

PanguLU 26.48

用户手册

C API 使用与参考

版本 26.48
NVIDIA GPU、海光 DTK、AMD ROCm
2026 年 11 月 21 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 选择平台与交付包 . . . 2
2.2 解压与保留完整目录 . . . 2
2.3 运行库定位 . . . 2
2.4 安装 License . . . 3
2.5 编译与运行公共示例 . . . 3
2.6 应用链接 . . . 4

3 通用说明 **5**
3.1 句柄与状态 . . . 5
3.2 执行阶段 . . . 5
3.3 内存所有权与生命周期 . . . 5
3.4 设备数据与结果 . . . 5
3.5 并发使用 . . . 6
3.6 匹配、重排与复用 . . . 6
3.7 复用条件 . . . 7

4 数据类型与存储格式 **8**
4.1 不透明对象 . . . 8
4.2 pangulu_value_type_t . . . 8
4.3 pangulu_index_type_t . . . 8
4.4 pangulu_memory_space_t . . . 9
4.5 稀疏存储格式 . . . 9
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() . . . 14
5.2 pangulu_config_set() . . . 15
5.3 重排选择与行列排列 . . . 15
5.4 设备内存接口 . . . 15
5.5 pangulu_matrix_create() . . . 16
5.6 pangulu_matrix_create_sym() . . . 17
5.7 pangulu_vector_create() . . . 18
5.8 pangulu_multiple_vector_create() . . . 19
5.9 pangulu_execute() . . . 20
5.10 pangulu_matrix_destroy() . . . 20
5.11 pangulu_vector_destroy() . . . 21
5.12 pangulu_destroy() . . . 21
5.13 内部错误编号 . . . 21
5.14 授权检查与兼容入口 . . . 21

6 使用示例 **22**
6.1 完整 C 示例 22
6.2 一次提交多个 RHS 23
6.3 同一矩阵，连续提交多组 RHS 24
6.4 相同稀疏结构，矩阵数值改变 24
6.5 运行完整的多 RHS 示例 24
6.6 为通用矩阵设置自定义重排 24

7 错误处理与结果验证 **25**
7.1 错误状态 25
7.2 常见集成问题 26
7.3 数值精度与残差 26
7.4 性能测量 27
7.5 问题报告 28

附录 A 示例输入文件 **29**
A.1 矩阵：Matrix Market 坐标格式 29
A.2 右端项：不带尺寸头的纯文本 29

附录 B 文档与支持 **30**
B.1 交付文件 30
B.2 技术支持 30

附录 C 版本日志 **31**
PanguLU 26.48 31
PanguLU 26.46 31
PanguLU 26.44 31
PanguLU 26.42 31

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，共享同一组数值因子。

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 26.48 标准版，覆盖 NVIDIA GPU、海光 DTK 和 AMD ROCm 三种交付平台。各版提供相同的 C API（ABI 版本 2）。

项目	支持范围
分解方法	支持稀疏 LU、Cholesky 和 LDLT 分解，按创建矩阵时声明的性质选择；对称及 Hermitian 矩阵支持完整或单三角输入。
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	使用主机内存或 GPU 显存中的列主序稠密存储，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，数值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`。

原有 14 参数 `pangulu_matrix_create()`、5 参数 `pangulu_vector_create()` 等既有调用继续保留。声明矩阵性质及存储三角时，使用 `pangulu_matrix_create_sym()`。第 3.6 节介绍本版的重排选择接口，第 5.4 节介绍设备内存接口。

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

选择与目标设备、驱动、运行库和操作系统匹配的交付包。NVIDIA、海光 DTK 和 AMD ROCm 版使用相同的公开 pangulu.h 和 C ABI。应用示例采用普通 C 编译器，各平台的运行依赖随所选交付包而定。

2.1 选择平台与交付包

下表列出 26.48 的 x86_64 Linux 交付包标称最低 glibc。存档包的 README.md 说明支持的目标架构、驱动与运行库版本；部署时以所选包的说明为准。

平台	运行环境	最低 glibc	设备检查
NVIDIA GPU	CUDA 12.1	2.17	使用 nvidia-smi 检查设备和驱动，核对包支持的计算能力。
NVIDIA GPU	CUDA 12.4	2.18	同上。
NVIDIA GPU	CUDA 12.9 / 13.1	2.27	同上。各包的支持架构可能不同。
海光 DCU	DTK 25.04.4 / 26.04	2.29	使用对应 DTK 驱动及运行库，核对包支持的 DCU 型号。
AMD GPU	ROCm 7.2.1	2.38	使用 rocminfo 检查设备架构。本包目标为 gfx900、gfx942 和 gfx1201。

使用 ldd--version 查询系统 glibc。编译公共示例需要支持 C11 的编译器及 GNU Make。应用也可使用 CMake 3.18 或更新版本。

2.2 解压与保留完整目录

将以下占位路径替换为压缩包的实际路径。解压后，将 PANGULU_ROOT 指向含有 include/ 和 lib/ 的目录。

```
mkdir -p "$HOME/pangulu-packages"
tar -xzf /absolute/path/to/download.tar.gz -C "$HOME/pangulu-packages"
export PANGULU_ROOT=/absolute/path/to/extracted/package
```

include/pangulu.h 为公开头文件，lib/ 保存主库与配套库。保留整包目录结构及所有运行文件。如果交付包提供 SHA256SUMS，可按其说明校验文件。

2.3 运行库定位

动态加载器通过系统库配置或 LD_LIBRARY_PATH 查找产品库及其依赖。部署时将同一平台的交付包库目录与配套运行库目录纳入搜索路径。避免把不同平台或不同版本的配套库混在同一目录。LD_LIBRARY_PATH 是 Linux 加载器的设置。

只调用公共 C API 的程序可用同一份源码构建。在 CPU 架构、系统 ABI、符号和所有运行依赖都兼容时，也可直接用目标平台的完整 26.48 交付包运行既有可执行文件。应用若直接调用了平台原生运行时接口，还需要满足这些调用的链接和执行要求。

2.4 安装 License

将适合产品、版本和设备的原始 License 放在默认位置。例如，将文件安装到用户目录。

```
mkdir -p "$HOME/.solved-tech/pangulu"
cp /absolute/path/to/license.key "$HOME/.solved-tech/pangulu/"
```

默认位置	用法
~/.solved-tech/pangulu/*.key	用户级 License 目录。
/etc/solved-tech/pangulu/*.key	系统级 License 目录。
~/.pangulu/license.key、/etc/pangulu/license.key	保留的单文件位置。
~/.pangulu/licenses/*.key	保留的多文件目录。
~/.pangulu/licenses.list	保留的清单，一行一个文件路径。

请保持签名文件完整并确保运行账户可读取。授权检查在本地完成。若将授权放在其他位置，可设置 PANGULU_LICENSE_PATHS。多个文件路径以英文逗号分隔。

```
export PANGULU_LICENSE_PATHS="/absolute/path/gpu-a.key,/absolute/path/gpu-b.key"
```

变量设置为非空时以指定清单为准；变量未设置或为空时使用默认位置。该变量指定授权文件路径。更新授权或签名撤销列表（CRL）后，结束当前计算并以新句柄加载。计算与结果提交仍受授权检查和有效期约束。

2.5 编译与运行公共示例

公共示例仓库提供 17 个独立示例、共用输入和构建规则。交付包里的示例较早或缺失时，请取用公共仓库。以下命令从示例库根目录执行。

```
make -C 01_double LIBRARY_ROOT="$PANGULU_ROOT"
make -C 01_double run LIBRARY_ROOT="$PANGULU_ROOT" \
  TOOLKIT_LIB_DIR=/absolute/path/to/platform/runtime/lib \
  PANGULU_LICENSE_PATHS=/absolute/path/to/license.key
```

TOOLKIT_LIB_DIR 是运行示例时的库搜索目录，可用冒号列出多个目录。若运行库已由系统加载器配置，也可直接执行编译好的程序。公共仓库的 makerun 默认指向包根目录的 license.key。要直接验证库的默认授权搜索路径，可运行已编译程序，并在其进程环境中取消 PANGULU_LICENSE_PATHS。更换交付包或编译器后先执行 makeclean。

示例	内容
01_double 至 04_float_complex	单双精度实数与复数的基本调用。
05_csr、06_cpu_matrix_input	按行压缩输入与主机矩阵输入。
07_l0_reordering、08_multi_rhs、09_refactor	匹配策略、多列 RHS 与数值重分解。
10_reordering_nd 至 17_reordering_none	每种重排选择各有一个小矩阵。各目录的 README.md 解释矩阵特征。

样例通过各阶段后会输出 `verification:PASS`。自行验证时应确认使用的范数，按第 7 章记录原方程的相对二范数残差。

2.6 应用链接

公共 C API 对应 `include/pangulu.h` 和 `libpangulu.so`。使用第 6 章的示例源码时，链接主库及 C 程序自身使用的数学库即可。以下命令以 GNU 链接器为例。把运行库路径换为目标平台对应目录；若系统已能加载全部依赖，可以只保留产品库路径。

```
cc -std=c11 -O2 -I"$PANGULU_ROOT/include" quick_start.c \
  -L"$PANGULU_ROOT/lib" -Wl,--no-copy-dt-needed-entries \
  -Wl,--allow-shlib-undefined -lpangulu -lm -o pangulu_demo
LD_LIBRARY_PATH="$PANGULU_ROOT/lib:/absolute/path/to/platform/runtime/lib" \
  ./pangulu_demo
```

也可在主程序或部署环境中配置包和运行库的搜索路径。上面的 GNU 链接选项将共享库的间接依赖留给目标机器的动态加载器解析。应用直接引用的 PanguLU 符号仍由链接器检查。使用 CMake 时可参照手册源中的 `examples/CMakeLists.txt`，并将 `PANGULU_ROOT` 设为解压目录。

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、数值因子和工作资源，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 b，结果写入 x。两个向量描述符可分别使用 HOST 或 DEVICE。
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。调用时同时提供 A、x、b。该枚举保持值 2047，流程不含独立的重分解调用。

REORDERING、SYMBOLIC_FACTORISATION、PREPROCESSING 是独立的阶段枚举，按重排、符号分解、预处理的顺序建立分析状态。通常使用 ANALYSIS 一次完成三项。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，先销毁向量和矩阵描述符。由 pangulu_device_alloc() 分配的缓冲区应通过 pangulu_device_free() 在销毁句柄前释放。其余缓冲区按各自分配方式释放。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 在 GPU 上执行右端项变换、求解和解的排回。对 DEVICE 描述符，数据在所选设备上，应用可继续在该设备处理结果或显式下载。对 HOST 描述符，SOLVE 每次上传 HOST 右端项，或下载

HOST 解。HOST 解在调用成功返回时可读；首次使用 HOST 右端项可能需要分配设备暂存区。只传输每列有效元素，输入 B 和列间填充保持不变。频繁求解、已经驻留 GPU 的数据或后续 GPU 计算，宜使用 PANGULU_MEMORY_DEVICE，减少跨主机传输。

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 匹配、重排与复用

匹配、重排算法和因子复用可分别设置。这些选项可能影响结果、用时和显存用量。复用设置决定分析和分解结果何时可以重复使用。

3.6.1 匹配和缩放

通过 pangulu_config_set(h,PANGULU_REORDERING_METHOD,value) 设置。在首次分析前，通用 LU 可选 L0（关闭匹配与缩放）或 L1（启用匹配与缩放）。对称或 Hermitian 分解需保持行列共用的对称结构。

3.6.2 重排算法

用 pangulu_reordering_algo_set() 选择算法。句柄的初始选择为 ND。下表列出 PANGULU_REORDERING_ALGO_ 后缀及固定枚举值。

枚举后缀	值	含义
ND	0	嵌套剖分，按图分隔点组织消元。
AMD	1	近似最小度排序，以减少消元填充。
RCM	2	反向 Cuthill–McKee 排序，用于缩小带宽。
BTF	3	块三角形式，按结构依赖分出对角块。
BTF_COLAMD	4	块三角形式，结合块内列近似最小度排序。
DM	5	Dulmage–Mendelsohn 分解，根据二部图匹配暴露结构块。
GIVEN_PERM	98	使用应用提供的行、列排列。
NONE	99	跳过所选图重排。需要同时关闭匹配与缩放时，通用 LU 还应设置 L0。

对称及 Hermitian 分解要求行列排列相同。如果某一内置方法对给定矩阵产生不同的行、列排列，请改用能保持对称结构的重排方案。新算法设置以最后一次成功设置为准。改变算法或已登记的排列后，应重新分析及分解，再提交求解。

3.6.3 应用指定行列排列

选择 GIVEN_PERM 后，用 ANALYSIS 或 REORDERING 提交矩阵以登记维度。收到 PANGULU_STATUS_REORDERING_PERM_NOT_SET 时，调用 pangulu_reordering_perm_set(h,row_perm,col_perm)，随后重试刚才的阶段。

两个数组均位于主机内存，类型为 int，长度等于矩阵阶数。对每个数组，新位置 k 对应原位置 perm[k]，下标从 0 开始，每个下标恰好出现一次。L1 匹配后的坐标可能不同于原矩阵，排列解释在匹配后的坐

标中；要按原矩阵坐标解释，请用 L0。通用矩阵的行、列排列可以不同。成功设置时库会复制数组。尺寸改变后应为新的阶数重新提交排列。

3.6.4 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。

匹配和复用级别应在首次依赖相应设置的阶段前配置。已进入计算流程后，如需变更匹配和复用级别，可销毁并重新创建句柄。

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 复用要求压缩指针、行列索引及存储项顺序保持一致，矩阵尺寸、布局、类型、性质与存储三角也保持不变。描述符及数组地址可以更新，系数数值可以改变。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。

PANGULU_C_128F 是保留值，当前不支持该输入类型。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_HOST 或 PANGULU_MEMORY_DEVICE	形状为 $n \times$ nrhs。DEVICE 数据位于句柄选定的 GPU。
解 X	PANGULU_MEMORY_HOST 或 PANGULU_MEMORY_DEVICE	形状和数值类型与 B 一致，内存位置可独立选择，缓冲区相互独立。

4.5 稀疏存储格式

系数矩阵须为方阵，满足 m=n>0。CSC、CSR、BSC 和 BSR 均可用于以下矩阵性质与存储视图。

4.5.1 矩阵性质与分解方法

通过 pangulu_matrix_create_sym() 声明矩阵性质。枚举值使用 PANGULU_MATRIX_ 前缀；下表列出后缀。

性质后缀	输入要求	分解方法
GENERAL	通用实数或复数方阵	LU
SYMMETRIC	实对称或复对称，$A = A^T$	LDLT，$A = LDL^T$
SPD	实对称正定矩阵	Cholesky，$A = LL^T$
HERMITIAN	复数 Hermitian，$A = A^H$	LDLH，$A = LDL^H$
HPD	复数 Hermitian 正定矩阵	Cholesky，$A = LL^H$

pangulu_execute() 根据描述符选择分解方法，调用形式保持不变。性质由应用声明，并非运行时自动判定；声明为正定的矩阵必须满足正定条件。复对称与 Hermitian 的镜像规则不同，应按实际方程选择。

4.5.2 完整矩阵与单三角输入

matrix_view 使用 PANGULU_MATRIX_VIEW_ 前缀，支持 FULL、LOWER 和 UPPER。兼容名称 PANGULU_MVIEW_FULL 和 PANGULU_MVIEW_HALF 分别等同于 FULL 和 LOWER。通用矩阵仅使用 FULL；其余性质可提供完整矩阵、下三角或上三角。单三角输入保留所选三角及对角，库依据声明的对称或共轭对称关系解释另一半，应用无需展开镜像项。

完整输入的两个三角必须符合声明的矩阵性质。块格式中，非对角块按所选块三角保存；对角块仍为完整的 $b \times b$ 数值块，计算使用其中声明的标量三角。所有输入数组保持由应用持有，库不会原地修改。

CSC 按列压缩，CSR 按行压缩，标量格式的 block_size 使用 1。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 nnzb^2 个元素。每块内部采用列主序，元素 (i,j) 位于块内偏移 $jb+i$。

4.5.3 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。

4.5.4 创建描述符前的检查

调用矩阵创建函数前，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。库可在 GPU 上归并重复坐标，其值相加；显式零值保留在稀疏模式中。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。

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

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，DEVICE 缓冲区位于句柄选定的 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、数值因子及工作区。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 的 19 个函数。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()	设置匹配与缩放策略或复用级别。
pangulu_reordering_algo_set()、pangulu_reordering_perm_set()	选择重排算法及给定的行、列排列。
pangulu_device_alloc()、pangulu_device_copy()、pangulu_device_free()	管理句柄选定 GPU 的设备缓冲区和数据传输。
pangulu_matrix_create()	为完整稀疏系数矩阵创建描述符。
pangulu_matrix_create_sym()	声明矩阵性质与完整或单三角输入。
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_get_last_internal_error_code()	读取有效句柄的最近一次内部故障编号。
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。
pangulu_license_check()、pangulu_license_status()	本地授权检查。
pangulu_print_gpu_info()	保留的设备信息兼容入口。

除两项授权检查返回 int 外，下列函数返回 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	已授权设备有序列表中的下标，从 0 开始；该序号可能与平台的物理设备编号不同。
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

函数完成授权和设备检查后，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()

为句柄设置匹配与缩放策略或复用级别，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。标准版每个句柄使用一个 GPU，头文件中的 PANGULU_GPU_COUNT 多卡取值适用于支持相应功能的版本。
value	输入	该配置项支持的取值。匹配策略支持 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 重排选择与行列排列

```
pangulu_status_t pangulu_reordering_algo_set(
    pangulu_handle_t handle, pangulu_reordering_algo_t algorithm);
pangulu_status_t pangulu_reordering_perm_set(
    pangulu_handle_t handle, int *row_perm, int *col_perm);
```

algorithm 使用第 3.6 节的枚举值。row_perm、col_perm 是主机上长度等于矩阵阶数的置换数组，并由库复制。先选择 GIVEN_PERM、提交矩阵维度，再设置这两个数组。对称及 Hermitian 分解要求两者内容相同。错误处理见第 7 章。

5.4 设备内存接口

```
pangulu_status_t pangulu_device_alloc(
    pangulu_handle_t handle, void **out, size_t bytes);
pangulu_status_t pangulu_device_copy(
    pangulu_handle_t handle, void *dst, const void *src,
    size_t bytes, pangulu_copy_direction_t direction);
pangulu_status_t pangulu_device_free(
    pangulu_handle_t handle, void *ptr);
```

out 接收设备地址；bytes 以字节为单位。dst 和 src 为不重叠的目标及源缓冲区。GPU 地址均属句柄选定的设备。方向枚举为 PANGULU_COPY_HOST_TO_DEVICE (0)、PANGULU_COPY_DEVICE_TO_HOST (1) 或 PANGULU_COPY_DEVICE_TO_DEVICE (2)。复制成功返回时传输已完成，零字节复制为空操作。

用同一句柄通过 pangulu_device_free() 释放此接口分配的 ptr，并在销毁句柄前完成释放。允许释放空指针。

在既有平台程序中，也可用原生接口分配和传输内存，再以 PANGULU_MEMORY_DEVICE 创建描述符。应先完成原生上传，使用句柄选定设备上的地址，并用原分配方式释放。只使用 PanguLU 设备内存接口的应用仅需包含 pangulu.h 及标准 C 头文件。

5.5 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 valueType,
    pangulu_operation_t operation,
    pangulu_memory_space_t memorySpace);
```

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。
valueType	输入	声明 values 数组的精度及实数、复数类型。
operation	输入	选择 PANGULU_LU、PANGULU_CHOLESKY 或 PANGULU_LDLT，分别描述通用、正定或对称完整输入。复数 Hermitian 输入使用扩展创建函数声明。
memorySpace	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。

5.6 pangulu_matrix_create_sym()

创建带矩阵性质及完整或单三角存储声明的矩阵描述符。使用公开的 26.48 头文件编译，并链接对应运行库。

```
pangulu_status_t pangulu_matrix_create_sym(
    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_matrix_type_t matrix_type,
    pangulu_matrix_view_t matrix_view,
    pangulu_memory_space_t memory_space);
```

参数与行为

matrix_type 声明通用、对称、对称正定、Hermitian 或 Hermitian 正定性质；matrix_view 声明完整、下三角或上三角输入。各枚举值与分解方法的对应关系见第 4 章。其他参数与 pangulu_matrix_create() 相同。

例如，以下调用以主机内存中的双精度实数 CSR 下三角创建正定矩阵描述符。nnz 为实际保存的下三角条目数。

```
pangulu_matrix_create_sym(
    &A, PANGULU_MATRIX_LAYOUT_CSR, n, n, nnz, 1,
    row_ptr, col_ind, values,
    PANGULU_R_64I, PANGULU_R_32I, PANGULU_R_64F,
    PANGULU_MATRIX_SPD, PANGULU_MATRIX_VIEW_LOWER,
    PANGULU_MEMORY_HOST);
```

之后仍通过 pangulu_execute() 执行分析、数值分解及求解，向量创建、复用和销毁方式均保持不变。本函数只创建描述符，不判断矩阵是否满足正定条件。

5.7 pangulu_vector_create()

为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。

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

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	连续缓冲区的地址，用于保存右端项或接收解。
value_type	输入	与系数矩阵一致的数值类型。
memory_space	输入	PANGULU_MEMORY_HOST 或 PANGULU_MEMORY_DEVICE，须与实际地址匹配。

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。创建描述符时只登记调用者提供的地址；输入与输出使用独立缓冲区。HOST 的传输在 SOLVE 时进行。

5.8 pangulu_multiple_vector_create()

创建列主序稠密 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 value_type,
    pangulu_memory_space_t memory_space);
```

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	主机或 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。
value_type	输入	与系数矩阵相同的数值类型。
memory_space	输入	PANGULU_MEMORY_HOST 或 PANGULU_MEMORY_DEVICE，须与实际地址匹配。

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。创建描述符时只登记地址，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。

5.9 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、类型匹配的 B/X，DEVICE 缓冲区位于句柄选定的 GPU；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，解在 x 所指的缓冲区内。若 x 为 HOST，则返回时应用可直接读取结果。原方程残差的计算与质量判定见第 7 章。

5.10 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。

5.11 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，对应的主机或设备缓冲区由应用管理。销毁后，该描述符失效。

5.12 pangulu_destroy()

销毁句柄并释放其持有的数值因子、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。

5.13 内部错误编号

```
pangulu_status_t pangulu_get_last_internal_error_code(
    pangulu_handle_t handle, uint64_t *error_code);
```

error_code 指向调用者的输出变量。函数读取有效句柄中记录的最近一次操作结果，数值 0 表示当前无内部错误。读取本身不会清除编号；建议在失败后的下一次句柄操作前读取。以 0x 加 16 位十六进制数字的格式向技术支持报告。公开状态码用于程序控制，内部编号用于故障诊断。

5.14 授权检查与兼容入口

```
int pangulu_license_check(char *error, size_t error_size);
int pangulu_license_status(char *error, size_t error_size);
pangulu_status_t pangulu_print_gpu_info(void);
```

两个授权函数在本地执行检查，返回 0 表示通过，非零表示失败。error 和 error_size 指定由应用分配的诊断缓冲区和字节容量。设备信息函数用于既有调用的二进制兼容，标准发布版保持静默。查看设备和驱动时使用平台的查询工具。

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 检查每次 PanguLU 调用；失败时读取有效句柄的内部诊断，再跳转到 `cleanup`。示例的设备内存操作仅使用 PanguLU 接口。

```c
#include <stdio.h>
#include <math.h>
#include <stdint.h>
#include <inttypes.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_); \
        uint64_t code_ = 0; \
        if (h && pangulu_get_last_internal_error_code(h, &code_) == 0 \
              && code_) \
            fprintf(stderr, "internal error: 0x%016" PRIx64 "\n", code_); \
        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};
    void *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));
    PTRY(pangulu_device_alloc(h, &d_b, sizeof(b)));
    PTRY(pangulu_device_alloc(h, &d_x, sizeof(x)));
    PTRY(pangulu_device_copy(h, d_b, b, sizeof(b),
                             PANGULU_COPY_HOST_TO_DEVICE));
    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));
    PTRY(pangulu_device_copy(h, x, d_x, sizeof(x),
                             PANGULU_COPY_DEVICE_TO_HOST));
    {
        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_device_free(h, d_x);
        pangulu_device_free(h, d_b);
        pangulu_destroy(h);
    }
    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。

手册源的 CMake 工程构建单 RHS 和多 RHS 两个示例。公共 pangulu-samples 仓库还提供 08_multi_rhs/ 示例。按第 2.5 节使用该目录中的 Makefile 编译。

6.6 为通用矩阵设置自定义重排

下例的两个排列来自公共样例的四阶矩阵，行列各不相同。对其他矩阵应替换为对应长度及含义的数组。调用时令 A 为已创建的矩阵描述符。

```
int row_perm[] = {2, 0, 3, 1};
int col_perm[] = {1, 3, 0, 2};
PTRY(pangulu_config_set(h, PANGULU_REORDERING_METHOD, L0));
PTRY(pangulu_reordering_algo_set(h, PANGULU_REORDERING_ALGO_GIVEN_PERM));

pangulu_status_t status =
    pangulu_execute(h, PANGULU_PHASE_ANALYSIS, A, NULL, NULL);
if (status != PANGULU_STATUS_REORDERING_PERM_NOT_SET)
    goto cleanup;
PTRY(pangulu_reordering_perm_set(h, row_perm, col_perm));
PTRY(pangulu_execute(h, PANGULU_PHASE_ANALYSIS, A, NULL, NULL));
```

分析成功后继续执行 NUMERIC 和 SOLVE。自定义排列示例包含完整的程序及小矩阵。

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	操作成功。对解另做数值质量检查。
1	NOT_INITIALIZED	句柄或前置计算状态未建立。
2	ALLOC_FAILED	已识别的主机内存或 GPU 显存分配失败。
3	EXECUTION_FAILED	执行失败。结合阶段、输入及内部错误编号排查。
4	LICENSE_ERROR	授权检查未通过。
5	NOT_SUPPORTED	已弃用的兼容值。对应的失败使用状态码 3。
6	INVALID_ARGUMENT	参数无效，例如重排算法枚举无效或排列指针为空。
7	REORDERING_UNKNOWN_MATRIX_SHAPE	给定排列前，须先通过 ANALYSIS 或 REORDERING 提交矩阵。
8	REORDERING_REDUNDANT_PERM	设置排列前，须先选 GIVEN_PERM。
9	REORDERING_PERM_NOT_SET	已选给定排列。设置行、列数组后重试分析。
10	INVALID_PERMUTATION	检查索引范围、重复项，以及对称分解要求的行列同排列。

创建成功的句柄可用 pangulu_get_last_internal_error_code() 获取内部诊断。应在清理或发出其他句柄操作之前读取，并一同保存公开状态和发生错误的阶段。状态码 3 涵盖不同失败原因；也应检查可用显存及目标平台运行环境。

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、内部错误编号和应用输出，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；DEVICE B/X 是否位于句柄选定的 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

给定排列缺失时可按第 3.6 节补齐并重试。其他失败应停止依赖该结果的计算，保留诊断，再清理资源。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	检查可执行文件中的 RPATH 或系统动态库加载配置，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。

升级已有程序时，可先用所选平台的完整交付包验证既有接口；采用本版新增函数时，使用 26.48 的 `pangulu.h` 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

即使函数返回成功，也应使用原始输入验证数值结果。对于相对残差异常或非有限的结果，应保留已返回的 API 状态，并单独记录应用验收结论。

注意：精度限制
病态矩阵可能产生很大残差。应按原始方程和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、公开状态码、可获取的 16 位十六进制内部错误编号及应用输出。

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。`pangulu-samples/08_multi_rhs/sample.c` 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。
`pangulu-samples/`	若包内提供，可用于验证环境和接口用法。最新示例见 公共仓库。
`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 26.48

- NVIDIA、海光 DTK 和 AMD ROCm 交付包使用统一的公开头文件和 C ABI。
- 增加重排算法选择和自定义行列排列，保留已有匹配与复用设置。
- 提供设备缓冲区分配、复制和释放接口，右端项及解也可使用主机缓冲区。
- 增加公开的内部错误编号查询和两处默认 License 目录。

PanguLU 26.46

- 引入声明通用、对称、对称正定、Hermitian 及 Hermitian 正定矩阵的接口。26.48 使用 `pangulu_matrix_create_sym()`。对称及 Hermitian 类矩阵支持完整矩阵、上三角或下三角输入。
- 扩展 `pangulu_execute()` 的分解方法，根据声明的矩阵性质自动选择 LU、Cholesky 或 LDLT/LDLH；Hermitian 矩阵采用相应的共轭转置形式，函数调用形式保持不变。
- 保留原有 C 接口与二进制调用约定，既有程序可直接替换匹配环境的动态库，无需重新编译。

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