LinglongAMG 26

用户手册

C API 使用与参考

版本：26.44

适用平台：NVIDIA CUDA, Hygon HIP, AMD HIP, Intel SYCL, Moore Threads MUSA, Lisuan OpenCL, MetaX MXMACA

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 交付目录 . . . 2
2.2 配置运行路径与 License . . . 2
2.3 授权生命周期 . . . 3
2.4 编译与运行标准样例 . . . 3
2.5 应用链接 . . . 4
2.6 CMake 接入 . . . 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 设备与并发 . . . 5
3.5 JSON 配置 . . . 6
3.6 复用条件 . . . 6
3.7 单周期预条件模式 . . . 6

4 数据类型与存储格式 **7**
4.1 不透明对象 . . . 7
4.2 数值类型与索引 . . . 7
4.3 CSR 存储 . . . 7
4.4 示例矩阵 . . . 7
4.5 向量与求解信息 . . . 8
4.6 缓冲区容量 . . . 8

5 函数参考 **9**
5.1 `linglongamg_create_from_config` . . . 9
5.2 `linglongamg_read_csr` . . . 10
5.3 `linglongamg_setup` . . . 11
5.4 `linglongamg_solve` . . . 11
5.5 `linglongamg_get_solve_info` . . . 12
5.6 `linglongamg_reset` . . . 13
5.7 `linglongamg_destroy` . . . 13
5.8 `linglongamg_csr_destroy` . . . 14
5.9 `linglongamg_status_string` . . . 14

6 使用示例 **15**
6.1 通用 CBD 测试程序 . . . 15
6.2 核心调用流程 . . . 15
6.3 同一矩阵连续求解 RHS . . . 16
6.4 执行一次 AMG 周期 . . . 16

7 错误处理与结果验证 **17**

7.1 错误状态 . . . 17
7.2 按阶段排查与恢复 . . . 17
7.3 常见集成问题 . . . 18
7.4 数值精度与残差 . . . 18
7.5 有限值与显示精度 . . . 18
7.6 性能测量 . . . 19
7.7 显存规划 . . . 19
7.8 问题报告 . . . 19

附录 A 示例输入文件 20
A.1 CBD 二进制布局 . . . 20
A.2 样例 RHS 与业务输入 . . . 20

附录 B 文档与支持 21
B.1 交付文件 . . . 21
B.2 技术支持 . . . 21

附录 C 版本日志 22
C.1 产品版本 . . . 22
C.2 手册修订 . . . 22

1 简介

LinglongAMG 是面向 GPU 的稀疏线性系统迭代求解库，通过公共 C API 求解

$$Ax = b, \qquad A \in \mathbb{R}^{n \times n}, \quad b, x \in \mathbb{R}^{n}.$$

其中 A 为系数矩阵，b 为右端项，x 为解向量。Setup 建立代数多重网格（AMG）层次；Solve 使用该层次执行配置指定的迭代。应用也可执行一次完整 AMG 周期，由外部 Krylov 迭代器控制收敛。

1.1 支持范围

项目	支持范围
数值类型	当前交付 ABI 使用实数 FP64（`double`），无复数输入接口。
稀疏存储	完整、零基、设备端 CSR；维度和索引为有符号 32 位整数。
右端项与解	每次调用一个 RHS；b, x 是长度 n 的连续设备向量。
设备使用	一个系统使用一个 GPU，无公开多卡协同或 MPI 分布式接口。
重复求解	矩阵不变时复用 AMG 层次，逐次求解新的 RHS。
内存范围	无自动主机内存溢出或外存计算流程。

1.2 文档约定

等宽字体表示函数、类型、字段和命令。产品版本 26.43 与文档修订号 1.9 分别标识运行库和手册。具体能力与系统、GPU、驱动及依赖要求，以随包头文件、README、配置和发布说明为准。不同平台的算法组合可能不同。

> **说明：适用性**
> 先确认数据规格与矩阵条件，再验证收敛和资源。PCG 需要相应的对称正定条件；奇异系统的投影或约束由应用处理。AMG 不保证任意系统均可收敛。

2 安装与快速开始

准备匹配目标 GPU、系统 ABI 与运行时的包及有效 License。示例链接随包共享库，无需重新构建 LinglongAMG。

2.1 交付目录

```
LinglongAMG/
|-- lib/liblinglongamg.so
|-- lib/linglongamg.h
|-- test_src/
|-- build_sample.sh
|-- CMakeLists.txt
|-- linglongamg_config.json
|-- spe10.cbd
|-- run_linglongamg.sh
`-- README.md
```

上图为包含标准样例的布局。原始 `license.key` 由用户另行放入包根目录，替换可能存在的空占位文件。保持库与头文件来自同一构建，保留完整配套文件。`build/` 在本地构建时生成，不预置给用户。

2.2 配置运行路径与 License

从包的父目录执行以下命令。`LINGLONGAMG_SDK` 是示例使用的 shell 变量，用于定位目录，不是库的鉴权参数。

```
cd ./LinglongAMG
export LINGLONGAMG_SDK="$PWD"
export LD_LIBRARY_PATH="$LINGLONGAMG_SDK/lib${LD_LIBRARY_PATH:+:$LD_LIBRARY_PATH}"
test -s "$LINGLONGAMG_SDK/license.key"
```

最后一条仅检查文件非空，不验证授权。变量只对当前终端及其启动程序有效；作业脚本也应配置运行环境。保留双引号以处理含空格路径。

使用有效的原始 License 文件，不手工编辑。授权在本地检查，运行时无需联网。当前鉴权构建从 SDK 本地查找 `license.key` 或 `licenses/*.key`；不要套用其他产品的授权路径环境变量。

2.3 授权生命周期

创建句柄时读取 JSON 并验证授权，检查产品、设备、版本范围、有效期、交付完整性及签名撤销列表（CRL）策略。通过平台运行时先选定 GPU；矩阵和向量位于该 GPU。创建函数不接收单独的“已授权 GPU 列表下标”。

正式鉴权构建在 `linglongamg_get_solve_info` 中再次检查授权并提交结果。只有查询成功后才能读取 x；失败时不能把缓冲区旧值当成本次结果。配置创建与后置查询位于 Setup/Solve 计时之外，参见第 7 章。

更新 License 或 CRL 后，先结束当前计算、销毁旧句柄，再创建新句柄。不要依赖已有句柄自动重新加载。按交付流程更新签名文件，不自行改写。

2.4 编译与运行标准样例

```
chmod +x ./build_sample.sh ./run_linglongamg.sh
./build_sample.sh
./run_linglongamg.sh
./run_linglongamg.sh -A ./my_matrix.cbd -c ./my_config.json
```

先加载随包要求的平台工具链与 CMake。默认输入取自包根目录中的 `spe10.cbd` 和 `linglongamg_config.json`；用户指定的相对路径以当前 shell 目录为基准。标准样例生成 $b = A\mathbf{1}$，使用 $x_0 = 0$，用于已知解检查；业务 RHS 由应用通过 API 提供。

保存 API 状态、阶段耗时和原方程残差。API 成功不等于精度达标；错误码、NaN、Inf 或残差异常按第 7 章处理。

2.5 应用链接

手册源码包的 `examples/cbd_sample.cpp` 是通用 CBD 测试程序；`cbd_reader.h` 负责输入检查。默认读取当前目录的 `spe10.cbd` 与 `linglongamg_config.json`，生成 $b = A\mathbf{1}$，以零初值求解并独立验算。完整库与授权保留在 SDK 中，示例通过 `AMG_ROOT` 指向该 SDK。头文件与库必须配套。以下假设示例目录与 LinglongAMG 目录并列：

```
Parent/
|-- LinglongAMG/
|   |-- lib/liblinglongamg.so
|   |-- lib/linglongamg.h
|   |-- license.key
|   |-- spe10.cbd
|   `-- linglongamg_config.json
`-- examples/
    |-- cbd_sample.cpp
    |-- cbd_reader.h
    `-- CMakeLists.txt
```

2.6 CMake 接入

从 examples 目录运行，使用匹配的开发工具链与运行库：

```
cmake -S . -B build -DBACKEND=CUDA -DAMG_ROOT=../LinglongAMG
cmake --build build --parallel
./build/cbd_sample -A ../LinglongAMG/spe10.cbd \
    -c ../LinglongAMG/linglongamg_config.json
```

CUDA 未自动找到时，在配置命令追加 `-DCUDAToolkit_ROOT="$CUDA_HOME"`。海光使用以下配置和匹配 DTK 环境：

```
cmake -S . -B build-hip -DBACKEND=HIP \
    -DAMG_ROOT=../LinglongAMG -DDTK_ROOT="$ROCM_PATH"
cmake --build build-hip --parallel
./build-hip/cbd_sample -A ../LinglongAMG/spe10.cbd \
    -c ../LinglongAMG/linglongamg_config.json
```

完整 CMake 文件随示例提供。构建时运行路径指向所选 SDK；移动目录或更换机器后应重新配置，保持授权与库的相对位置不变。

3 通用说明

3.1 句柄与状态

句柄保存配置、AMG 层次和工作资源。以下流程按顺序执行；任何一步失败，应停止依赖该状态的后续调用。

```
select device; create_from_config(config)
upload CSR, b, x0; read_csr(...)
setup(solver, A)
solve(solver, b, x)
get_solve_info(solver, info)
read x; verify original-system residual
destroy(solver); csr_destroy(A); free device arrays
```

3.2 执行阶段

阶段	作用
配置创建	读取 JSON，检查授权并创建句柄。
Setup	由 CSR 矩阵构建多层结构与工作数据。
Solve	执行配置指定的求解或周期；鉴权构建暂存结果。
结果查询	检查后置授权、提交解并返回信息。

3.3 内存所有权与生命周期

CSR 描述符引用已有设备数组，不复制或接管其所有权。应用负责分配、传输和释放。使用期间保持矩阵、b, x 有效，结果查询成功前不要读取 x 为本次结果。先销毁求解器，再销毁 CSR 描述符，最后释放设备缓冲区。

3.4 设备与并发

同一句柄串行使用，不进行并发或重入调用；生命周期内保持设备选择一致。独立系统分别管理句柄与数据依赖，不能将设备指针直接交给另一 GPU 使用。

3.5 JSON 配置

`setup.max_levels`	层数上限。
`setup.min_coarse_size`	最粗层规模阈值。
`setup.strong_threshold`	强连接阈值。
`setup.presweeps` `setup.postsweeps`	前、后平滑次数。
`solve.method`	求解方法或单周期模式。
`solve.smoother` `solve.cycle`	层内平滑器与循环类型。
`solve.max_iters` `solve.tolerance`	迭代上限与内部容差。
`solve.norm` `solve.convergence`	范数与归一化口径。

以随包配置为起点，保留字段所在的 setup/solve 对象。手册示例使用 GMRES、AMG 预条件、V-cycle 与 L1-Jacobi，并不代表全平台默认值。解析某字段不表示目标后端已支持相应功能。更改配置后重新创建句柄并 Setup。

3.6 复用条件

变化	处理方式
仅 b 改变	保留未改变的矩阵与层次，初始化 x 后再次 Solve 并查询。
A 数值改变	重新 Setup；不默认旧层次仍可用。
结构或维度改变	重建矩阵描述符与求解器，完整 Setup。
Reset 后	重新 Setup 后才调用 Solve；改变配置需重建句柄。

3.7 单周期预条件模式

使用适配配置中的 `method=amg`、`max_iters=1`。每次输入外层残差 r，清零修正向量 z，执行完整周期得到 $z \approx M^{-1}r$，查询成功后交给外层。极小内部容差用于抑制提前停止，不是业务验收精度。

4 数据类型与存储格式

4.1 不透明对象

```
typedef struct linglongamg_solver_st *linglongamg_handle_t;
typedef struct linglongamg_csr linglongamg_csr_t;
```

应用保存对象指针，不访问内部字段。使用与库配套的头文件，不手工修改 ABI 声明。

4.2 数值类型与索引

当前交付的 `linglongamg_real_t` 是 `double`，不要定义 `LINGLONGAMG_USE_FP32` 后仍链接同一 FP64 库。维度、nnz、行指针及列索引均使用有符号 32 位 int；转换时检查溢出与数组容量。

4.3 CSR 存储

`row_ptr`	$n+1$ 项，首项 0，非递减，末项 nnz。
`col_idx`	nnz 项，范围 $[0, n)$，行内按列排序并预先合并重复项。
`values`	nnz 项 FP64 实数，与列索引对应，不含 NaN/Inf。

要求非空方阵。对称矩阵也须提供完整 CSR，不只传一侧三角。三个数组位于同一 GPU；主机数据由应用预先上传。

4.4 示例矩阵

$$A = \begin{bmatrix} 4 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 3 \end{bmatrix}, \quad b = \begin{bmatrix} 3 \\ 2 \\ 2 \\ 2 \end{bmatrix}, \quad x_* = \mathbf{1}.$$

```
row_ptr = [0, 2, 5, 8, 10]
col_idx = [0,1, 0,1,2, 1,2,3, 2,3]
values  = [4,-1, -1,4,-1, -1,4,-1, -1,3]
```

4.5 向量与求解信息

```
typedef struct {
    int n;
    linglongamg_real_t *val;
} linglongamg_vec_t;

typedef struct {
    int converged;
    int iterations;
    linglongamg_real_t initial_residual;
    linglongamg_real_t final_residual;
    linglongamg_real_t relative_residual;
    double setup_ms;
    double solve_ms;
} linglongamg_solve_info_t;
```

向量描述结构在主机上，val 指向设备数组。b 是输入，x 保存初值及查询提交后的解；分别分配，避免别名。

converged	是否满足内部停止判据，不代替独立验收。
iterations	已执行迭代数。
initial_residual final_residual	配置定义下的初始与最终残差。
relative_residual	按配置归一化的相对残差。
setup_ms solve_ms	库内部耗时，单位毫秒。

4.6 缓冲区容量

FP64、32 位索引下，输入 CSR 基本容量为 $4(n+1)+12\,\mathrm{nnz}$ 字节，每个向量 $8n$ 字节；不包含多层矩阵、插值、工作向量、运行时及结果暂存。峰值显存需实测。

5 函数参考

本章列出公开头文件 `linglongamg.h` 中的核心函数。参数中的输入由应用提供，输出用于接收创建的对象或计算结果。句柄、描述符与数据缓冲区的生命周期约定见第 3 章，数据类型与存储格式见第 4 章。

函数	用途
`linglongamg_create_from_config`	读取 JSON 配置，检查授权并创建求解器句柄。
`linglongamg_read_csr`	为应用提供的 CSR 数组创建矩阵描述符。
`linglongamg_setup`	建立 AMG 层次和求解工作数据。
`linglongamg_solve`	按配置执行求解，生成待查询确认的解。
`linglongamg_get_solve_info`	检查后置授权、提交解并返回求解信息。
`linglongamg_reset`	清理求解状态与层次数据，供后续重新 Setup。
`linglongamg_destroy`	销毁求解器句柄并释放库内部资源。
`linglongamg_csr_destroy`	销毁 CSR 描述符，保留应用持有的数组。
`linglongamg_status_string`	将状态码转换为只读诊断字符串。

除 `linglongamg_csr_destroy` 无返回值、`linglongamg_status_string` 返回字符串外，表中其余函数均返回 `linglongamg_status_t`，成功值为 `LINGLONGAMG_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。只有 `linglongamg_get_solve_info` 成功后才能读取或下载解向量，不得将缓冲区中的旧值视为本次结果。

5.1 `linglongamg_create_from_config`

```
linglongamg_status_t linglongamg_create_from_config(
    linglongamg_handle_t *solver, const char *config_file);
```

参数

`solver`	输出句柄变量地址。
`config_file`	可读 JSON 路径；相对路径以当前进程目录为基准。

行为与返回

选择 GPU 后调用。读取配置并检查授权；成功后才能 Setup。创建失败则停止后续调用。

5.2 linglongamg_read_csr

```
linglongamg_status_t linglongamg_read_csr(
    linglongamg_csr_t **A, int nrows, int ncols, int nnz,
    const int *device_row_ptr, const int *device_col_idx,
    const linglongamg_real_t *device_values);
```

参数

A	CSR 描述符输出地址。
nrows, ncols, nnz	方阵维度及存储项数，满足 int32 范围并与数组一致。
device_row_ptr	设备行指针，长度 nrows+1。
device_col_idx	设备列索引，长度 nnz。
device_values	设备 FP64 数值，长度 nnz。

行为与返回

只创建引用现有数组的描述符，不读磁盘，不执行 Setup，不接管输入内存。所有数组满足第 4 章规范。

5.3 linglongamg_setup

```
linglongamg_status_t linglongamg_setup(
    linglongamg_handle_t solver, const linglongamg_csr_t *A);
```

参数

solver	有效且已配置的句柄。
A	指向有效设备 CSR 的描述符。

行为与返回

建立 AMG 层次及求解工作数据。成功后方可 Solve。前次求解须已完成结果查询；矩阵变化时的重建规则见第 3 章。

5.4 linglongamg_solve

```
linglongamg_status_t linglongamg_solve(
    linglongamg_handle_t solver, const linglongamg_vec_t *b,
    linglongamg_vec_t *x);
```

参数

solver	Setup 成功且无待查询结果的句柄。
b	输入向量的主机描述结构，val 指向同一 GPU 上的 RHS。
x	初值与输出缓冲区的主机描述结构，长度与 b 一致。

行为与返回

执行配置指定的求解或周期。返回成功不保证业务精度；正式鉴权构建的解仍待查询提交。每次 Solve 后成功查询，再读取解或再次 Solve。

5.5 linglongamg_get_solve_info

```
linglongamg_status_t linglongamg_get_solve_info(
    linglongamg_handle_t solver, linglongamg_solve_info_t *info);
```

参数

solver	本次 Solve 已成功返回的句柄。
info	主机端信息结构的可写地址。

行为与返回

检查后置授权并提交解，返回第 4 章的信息结构。只有 SUCCESS 后才能读取或下载 x。失败时停止使用结果；第 7 章给出恢复流程。

5.6 linglongamg_reset

```
linglongamg_status_t linglongamg_reset(linglongamg_handle_t solver);
```

参数

`solver`	尚未销毁的有效句柄。

行为与返回

清理当前求解状态及层次工作数据，下次 Solve 前重新 Setup。不读取新 JSON，不代替更新 License 后重新创建句柄。错误恢复优先销毁并重建。

5.7 linglongamg_destroy

```
linglongamg_status_t linglongamg_destroy(linglongamg_handle_t solver);
```

参数

`solver`	要结束生命周期的句柄。

行为与返回

释放句柄内部资源，随后不再访问该句柄。应用仍需销毁 CSR 描述符并释放自己的设备数组。

5.8 linglongamg_csr_destroy

```
void linglongamg_csr_destroy(linglongamg_csr_t *A);
```

参数

A	不再被求解器使用的 CSR 描述符。

行为与返回

销毁描述符，但不释放引用的设备数组。本函数没有返回状态。

5.9 linglongamg_status_string

```
const char *linglongamg_status_string(linglongamg_status_t status);
```

参数

status	公共 API 返回状态。

行为与返回

返回只读诊断字符串，不修改或释放。程序逻辑按枚举判断，排查时保留原始日志。

> **说明：授权诊断扩展**
> 部分头文件还声明机器码和授权诊断函数；这些不代替创建与结果查询中的正式验证。激活使用当前包指定的采集工具和流程。

6 使用示例

6.1 通用 CBD 测试程序

程序读取矩阵和 JSON 配置，链接共享库并独立验算原系统残差。支持 CUDA 和 HIP 构建；输入格式见附录 A。未提供参数时，从当前工作目录读取 spe10.cbd 与 `linglongamg_config.json`。

```
./build/cbd_sample
./build/cbd_sample -A ./spe10.cbd -c ./linglongamg_config.json
./build/cbd_sample -A ./other.cbd -c ./config.json \
    --rtol 1e-8 --device 0
```

`--rtol` 默认 10^{-6}，独立于 JSON 内部容差；`--device` 默认为 0。程序不改写配置。为已知解测试生成 $b = A\mathbf{1}$，初值为零，不读取业务 RHS 文件。

6.2 核心调用流程

以下摘录展示数据上传后如何调用库；完整的参数解析、CBD 读取、资源清理和验算见源码包。

```
amg_check(linglongamg_create_from_config(&solver.handle,config));
amg_check(linglongamg_read_csr(&solver.matrix,a.n,a.n,a.nnz,
                               dr.ptr,dc.ptr,da.ptr));
linglongamg_vec_t rhs{a.n,db.ptr}, solution{a.n,dx.ptr};
gpu_check(GPU(DeviceSynchronize)()); auto t0=Clock::now();
amg_check(linglongamg_setup(solver.handle,solver.matrix));
gpu_check(GPU(DeviceSynchronize)()); auto t1=Clock::now();
amg_check(linglongamg_solve(solver.handle,&rhs,&solution));
gpu_check(GPU(DeviceSynchronize)()); auto t2=Clock::now();
linglongamg_solve_info_t info{};
amg_check(linglongamg_get_solve_info(solver.handle,&info));
gpu_check(GPU(DeviceSynchronize)()); auto t3=Clock::now();
// Download x only after the query succeeds.
```

耗时采用两位小数并注明 ms/us，残差与容差采用两位小数科学计数法。独立验算使用 long double 累加与稳定范数，确认设备 RHS 未改变。退出码 0 表示验收通过，2 表示残差未达标，1 表示参数、文件、GPU、API 或授权错误。零 RHS 时注明相对残差未定义，并要求绝对残差为零。

6.3 同一矩阵连续求解 RHS

```
/* Keep A unchanged; run Setup once. */
for (int k = 0; k < rhs_count; ++k) {
    upload_rhs(k, b_device);  /* application operation */
    zero_solution(x_device); /* application operation */
    check(linglongamg_solve(solver, &b, &x));
    check(linglongamg_get_solve_info(solver, &info));
    consume_and_verify(k, x_device);
}
```

上传、清零、检查和验算是应用伪代码，不是库 API。每个 RHS 单独记录状态、耗时和残差，不以平均值掩盖异常。

6.4 执行一次 AMG 周期

```
/* Compatible configuration: method=amg, max_iters=1. */
zero_solution(z_device);
linglongamg_vec_t r = {n, residual_device};
linglongamg_vec_t z = {n, z_device};
check(linglongamg_solve(solver, &r, &z));
check(linglongamg_get_solve_info(solver, &info));
/* The outer iteration may now consume z_device. */
```

一次周期后的不收敛标记可以是正常行为，外层算法负责整体收敛与业务验收。不要为改变内部标记而放宽业务残差要求。

7 错误处理与结果验证

API 状态说明操作是否完成，数值验收说明解是否满足要求；二者分别记录。

7.1 错误状态

0 / `SUCCESS`	本次操作成功，仍需独立验算。
1 / `INVALID_VALUE`	参数、尺寸或调用状态无效。
2 / `ALLOC_FAILED`	资源分配失败，检查内存与显存。
3 / `CUDA_ERROR`	GPU 操作错误；枚举名称保留 CUDA。
4 / `NOT_IMPLEMENTED`	当前实现不支持请求的功能。
5 / `RUNTIME_ERROR`	运行时错误，保留原始诊断。
6 / `LICENSE_ERROR`	授权、设备、期限或完整性检查失败。

枚举加 `LINGLONGAMG_STATUS_` 前缀。日志中的内部负值不是这些公共枚举；保留第一处失败 API、返回码及完整原始日志。

7.2 按阶段排查与恢复

创建句柄	JSON 路径、原始授权、设备选择、配套库和文件权限。
描述符／ Setup	CSR 尺寸、索引、排序、存储位置、配置及显存。
Solve	Setup 成功、向量长度与设备、有限值及前次结果查询。
结果查询	Solve 是否成功与授权是否有效；失败时不使用解。

失败后停止后续依赖计算，清理已创建资源。修正问题后重新创建句柄并 Setup，旧 x 不作为本次结果。

7.3 常见集成问题

找不到头文件或库	将搜索路径指向同一交付包的 lib 目录。
缺少.so 或符号	检查运行库路径、完整依赖和 ABI，不拼接不匹配的包。
授权失败	核对原始 key、GPU、版本、期限与完整错误信息。
残差不下降	核对矩阵条件、初值、平滑器与迭代配置，独立验算。

7.4 数值精度与残差

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

使用原始系统数据。若 $\|b\|_2 = 0$，报告绝对残差，注明相对残差未定义。配置为 `rel_ini` 时，库以初始残差归一化，非零初值下不一定等于上述 RHS 归一化指标。

7.5 有限值与显示精度

1. 检查输入、解、残差各分量、范数与最终比值是否有限。
2. 采用稳定范数算法，必要时提高累加精度，避免上溢或下溢。
3. NaN/Inf 如实输出，不替换为 0；残差使用科学计数法。
4. 耗时显示两位小数并注明 ms 或 s，计数保持整数；验收使用未格式化数值。

> **说明：残差与解误差**
> 病态矩阵的小残差不一定表示小解误差。存在参考解时另行计算相对解误差。数值验收失败与 API 执行失败分别保存。

7.6 性能测量

$$T_{\text{solver}} = T_{\text{setup}} + T_{\text{solve}}.$$

该和式不包括读文件、转换、传输、配置创建、后置查询与独立验算。业务总耗时另行说明范围。库内部毫秒计时与同步后的墙钟时间可能覆盖不同工作，不混用。

```
synchronize_device(); t0 = clock_now();
check(linglongamg_setup(solver, A));
synchronize_device(); t1 = clock_now();
check(linglongamg_solve(solver, &b, &x));
synchronize_device(); t2 = clock_now();
check(linglongamg_get_solve_info(solver, &info));
synchronize_device(); t3 = clock_now();
/* setup=t1-t0; solve=t2-t1; query_publish=t3-t2 */
```

同步与时钟函数为应用伪代码。确保 GPU 工作完成后结束计时，避免只测命令提交开销。比较时保持数据、配置、初值、容差、硬件与计时边界一致；分别报告首次运行、复用运行及波动，保留超时和失败。

7.7 显存规划

多层结构与工作区取决于矩阵结构、粗化、插值密度、平滑器和迭代空间，不能仅用原始 nnz 或文件大小推断。测量目标设备峰值，预留运行时资源，控制并发任务占用。

7.8 问题报告

提供包名及校验值、头文件版本、系统、GPU、驱动、运行时、配置、最小调用程序、首次失败 API、返回码和原始日志。矩阵问题补充维度、nnz、格式、显存与原方程残差；可以共享时提供最小输入。不公开授权内容或敏感凭据。

附录 A　示例输入文件

本附录说明标准读取器支持的 CBD 文件。公共 API 只接收设备 CSR 数组，业务格式由应用解析、转换并按第 4 章校验。

A.1　CBD 二进制布局

顺序／字段	类型与数量
1 / nrows, ncols	各一个 32 位整数。
2 / nnz	一个 64 位整数。
3 / row_ptr	$n+1$ 个 64 位整数。
4 / col_idx	nnz 个 32 位整数。
5 / values	nnz 个 FP64 实数。

读取器使用本机二进制布局，当前交付平台使用小端序，不自动交换字节序。读入后转换为 API int32 行指针，检查溢出、末项、排序和索引范围。文件后缀不能代替格式验证。

A.2　样例 RHS 与业务输入

标准样例生成 $b = A\mathbf{1}$ 并使用零初值。业务 b, x_0 由应用单独准备并上传。独立验收必须保留原系统；不能以样例构造 RHS 替代业务 RHS。

附录 B　文档与支持

使用与运行库配套的文件。手册源码只含文档与公开接口示例，不包含实现库或有效授权。

B.1　交付文件

`lib/linglongamg.h`	公开 C API 声明。
`README.md`	平台环境、操作步骤和限制。
`linglongamg_config.json`	配套默认配置。
`test_src/`	交付样例源码。
发布说明与校验记录（如提供）	构建变更、已知问题和文件完整性。

B.2　技术支持

准备第 7 章的复现信息，通过末页企业微信联系技术支持。授权入口为 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

产品版本标识运行库，文档修订号标识手册。以下不构成全平台具有相同功能或性能的承诺。

C.1　产品版本

C.1.1　LinglongAMG 26.43

手册依据该版本的公共 CSR、Setup/Solve 和结果查询接口编写。正式鉴权结果在查询成功后使用。具体能力、运行要求及兼容性以交付说明为准；更新文档不表示运行库已重新编译或上线。

C.2　手册修订

C.2.1　1.9

按统一框架整理中英文手册，补充参数、授权生命周期、阶段依赖、复用和验证。区分 API 状态、内部收敛与独立验收，保留当前企业微信入口。

C.2.2　1.6

整理 LinglongAMG 专用能力说明、输入输出规格、调用示例与部署流程，采用简洁技术手册版式。

授权获取

企业微信

store.solved-tech.com