HYLU 使用说明

（版本 20260126）

陈晓明（chenxiaoming@ict.ac.cn）

HYLU (Hybrid Parallel Sparse LU Factorization)是一款为多核共享内存架构设计的通用并行求解器，用于高效求解稀疏线性方程组（**Ax = b**）。该求解器采用创新的并行向上看（up-looking）LU 分解算法，通过集成混合的数值内核以动态适应不同的矩阵稀疏度，其数值稳定性通过结合静态选主元、超节点对角块动态选主元、动态数值缩放，以及迭代修正技术来保障。

HYLU 为大规模稀疏线性方程组提供高性能 LU 分解方案，其适用范围涵盖电路仿真、电力系统、计算流体力学、电磁学与结构分析等多个工程领域，可高效求解源于有限元分析、2D/3D 建模及优化问题的线性方程组。

HYLU 基于 C 语言实现，可方便地与 C/C++应用程序集成。HYLU 的多线程基于操作系统的原生线程接口实现（无需 OpenMP）。

1. 函数

HYLU 提供 7 组函数用于求解稀疏线性方程组，每组函数均提供了支持实数和复数线性系统求解的 32 位和 64 位整数版本。下表列出了 HYLU 函数名的格式和对应的版本。下文中，将以 32 位整数的实数版本的函数名来说明 HYLU 功能。在 32 位整数版本中，仅输入矩阵的索引使用 32 位整数，而内部 LU 因子的数据结构使用 64 位整数以确保扩展性。HYLU 的函数原型以及函数参数的说明，请参阅<hylu.h>头文件。

函数名	版本	库文件
HYLU_XXXX	32 位整数/实数	libhylu.so|hylu.dll
HYLU_L_XXXX	64 位整数/实数	libhylu_l.so|hylu_l.dll
HYLU_C_XXXX	32 位整数/复数	libhylu_c.so|hylu_c.dll
HYLU_CL_XXXX	64 位整数/复数	libhylu_cl.so|hylu_cl.dll

(1) HYLU_CreateSolver

该函数用于创建求解器实例，获取参数数组指针，并启动工作线程以实现并行计算。在此过程中，所有输入参数被初始化为默认值。工作线程的数量不应超过空闲核心的数量。

(2) HYLU_DestroySolver

该函数释放所有已分配内存，终止工作线程，并销毁求解器实例。

(3) HYLU_Analyze

该函数执行预处理步骤，包括静态选主元、矩阵重排序和符号分解。输入矩阵格式是**压缩稀疏行**（compressed sparse row, CSR），每行中的列索引**无需有序**。对于对称矩阵，输入的 CSR 格式仍须存储矩阵的**全部**元素。

该函数的第二个参数"repeat"用于指定是否需要对具有相同矩阵结构的线性方程组进行重复求解（常见于电路仿真等实际应用）。在这种场景中，**预处理仅需执行一次**。启用此参数时，将影响部分参数的默认值设置。在重复求解场景下，HYLU 会优先最小化 LU 因子中的非零元数量，虽然可能增加预处理时间，但能显著提升后续分解效率。

在预处理阶段提供矩阵数值（"ax"参数）是可选的，但强烈建议提供。在没有矩阵数值的情况下，不能实施静态选主元和静态数值缩放。

(4) `HYLU_Analyze2`

该函数基于用户提供的矩阵排序，执行与 `HYLU_Analyze` 类似的功能。用户提供的排序必须确保换序后的矩阵拥有**非零对角线**。静态选主元和静态数值缩放将被禁用。

(5) `HYLU_Factorize`

该函数对预处理后的矩阵执行数值 LU 分解，将其分解为下三角矩阵（**L**）和上三角矩阵（**U**）。

数值分解过程中将实施超节点对角块动态选主元策略。若某行无法找到合适的主元，求解器自动采用主元扰动策略。

(6) `HYLU_Solve`

该函数通过前代（$\mathbf{Ly = b}$）与回代（$\mathbf{Ux=y}$）运算，利用已计算的 LU 因子求解目标向量。

若上一次数值 LU 分解中曾触发主元扰动，求解过程将自动执行迭代修正以修正解向量。

(7) `HYLU_MSolve`

该函数用于求解具有相同系数矩阵的多个线性方程组。当启用多线程时，则对不同右端向量进行并行求解。

2. 参数

下表详细说明了参数数组。所有输入参数在调用 `HYLU_CreateSolver` 时将被初始化为默认值（标有星号*的参数）。用户应在调用 `HYLU_Analyze` 或 `HYLU_Analyze2` 之前完成对输入参数的设置（如需要）。

参数	描述
`parm[0]`: 输出	软件版本。
`parm[1]`: **输入**	计时器。当启用时，`parm[7]`记录最近一次函数调用的运行时间。

	0*	不启用。
	>0	高精度计时器（微秒精度）。
	<0	低精度计时器（毫秒精度）。
parm[2]: **输入**	减少填入的排序算法。	
	0*	排序方法根据矩阵维度和 HYLU_Analyze 的 "repeat" 参数自动决定。
	1	近似最小度算法。
	2	近似最小度算法变种。
	3	嵌套剖分算法 1。
	4	嵌套剖分算法 2。
	5	1 和 2 中的最佳方案。
	6	3 和 4 中的最佳方案。
	7	1-4 中的最佳方案。
parm[3]: **输入**	排序方法切换阈值。在嵌套剖分中，若子图的尺寸小于 parm[3] 设定值，系统自动切换至约束最小度排序方法。较小的 parm[3] 通常会提升嵌套剖分排序质量，但同时会增加排序计算时间。	
	0*	自动控制，其值根据 HYLU_Analyze 的 "repeat"参数自动决定。
	>=64	允许的范围。
parm[4]: 输出	选择的排序算法（1-4），由 HYLU_Analyze 输出。	
parm[5]: **输入**	超节点的最小列数。每个超节点至少包含 parm[5]列。	
	32*	默认值。
	>=8	允许的范围。
parm[6]: **输入**	超节点最大行数限制。每个超节点允许的最大行数是 parm[6]。当超节点行数超过此阈值时，求解器将其拆分为多个超节点。	
	0*	自动控制，其值由是否已创建工作线程来决定。
	>=8	允许的范围。
parm[7]: 输出	最近一次函数调用的运行时间（单位为**微秒**）。	
parm[8]: 输出	非对角线主元数量，由 HYLU_Factorize 输出。	
parm[9]: 输出	超节点数量，由 HYLU_Analyze 或 HYLU_Analyze2 输出。	
parm[10]: **输入**	主元扰动系数。当出现零主元或小主元时，主元被替换为 $\text{sign(pivot)} \times 10^{\text{parm[10]}} \times \|\mathbf{A}\|_\infty$。	
	-15*	默认值。
	<0	允许的范围。
parm[11]: 输出	扰动的主元数量，由 HYLU_Factorize 输出。	
parm[12]: 输出	当前内存使用量（字节），当函数返回-4 时则表示为所需内存大小（字节），由 HYLU_Analyze 或 HYLU_Analyze2 输出。	
parm[13]: 输出	最大内存使用量（字节），由 HYLU_Analyze 或 HYLU_Analyze2 输出。	
parm[14]: 输出	该参数使用 3 个 short 类型数据存储线程数量信息，可通过以	

	下方式获取： const short *threads = (short *)&parm[14]; threads[0]: 物理核心数（可能不正确）。 threads[1]: 逻辑核心数。 threads[2]: 已创建的工作线程数。	
parm[15]: **输入**	迭代修正的最大迭代次数。	
	0*	自动控制是否执行迭代修正以及迭代次数。
	>0	如果 HYLU 决定执行迭代修正，执行 parm[15]次迭代。
	<0	执行-parm[15]次迭代。
parm[16]: 输出	已执行的迭代修正的迭代次数，由 HYLU_Solve 输出。	
parm[17]: 输出	**L** 的非零元数量（包括对角线），由 HYLU_Analyze 或 HYLU_Analyze2 输出。	
parm[18]: 输出	**U** 的非零元数量（不包括对角线），由 HYLU_Analyze 或 HYLU_Analyze2 输出。	
parm[19]: 输出	数值分解的浮点计算次数（不包括缩放），由 HYLU_Analyze 或 HYLU_Analyze2 输出。	
parm[20]: 输出	求解的浮点计算次数（不包括缩放），由 HYLU_Analyze 或 HYLU_Analyze2 输出。	
parm[21]: **输入**	矩阵缩放方法。	
	>0*	动态数值缩放。
	0	不启用。
	<0	静态数值缩放。
parm[22]: **输入**	符号分解方法。对称符号分解可减少预处理时间，但对非结构对称的矩阵会增加填入。	
	>0	非对称符号分解。
	<0	对称符号分解。
	0*	自动控制。
parm[23]: **输入**	串行与并行计算的符号结果的一致性。	
	!=0*	启用。
	0	不启用。

3. 函数返回值

所有 HYLU 函数均通过整型返回值传递错误代码，具体含义如下表所示。

返回值	描述
0	函数执行成功。
-1	无效的实例句柄。
-2	函数参数错误（例如，矩阵维度为负、空指针）。
-3	非法矩阵（例如，矩阵索引错误）。

-4	内存不足，parm[12]将记录所需内存。
-5	矩阵结构奇异。
-6	矩阵数值奇异。
-7	线程操作失败。
-8	调用顺序错误。
-9	整数溢出，请使用 HYLU_(C)L_*系列函数。
-10	内部错误。

4. 使用 HYLU 库

HYLU 以 x64 动态链接库的形式提供。**运行 HYLU 需要支持 FMA 和 AVX2 指令集的 x64 处理器。**推荐使用支持 AVX-512 指令集的处理器。

请根据所使用的操作系统平台，参考以下说明链接 HYLU 库。

- **Linux 平台**

在编译链接最终可执行文件时，请使用`-L<库所在目录> -l<库名>`这两个选项来指明 HYLU 库的位置和名称。

有可能还需要链接一些系统库。通常需要额外加上`-lpthread`、`-lm`和`-ldl`这几个链接选项。对于较旧的 Linux 发行版，可能还需要添加`-lrt`。一个完整的编译链接命令示例如下：`gcc -o myprogram myprogram.c -L/path/to/hylu -lhylu -lpthread -lm -ldl`。

在运行程序前，确保系统能定位 HYLU 的动态库。最常用的方法是设置 LD_LIBRARY_PATH 环境变量，例如：`export LD_LIBRARY_PATH=<库所在目录>:$LD_LIBRARY_PATH`。

- **Windows 平台（使用 Visual Studio）**

在源代码文件的任意位置（通常在文件开头），添加一行编译指示指令：`#pragma comment(lib, "<库名>.lib")`。在链接阶段，仅需要.lib 导入库文件。

程序运行时只需要.dll 文件。推荐且最简单的方法是，将对应的.dll 文件复制到可执行文件（.exe）所在的目录下。

5. 注意事项

(1) **用户不要释放参数数组内存**，该内存由求解器内部管理。

(2) HYLU 默认采用**行优先**存储，对列优先存储的矩阵，需设置 HYLU_Solve 的"transpose"参数为 true 以求解转置的线性方程组（$\mathbf{A}^{\mathrm{T}}\mathbf{x}=\mathbf{b}$）。但是，列模式求解的并行扩展性不如行模式求解。

(3) 对于对称矩阵，输入的 CSR 格式仍须存储矩阵的全部元素。

(4) 请确保 HYLU 创建的线程数量小于等于空闲核数，否则 HYLU 计算性能将急剧下降。

(5) 由于有限的主元选择范围，对于某些线性方程组，HYLU 可能会求得不精确的解。在这种情况下，建议将 parm[23]设为 0，parm[6]调至更大数值并改用

串行分解模式。parm[10]和 parm[21]也可能会影响结果的精度。

(6) 在使用英特尔 MKL BLAS 的 HYLU 库中观察到内存泄漏问题，该问题归因于 MKL BLAS 函数自身。

(7) 设置 parm[23]!=0 确保串行与并行计算时符号结果的一致性，但数值结果可能会略有不同。若 parm[23]=0 且使用多线程，则并行嵌套剖分的排序结果可能具有随机性。