

HYLU 使用说明

(版本 20250903)

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 ($\mathbf{Ax} = \mathbf{b}$)。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、动态对角超节点选主元、动态数值缩放，以及迭代修正技术来保障。

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 5 个用户友好的函数用于求解稀疏线性方程组。该函数库同时支持 32 位和 64 位整数版本 (64 位整数版本通过 `_L` 后缀标识)。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 因子的数据结构仍使用 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。

(1) `HYLU(_L)_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，所有输入参数将被初始化为默认值。工作线程的数量不应超过空闲核心的数量。

(2) `HYLU(_L)_DestroySolver`

该函数释放所有已分配内存，终止已创建的线程，并销毁求解器实例。

(3) `HYLU(_L)_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**。

该函数的第二个参数 `"repeat"` 用于指定是否需要具有相同矩阵结构的线性方程组进行重复求解 (该场景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。

(4) HYL_U(_L)_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动态对角超节点选主元策略。若无法找到合适主元，求解器将自动采用主元扰动策略。

(5) HYL_U(_L)_Solve

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

若上次数值 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。

2. 参数

下表详细说明了参数数组的每个元素。所有输入参数在调用 HYL_U(_L)_CreateSolver 时将被初始化为默认值 (标有星号*的参数)。用户应在调用 HYL_U(_L)_Analyze 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，用户可从 parm[7] 来获得上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*	排序方法将根据矩阵维度和 HYL _U (_L)_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。当采用嵌套剖分排序时，若子图的尺寸小于 parm[3] 设定值，系统将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，其值将根据 HYL _U (_L)_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 HYL _U (_L)_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, 求解器将自动将其拆分为多个超节点。	
	0*	自动控制, 其值将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 HYL _U (_L)_Factorize 输出。	
parm[9]: 输出	超节点数量, 由 HYL _U (_L)_Analyze 输出。	
parm[10]: 输入	主元扰动系数。当出现零主元或小主元时, 主元将被替换为 $\text{sign}(\text{pivot}) \times 10^{\text{parm}[10]} \times \ \mathbf{A}\ _{\infty}$ 。	
	-15*	默认值。
	<0	允许的范围。
parm[11]: 输出	扰动的主元数量, 由 HYL _U (_L)_Factorize 输出。	
parm[12]: 输出	当前内存使用量 (字节), 当函数返回 -4 时表示为所需内存大小 (字节), 由 HYL _U (_L)_Analyze 输出。	
parm[13]: 输出	最大内存使用量 (字节), 由 HYL _U (_L)_Analyze 输出。	
parm[14]: 输出	<p>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:</p> <pre>const short *threads = (short *)&parm[14];</pre> <p>threads[0]: 物理核心数 (可能不正确)。 threads[1]: 逻辑核心数。 threads[2]: 已创建的线程数。</p>	
parm[15]: 输入	迭代修正的最大迭代次数。	
	0*	自动控制。
	>0	允许的范围。
parm[16]: 输出	已执行的迭代修正的迭代次数, 由 HYL _U (_L)_Solve 输出。	
parm[17]: 输出	L 的非零元数量 (包括对角线), 由 HYL _U (_L)_Analyze 输出。	
parm[18]: 输出	U 的非零元数量 (不包括对角线), 由 HYL _U (_L)_Analyze 输出。	
parm[19]: 输出	数值分解的浮点计算次数 (不包括缩放), 由 HYL _U (_L)_Analyze 输出。	
parm[20]: 输出	求解的浮点计算次数 (不包括缩放), 由 HYL _U (_L)_Analyze 输出。	
parm[21]: 输入	矩阵缩放。	
	1*	动态数值缩放。
	0	不启用。

	-1	静态数值缩放。
parm[22]: 输入		对称符号分解。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	不启用。
	1	启用。
	-1*	自动控制。

3. 函数返回值

所有 HYL U 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]将返回所需内存。
-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 HYL U_L_*系列函数。
-10	内部错误。

4. 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) HYL U 默认采用行优先存储，对列优先存储的矩阵，需设置 HYL U(_L)_Solve 的"transpose"参数为 true 以求解转置的线性方程组 ($\mathbf{A}^T \mathbf{x} = \mathbf{b}$)。但是，列模式求解的并行扩展性不如行模式求解。
- (3) 请确保 HYL U 创建的线程数量小于等于空闲核数，否则性能将急剧下降。
- (4) 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[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。
- (5) 使用 MKL BLAS 的 HYL U 库存在内存泄露，这是由于 MKL BLAS 的函数造成的。