(h, PANGULU_PHASE_ANALYSIS, A, NULL, NULL);
pangulu_execute(h, PANGULU_PHASE_NUMERIC_FACTORISATION,
                A, NULL, NULL);
pangulu_execute(h, PANGULU_PHASE_SOLVE, A, X, B);
```

布局、列间距和显存容量的要求见第 4 章。

6.3 同一矩阵，连续提交多组 RHS

若分解后还要连续提交新的 RHS，在首次计算前启用 L1 复用。完成 ANALYSIS 和 NUMERIC 后，更新 GPU 输入缓冲区并执行 SOLVE；每次可提交一列或一组 RHS。

```
/* Before initial analysis and factorisation: */
pangulu_config_set(h, PANGULU_REUSE_LEVEL, L1);
/* After factorisation, for each new RHS group: */
pangulu_execute(h, PANGULU_PHASE_SOLVE, A, X, B);
```

若沿用相同地址和布局，可直接更新缓冲区内容；地址或布局改变时，需创建对应的新描述符。写入下一组输入前，应确认上一组计算已结束，并始终保持系数矩阵不变。

6.4 相同稀疏结构，矩阵数值改变

若稀疏结构不变、仅系数值更新，在首次计算前启用 L2，并完成 ANALYSIS 和 NUMERIC。后续每轮用 REFACTORISATION 更新因子，再执行 SOLVE。

```
/* After initial analysis and numeric factorisation, with reuse L2: */
/* Supply A_next with the identical sparse pattern and new values. */
pangulu_execute(h, PANGULU_PHASE_REFACTORISATION,
                A_next, NULL, NULL);
pangulu_execute(h, PANGULU_PHASE_SOLVE, A_next, X, B);
```

`A_next` 描述更新后的矩阵。使用新数组时，需创建相应的矩阵描述符。复用前按第 3 章核对稀疏结构；若存储模式改变，应重新分析。

6.5 运行完整的多 RHS 示例

手册源码包中的 `examples/multi_rhs.c` 使用两列 RHS 和 `ld=4`。按第 2.6 节的直接编译命令，将源文件改为 `multi_rhs.c`，输出程序改为 `pangulu_multi_rhs`，然后执行 `./pangulu_multi_rhs`。两列参考解分别为 $[1,1,1]^{\mathsf{T}}$ 和 $[2,2,2]^{\mathsf{T}}$，运行后应逐列检查残差。

随手册提供的 CMake 工程默认构建单 RHS 示例。若使用安装包自带的多 RHS 示例 `example_08_multi_rhs.c`，则按第 2.5 节在安装目录的示例目录中用 Makefile 编译。

7 错误处理与结果验证

本章分别说明执行错误和数值结果的检查方法：先根据 API 状态确认操作是否完成，再用原始方程的残差判断结果质量。二者应分别记录。

7.1 错误状态

下表列出公开状态码和处理方向。枚举后缀需加上 PANGULU_STATUS_ 前缀，例如 PANGULU_STATUS_ALLOC_FAILED。

值	枚举后缀	含义与处理
0	SUCCESS	本次 API 操作成功。若为求解调用，仍需独立检查解的质量。
1	NOT_INITIALIZED	句柄或前置计算状态未建立。检查对象是否创建成功、阶段是否按顺序执行。
2	ALLOC_FAILED	已识别的主机内存或 GPU 显存分配失败。检查资源占用并减少同时运行的任务。
3	EXECUTION_FAILED	参数或执行过程发生错误。结合失败阶段、输入配置和原始输出排查。
4	LICENSE_ERROR	授权、设备、期限或构建配套检查未通过。检查 License 和所用交付文件。

已弃用状态码。 PANGULU_STATUS_NOT_SUPPORTED（值 5）仅为源代码和二进制兼容保留在头文件中，并标记为 deprecated。当前版本将相关配置或功能错误统一归入 PANGULU_STATUS_EXECUTION_FAILED（值 3），不再返回值 5。

> **注意：分析阶段的已知报错例外**
> 重排模块可能将部分 GPU 分配失败归并为内部错误 -5061，使 ANALYSIS 返回 3 而非 2。因此，分析阶段的状态码 3 不能排除显存不足；应结合原始诊断和显存占用排查。这是 26.44 版分析阶段的一种已知报错情形。

7.1.1 按失败阶段排查

定位时，从第一次失败的调用或阶段开始，保留其返回码和原始输出，再按下表检查。

失败位置	优先检查
创建句柄	License 路径及读取权限；授权对应的产品、设备、版本和期限；可见 GPU；运行库是否来自同一交付包。
创建描述符	方阵条件、稀疏布局、数组容量、数据与索引类型，以及声明的存储位置。
ANALYSIS	稀疏结构是否合法、所选重排配置、当前可用显存和原始诊断。
NUMERIC / REFACTOR	前置阶段是否成功、结构复用条件是否满足，以及显存占用和数值异常。
SOLVE	因子是否建立成功；B/X 是否位于句柄对应的 GPU；长度、列数和类型是否匹配；授权是否到期。

7.1.2 失败后的恢复

失败后，停止后续依赖计算并清理已创建的资源。修正输入、授权或资源条件后，重新创建句柄执行流程；X 中保留的旧值不得作为本次结果。

7.2 常见集成问题

若程序尚未进入计算便报错，先按下表检查编译、链接及动态库加载环境。

现象	检查与处理
找不到头文件，或链接时找不到 `-lpangulu`	检查 `-I` 是否指向安装目录的 `include/`，`-L` 是否指向同一安装目录的 `lib/`，并确认相应文件存在。
程序启动时提示某个 `.so` 缺失	检查 LD_LIBRARY_PATH 或可执行文件中的运行时搜索路径，并确认配套库完整。不要从其他交付包随意复制同名库补齐。
提示系统 ABI 或符号版本不满足	按第 2 章检查 Linux、运行库和驱动要求，选择匹配的交付包；不要通过改写 ELF 版本标签绕过依赖检查。

可在自行构建且可信的程序所在目录运行 `ldd./pangulu_demo`，查看动态库解析路径。若出现 `notfound`，补齐运行依赖或搜索路径。

从 26.42 升级到 26.44，可先直接运行原有最小示例验证换库；使用新增接口时，再采用新头文件编译。

7.3 数值精度与残差

用原始系数矩阵和右端项验证返回的解。对单个 RHS，残差向量和相对残差定义为：

$$r = A_{\text{original}}x - b_{\text{original}}, \qquad \eta = \frac{\|r\|_2}{\|b_{\text{original}}\|_2}.$$

当 $\|b_{\text{original}}\|_2 > 0$ 且参与计算的数值均有限时，使用上述比值。若右端项范数为 0，则报告绝对残差 $\|r\|_2$，并注明相对残差未定义。

7.3.1 避免将异常结果显示为 0

校验过程也可能溢出或产生非有限值。为避免将异常误报为零残差，应执行以下检查：

1. 检查输入、解向量、残差各分量、范数及最终比值是否有限。复数应同时检查实部和虚部。
2. 使用稳定的范数计算方法，避免直接平方求和产生上溢或下溢，进而得到错误比值。
3. 遇到 NaN 或 Inf 时，明确输出 `non-finite` 或具体异常说明，不要用 0 代替。
4. 使用科学计数法或足够的有效位显示小残差，避免仅因格式化保留位数过少而显示为“0”。

大残差或非有限残差应如实报告，不能仅因此将已完成的求解改报为 API 错误。应用可设置独立的质量阈值，将验收结论与 API 状态分别保存。

7.3.2 零对角与数值困难矩阵

LU 分解按分解前确定的主元顺序执行。对零对角或缺失对角等情况，库可能在内部 GPU 副本上执行固定 10^{-12} 的对角修复，必要时使用已有的小规模 GPU 部分选主元 LU 恢复路径。修复不改变原始输入，修复量也不自适应增大。

该处理可能影响因子填充、耗时、显存和数值结果。验证时仍应使用原始方程，不能以修复后方程的残差替代。

> **注意：精度限制**
> 匹配、缩放和对角修复不构成任意输入下的精度保证。病态或数值困难矩阵可能产生很大残差，应按应用误差要求判断是否接受。

7.3.3 范数型后向误差

当矩阵、解和右端项的尺度差异较大时，可补充报告范数型后向误差：

$$\beta_\infty = \frac{\|A_{\text{original}}x - b_{\text{original}}\|_\infty}{\|A_{\text{original}}\|_\infty\|x\|_\infty + \|b_{\text{original}}\|_\infty}.$$

矩阵无穷范数为最大绝对行和，复数元素使用复数模。β_∞ 与 η 定义不同，报告时应注明公式，并分别设置判断阈值。

计算该指标时，沿用前述有限性及稳定范数检查；分母为零时，另记分子和退化情形。残差与范数的累加精度至少与求解相同，必要时使用更高精度或缩放算法。

> **说明：残差与解误差的区别**
> 残差衡量解代入方程后的偏差。对于病态矩阵，小残差仍可能对应较大的解误差；有参考解时，可另外计算相对解误差。

7.3.4 逐个 RHS 检查

每列结果应记录 RHS 标识、API 状态、原方程残差及有限性。汇总可列出最大残差和异常列数，但仍需保留逐列结果，避免平均值掩盖个别异常。

一次 SOLVE 提交多列时，记录整组耗时和 RHS 数量；逐列调用时，分别记录各次耗时，不将整组时间直接解释为单列时间。

7.4 性能测量

一次完整分解与求解的三阶段耗时定义为：

$$T_{\text{solver}} = T_{\text{analysis}} + T_{\text{numeric}} + T_{\text{solve}}.$$

该和式不包含文件读取、格式转换、句柄创建、应用侧数据传输或残差计算。报告业务总耗时时，应另行说明是否包含这些操作。

为便于复现，记录交付包标识、主机与 GPU、驱动、精度、布局、重排和复用配置，并分别报告首次运行与重复运行的耗时和波动。

7.4.1 GPU 计时边界

使用主机计时器时，测量终点应覆盖相关 GPU 工作完成，避免只测到命令提交时间。同步方法应与应用的执行方式相符，并在对比测试中保持一致。

7.4.2 显存规划

分解显存由矩阵结构、重排、因子填充和工作区共同决定，不能仅由原始 nnz 或文件大小准确推算。PanguLU 仅支持显存内计算；应在目标 GPU 上测量峰值，为运行时资源预留空间，并控制并发任务的显存占用。

7.4.3 保持可比条件

对比性能时，建议遵循以下约定：

- 区分首次完整流程与复用因子后的 SOLVE，分别报告其耗时。
- 使用相同的输入矩阵和精度，并明确匹配设置及残差定义。
- 保留完整原始输出，将超时、执行失败和异常残差分别作为结果状态记录。
- 保持同步、计时方式和运行环境一致。

7.5 问题报告

提交问题时，提供下载包文件名、操作系统、GPU、驱动、最小调用顺序和配置，并附首次失败阶段、返回码及完整原始输出。

矩阵相关问题还需提供尺寸、存储项数、布局、数值及索引类型、显存占用和原方程残差。允许共享时，附最小复现输入；公开记录中不要包含 License 正文或敏感凭据。

附录 A　示例输入文件

本附录规定安装包示例读取器接受的文件格式。应用可采用其他格式，但调用公共 API 前，需先转换为第 4 章规定的数组。

A.1　矩阵：Matrix Market 坐标格式

下例对应第 4 章的矩阵。第一行声明坐标格式、实数类型和一般矩阵，第二行给出行数、列数及文件中的存储项数。后续每行依次为从 1 开始的行号、列号和数值。

```
%%MatrixMarket matrix coordinate real general
3 3 6
1 1 4
2 1 2
1 2 1
2 2 5
2 3 3
3 3 6
```

复数文件头使用 `complex`，每个存储项按“行号、列号、实部、虚部”排列。实数读取器还支持 `integer` 和 `pattern`；后者只给出位置，读取器为每项赋值 1。

示例读取器会展开 `symmetric`、`hermitian` 或 `skew-symmetric` 的三角存储，并将索引转为从 0 开始。Hermitian 镜像项取复共轭，skew-symmetric 镜像项取负值。文件内容应与声明一致，不要同时提供三角项及其重复镜像。

A.2　右端项：不带尺寸头的纯文本

实数 RHS 文件包含 n 个由空白分隔的数值，复数文件包含 n 对实部和虚部。不要添加尺寸行或 Matrix Market 头，以免被误读为向量数据。以下两种写法表示同一个右端项：

实数 RHS

```
5
10
6
```

复数 RHS：实部与虚部

```
5 0
10 0
6 0
```

> **说明：文件、数据类型与示例应一致**
> 按实际数值类型选择实数或复数示例，不以文件后缀判断。读取后仍需检查数组尺寸、索引和数值。`example_08_multi_rhs` 接受两个独立的 RHS 文件，参数顺序见程序用法说明。

附录 B　文档与支持

本附录汇总开发、部署和排查问题时使用的文档及支持入口。查阅时，请选择与当前运行库配套的交付文件。

B.1　交付文件

按需要查阅以下文件：

文件或目录	用途
`include/pangulu.h`	提供本交付包的公开 C 接口声明，供应用编译使用。
`examples/`	提供可编译的调用示例及样例输入，便于验证环境和接口用法。
`licenses/`	保存第三方许可及相关声明。
`SHA256SUMS`	提供随包文件的校验值，用于检查文件内容是否与交付记录一致。

B.2　技术支持

需要协助时，可通过末页企业微信入口联系技术支持，并准备第 7 章列出的复现信息。

授权获取网页为 https://store.solved-tech.com/，二维码见末页。更新授权后的操作见第 2 章。

附录 C　版本日志

手册与软件使用相同的版本号。以下按从新到旧的顺序列出产品的主要更新。

PanguLU 26.44

- 新增 `pangulu_multiple_vector_create()` 函数接口，支持列主序稠密多 RHS，一组右端项可通过一次 SOLVE 提交。
- 保留 `pangulu_vector_create()` 函数接口的五参数声明及既有二进制调用约定，26.42 程序可直接替换匹配环境的 26.44 动态库，无需重新编译。
- 清理未使用的 GPU 内核，缩减库文件体积。
- 新增 CUDA 12.1、glibc 2.17 交付包。
- 增加 BSC/BSR 块结构矩阵输入支持。
- 将已弃用的状态码 5 统一归入运行失败（状态码 3）。

PanguLU 26.42

- PanguLU 的首个发布版本。

授权获取 企业微信

store.solved-tech.com