arXiv:2301.07964v1 [cs.DC] 19 Jan 2023

Pa并行两阶段约化至 Hessenberg-三角形式

Thijs Steel RafVandebril

2023年1月20日

摘要

我们提出了一种两阶段算法，用于将矩阵束并行约化为 Hessenberg-三角形式。传统的两阶段海森堡-三角约化算法在第一阶段性能很高，但在第二阶段难以实现高性能。我们的算法扩展了 Karlsson等人所述的技术 [9] ，使其在第二阶段也能实现高性能。在共享内存环境中进行的实验表明，该算法的性能可优于最先进的实现。

1 引言

我们研究将矩阵束约化为 Hessenberg-三角形式的并行算法。对于矩阵束 (A, B) A, $B \in \mathbb{C}^{n\times n}$，这包括求出酉矩阵 Q 和 Z、Hessenberg 矩阵 H 以及上三角矩阵 T，使得

$$Q(H,T)Z^* = (A,B).$$

这种分解最常见的用途是作为 QZ 算法的预处理步骤。QZ 算法用于求解广义特征值问题，由 C.B. Moler 和 G.W. Stewart 提出 [10]。本文旨在介绍并评估一种新的 Hessenberg-三角约化算法，该算法能够在多核机器上实现良好的扩展性。

并行化稠密线性代数算法的一种常见方法，是将算法设计为尽可能多地通过大型矩阵-矩阵乘法执行浮点运算次数。这样，这些算法的实现便可以依赖高度优化、并行性能良好的矩阵-矩阵乘法实现。2008 年，B. K˚agstr¨om、D. Kressner、E.S. Quintana-Ort´ı 和 G. Quintana-Orti [7] 提出了一种将矩阵束约化为 Hessenberg-三角形式的算法。该算法至少 60% 的运算通过矩阵乘法完成。与 Moler 和 Stewart 最初提出的约化方法相比，这是一个重大改进，但在考虑并行化时仍存在问题。如果我们只依赖矩阵乘法的并行化，那么 40% 的工作将无法并行化。

一种可行的解决方案是采用两阶段算法。在第一阶段，将矩阵束约化为 r-海森堡-三角形式。当 T 为上三角矩阵，且 $H_{i,j} = 0 \forall i > j + r$ 时，矩阵束 (H, T) 即为 r-海森堡-三角形式，也就是说， H 至多有 r 条非零次对角线。将矩阵束高效并行化地约化为 r-海森堡-三角形式要容易得多。K. Dackland 和 B. K˚agstr¨om [6] 首次提出了这一约化的并行算法，随后 K˚agstr¨om 等人 [7] 对其进行了改进。在第二阶段，再将处于 r-海森堡-三角形式的矩阵束约化为 Hessenberg-三角形式。据我们所知，文献中尚未出现第二阶段的有效并行化方法。此外，这两阶段的组合

比单阶段算法所需的浮点运算次数约多 50%。尽管存在这些缺点，K˚agstr¨om 等人报告称，两阶段方法能够取得良好的性能。

将矩阵约化为海森堡形式也有类似的两阶段算法。特别是，L. Karlsson 和 B. K˚agstr¨om [8,9] 成功实现了海森堡矩阵第二阶段的高效并行化。本文的主要贡献是将他们的工作扩展到 Hessenberg-三角约化。

B. Adlerborn、L. Karlsson 和 B. K˚agstr¨om [1]的研究提供了另一种解决方案。他们也认识到，依赖矩阵乘法的并行化并非有效策略。他们没有改用两阶段算法，而是非常谨慎地对 K˚agstr¨om 等人 [7] 提出的单阶段约化的每一步进行并行化。

本文其余部分安排如下。首先，我们回顾 K˚agstr¨om 等人 [7] 提出的将矩阵束约化为 r-海森伯格-三角形铅笔形式的分块约化，并在第 2 节讨论其并行化方法。其次，我们在第 3 节介绍一种新的并行算法，用于将 r-海森伯格-三角形铅笔约化为 Hessenberg-三角形式。最后，我们在第 4 节评估并行两阶段算法的性能。

2 化简为 r-海森堡-三角形式

2.1 豪斯霍尔德反射的 WY 形式

在后文将介绍的算法中，我们经常需要对一个矩阵应用一系列豪斯霍尔德反射。逐个应用这些反射通常不是最优做法。为了获得更好的缓存性能，我们可以使用这一系列反射的 WY表示。如果 $Q = Q_r Q_{r-1} \dots Q_1$ 是这些反射的乘积，那么我们可以将 Q 分解为

$$Q = I - WY^*,$$

其中 Q 是一个 $n \times n$ 矩阵，而 W 和 Y 是 $n \times r$ 矩阵。计算矩阵 W 和 Y 的算法由 C. Bischof 和 C. Van Loan 提出 [4]。将 Q 与一个 $n \times m$ 矩阵 A 相乘所需的浮点运算次数，大致与逐个应用反射子相同，但只需进行两次矩阵-矩阵乘法，而非 r 次反射子应用。若有高度优化的矩阵-矩阵乘法可用，使用 WY表示通常会高效得多。

在本文其余部分中，当我们称某个对象为块反射器时，指的是它是由一系列反射子组成，并使用其 WY 表示来表示。

2.2 分块第一阶段

基本上，分块第一阶段算法（约化为 r-上 Hessenberg-三角形式）是一种面板约化。在每次迭代中，该算法使用块反射器从左侧将 n_b 列的 A 约化为一个面板，同时使用块反射器从右侧操作，以保持 B的结构。该算法之所以能进行面板约化，关键在于计算反射子时只更新 A和 B 的子矩阵。计算完成后，其余更新便可通过矩阵乘法高效完成。下面来看该算法的一次迭代，其中我们要约化从 j 到 $j + n_b - 1$的列。图1展示了这次迭代。本小节其余部分将详细说明该算法。

矩阵束的初始状态如图1a所示。计算左侧反射子的最简单方法，是对块 $A(j+n_b:\ n, j:\ j+n_b-1)$进行 QR分解。该 QR分解的正交因子会约化 A中的整个面板。然而，应用这一块反射器会在 B中产生大量填充。为避免这种填充，我们将 $A(j+n_b:\ n, j:\ j+n_b-1)$拆分成若干个大小为 $pn_b \times n_b$的块，并对这些块进行 QR分解，其中 p 是一个实现参数。计算反射子时只需更新面板。之后，可以累积反射子，并通过矩阵乘法将其应用于矩阵束的其余部分。约化后的块如图1b所示。若 p 较大，则块反射器较大、数量较少，通常效率更高。然而，若 p 较大，也会导致大量填充。K˚agstr¨om 等人 [7]报告称，5 到 12 之间的适中取值通常最优。

经过左乘后，矩阵束处于图1c所示的状态。现在， A 中的块列已完成约化，我们需要消除 B 中的填充，同时不扰动 A中已约化的列。将 B 还原为上三角形式的最简单方法，是对所有产生填充的子块进行 RQ 分解，并从右侧应用其正交因子。K˚agstr¨om 等人 [7] 指出，无需将 B 完全还原为上三角形式，只需约化子块的前 n_b 列。

使用从右侧施加的反射子来约化选定的列并非易事。通常，从右侧施加反射子约化的是选定的行，而不是列。为完成这项任务，可以采用反向豪斯霍尔德反射器。D. Watkins [12], 最先指出，从右侧施加反向豪斯霍尔德反射器可以约化一列。要使用这一技巧，我们对该子块的 RQ 分解所得正交因子的前 n_b 行进行 LQ 分解。若从右侧将此 LQ 分解的正交因子施加到 B，便可约化子块的前 n_b 列。详细分析请参阅 K˚agstr¨om 等人 [7]。

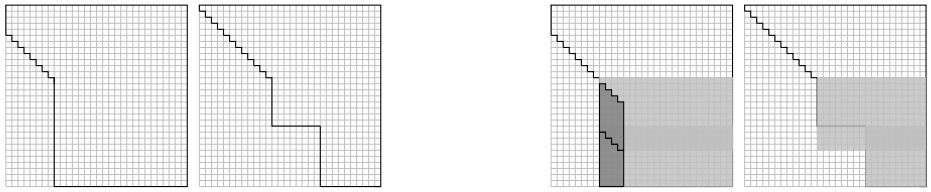
生成反向反射子的方式看起来可能有些奇怪。我们使用反向反射子来避免对子块进行RQ分解，但要生成反向反射子，我们仍然需要计算RQ分解。当考虑将反射子应用于矩阵其余部分的代价时，这种做法的优势便显而易见。RQ分解使用 pn_b 个反射子，而LQ分解只使用 n_b 个反射子。从右侧进行完整约化的过程如图1d所示。请注意，这些乘法不会影响 A的结构。

图e展示了本次迭代后矩阵束的最终状态。 B 中的次对角块不会干扰后续迭代。在每次迭代中，B 中的块都会向下移动 n_b 个位置，最终移出矩阵束的边缘。

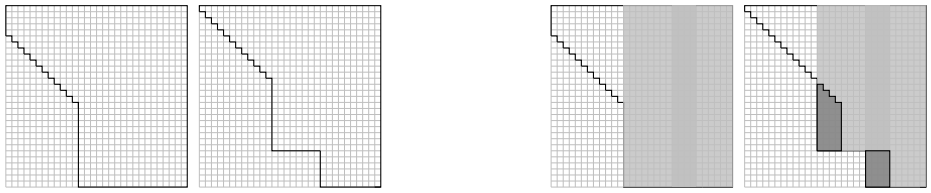
最后，我们来考虑第一阶段所需的浮点运算次数。如果将更新 Q 和 Z所需的浮点运算次数也计算在内，算法1需要$\frac{28p+14}{3}(p-1)n^3 + O(n^2)$次浮点运算。在我们的实现中，我们选择了 $p=8$，因此计算代价为 $11.33\,n^3 + O(n^2)$次浮点运算。

2.3 并行第一阶段

由于算法1是一种分块算法，因此已经具备一定的并行化潜力。将块反射器应用于矩阵需要进行多次矩阵乘法，而这些运算通常可以很好地实现并行化。这本质上就是 Dackland 等人所采用的并行化策略 [6]。然而，对于 r 和 p 的典型取值，这些矩阵的维度太小，无法对其进行有效的并行化。在此，我们提出一种不同的第一阶段并行化方法。



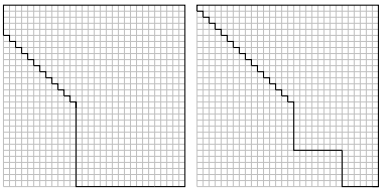
(a) 面板约化开始时矩阵束的初始状态。(b) 通过从左侧进行乘法来约化面板。



(c) 从左侧进行乘法后的矩阵束

。

(d) 约化填充部分



(e) 填充部分约化后的状态

图1：算法1中第一阶段单个面板约化过程的示意图，参数为：$n = 30$、 $n_b = 4$、 $p = 3$。灰色标出的块将被约化。更新将影响矩阵束中浅灰色标出的部分（稍深的灰色表示这些更新的重叠区域）。

我们将采用两级并行化。首先，我们将识别出若干粗粒度任务。其次，我们将把其中一些粗粒度任务拆分为细粒度任务。

算法1的执行方式如下：计算一个块反射器，然后将该块反射器应用于相关矩阵。对于从左侧进行的更新，一个重要的认识是，块反射器序列仅取决于正在约化的面板。我们可以通过将反射子序列的生成作为一项任务（ G_L 任务），然后并行地将该序列应用于 A、 B和 Q （ L_A、 L_B 和 L_Q 任务），来增加并行度。对于从右侧进行的更新，若不更新B，就无法完全计算出反射子序列。我们仍然可以将工作拆分为反射子序列的生成（其中包括将该序列应用于 B ，即 G_R 任务），以及将该序列应用于 A 和 Z （即R_A 和 R_Z 任务）。图2展示了这些任务之间的依赖关系。

这些粗粒度任务可以实现一定程度的并行，但以这种方式最多只能并行执行三个任务。为了获得高效的算法，必须将任务拆分成更小的任务。应用任务： L_Q、 L_A 和 L_B 将一系列块反射器应用于各自的矩阵。要更新这些矩阵中的某一列，任务只需要该特定列，因此可以将矩阵拆分为多个列切片并并行更新这些切片，从而轻松并行化 L_Q、 L_A 和 L_B 。同样的技巧也可用于 R_Z 和 R_A，只是将矩阵拆分为行切片，而不是列切片。图示如下

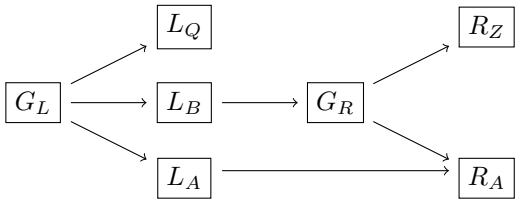
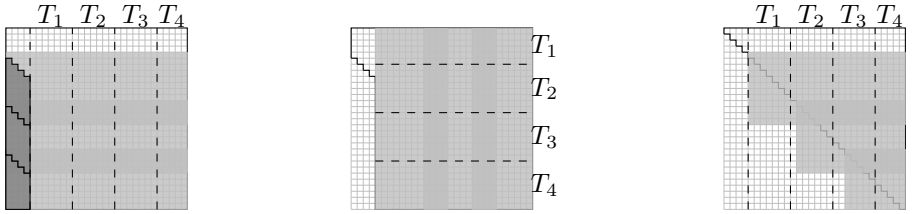


图2：并行阶段一的依赖图。



(a) L_A (b) R_A (c) L_B

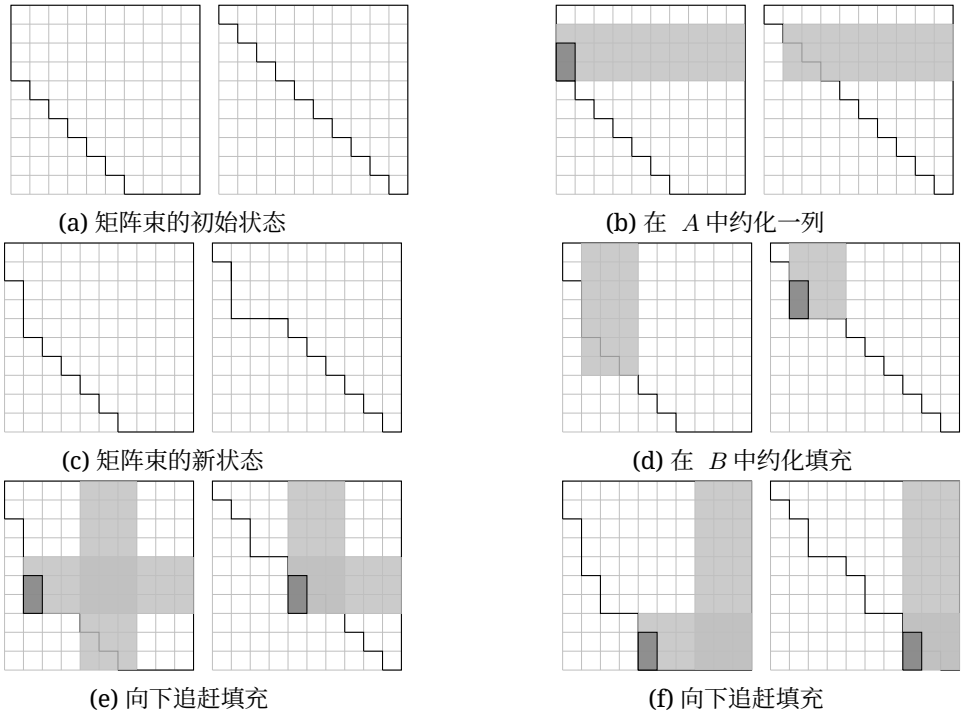
图3：展示并行阶段 1 算法的 L_A、 R_A 和 L_B 任务如何分配给不同的子任务 T_1、T_2，...。深灰色区域表示被约化的块。浅灰色区域表示受乘法影响的矩阵部分。

图3。我们强调，这种矩阵分布可能并不适合分布式内存实现，因为各个核心会不断切换，交替处理矩阵的行和列。我们还注意到， L_B 可能存在负载均衡问题。由于 B的上三角结构， L_B 的不同子任务所需的浮点运算次数各不相同。可以让影响行数较少的任务处理更多列，以此均衡浮点运算次数来解决这一问题。在我们的实现中，我们选择由动态调度器来处理这些负载不均衡。

从某种意义上说，这种并行化类似于本节开头提到的矩阵-矩阵乘法的简单并行化。不同之处在于，我们分配的不是单个块反射器的应用，而是一系列块反射器的应用。这样可以获得相同程度的并行性，但同步点更少。最后，我们考虑生成任务 G_L 和 G_R 的并行化。 G_L 是一个小任务，不值得并行化。 G_R 是最难并行化的任务。计算下一个块反射器需要将前一个块反射器应用于 B，因此此处无法采用我们用于 R_A 的并行化方法。这里只能对矩阵乘法进行简单并行化。

3 化简为 Hessenberg-三角形式

为了将处于 r-海森堡-三角形式的矩阵束约化为 Hessenberg-三角形式，我们将 Karlsson等人针对 r-海森堡矩阵提出的约化算法 [8, 9] 应用于 r-海森堡-三角矩阵束。我们先介绍非分块算法，随后再说明如何延迟更新并高效地应用更新。



(a) 矩阵束的初始状态　(b) 在 A 中约化一列

(c) 矩阵束的新状态　(d) 在 B 中约化填充

(e) 向下追赶填充　(f) 向下追赶填充

图 4：算法 2 示意图。

3.1 非分块第二阶段

与第一阶段一样，我们将通过从左侧应用反射子来约化 A 中的元素，同时通过从右侧应用反射子来消除 B 中的填充。该算法由多次扫描组成。每次扫描中，我们将约化 A 中的一列并消除填充。在扫描 j开始时，前 $j-1$ 列已处于 Hessenberg-三角形式。一次扫描的第一步是从左侧应用反射子 $\hat{}Q_0^j$，以约化 $\hat{}Q_0^j A(j+1:\ j+r, j)$。这会在 B 中产生填充，使得 $B(j+1:\ j+r, j+1:\ j+r)$ 中出现一个块。第一步如图 4b 所示。接下来，我们应用反向反射子 $\hat{}Z_0^j$，以约化 $B(j+1:\ j+r, j+1:\ j+r)\ \hat{}Z_0^j$ 的第一列。第二步如图 4d 所示。请注意，$\hat{}Z_0^j$ 会在 A中产生一些填充。我们可以使用 $\hat{}Q_1^j$ 消除这些填充，再使用 $\hat{}Z_1^j$ 消除由此在 B 中产生的填充。如图 4e 和 4f 所示，这会形成一种鼓包追赶算法：消除填充（有时称为鼓包）会在矩阵更靠下的位置产生更多填充。重复这一过程，直至鼓包被追赶出矩阵末端。完整过程见算法 2。

注意，与第一阶段一样，第一次扫描后 B 是块上三角（而非上三角）。同样，A 并非处于 r-海森堡-三角形式。这些块不会影响后续扫描，并会随着更多列被约化而向下移动。传统上，Hessenberg-三角矩阵的第二阶段采用 Givens 旋转实现；K˚agstr¨om 等人 [7]的研究中对此算法有相关描述。Givens 旋转无需计算反向反射子所需的 RQ 分解，但其浮点运算次数（渐近而言）略多。

如果将更新 Q 和 Z所需的浮点运算次数也计算在内，算法 2 需要 $10n^3 + O(n^2)$ 次浮点运算。请注意，即使渐近上可以忽略不计，当 r 较大时，RQ 分解的 r^2n^2 开销也可能相当可观。在我们的实现中，通常选择 $r = 16$。Moler 和 Stewart 的原始约化算法只需 $14n^3 + O(n^2)$ 次浮点运算，而两阶段约化需要 $21.33n^3 + O(n^2)$ 次浮点运算。这意味着计算量增加了 40% 以上。在适当的情况下，并行化可以弥补计算成本的增加。

3.2 分块第二阶段

我们的分块第二阶段算法基于一个简单的原则：生成反射子时尽可能少地更新矩阵元素，然后利用额外的自由度重新安排操作顺序，以便在完整应用更新时提高效率。我们将先解释如何重新排序并应用更新，最后说明如何生成反射子。这分别是分块算法的应用阶段和生成阶段。

应用反射子 暂且假设我们能够以某种方式生成用于 q 个连续列的反射子，而不更新整个矩阵。即便如此，高效地应用反射子也并非易事。非分块变体按以下顺序应用反射子

$$\hat{Q}_0^j, \hat{Z}_0^j, \hat{Q}_1^j, \hat{Z}_1^j, \ldots, \hat{Q}_0^{j+1}, \hat{Z}_0^{j+1}, \hat{Q}_1^{j+1}, \hat{Z}_1^{j+1}, \ldots$$

$\hat{}$ 一个重要的认识是，如果我们已经拥有全部反射子，就可以先应用所有 Z 个 $\hat{}$，再应用 Q，这是因为结合律。由此得到以下顺序

$$\hat{Z}_0^j, \hat{Z}_1^j, \ldots, \hat{Z}_0^{j+1}, \hat{Z}_1^{j+1}, \ldots, \hat{Q}_0^j, \hat{Q}_1^j, \ldots, \hat{Q}_0^{j+1}, \hat{Q}_1^{j+1}, \ldots$$

人们或许会认为，由于现在左右两侧的更新已分开，应用它们会更高效。遗憾的是，缓存重用并不多。受 $\hat{Q}_k^j$ 影响的元素与受 $\hat{Q}_{k+1}^j$影响的元素没有重叠，因此按此顺序应用反射子时几乎没有缓存重用。我们在选择顺序时还有一些额外的自由度。彼此不重叠的反射子可以交换。这意味着，只要反射子 $\hat{Q}_k^j$ 排在 $\hat{Q}_k^{j-1}$ 和 $\hat{Q}_{k+1}^{j-1}$ 之后（$\hat{Z}$ 的情况也类似），我们就可以任意选择顺序。反射子 $\hat{Q}_k^j$ 会影响 r 列（若是 $\hat{Z}_k^j$，则影响行），其中有 $r-1$ 列与 $\hat{Q}_k^{j+1}$ 共享，因此缓存重用的潜力大得多。按 k 而不是 j 对反射子分组，可能更有效。请看以下顺序

$$\hat{Z}_{l-1}^j, \hat{Z}_{l-1}^{j+1}, \ldots, \hat{Z}_{l-2}^j, \hat{Z}_{l-2}^{j+1}, \ldots, \hat{Q}_{l-1}^j, \hat{Q}_{l-1}^{j+1}, \ldots, \hat{Q}_{l-2}^j, \hat{Q}_{l-2}^{j+1}, \ldots,$$

其中，l 是 k的最大值。很容易看出，现在这些反射子按 k 分组，且不会违反前面提到的约束。C. Bishof、X. Sun 和 B. Lang 最初在 [3] 中针对三对角化约简发现了这一技术。按此顺序处理反射子还使我们能够使用 WY 表示。属于同一k 的每组 q 个反射子都可以累积成一个块反射器，从而使用矩阵乘法。

生成反射子 现在我们已经知道如何高效地应用反射子，接下来需要一种方法，在更新尽可能少的元素的同时，计算多个扫描所需的反射子。这相当于完成以下任务。为列 $j = j_1$：$j_1 + q - 1$生成反射子，同时仅将反射子 $\hat{Q}_k^j$ 应用于 A 的列 $c_{1A}(k,j)$：$c_{2A}(k,j)$ 范围，以及 B 的列$c_{1B}(k,j)$：$c_{2B}(k,j)$ 范围；并将反射子 $\hat{Z}_k^j$ 应用于 A 的行 $r_{1A}(k,j)$：$r_{2A}(k,j)$ 范围，以及 B 的行 $r_{1B}(k,j)$：$r_{2B}(k,j)$ 范围。

我们先推导 A 的表达式。为了生成反射子 $\hat{}Q_k^{j+1}$ ，

$$r_{1A}(k,j) \le j_1 + kr + 1 - r \tag{1}$$

并且为了生成反射子 $\hat{}Q_{k+1}^{j}$，

$$r_{2A}(k,j) \ge \min(j + (k+2)r, n). \tag{2}$$

我们还必须遵守前面提到的约束： Z_k^j 必须位于Z_k^{j-1} 和 Z_{k+1}^{j-1} 之前，这意味着

$$r_{1A}(k,j) \ge r_{1A}(k,j-1) \tag{3}$$

and

$$r_{1A}(k,j) \ge r_{1A}(k-1,j-1)$$

令 $r_{1A}(k, j_1 + q - 1) = j_1 + kr + 1 - r$，并将其与方程 (3.2) 结合，可得$r_{1A}(k, j_1 + q - 2)$ $= j_1 + kr + 1 - 2r$。沿用这一思路，我们可以得出 $r_{1A}(k, j_1 + q - 3) = j_1 + kr + 1 - 3r$，最终得到

$$\begin{aligned} r_{1A}(k,j) &= j_1 + 1 + \max(0, kr - r - (j_1 + q - 1 - j)r) \\ r_{2A}(k,j) &= \min(j + (k+2)r, n). \end{aligned} \tag{4}$$

对于 Q_k^j，取值范围更简单，我们只需令

$$c_{1A}(k,j) = j + \max(0, (k-1)r + 1)$$

and

$$c_{2A}(k,j) = j_1 + q - 1 + \max(0, (k-1)r + 1)$$

这些是生成反射子所需的最小范围，无需为考虑重叠更新而扩大。L. Karlsson等人最早为海森堡矩阵提出了这两个范围。 [8]

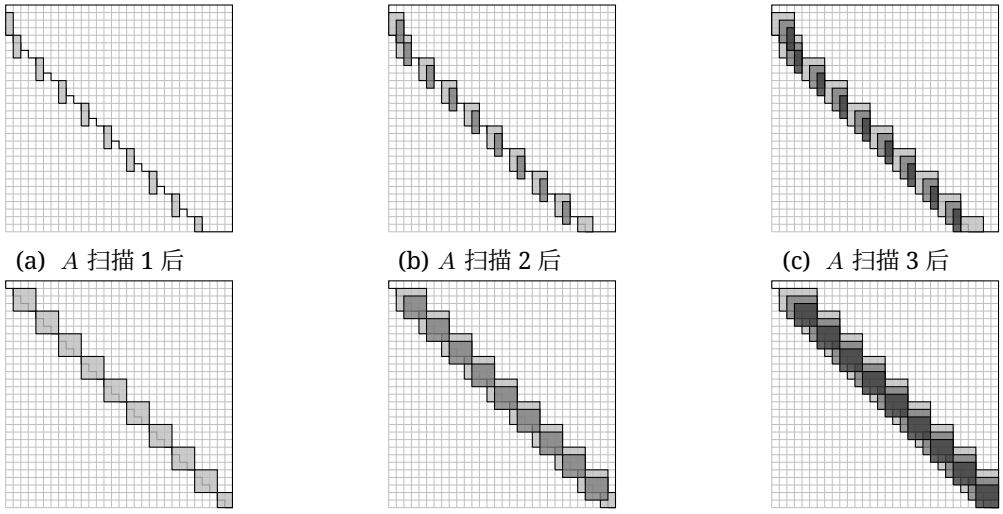
对 B 进行类似推导，可得:

$$\begin{aligned} r_{1B}(k,j) &= j_1 + 1 + \max(0, kr - r - (j_1 + q - 1 - j)r) \\ r_{2B}(k,j) &= \min(j + (k+1)r, n) \end{aligned} \tag{5}$$

and

$$\begin{aligned} c_{1B}(k,j) &= j + kr + 1 \\ c_{2B}(k,j) &= \min(j_1 + q - 1 + (k+1)r, n). \end{aligned} \tag{6}$$

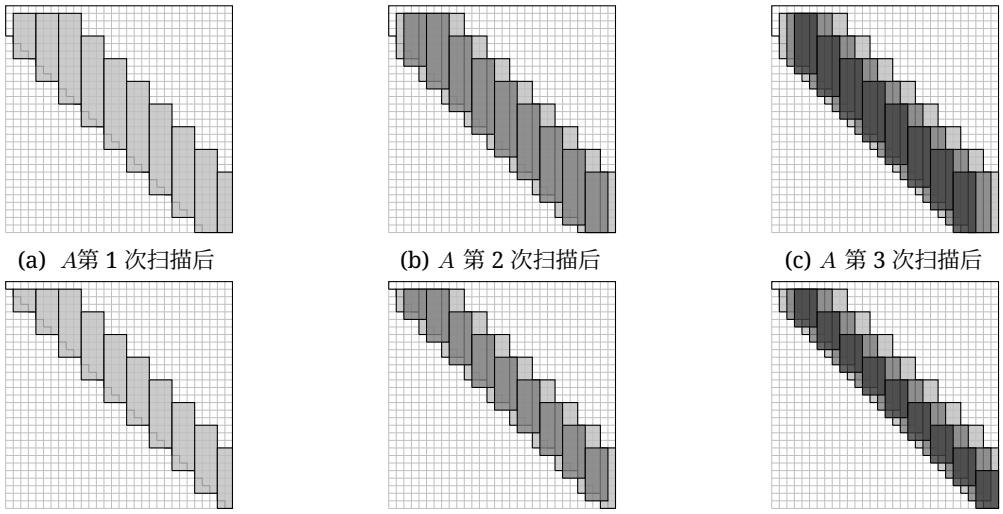
算法 3 在仅更新 A 和 B 的一小部分的同时，为一系列列生成反射子。我们建议读者研究算法 2 与算法 3 之间的差异。算法 4 使用重新排序后的反射子和块反射器应用其余更新。图 5 和图 6 展示了生成反射子时从左侧和右侧更新的矩阵束部分。



(a) A 扫描 1 后　(b) A 扫描 2 后　(c) A 扫描 3 后

(d) B 扫描 1 后　(e) B 扫描 2 后　(f) B 扫描 3 后

图 5：生成阶段从左侧更新的矩阵 A 和 B 的部分。浅灰色、灰色和深灰色部分分别由第 1、2 和 3 次扫描中的反射子更新。注意，在每次扫描中，我们都会将前几次扫描中的反射子应用到多一列上。这无法更早进行，因为那样会干扰从右侧进行的更新。



(a) A第 1 次扫描后　(b) A 第 2 次扫描后　(c) A 第 3 次扫描后

(d) B 第 1 次扫描后　(e) B 第 2 次扫描后　(f) B 扫描 3 之后

图 6：生成阶段从右侧更新的矩阵 A 和 B 的部分。浅灰色、灰色和深灰色部分分别表示由扫描 1、2 和 3 中的反射子更新的部分。请注意，与左侧更新相比，矩阵中被更新的部分要大得多。

3.3 第二阶段并行化

第二阶段的并行化策略遵循与第一阶段相同的基本原则。我们将反射子的生成与对不同矩阵的更新拆分为不同的任务。这里的特殊之处在于，我们还将应用任务拆分为前瞻任务和较小的应用任务。在这些前瞻任务中，我们只需对 A 和 B 进行适量更新，便可确保生成反射子所需的 $O(rq)$ 带得到完整更新。这样一来，矩阵更新便可与反射子生成重叠执行。这一点很重要，因为反射子的生成既不可忽略，也不易并行化。图 7 展示了不同迭代的任务之间的依赖关系。

接下来，我们按照第一阶段中的相同方式并行化大型任务。对于前瞻任务和更新任务，我们都将矩阵拆分为列切片或行切片，并并行执行更新。图 8 展示了这种分布。

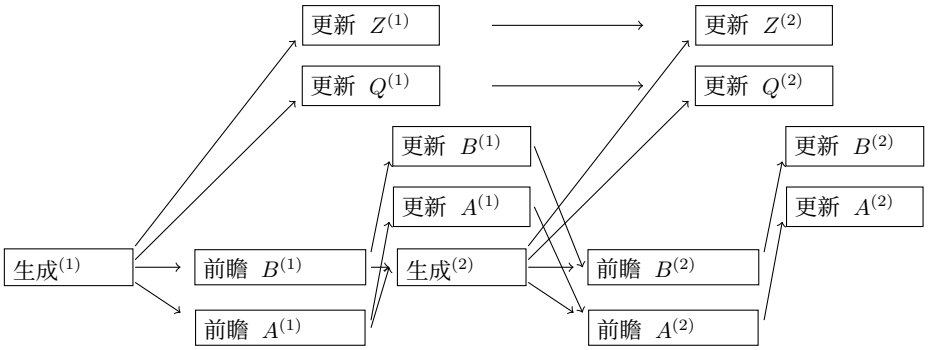


图 7：并行阶段二的依赖图。上标表示任务所属的迭代。

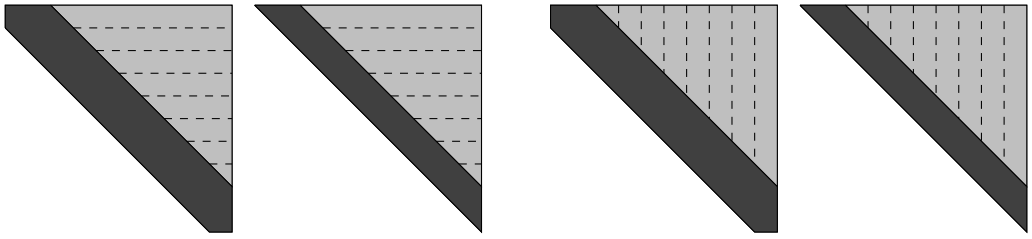
4 实验

我们比较以下实现

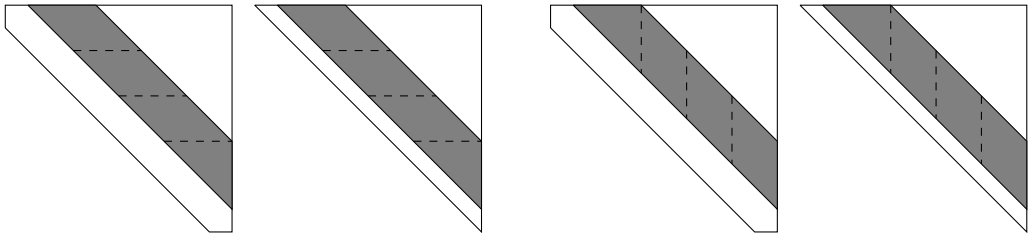
- ParaHT：本文提出的算法
- IterHT：T. Steel 和 R. Vandebril 提出的迭代约化算法 [11]
- DGGHD3：K˚agstr¨om 等人提出的分块海森堡-三角约化算法 [7]，如 LAPACK 3.9 [2] 中的实现。
- HouseHT：基于豪斯霍尔德反射器的约化方法，作者为 Bujanovic、L. Karlsson 和 D. Kress-ner [5]。

测试在一台配备两颗 Intel Xeon E-2697v3 CPU（每颗 14 个核心）、共享 128GB 内存的机器上进行。该 CPU 的单个节点理论双精度峰值性能为 20.8 Gflops。

ParaHT、IterHT 和 DGGHD3 使用 Fortran 编写，并使用 GNU Fortran 7.5.0 版和优化标志 -O3 -march=native 编译。HouseHT 使用 C++ 编写，并使用 GNU C 编译器 7.5.0 版和优化标志 -O3 -march=native 编译。代码与 MKL 2019.0.1 版链接。



(a) 从右侧进行完整更新；(b) 从左侧进行完整更新并生成反射子



(c) 从右侧进行前瞻更新　　(d) 从左侧进行前瞻更新

图 8：阶段 2 的更新、前瞻和生成任务划分为多个子任务的情况。深灰色表示生成任务，该任务无法有效并行化，因此不会拆分为子任务。浅灰色表示完整更新任务。这些任务包含算法中大部分浮点运算次数，因此拆分成最多的子任务。灰色表示前瞻更新。这些任务涉及的浮点运算次数较少，因此拆分成较大的块。

对于 HouseHT，参数 n_b 设为 64。对于 ParaHT，参数 r、p 和 q 分别设为 16、8 和 8。这些参数通过在随机生成的不同规模矩阵束上运行算法进行调优，并选择平均表现最佳的一组参数。

实验将重点关注算法的运行时间。我们不会详细讨论这些方法的精度。我们测试的每种算法始终都能产生结果，其相对后向误差处于机器精度的量级。

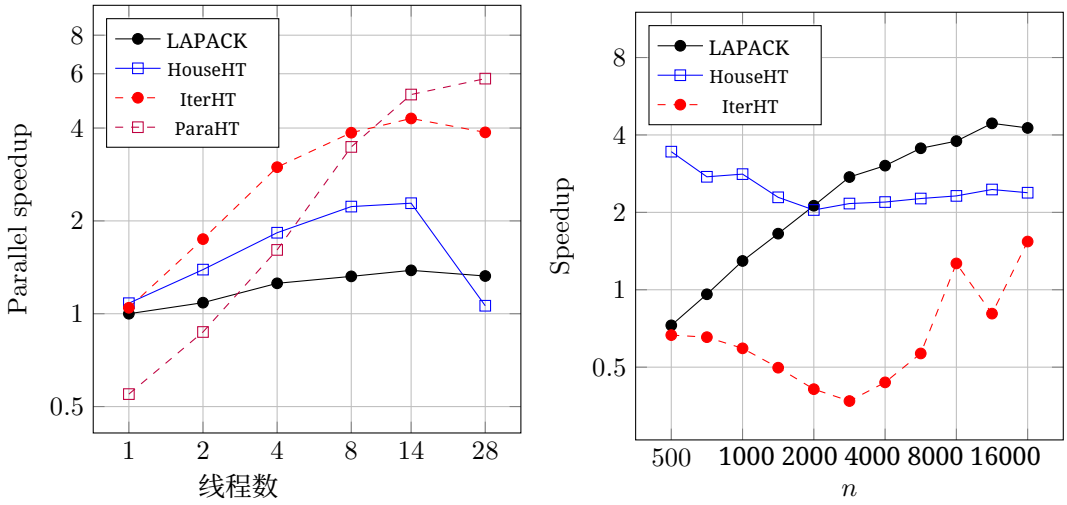
随机矩阵束测试 为评估我们算法的整体性能，我们对不同尺寸的随机生成矩阵进行测试，并将其与其他实现进行比较。生成随机矩阵束后，我们对 B 进行 QR分解，使其成为上三角矩阵。请注意，随机矩阵通常是良态的，这对我们进行比较的两种算法很重要。IterHT，以及在较小程度上 HouseHT，都是迭代算法；如果矩阵束含有许多无穷特征值，就需要进行更多迭代。矩阵 B 条件数较小的矩阵束可以避免这些额外迭代。

图 9a 展示了在不同线程数下，几种算法相对于 LAPACK 顺序运行时间的运行时间。由于浮点运算次数较多，我们的算法在单核上明显慢于其他算法。随着线程数增加，有效的并行化弥补了增加的开销。我们还注意到，HouseHT 和 IterHT 在使用 14 个线程时达到最高并行加速比。在其他未明确改变线程数的测试中，我们将 HouseHT 和 IterHT 的线程数限制为 14，以便进行公平比较。

图 9b 展示了我们的算法相较于 LAPACK、HouseHT 和 IterHT 所实现的加速效果。

相较于 HouseHT，我们的运行速度提升了 2 倍。与 LAPACK 相比，我们的算法在处理小型矩阵时稍慢，但在处理大型矩阵时可实现 4 倍加速。大多数情况下，IterHT 的速度都快于我们的算法。如前所述，IterHT 的运行时间很大程度上取决于矩阵束中无穷特征值的数量以及 B 的条件。大多数测试中，由于 B 条件良好，只需进行一次迭代。对于 $n = 8000$ 和 $n = 16000$，则需要进行两次迭代；在这些情况下，我们的算法略快于 IterHT。

图 10 更详细地展示了 ParaHT 的并行性能。首先请注意，尽管阶段 1 所需的浮点运算次数略多，但算法的大部分运行时间都花在阶段 2 中。这表明阶段 2 的效率较低。其次，阶段 1 和阶段 2（进而整个算法）的并行加速比非常相似。导致阶段 2 在单核上效率较低的问题，在并行执行这些阶段时依然存在。最后，我们还可以看到，矩阵必须足够大才能获得良好的加速比。对于 $n = 1000$，我们的算法只能达到约 2 的并行加速比；而对于 $n = 8000$，则能达到 10。



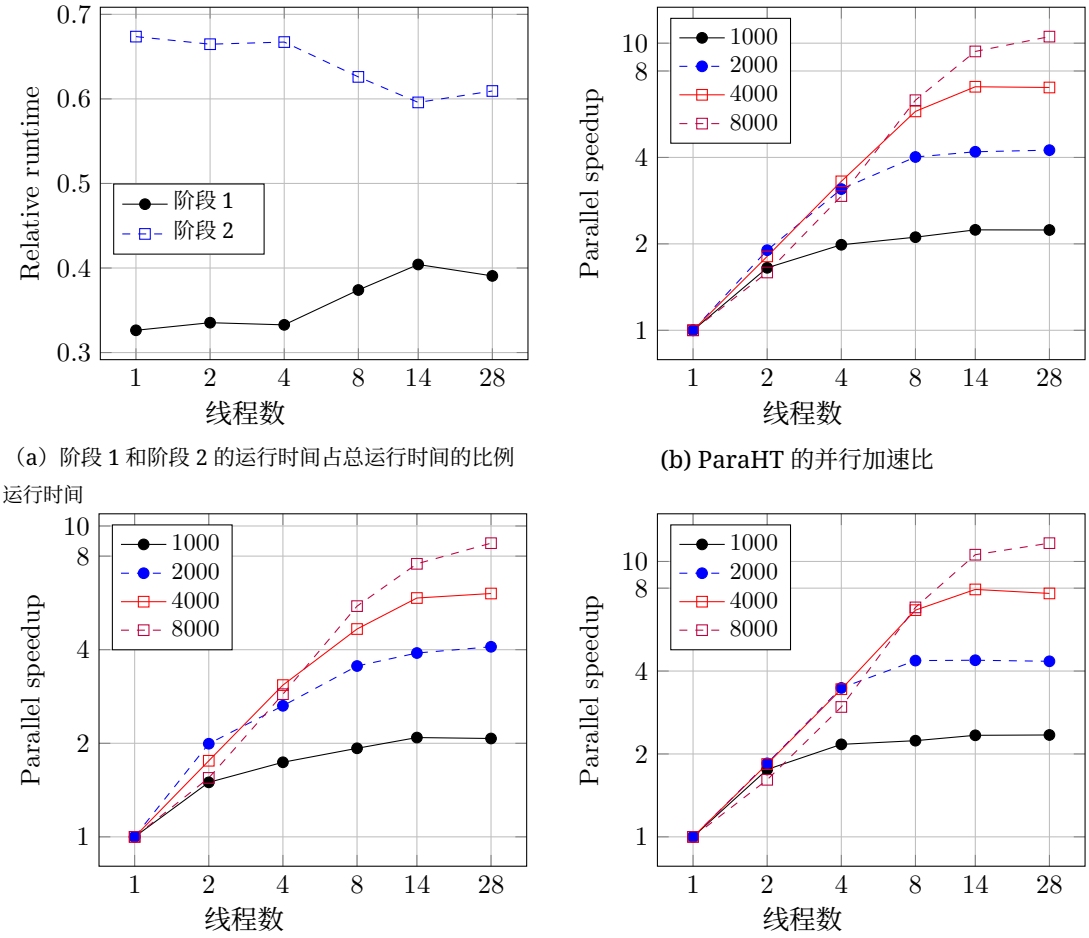
（a）并行加速比　　（b）不同 n 值下的加速比

图 9：左图为随机生成的 8000 阶矩阵束的并行加速比（LAPACK 单线程运行时间与相关算法多线程运行时间之比）。右图为不同矩阵束规模下 ParaHT 相较于其他算法所取得的加速比。

鞍点问题测试 按照 Bujanovic 等人 [5], 的做法，我们也在鞍点问题上测试了我们的算法。这些矩阵束具有如下形式：

$$(A, B) = \left(\begin{bmatrix} X & Y \\ Y^* & 0 \end{bmatrix}, \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} \right),$$

其中，I 是单位矩阵，Y 是随机矩阵，X 是随机对称正定矩阵。通过选择 X 和单位矩阵的维数，我们可以控制矩阵束中无穷特征值的数量。在我们的实验中，维数的选取使得矩阵束中有 25% 的特征值为无穷特征值。这样的矩阵束对 HouseHT 和 IterHT 尤其棘手。



（a）阶段 1 和阶段 2 的运行时间占总运行时间的比例

(b) ParaHT 的并行加速比

(c) 阶段 1 的并行加速比

(d) 阶段 2 的并行加速比

图 10：我们的算法及其各阶段的并行加速比和相对运行时间。

我们注意到，尽管实际中存在含有大量无穷特征值的矩阵束，但通常可以在海森堡-三角约化之前消去这些特征值。LAPACK 可以根据矩阵束的稀疏结构检测无穷特征值；而我们在先前论文 [11] 中提出的另一种消去技术，可以消去并非严格无穷、但在有限精度下可视为无穷的特征值。

图11展示了 ParaHT 在不同规模鞍点问题上的加速比。正如预期，ParaHT 相对于 LAPACK 的加速比没有显著变化，因为 LAPACK 和 ParaHT 的运行时间不取决于无穷特征值的数量。ParaHT 相对于 HouseHT 的加速比则比之前大得多，这是因为 HouseHT 必须进行更多迭代改进，才能处理病态矩阵。IterHT 对大量无穷特征值的存在更加敏感，并且无法收敛。

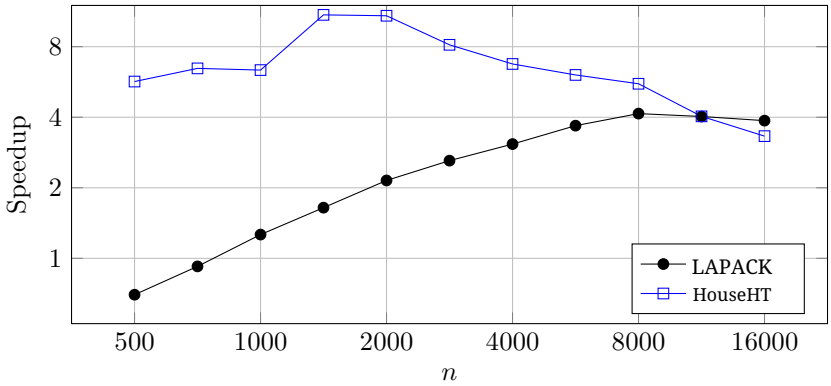


图11：对于不同规模的鞍点铅笔，ParaHT 相较于其他算法所实现的加速比。未列出 IterHT，因为它在 10 次迭代改进迭代内未能收敛。

5 结论

我们提出了一种并行算法，用于
海森堡-三角约化。该算法分为两个阶段：先约化为 r-Hessenberg-三角形式，再约化为 Hessenberg-三角形式。两阶段约化通常比单阶段约化需要更多运算，但更易于并行化，我们提出的新方案也不例外。在共享内存环境下进行的实验表明，该算法具有良好的精度，并且借助并行基本线性代数子程序，其性能可优于当前最先进的串行算法。未来的工作需要确定该算法是否也能胜过这些算法的并行版本。此外，当矩阵束中存在无穷特征值时，该算法不会因此受到性能惩罚。

致谢

感谢 Bujanovic、Karlsson 和 Kressner 提供 HouseHT 的实现，供我们进行比较。

参考文献

[1] Bj¨orn Adlerborn、Lars Karlsson 和 Bo K˚agstr¨om。采用波前调度的分布式单阶段 Hessenberg-三角约化。《SIAM科学计算杂志》，40(2):C157–C180，2018。[2] E. Anderson、Z. Bai、C. Bischof、S. Blackford、J. Demmel、J. Dongarra、J. Du Croz、A. Greenbaum、S. Hammarling、A. McKenney 和 D. Sorensen。《LAPACK用户指南》。工业与应用数学学会，费城，宾夕法尼亚州，第3版，1999。[3] Christian Bischof、Xiaobai Sun 和 Bruno Lang。通过两步带状约化实现并行三对角化。载于《IEEE可扩展高性能计算会议论文集》，第23–27页。IEEE，1994。[4] Christian Bischof 和 Charles Van Loan。Householder矩阵乘积的WY表示。《SIAM科学与统计计算杂志》，8(1):s2–s13，1987。[5] Zvonimir Bujanovic、Lars Karlsson 和 Daniel Kressner。一种用于海森堡-三角约化的基于Householder变换的算法。《SIAM矩阵分析与应用杂志》，39(3):1270–1294，2018。[6] Krister Dackland 和 Bo K˚agstr¨om。一种将正则矩阵对约化为块Hessenberg-三角形式的ScaLAPACK风格算法。载于《应用并行计算国际研讨会》，第95–103页。施普林格，1998。[7] Bo K˚agstr¨om、Daniel Kressner、ES Quintana-Ort´ı 和 Gregorio Quintana-Orti。重新探讨约化为Hessenberg-三角形式的分块算法。《BIT数值数学》，48(3):563–584，2008。[8] Lars Karlsson 和 Bo K˚agstr¨om。利用共享内存将块Hessenberg形式高效约化为海森堡形式。载于《应用并行计算国际研讨会》，第258–268页。施普林格，2010。[9] Lars Karlsson 和 Bo K˚agstr¨om。在共享内存架构上使用动态调度并行约化为海森堡形式。《并行计算》，37(12):771–782，2011。[10] C. B. Moler 和 G. W. Stewart。一种求解广义矩阵特征值问题的算法。《SIAM数值分析杂志》，10(2):241–256，1973。[11] Thijs Steel 和 Raf Vandebril。一种新颖的分块算法，用于约化为Hessenberg-三角形式。《线性代数电子期刊》，已接受发表，2023。[12] David S Watkins。存在无穷特征值时QZ 算法的性能。《SIAM矩阵分析与应用杂志》，22(2):364–375，2000。

附录

算法1：分块约化为 r-海森堡-三角形式 [6]

输入： $n \times n$ 矩阵束 (A, B)，其中 B 为上三角

输出：分解 $(A_{orig}, B_{orig}) = Q(A, B)Z^*$，其中 (A, B) 为 r-海森堡-三角形式 form

$Q = I$

$Z = I$

对于 $j = 1 : n_b : n - 2$ 执行

  $j_1 = j$

  $j_2 = $ 最小值$(n, j + n_b - 1)$

  $n_{blocks} = \lceil (n - nb - j + 1)/((p - 1)n_b) \rceil$

  对于 $k = n_{blocks} - 1 : -1 : 0$ 执行

    $i_1 = j + n_b + k(p - 1)n_b$

    $i_2 = $ 最小值$(n, i_1 + p * n_b - 1)$

    $A(i_1 : i_2, j_1 : j_2) = \hat{Q}_k^j \hat{R}$ {QR分解}

    $A(i_1 : i_2, j_1 : j_2) = \hat{R}$

    $A(i_1 : i_2, j_2 + 1 : n) = \hat{Q}_k^{j*} A(i_1 : i_2, j_2 + 1 : n)$

    $B(i_1 : i_2, i_1 : n) = \hat{Q}_k^{j*} B(i_1 : i_2, i_1 : n)$

    $Q(1 : n, i_1 : i_2) = Q(1 : n, i_1 : i_2) \hat{Q}_k^j$

  结束循环

  对 $i = j + n_b + (n_{blocks} - 1)(p - 1)n_b : -(p - 1)n_b : j + n_b$ 执行

    $i_1 = i$

    $i_2 = \min(n, i + p * n_b - 1)$

    $B(i_1 : i_2, i_1 : i_2) = \tilde{R}\tilde{Q}$ {RQ分解}

    $\tilde{Q}(1 : \min(n_b, i_2 - i_1 + 1), 1 : i_2 - i_1 + 1) = L \hat{Z}$ {LQ分解}

    $A(1 : n, i_1 : i_2) = A(1 : n, i_1 : i_2) \hat{Z}$

    $B(1 : i_2, i_1 : i_2) = B(1 : i_2, i_1 : i_2) \hat{Z}$

    $Z(1 : n, i_1 : i_2) = Z(1 : n, i_1 : i_2) \hat{Z}$

  结束循环

结束循环

算法 2：无分块地约化为 Hessenberg-三角形式

输入：处于 r-海森堡-三角形式的 $n \times n$ 矩阵束（A, B）

输出：分解（A_{orig}, B_{orig}）$= Q$（A, B）Z^*，其中（A, B）处于 Hessenberg-三角形式 form

$Q = I$

$Z = I$

对 $j = 1 : n - 2$ 执行

  $n_{blocks} = 1 + \lfloor \frac{n-j-2}{r} \rfloor$

  对于 $k = 0 : n_{blocks} - 1$ 执行

    $j_b = j + \max(0, (k-1)r + 1)$

    $i_1 = j + kr + 1$

    $i_2 = \min(j + (k+1)r, n)$

    $i_3 = \min(j + (k+2)r, n)$

    生成一个反射子 $\hat{}Q_k^j = I - \tau_l v_l v_l^*$，以化简 $A(i_1 : i_2, j_b)$

    $A(i_1 : i_2, j_b : n) = \hat{}Q_k^j A(i_1 : i_2, j_b : n)$

    $B(i_1 : i_2, i_1 : n) = \hat{}Q_k^j B(i_1 : i_2, i_1 : n)$

    $Q(: i_1 : i_2) = Q(: i_1 : i_2) \hat{}Q_k^j$

    $B(i_1 : i_2, i_1 : i_2) = \tilde{R}\tilde{Q}$ {RQ分解}

    生成一个反射子 $\hat{}Z_k^j = I - \tau_r v_r v_r^*$，将 $\tilde{Q}(1, 1 : i_2 - i_1 + 1)$ 化简为

    $A(1 : i_3, i_1 : i_2) = A(1 : i_3, i_1 : i_2) \hat{}Z_k^j$

    $B(1 : i_2, i_1 : i_2) = B(1 : i_2, i_1 : i_2) \hat{}Z_k^j$

    $Z(: i_1 : i_2) = Z(: i_1 : i_2) \hat{}Z_k^j$

  结束循环

结束循环

算法 3：第二阶段的生成阶段

输入：处于 r-海森堡-三角形式的矩阵束 (A, B)，其中 A 在以下列中处于海森堡形式列 $1: j_1 - 1$。

对于 $j = j_1 : j_1 + q - 1$ 执行
  $n_{blocks} = 2 + \lfloor \frac{n-j-1}{r} \rfloor$
  对于 $k = 0 : n_{blocks} - 1$ 执行
    $j_b = j + \max(0, (k-1)r + 1)$
    $i_1 = j + kr + 1$
    $i_2 = \min(j + (k+1)r, n)$
    $i_3 = \min(j + (k+2)r, n)$
    $i_4 = j_1 + 1 + \max(0, (k + j - j_1 - q + 2)r)$
    for $\hat{j} = j_1 : j - 1$ 执行
      $\hat{i_1} = \hat{j} + kr + 1$
      $\hat{i_2} = \min(\hat{j} + (k+1)r, n)$
      if $\hat{i_2} - \hat{i_1} \geq 1$ 然后
        $A(\hat{i_1} : \hat{i_2}, j_b) = \hat{Q}_k^{\hat{j}} A(\hat{i_1} : \hat{i_2}, j_b)$
        如果 $i_1 + r - 1 \leq n$ 那么
          $B(\hat{i_1} : \hat{i_2}, i_1 + r - 1) = \hat{Q}_k^{\hat{j}} B(\hat{i_1} : \hat{i_2}, i_1 + r - 1)$
        结束如果
      结束如果
    结束循环
    生成一个反射子 $\hat{Q}_k^j = I - \tau_l v_l v_l^*$，用于约化 $A(i_1 : i_2, j_b)$
    $A(i_1 : i_2, j_b) = \hat{Q}_k^j A(i_1 : i_2, j_b)$
    $B(i_1 : i_2, i_1 : i_2) = \hat{Q}_k^j B(i_1 : i_2, i_1 : i_2)$
    $B(i_1 : i_2, \ i_1 : i_2) = \tilde{R}\tilde{Q}$ {RQ分解}
    生成一个反射子 $\hat{Z}_k^j = I - \tau_r v_r v_r^*$，将 $\tilde{Q}(1, 1 : \ i_2 - i_1 + 1)$ 化简为
    $A(i_4 : \ i_3, \ i_1 : \ i_2) = A(i_4 : \ i_3, \ i_1 : \ i_2)\hat{Z}_k^j$
    $B(i_4 : \ i_2, \ i_1 : \ i_2) = B(i_4 : \ i_2, \ i_1 : \ i_2)\hat{Z}_k^j$
  结束循环
结束循环

算法 4：第二阶段的应用阶段

输入：处于 r-海森堡-三角形式的矩阵束 (A, B)，其中 A 在以下范围内呈海森堡形式列 $1:\ j_1 - 1$。

$n_{blocks} = 1 + \lfloor \frac{n-j_1-2}{r} \rfloor$

对于 $k = n_{blocks} - 1 :\ -1 : 0$，执行

  $i_5 = j_1 + 1 + \max(0, (k-q+2)r)$

  对于 $j = j_1 + 1 :\ j_1 + q - 1$ 执行

    $i_1 = j + kr + 1$

    $i_2 = \min(j + (k+1)r, n)$

    $i_4 = j_1 + $ 最大值$(0, (k + j - j_1 - q + 2)r)$

    $A(i_5 :\ i_4, i_1 :\ i_2) = A(i_5 :\ i_4, i_1 :\ i_2)\ \hat{}\ Z_k^j$

    $B(i_5 :\ i_4, i_1 :\ i_2) = B(i_5 :\ i_4, i_1 :\ i_2)\ \hat{}\ Z_k^j$

  结束循环

  $\hat{}Z_k = \hat{}Z_k^{j_1}\hat{}Z_k^{j_2} \ldots \hat{}Z_k^{j_1+q-1}$ {形成紧凑 WY}

  $i_1 = j_1 + kr + 1$

  $i_2 = \min(j_1 + q - 1 + (k+1)r, n)$

  $i_5 = j_1 + $ 最大值$(0, (k-q+2)r)$

  $A(1 :\ i_5, i_1 :\ i_2) = A(1 :\ i_5, i_1 :\ i_2)\hat{Z}_k$

  $B(1 :\ i_5, i_1 :\ i_2) = B(1 :\ i_5, i_1 :\ i_2)\hat{Z}_k$

  $Z(:\ i_1 :\ i_2) = Z(:\ i_1 :\ i_2)\hat{Z}_k$

</style}结束循环

对于 $k = n_{blocks} - 1 :\ -1 : 0$，执行

  $\hat{}Q_k = \hat{}Q_k^{j_1}\hat{}Q_k^{j_2} \ldots \hat{}Q_k^{j_1+q-1}$ {形成紧凑 WY}

  $i_1 = j_1 + kr + 1$

  $i_2 = \min(j_1 + q - 1 + (k+1)r, n)$

  $i_5 = j_1 + q - 1 + \max(0, (k-1)r + 1)$

  $i_6 = j_1 + q + (k+1)r - 1$

  $A(i_1 :\ i_2, i_5 :\ n) = \hat{Q}_k^* A(i_1 :\ i_2,\ \ i_5 :\ n)$

  $B(i_1 :\ i_2, i_6 :\ n) = \hat{Q}_k^* B(i_1 :\ i_2,\ \ i_6 :\ n)$

  $Q(:\ i_1 :\ i_2) = Q(:\ i_1 :\ i_2)\hat{Q}_k$

结束于