LCOV - code coverage report

Current view:	top level - src/plugins/intel_cpu/src/nodes/kernels/scaled_attn - transpose_kernel.hpp (source / functions)	Coverage	
Test:	full_covergae.info	**Lines:**	47.2 %
Test Date:	2025-04-10 19:26:12	**Functions:**	28.6 %

```
      Line data    Source code
    1             : // Copyright (C) 2018-2025 Intel Corporation
    2             : // SPDX-License-Identifier: Apache-2.0
    3             : //
    4             : #pragma once
    5             :
    6             : #include <array>
    7             : #include <cstddef>
    8             : #include <cstdint>
    9             : #include <vector>
   10             :
   11             : #include "common.hpp"
   12             : #include "openvino/core/type/element_type.hpp"
   13             :
   14             : #if defined(HAVE_SVE)
   15             : #    include "arm_sve.h"
   16             : #endif
   17             :
   18             : namespace ov::Extensions::Cpu::XARCH {
   19             :
   20             : #if defined(HAVE_AVX512F)
   21             : inline void transpose_m512i_16x16(__m512i& r0,
   22             :                                   __m512i& r1,
   23             :                                   __m512i& r2,
   24             :                                   __m512i& r3,
   25             :                                   __m512i& r4,
   26             :                                   __m512i& r5,
   27             :                                   __m512i& r6,
   28             :                                   __m512i& r7,
   29             :                                   __m512i& r8,
   30             :                                   __m512i& r9,
   31             :                                   __m512i& ra,
   32             :                                   __m512i& rb,
   33             :                                   __m512i& rc,
   34             :                                   __m512i& rd,
   35             :                                   __m512i& re,
   36             :                                   __m512i& rf) {
   37             :     __m512i t0, t1, t2, t3, t4, t5, t6, t7, t8, t9, ta, tb, tc, td, te, tf;
   38             :
   39             :     t0 = _mm512_unpacklo_epi32(r0, r1);  //   0  16   1  17   4  20   5  21   8  24   9  25  12  28  13  29
   40             :     t1 = _mm512_unpackhi_epi32(r0, r1);  //   2  18   3  19   6  22   7  23  10  26  11  27  14  30  15  31
   41             :     t2 = _mm512_unpacklo_epi32(r2, r3);  //  32  48  33  49 ...
   42             :     t3 = _mm512_unpackhi_epi32(r2, r3);  //  34  50  35  51 ...
   43             :     t4 = _mm512_unpacklo_epi32(r4, r5);  //  64  80  65  81 ...
   44             :     t5 = _mm512_unpackhi_epi32(r4, r5);  //  66  82  67  83 ...
   45             :     t6 = _mm512_unpacklo_epi32(r6, r7);  //  96 112  97 113 ...
   46             :     t7 = _mm512_unpackhi_epi32(r6, r7);  //  98 114  99 115 ...
   47             :     t8 = _mm512_unpacklo_epi32(r8, r9);  // 128 ...
   48             :     t9 = _mm512_unpackhi_epi32(r8, r9);  // 130 ...
   49             :     ta = _mm512_unpacklo_epi32(ra, rb);  // 160 ...
   50             :     tb = _mm512_unpackhi_epi32(ra, rb);  // 162 ...
   51             :     tc = _mm512_unpacklo_epi32(rc, rd);  // 196 ...
   52             :     td = _mm512_unpackhi_epi32(rc, rd);  // 198 ...
   53             :     te = _mm512_unpacklo_epi32(re, rf);  // 228 ...
   54             :     tf = _mm512_unpackhi_epi32(re, rf);  // 230 ...
   55             :
   56             :     r0 = _mm512_unpacklo_epi64(t0, t2);  //   0  16  32  48 ...
   57             :     r1 = _mm512_unpackhi_epi64(t0, t2);  //   1  17  33  49 ...
   58             :     r2 = _mm512_unpacklo_epi64(t1, t3);  //   2  18  34  49 ...
   59             :     r3 = _mm512_unpackhi_epi64(t1, t3);  //   3  19  35  51 ...
   60             :     r4 = _mm512_unpacklo_epi64(t4, t6);  //  64  80  96 112 ...
   61             :     r5 = _mm512_unpackhi_epi64(t4, t6);  //  65  81  97 114 ...
   62             :     r6 = _mm512_unpacklo_epi64(t5, t7);  //  66  82  98 113 ...
   63             :     r7 = _mm512_unpackhi_epi64(t5, t7);  //  67  83  99 115 ...
   64             :     r8 = _mm512_unpacklo_epi64(t8, ta);  // 128 144 160 176 ...
   65             :     r9 = _mm512_unpackhi_epi64(t8, ta);  // 129 145 161 178 ...
   66             :     ra = _mm512_unpacklo_epi64(t9, tb);  // 130 146 162 177 ...
   67             :     rb = _mm512_unpackhi_epi64(t9, tb);  // 131 147 163 179 ...
   68             :     rc = _mm512_unpacklo_epi64(tc, te);  // 192 208 228 240 ...
   69             :     rd = _mm512_unpackhi_epi64(tc, te);  // 193 209 229 241 ...
   70             :     re = _mm512_unpacklo_epi64(td, tf);  // 194 210 230 242 ...
   71             :     rf = _mm512_unpackhi_epi64(td, tf);  // 195 211 231 243 ...
   72             :
   73             :     t0 = _mm512_shuffle_i32x4(r0, r4, 0x88);  //   0  16  32  48   8  24  40  56  64  80  96  112 ...
   74             :     t1 = _mm512_shuffle_i32x4(r1, r5, 0x88);  //   1  17  33  49 ...
   75             :     t2 = _mm512_shuffle_i32x4(r2, r6, 0x88);  //   2  18  34  50 ...
   76             :     t3 = _mm512_shuffle_i32x4(r3, r7, 0x88);  //   3  19  35  51 ...
   77             :     t4 = _mm512_shuffle_i32x4(r0, r4, 0xdd);  //   4  20  36  52 ...
   78             :     t5 = _mm512_shuffle_i32x4(r1, r5, 0xdd);  //   5  21  37  53 ...
   79             :     t6 = _mm512_shuffle_i32x4(r2, r6, 0xdd);  //   6  22  38  54 ...
   80             :     t7 = _mm512_shuffle_i32x4(r3, r7, 0xdd);  //   7  23  39  55 ...
   81             :     t8 = _mm512_shuffle_i32x4(r8, rc, 0x88);  // 128 144 160 176 ...
   82             :     t9 = _mm512_shuffle_i32x4(r9, rd, 0x88);  // 129 145 161 177 ...
   83             :     ta = _mm512_shuffle_i32x4(ra, re, 0x88);  // 130 146 162 178 ...
   84             :     tb = _mm512_shuffle_i32x4(rb, rf, 0x88);  // 131 147 163 179 ...
   85             :     tc = _mm512_shuffle_i32x4(r8, rc, 0xdd);  // 132 148 164 180 ...
   86             :     td = _mm512_shuffle_i32x4(r9, rd, 0xdd);  // 133 149 165 181 ...
   87             :     te = _mm512_shuffle_i32x4(ra, re, 0xdd);  // 134 150 166 182 ...
   88             :     tf = _mm512_shuffle_i32x4(rb, rf, 0xdd);  // 135 151 167 183 ...
   89             :
   90             :     r0 = _mm512_shuffle_i32x4(t0, t8, 0x88);  //   0  16  32  48  64  80  96 112 ... 240
   91             :     r1 = _mm512_shuffle_i32x4(t1, t9, 0x88);  //   1  17  33  49  66  81  97 113 ... 241
   92             :     r2 = _mm512_shuffle_i32x4(t2, ta, 0x88);  //   2  18  34  50  67  82  98 114 ... 242
   93             :     r3 = _mm512_shuffle_i32x4(t3, tb, 0x88);  //   3  19  35  51  68  83  99 115 ... 243
```

```
94              :     r4 = _mm512_shuffle_i32x4(t4, tc, 0x88);  //   4 ...
95              :     r5 = _mm512_shuffle_i32x4(t5, td, 0x88);  //   5 ...
96              :     r6 = _mm512_shuffle_i32x4(t6, te, 0x88);  //   6 ...
97              :     r7 = _mm512_shuffle_i32x4(t7, tf, 0x88);  //   7 ...
98              :     r8 = _mm512_shuffle_i32x4(t0, t8, 0xdd);  //   8 ...
99              :     r9 = _mm512_shuffle_i32x4(t1, t9, 0xdd);  //   9 ...
100             :     ra = _mm512_shuffle_i32x4(t2, ta, 0xdd);  //  10 ...
101             :     rb = _mm512_shuffle_i32x4(t3, tb, 0xdd);  //  11 ...
102             :     rc = _mm512_shuffle_i32x4(t4, tc, 0xdd);  //  12 ...
103             :     rd = _mm512_shuffle_i32x4(t5, td, 0xdd);  //  13 ...
104             :     re = _mm512_shuffle_i32x4(t6, te, 0xdd);  //  14 ...
105             :     rf = _mm512_shuffle_i32x4(t7, tf, 0xdd);  //  15  31  47  63  79  96 111 127 ... 255
106             : }
107             :
108             : template <typename T>
109             : inline void transpose_16x16_kernel(float* _dst, T* src, size_t dst_stride, size_t src_stride) {
110             :     auto* dst = reinterpret_cast<uint32_t*>(_dst);
111             :     __m512i r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, ra, rb, rc, rd, re, rf;
112             :     r0 = _mm512_castps_si512(mm512_uni_loadu_ps(src));
113             :     r1 = _mm512_castps_si512(mm512_uni_loadu_ps(src + src_stride));
114             :     r2 = _mm512_castps_si512(mm512_uni_loadu_ps(src + 2 * src_stride));
115             :     r3 = _mm512_castps_si512(mm512_uni_loadu_ps(src + 3 * src_stride));
116             :     r4 = _mm512_castps_si512(mm512_uni_loadu_ps(src + 4 * src_stride));
117             :     r5 = _mm512_castps_si512(mm512_uni_loadu_ps(src + 5 * src_stride));
118             :     r6 = _mm512_castps_si512(mm512_uni_loadu_ps(src + 6 * src_stride));
119             :     r7 = _mm512_castps_si512(mm512_uni_loadu_ps(src + 7 * src_stride));
120             :     r8 = _mm512_castps_si512(mm512_uni_loadu_ps(src + 8 * src_stride));
121             :     r9 = _mm512_castps_si512(mm512_uni_loadu_ps(src + 9 * src_stride));
122             :     ra = _mm512_castps_si512(mm512_uni_loadu_ps(src + 10 * src_stride));
123             :     rb = _mm512_castps_si512(mm512_uni_loadu_ps(src + 11 * src_stride));
124             :     rc = _mm512_castps_si512(mm512_uni_loadu_ps(src + 12 * src_stride));
125             :     rd = _mm512_castps_si512(mm512_uni_loadu_ps(src + 13 * src_stride));
126             :     re = _mm512_castps_si512(mm512_uni_loadu_ps(src + 14 * src_stride));
127             :     rf = _mm512_castps_si512(mm512_uni_loadu_ps(src + 15 * src_stride));
128             :
129             :     transpose_m512i_16x16(r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, ra, rb, rc, rd, re, rf);
130             :
131             :     _mm512_storeu_si512(dst, r0);
132             :     _mm512_storeu_si512(dst + dst_stride, r1);
133             :     _mm512_storeu_si512(dst + 2 * dst_stride, r2);
134             :     _mm512_storeu_si512(dst + 3 * dst_stride, r3);
135             :     _mm512_storeu_si512(dst + 4 * dst_stride, r4);
136             :     _mm512_storeu_si512(dst + 5 * dst_stride, r5);
137             :     _mm512_storeu_si512(dst + 6 * dst_stride, r6);
138             :     _mm512_storeu_si512(dst + 7 * dst_stride, r7);
139             :     _mm512_storeu_si512(dst + 8 * dst_stride, r8);
140             :     _mm512_storeu_si512(dst + 9 * dst_stride, r9);
141             :     _mm512_storeu_si512(dst + 10 * dst_stride, ra);
142             :     _mm512_storeu_si512(dst + 11 * dst_stride, rb);
143             :     _mm512_storeu_si512(dst + 12 * dst_stride, rc);
144             :     _mm512_storeu_si512(dst + 13 * dst_stride, rd);
145             :     _mm512_storeu_si512(dst + 14 * dst_stride, re);
146             :     _mm512_storeu_si512(dst + 15 * dst_stride, rf);
147             : }
148             :
149             : template <typename T>
150             : inline void transpose_16xK_kernel(float* _dst, T* src, size_t K, size_t dst_stride, size_t src_stride) {
151             :     auto* dst = reinterpret_cast<uint32_t*>(_dst);
152             :     __m512i r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, ra, rb, rc, rd, re, rf;
153             :     r0 = _mm512_castps_si512(mm512_uni_loadu_tail_ps(src, K));
154             :     r1 = _mm512_castps_si512(mm512_uni_loadu_tail_ps(src + src_stride, K));
155             :     r2 = _mm512_castps_si512(mm512_uni_loadu_tail_ps(src + 2 * src_stride, K));
156             :     r3 = _mm512_castps_si512(mm512_uni_loadu_tail_ps(src + 3 * src_stride, K));
157             :     r4 = _mm512_castps_si512(mm512_uni_loadu_tail_ps(src + 4 * src_stride, K));
158             :     r5 = _mm512_castps_si512(mm512_uni_loadu_tail_ps(src + 5 * src_stride, K));
159             :     r6 = _mm512_castps_si512(mm512_uni_loadu_tail_ps(src + 6 * src_stride, K));
160             :     r7 = _mm512_castps_si512(mm512_uni_loadu_tail_ps(src + 7 * src_stride, K));
161             :     r8 = _mm512_castps_si512(mm512_uni_loadu_tail_ps(src + 8 * src_stride, K));
162             :     r9 = _mm512_castps_si512(mm512_uni_loadu_tail_ps(src + 9 * src_stride, K));
163             :     ra = _mm512_castps_si512(mm512_uni_loadu_tail_ps(src + 10 * src_stride, K));
164             :     rb = _mm512_castps_si512(mm512_uni_loadu_tail_ps(src + 11 * src_stride, K));
165             :     rc = _mm512_castps_si512(mm512_uni_loadu_tail_ps(src + 12 * src_stride, K));
166             :     rd = _mm512_castps_si512(mm512_uni_loadu_tail_ps(src + 13 * src_stride, K));
167             :     re = _mm512_castps_si512(mm512_uni_loadu_tail_ps(src + 14 * src_stride, K));
168             :     rf = _mm512_castps_si512(mm512_uni_loadu_tail_ps(src + 15 * src_stride, K));
169             :
170             :     transpose_m512i_16x16(r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, ra, rb, rc, rd, re, rf);
171             :
172             : #    define S(m) _mm512_storeu_si512(dst + 0x##m * dst_stride, r##m)
173             : #    define S8() \
174             :         S(0);    \
175             :         S(1);    \
176             :         S(2);    \
177             :         S(3);    \
178             :         S(4);    \
179             :         S(5);    \
180             :         S(6);    \
181             :         S(7);
182             :     switch (K) {
183             :     case 8:
184             :         S8();
185             :         break;
186             :     case 9:
187             :         S8() S(8);
188             :         break;
189             :     case 10:
190             :         S8();
191             :         S(8);
192             :         S(9);
193             :         break;
194             :     case 11:
195             :         S8();
196             :         S(8);
197             :         S(9);
```

```
198              :         S(a);
199              :         break;
200              :     case 12:
201              :         S8();
202              :         S(8);
203              :         S(9);
204              :         S(a);
205              :         S(b);
206              :         break;
207              :     case 13:
208              :         S8();
209              :         S(8);
210              :         S(9);
211              :         S(a);
212              :         S(b);
213              :         S(c);
214              :         break;
215              :     case 14:
216              :         S8();
217              :         S(8);
218              :         S(9);
219              :         S(a);
220              :         S(b);
221              :         S(c);
222              :         S(d);
223              :         break;
224              :     case 15:
225              :         S8();
226              :         S(8);
227              :         S(9);
228              :         S(a);
229              :         S(b);
230              :         S(c);
231              :         S(d);
232              :         S(e);
233              :         break;
234              :     case 1:
235              :         S(0);
236              :         break;
237              :     case 2:
238              :         S(0);
239              :         S(1);
240              :         break;
241              :     case 3:
242              :         S(0);
243              :         S(1);
244              :         S(2);
245              :         break;
246              :     case 4:
247              :         S(0);
248              :         S(1);
249              :         S(2);
250              :         S(3);
251              :         break;
252              :     case 5:
253              :         S(0);
254              :         S(1);
255              :         S(2);
256              :         S(3);
257              :         S(4);
258              :         break;
259              :     case 6:
260              :         S(0);
261              :         S(1);
262              :         S(2);
263              :         S(3);
264              :         S(4);
265              :         S(5);
266              :         break;
267              :     case 7:
268              :         S(0);
269              :         S(1);
270              :         S(2);
271              :         S(3);
272              :         S(4);
273              :         S(5);
274              :         S(6);
275              :         break;
276              :     }
277              : }
278              :
279              : inline void transpose_16x16_kernel(uint32_t* dst, uint32_t* src, size_t dst_stride, size_t src_stride) {
280              :     __m512i r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, ra, rb, rc, rd, re, rf;
281              :     r0 = _mm512_loadu_si512(src);
282              :     r1 = _mm512_loadu_si512(src + src_stride);
283              :     r2 = _mm512_loadu_si512(src + 2 * src_stride);
284              :     r3 = _mm512_loadu_si512(src + 3 * src_stride);
285              :     r4 = _mm512_loadu_si512(src + 4 * src_stride);
286              :     r5 = _mm512_loadu_si512(src + 5 * src_stride);
287              :     r6 = _mm512_loadu_si512(src + 6 * src_stride);
288              :     r7 = _mm512_loadu_si512(src + 7 * src_stride);
289              :     r8 = _mm512_loadu_si512(src + 8 * src_stride);
290              :     r9 = _mm512_loadu_si512(src + 9 * src_stride);
291              :     ra = _mm512_loadu_si512(src + 10 * src_stride);
292              :     rb = _mm512_loadu_si512(src + 11 * src_stride);
293              :     rc = _mm512_loadu_si512(src + 12 * src_stride);
294              :     rd = _mm512_loadu_si512(src + 13 * src_stride);
295              :     re = _mm512_loadu_si512(src + 14 * src_stride);
296              :     rf = _mm512_loadu_si512(src + 15 * src_stride);
297              :
298              :     transpose_m512i_16x16(r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, ra, rb, rc, rd, re, rf);
299              :
300              :     _mm512_storeu_si512(dst, r0);
301              :     _mm512_storeu_si512(dst + dst_stride, r1);
```

```
302              :     _mm512_storeu_si512(dst + 2 * dst_stride, r2);
303              :     _mm512_storeu_si512(dst + 3 * dst_stride, r3);
304              :     _mm512_storeu_si512(dst + 4 * dst_stride, r4);
305              :     _mm512_storeu_si512(dst + 5 * dst_stride, r5);
306              :     _mm512_storeu_si512(dst + 6 * dst_stride, r6);
307              :     _mm512_storeu_si512(dst + 7 * dst_stride, r7);
308              :     _mm512_storeu_si512(dst + 8 * dst_stride, r8);
309              :     _mm512_storeu_si512(dst + 9 * dst_stride, r9);
310              :     _mm512_storeu_si512(dst + 10 * dst_stride, ra);
311              :     _mm512_storeu_si512(dst + 11 * dst_stride, rb);
312              :     _mm512_storeu_si512(dst + 12 * dst_stride, rc);
313              :     _mm512_storeu_si512(dst + 13 * dst_stride, rd);
314              :     _mm512_storeu_si512(dst + 14 * dst_stride, re);
315              :     _mm512_storeu_si512(dst + 15 * dst_stride, rf);
316              : }
317              :
318              : inline void transpose_16xK_kernel(uint32_t* dst, uint32_t* src, size_t K, size_t dst_stride, size_t src_stride) {
319              :     __m512i r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, ra, rb, rc, rd, re, rf;
320              :     __mmask16 k = 0xffff >> (16 - K);
321              :
322              :     r0 = _mm512_maskz_loadu_epi32(k, src);
323              :     r1 = _mm512_maskz_loadu_epi32(k, src + src_stride);
324              :     r2 = _mm512_maskz_loadu_epi32(k, src + 2 * src_stride);
325              :     r3 = _mm512_maskz_loadu_epi32(k, src + 3 * src_stride);
326              :     r4 = _mm512_maskz_loadu_epi32(k, src + 4 * src_stride);
327              :     r5 = _mm512_maskz_loadu_epi32(k, src + 5 * src_stride);
328              :     r6 = _mm512_maskz_loadu_epi32(k, src + 6 * src_stride);
329              :     r7 = _mm512_maskz_loadu_epi32(k, src + 7 * src_stride);
330              :     r8 = _mm512_maskz_loadu_epi32(k, src + 8 * src_stride);
331              :     r9 = _mm512_maskz_loadu_epi32(k, src + 9 * src_stride);
332              :     ra = _mm512_maskz_loadu_epi32(k, src + 10 * src_stride);
333              :     rb = _mm512_maskz_loadu_epi32(k, src + 11 * src_stride);
334              :     rc = _mm512_maskz_loadu_epi32(k, src + 12 * src_stride);
335              :     rd = _mm512_maskz_loadu_epi32(k, src + 13 * src_stride);
336              :     re = _mm512_maskz_loadu_epi32(k, src + 14 * src_stride);
337              :     rf = _mm512_maskz_loadu_epi32(k, src + 15 * src_stride);
338              :
339              :     transpose_m512i_16x16(r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, ra, rb, rc, rd, re, rf);
340              :
341              :     switch (K) {
342              :     case 8:
343              :         S8();
344              :         break;
345              :     case 9:
346              :         S8() S(8);
347              :         break;
348              :     case 10:
349              :         S8();
350              :         S(8);
351              :         S(9);
352              :         break;
353              :     case 11:
354              :         S8();
355              :         S(8);
356              :         S(9);
357              :         S(a);
358              :         break;
359              :     case 12:
360              :         S8();
361              :         S(8);
362              :         S(9);
363              :         S(a);
364              :         S(b);
365              :         break;
366              :     case 13:
367              :         S8();
368              :         S(8);
369              :         S(9);
370              :         S(a);
371              :         S(b);
372              :         S(c);
373              :         break;
374              :     case 14:
375              :         S8();
376              :         S(8);
377              :         S(9);
378              :         S(a);
379              :         S(b);
380              :         S(c);
381              :         S(d);
382              :         break;
383              :     case 15:
384              :         S8();
385              :         S(8);
386              :         S(9);
387              :         S(a);
388              :         S(b);
389              :         S(c);
390              :         S(d);
391              :         S(e);
392              :         break;
393              :     case 1:
394              :         S(0);
395              :         break;
396              :     case 2:
397              :         S(0);
398              :         S(1);
399              :         break;
400              :     case 3:
401              :         S(0);
402              :         S(1);
403              :         S(2);
404              :         break;
405              :     case 4:
```

```
406              :         S(0);
407              :         S(1);
408              :         S(2);
409              :         S(3);
410              :         break;
411              :     case 5:
412              :         S(0);
413              :         S(1);
414              :         S(2);
415              :         S(3);
416              :         S(4);
417              :         break;
418              :     case 6:
419              :         S(0);
420              :         S(1);
421              :         S(2);
422              :         S(3);
423              :         S(4);
424              :         S(5);
425              :         break;
426              :     case 7:
427              :         S(0);
428              :         S(1);
429              :         S(2);
430              :         S(3);
431              :         S(4);
432              :         S(5);
433              :         S(6);
434              :         break;
435              :     }
436              : #    undef S
437              : #    undef S8
438              : }
439              :
440              : #elif defined(HAVE_AVX2)
441              :
442              : // https://stackoverflow.com/questions/25622745/transpose-an-8x8-float-using-avx-avx2
443              : inline void
444              : transpose_8x8(__m256& r0, __m256& r1, __m256& r2, __m256& r3, __m256& r4, __m256& r5, __m256& r6, __m256& r7) {
445              :     __m256 t0, t1, t2, t3, t4, t5, t6, t7;
446              :     __m256 tt0, tt1, tt2, tt3, tt4, tt5, tt6, tt7;
447              :     t0 = _mm256_unpacklo_ps(r0, r1);
448              :     t1 = _mm256_unpackhi_ps(r0, r1);
449              :     t2 = _mm256_unpacklo_ps(r2, r3);
450              :     t3 = _mm256_unpackhi_ps(r2, r3);
451              :     t4 = _mm256_unpacklo_ps(r4, r5);
452              :     t5 = _mm256_unpackhi_ps(r4, r5);
453              :     t6 = _mm256_unpacklo_ps(r6, r7);
454              :     t7 = _mm256_unpackhi_ps(r6, r7);
455              :     tt0 = _mm256_shuffle_ps(t0, t2, _MM_SHUFFLE(1, 0, 1, 0));
456              :     tt1 = _mm256_shuffle_ps(t0, t2, _MM_SHUFFLE(3, 2, 3, 2));
457              :     tt2 = _mm256_shuffle_ps(t1, t3, _MM_SHUFFLE(1, 0, 1, 0));
458              :     tt3 = _mm256_shuffle_ps(t1, t3, _MM_SHUFFLE(3, 2, 3, 2));
459              :     tt4 = _mm256_shuffle_ps(t4, t6, _MM_SHUFFLE(1, 0, 1, 0));
460              :     tt5 = _mm256_shuffle_ps(t4, t6, _MM_SHUFFLE(3, 2, 3, 2));
461              :     tt6 = _mm256_shuffle_ps(t5, t7, _MM_SHUFFLE(1, 0, 1, 0));
462              :     tt7 = _mm256_shuffle_ps(t5, t7, _MM_SHUFFLE(3, 2, 3, 2));
463              :     r0 = _mm256_permute2f128_ps(tt0, tt4, 0x20);
464              :     r1 = _mm256_permute2f128_ps(tt1, tt5, 0x20);
465              :     r2 = _mm256_permute2f128_ps(tt2, tt6, 0x20);
466              :     r3 = _mm256_permute2f128_ps(tt3, tt7, 0x20);
467              :     r4 = _mm256_permute2f128_ps(tt0, tt4, 0x31);
468              :     r5 = _mm256_permute2f128_ps(tt1, tt5, 0x31);
469              :     r6 = _mm256_permute2f128_ps(tt2, tt6, 0x31);
470              :     r7 = _mm256_permute2f128_ps(tt3, tt7, 0x31);
471              : }
472              :
473              : template <typename T>
474              : inline void transpose_16x16_kernel(float* dst, T* src, size_t dst_stride, size_t src_stride) {
475              :     __m256 r0, r1, r2, r3, r4, r5, r6, r7;
476              :
477              :     for (int i = 0; i < 16; i += 8) {
478              :         for (int j = 0; j < 16; j += 8) {
479              :             r0 = mm256_uni_loadu_ps(src + src_stride * j);
480              :             r1 = mm256_uni_loadu_ps(src + src_stride * (1 + j));
481              :             r2 = mm256_uni_loadu_ps(src + src_stride * (2 + j));
482              :             r3 = mm256_uni_loadu_ps(src + src_stride * (3 + j));
483              :             r4 = mm256_uni_loadu_ps(src + src_stride * (4 + j));
484              :             r5 = mm256_uni_loadu_ps(src + src_stride * (5 + j));
485              :             r6 = mm256_uni_loadu_ps(src + src_stride * (6 + j));
486              :             r7 = mm256_uni_loadu_ps(src + src_stride * (7 + j));
487              :
488              :             transpose_8x8(r0, r1, r2, r3, r4, r5, r6, r7);
489              :
490              :             _mm256_storeu_ps(dst + j, r0);
491              :             _mm256_storeu_ps(dst + j + dst_stride, r1);
492              :             _mm256_storeu_ps(dst + j + dst_stride * 2, r2);
493              :             _mm256_storeu_ps(dst + j + dst_stride * 3, r3);
494              :             _mm256_storeu_ps(dst + j + dst_stride * 4, r4);
495              :             _mm256_storeu_ps(dst + j + dst_stride * 5, r5);
496              :             _mm256_storeu_ps(dst + j + dst_stride * 6, r6);
497              :             _mm256_storeu_ps(dst + j + dst_stride * 7, r7);
498              :         }
499              :         src += 8;
500              :         dst += 8 * dst_stride;
501              :     }
502              : }
503              :
504              : template <typename T>
505              : inline void transpose_16xK_kernel(float* dst, T* src, size_t K, size_t dst_stride, size_t src_stride) {
506              :     __m256 r0, r1, r2, r3, r4, r5, r6, r7;
507              :
508              :     if (K >= 8) {
509              :         for (int j = 0; j < 16; j += 8) {
```

```
510              :             r0 = mm256_uni_loadu_ps(src + src_stride * j);
511              :             r1 = mm256_uni_loadu_ps(src + src_stride * (1 + j));
512              :             r2 = mm256_uni_loadu_ps(src + src_stride * (2 + j));
513              :             r3 = mm256_uni_loadu_ps(src + src_stride * (3 + j));
514              :             r4 = mm256_uni_loadu_ps(src + src_stride * (4 + j));
515              :             r5 = mm256_uni_loadu_ps(src + src_stride * (5 + j));
516              :             r6 = mm256_uni_loadu_ps(src + src_stride * (6 + j));
517              :             r7 = mm256_uni_loadu_ps(src + src_stride * (7 + j));
518              :
519              :             transpose_8x8(r0, r1, r2, r3, r4, r5, r6, r7);
520              :
521              :             _mm256_storeu_ps(dst + j, r0);
522              :             _mm256_storeu_ps(dst + j + dst_stride, r1);
523              :             _mm256_storeu_ps(dst + j + dst_stride * 2, r2);
524              :             _mm256_storeu_ps(dst + j + dst_stride * 3, r3);
525              :             _mm256_storeu_ps(dst + j + dst_stride * 4, r4);
526              :             _mm256_storeu_ps(dst + j + dst_stride * 5, r5);
527              :             _mm256_storeu_ps(dst + j + dst_stride * 6, r6);
528              :             _mm256_storeu_ps(dst + j + dst_stride * 7, r7);
529              :         }
530              :         src += 8;
531              :         dst += 8 * dst_stride;
532              :         K -= 8;
533              :     }
534              :     if (K > 0) {
535              :         for (int j = 0; j < 16; j += 8) {
536              :             r0 = mm256_uni_loadu_tail_ps(src + src_stride * j, K);
537              :             r1 = mm256_uni_loadu_tail_ps(src + src_stride * (1 + j), K);
538              :             r2 = mm256_uni_loadu_tail_ps(src + src_stride * (2 + j), K);
539              :             r3 = mm256_uni_loadu_tail_ps(src + src_stride * (3 + j), K);
540              :             r4 = mm256_uni_loadu_tail_ps(src + src_stride * (4 + j), K);
541              :             r5 = mm256_uni_loadu_tail_ps(src + src_stride * (5 + j), K);
542              :             r6 = mm256_uni_loadu_tail_ps(src + src_stride * (6 + j), K);
543              :             r7 = mm256_uni_loadu_tail_ps(src + src_stride * (7 + j), K);
544              :
545              :             transpose_8x8(r0, r1, r2, r3, r4, r5, r6, r7);
546              :
547              : #    define S(m) _mm256_storeu_ps(dst + j + m * dst_stride, r##m)
548              :             switch (K) {
549              :             case 1:
550              :                 S(0);
551              :                 break;
552              :             case 2:
553              :                 S(0);
554              :                 S(1);
555              :                 break;
556              :             case 3:
557              :                 S(0);
558              :                 S(1);
559              :                 S(2);
560              :                 break;
561              :             case 4:
562              :                 S(0);
563              :                 S(1);
564              :                 S(2);
565              :                 S(3);
566              :                 break;
567              :             case 5:
568              :                 S(0);
569              :                 S(1);
570              :                 S(2);
571              :                 S(3);
572              :                 S(4);
573              :                 break;
574              :             case 6:
575              :                 S(0);
576              :                 S(1);
577              :                 S(2);
578              :                 S(3);
579              :                 S(4);
580              :                 S(5);
581              :                 break;
582              :             case 7:
583              :                 S(0);
584              :                 S(1);
585              :                 S(2);
586              :                 S(3);
587              :                 S(4);
588              :                 S(5);
589              :                 S(6);
590              :                 break;
591              :             }
592              : #    undef S
593              :         }
594              :     }
595              : }
596              :
597              : #elif defined(HAVE_SVE)
598              : template <typename TSRC, typename TDST>
599              : inline void transpose_16x16_kernel(TDST* dst, TSRC* src, size_t dst_stride, size_t src_stride) {
600              :     for (size_t i = 0; i < 16; i++) {
601              :         for (size_t j = 0; j < 16; j++) {
602              :             dst[i * dst_stride + j] = static_cast<TDST>(src[i + j * src_stride]);
603              :         }
604              :     }
605              : }
606              :
607              : template <typename TSRC, typename TDST>
608            0 : inline void transpose_16xK_kernel(TDST* dst, TSRC* src, size_t K, size_t dst_stride, size_t src_stride) {
609            0 :     for (size_t i = 0; i < K; i++) {
610            0 :         for (size_t j = 0; j < 16; j++) {
611            0 :             dst[i * dst_stride + j] = static_cast<TDST>(src[i + j * src_stride]);
612              :         }
613              :     }
```

```
      614          0 : }
      615            :
      616      85031 : inline void transpose_8x16_kernel(float16_t* dst, float16_t* src, size_t dst_stride, size_t src_stride) {
      617            :     // load 8 rows of 16 elements
      618            :     // a -> a0 a1 a2 a3 a4 a5 a6 a7 a8 ...
      619            :     // b -> b0 b1 b2 b3 b4 b5 b6 b7 b8 ...
      620            :     // c -> c0 c1 c2 c3 c4 c5 c6 c7 c8 ...
      621            :     // d -> d0 d1 d2 d3 d4 d5 d6 d7 d8 ...
      622            :     // ...
      623      85031 :     auto a = svld1_f16(svptrue_b16(), src);
      624      85031 :     auto b = svld1_f16(svptrue_b16(), src + 1 * src_stride);
      625      85031 :     auto c = svld1_f16(svptrue_b16(), src + 2 * src_stride);
      626      85031 :     auto d = svld1_f16(svptrue_b16(), src + 3 * src_stride);
      627      85031 :     auto e = svld1_f16(svptrue_b16(), src + 4 * src_stride);
      628      85031 :     auto f = svld1_f16(svptrue_b16(), src + 5 * src_stride);
      629      85031 :     auto g = svld1_f16(svptrue_b16(), src + 6 * src_stride);
      630      85031 :     auto h = svld1_f16(svptrue_b16(), src + 7 * src_stride);
      631            :
      632            :     // a -> a0 b0 | a2 b2 | a4 b4 | a6 b6 | a8 b8 ...
      633            :     // b -> a1 b1 | a3 b3 | a5 b5 | a7 b7 | a9 b9 ...
      634            :     // c -> c0 d0 | c2 d2 | c4 d6 | c6 d8 | c8 d8 ...
      635            :     // d -> c1 d1 | c3 d3 | c5 d5 | c7 d7 | c9 d9 ...
      636            :     // ...
      637            :
      638      85031 :     auto ta = svtrn1_f16(a, b);
      639      85031 :     auto tb = svtrn2_f16(a, b);
      640      85031 :     auto tc = svtrn1_f16(c, d);
      641      85031 :     auto td = svtrn2_f16(c, d);
      642      85031 :     auto te = svtrn1_f16(e, f);
      643      85031 :     auto tf = svtrn2_f16(e, f);
      644      85031 :     auto tg = svtrn1_f16(g, h);
      645      85031 :     auto th = svtrn2_f16(g, h);
      646            :
      647            :     // a -> a0 b0 c0 d0 | a4 b4 c4 d6 | a8 b8 ...
      648            :     // b -> a2 b2 c2 d2 | a6 b6 c6 d8 | c8 d8 ...
      649            :     // c -> a1 b1 c1 d1 | a5 b5 c5 d5 | a9 b9 ...
      650            :     // d -> a3 b3 c3 d3 | a7 b7 c7 d7 | c9 d9 ...
      651            :     // ...
      652            :
      653      85031 :     auto a1 = svtrn1_f32(svreinterpret_f32_f16(ta), svreinterpret_f32_f16(tc));
      654      85031 :     auto b1 = svtrn2_f32(svreinterpret_f32_f16(ta), svreinterpret_f32_f16(tc));
      655      85031 :     auto c1 = svtrn1_f32(svreinterpret_f32_f16(tb), svreinterpret_f32_f16(td));
      656      85031 :     auto d1 = svtrn2_f32(svreinterpret_f32_f16(tb), svreinterpret_f32_f16(td));
      657      85031 :     auto e1 = svtrn1_f32(svreinterpret_f32_f16(te), svreinterpret_f32_f16(tg));
      658      85031 :     auto f1 = svtrn2_f32(svreinterpret_f32_f16(te), svreinterpret_f32_f16(tg));
      659      85031 :     auto g1 = svtrn1_f32(svreinterpret_f32_f16(tf), svreinterpret_f32_f16(th));
      660      85031 :     auto h1 = svtrn2_f32(svreinterpret_f32_f16(tf), svreinterpret_f32_f16(th));
      661            :
      662      85031 :     auto a2 = svtrn1_f64(svreinterpret_f64_f32(a1), svreinterpret_f64_f32(e1));
      663      85031 :     auto b2 = svtrn2_f64(svreinterpret_f64_f32(a1), svreinterpret_f64_f32(e1));
      664      85031 :     auto c2 = svtrn1_f64(svreinterpret_f64_f32(b1), svreinterpret_f64_f32(f1));
      665      85031 :     auto d2 = svtrn2_f64(svreinterpret_f64_f32(b1), svreinterpret_f64_f32(f1));
      666      85031 :     auto e2 = svtrn1_f64(svreinterpret_f64_f32(c1), svreinterpret_f64_f32(g1));
      667      85031 :     auto f2 = svtrn2_f64(svreinterpret_f64_f32(c1), svreinterpret_f64_f32(g1));
      668      85031 :     auto g2 = svtrn1_f64(svreinterpret_f64_f32(d1), svreinterpret_f64_f32(h1));
      669      85031 :     auto h2 = svtrn2_f64(svreinterpret_f64_f32(d1), svreinterpret_f64_f32(h1));
      670            :
      671      85031 :     auto pgh = svnot_b_z(svptrue_b64(), svptrue_pat_b64(SV_VL2));
      672      85031 :     auto pgl = svptrue_pat_b64(SV_VL2);
      673      85031 :     svst1_f64(pgl, reinterpret_cast<float64_t*>(dst), a2);
      674      76516 :     svst1_f64(pgl, reinterpret_cast<float64_t*>(dst + dst_stride * 1), e2);
      675      76516 :     svst1_f64(pgl, reinterpret_cast<float64_t*>(dst + dst_stride * 2), c2);
      676      76522 :     svst1_f64(pgl, reinterpret_cast<float64_t*>(dst + dst_stride * 3), g2);
      677      76522 :     svst1_f64(pgl, reinterpret_cast<float64_t*>(dst + dst_stride * 4), b2);
      678      75655 :     svst1_f64(pgl, reinterpret_cast<float64_t*>(dst + dst_stride * 5), f2);
      679      84868 :     svst1_f64(pgl, reinterpret_cast<float64_t*>(dst + dst_stride * 6), d2);
      680      84868 :     svst1_f64(pgl, reinterpret_cast<float64_t*>(dst + dst_stride * 7), h2);
      681            :
      682      86136 :     svst1_f64(pgh, reinterpret_cast<float64_t*>(dst + dst_stride * 8) - 2, a2);
      683      86108 :     svst1_f64(pgh, reinterpret_cast<float64_t*>(dst + dst_stride * 9) - 2, e2);
      684      86004 :     svst1_f64(pgh, reinterpret_cast<float64_t*>(dst + dst_stride * 10) - 2, c2);
      685      86004 :     svst1_f64(pgh, reinterpret_cast<float64_t*>(dst + dst_stride * 11) - 2, g2);
      686      94500 :     svst1_f64(pgh, reinterpret_cast<float64_t*>(dst + dst_stride * 12) - 2, b2);
      687      94808 :     svst1_f64(pgh, reinterpret_cast<float64_t*>(dst + dst_stride * 13) - 2, f2);
      688      97830 :     svst1_f64(pgh, reinterpret_cast<float64_t*>(dst + dst_stride * 14) - 2, d2);
      689      97830 :     svst1_f64(pgh, reinterpret_cast<float64_t*>(dst + dst_stride * 15) - 2, h2);
      690      97833 : }
      691            :
      692      41518 : inline void transpose_16x16_kernel(ov::float16* dst, ov::float16* src, size_t dst_stride, size_t src_stride) {
      693            :     // check for SVE256
      694      41518 :     if (svcnth() == 16) {
      695      41518 :         transpose_8x16_kernel(reinterpret_cast<float16_t*>(dst),
      696            :                               reinterpret_cast<float16_t*>(src),
      697            :                               dst_stride,
      698            :                               src_stride);
      699      48957 :         transpose_8x16_kernel(reinterpret_cast<float16_t*>(dst + 8),
      700      48957 :                               reinterpret_cast<float16_t*>(src + 128),
      701            :                               dst_stride,
      702            :                               src_stride);
      703            :     } else {
      704          0 :         for (size_t i = 0; i < 16; i++) {
      705          0 :             for (size_t j = 0; j < 16; j++) {
      706          0 :                 dst[i * dst_stride + j] = (src[i + j * src_stride]);
      707            :             }
      708            :         }
      709            :     }
      710      56641 : }
      711            :
      712          0 : inline void transpose_8x8_kernel(float* src, size_t ld_src, float* dst, size_t ld_dst) {
      713            :     // load from src to registers
      714            :     // a: a0  a1  a2  a3  a4  a5  a6  a7
      715            :     // b: b0  b1  b2  b3  b4  b5  b6  b7
      716            :     // c: c0  c1  c2  c3  c4  c5  c6  c7
      717            :     // d: d0  d1  d2  d3  d4  d5  d6  d7
```

```
718            :     // e: e0  e1  e2  e3  e4  e5  e6  e7
719            :     // f: f0  f1  f2  f3  f4  f5  f6  f7
720            :     // g: g0  g1  g2  g3  g4  g5  g6  g7
721            :     // h: h0  h1  h2  h3  h4  h5  h6  h7
722          0 :     svfloat32_t a = svld1_f32(svptrue_b8(), &src[0 * ld_src]);
723          0 :     svfloat32_t b = svld1_f32(svptrue_b8(), &src[1 * ld_src]);
724          0 :     svfloat32_t c = svld1_f32(svptrue_b8(), &src[2 * ld_src]);
725          0 :     svfloat32_t d = svld1_f32(svptrue_b8(), &src[3 * ld_src]);
726          0 :     svfloat32_t e = svld1_f32(svptrue_b8(), &src[4 * ld_src]);
727          0 :     svfloat32_t f = svld1_f32(svptrue_b8(), &src[5 * ld_src]);
728          0 :     svfloat32_t g = svld1_f32(svptrue_b8(), &src[6 * ld_src]);
729          0 :     svfloat32_t h = svld1_f32(svptrue_b8(), &src[7 * ld_src]);
730            :     // unpacking and interleaving 32-bit elements
731            :     // a0  b0  a1  b1  a4  b4  a5  b5
732            :     // a2  b2  a3  b3  a6  b6  a7  b7
733            :     // c0  d0  c1  d1 ...
734            :     // c2  d2  c3  d3 ...
735            :     // e0  f0  e1  f1 ...
736            :     // e2  f2  e3  f3 ...
737            :     // g0  h0  g1  h1 ...
738            :     // g2  h2  g3  h3 ...
739          0 :     svfloat32_t ta = svtrn1_f32(a, b);
740          0 :     svfloat32_t tb = svtrn2_f32(a, b);
741          0 :     svfloat32_t tc = svtrn1_f32(c, d);
742          0 :     svfloat32_t td = svtrn2_f32(c, d);
743          0 :     svfloat32_t te = svtrn1_f32(e, f);
744          0 :     svfloat32_t tf = svtrn2_f32(e, f);
745          0 :     svfloat32_t tg = svtrn1_f32(g, h);
746          0 :     svfloat32_t th = svtrn2_f32(g, h);
747            :     // unpacking and interleaving 64-bit elements
748            :     //  a0  b0  c0  d0  a4  b4  c4  d4
749            :     //  a1  b1  c1  d1 ...
750            :     //  a2  b2  c2  d2 ...
751            :     //  a3  b3  c3  d3 ...
752            :     //  e0  f0  g0  h0  e4  f4  g4  h4
753            :     //  e1  f1  g1  h1 ...
754            :     //  e2  f2  g2  h2 ...
755            :     //  e3  f3  g3  h3 ...
756          0 :     a = svreinterpret_f32_f64(svtrn1_f64(svreinterpret_f64_f32(ta), svreinterpret_f64_f32(tc)));
757          0 :     b = svreinterpret_f32_f64(svtrn2_f64(svreinterpret_f64_f32(ta), svreinterpret_f64_f32(tc)));
758          0 :     c = svreinterpret_f32_f64(svtrn1_f64(svreinterpret_f64_f32(tb), svreinterpret_f64_f32(td)));
759          0 :     d = svreinterpret_f32_f64(svtrn2_f64(svreinterpret_f64_f32(tb), svreinterpret_f64_f32(td)));
760          0 :     e = svreinterpret_f32_f64(svtrn1_f64(svreinterpret_f64_f32(te), svreinterpret_f64_f32(tg)));
761          0 :     f = svreinterpret_f32_f64(svtrn2_f64(svreinterpret_f64_f32(te), svreinterpret_f64_f32(tg)));
762          0 :     g = svreinterpret_f32_f64(svtrn1_f64(svreinterpret_f64_f32(tf), svreinterpret_f64_f32(th)));
763          0 :     h = svreinterpret_f32_f64(svtrn2_f64(svreinterpret_f64_f32(tf), svreinterpret_f64_f32(th)));
764            :     //  shuffle 128-bits (composed of 4 32-bit elements)
765            :     //  a0  b0  c0  d0  e0  f0  g0  h0
766            :     //  a1  b1  c1  d1 ...
767            :     //  a2  b2  c2  d2 ...
768            :     //  a3  b3  c3  d3 ...
769            :     //  a4  b4  c4  d4 ...
770            :     //  a5  b5  c5  d5 ...
771            :     //  a6  b6  c6  d6 ...
772            :     //  a7  b7  c7  d7 ...
773          0 :     svfloat32_t t1a = svext_f32(a, a, 4);
774          0 :     svfloat32_t t1b = svext_f32(b, b, 4);
775          0 :     svfloat32_t t1c = svext_f32(c, c, 4);
776          0 :     svfloat32_t t1d = svext_f32(d, d, 4);
777          0 :     ta = svext_f32(t1a, e, 4);
778          0 :     tb = svext_f32(t1b, f, 4);
779          0 :     tc = svext_f32(t1c, g, 4);
780          0 :     td = svext_f32(t1d, h, 4);
781          0 :     te = svsel_f32(svptrue_pat_b32(SV_VL4), t1a, e);
782          0 :     tf = svsel_f32(svptrue_pat_b32(SV_VL4), t1b, f);
783          0 :     tg = svsel_f32(svptrue_pat_b32(SV_VL4), t1c, g);
784          0 :     th = svsel_f32(svptrue_pat_b32(SV_VL4), t1d, h);
785            :     // Store the transposed result in destination
786          0 :     svst1_f32(svptrue_b8(), &dst[0 * ld_dst], ta);
787          0 :     svst1_f32(svptrue_b8(), &dst[1 * ld_dst], tc);
788          0 :     svst1_f32(svptrue_b8(), &dst[2 * ld_dst], tb);
789          0 :     svst1_f32(svptrue_b8(), &dst[3 * ld_dst], td);
790          0 :     svst1_f32(svptrue_b8(), &dst[4 * ld_dst], te);
791          0 :     svst1_f32(svptrue_b8(), &dst[5 * ld_dst], tg);
792          0 :     svst1_f32(svptrue_b8(), &dst[6 * ld_dst], tf);
793          0 :     svst1_f32(svptrue_b8(), &dst[7 * ld_dst], th);
794          0 : }
795            :
796          0 : inline void transpose_16x16_kernel(float* dst, float* src, size_t dst_stride, size_t src_stride) {
797          0 :     transpose_8x8_kernel(src, src_stride, dst, dst_stride);
798          0 :     transpose_8x8_kernel(src + 8, src_stride, dst + 8 * dst_stride, dst_stride);
799          0 :     transpose_8x8_kernel(src + 8 * src_stride, src_stride, dst + 8, dst_stride);
800          0 :     transpose_8x8_kernel(src + 8 * src_stride + 8, src_stride, dst + 8 * dst_stride + 8, dst_stride);
801          0 : }
802            :
803            : #else
804            :
805            : template <typename TSRC, typename TDST>
806          0 : inline void transpose_16x16_kernel(TDST* dst, TSRC* src, size_t dst_stride, size_t src_stride) {
807          0 :     for (size_t i = 0; i < 16; i++) {
808          0 :         for (size_t j = 0; j < 16; j++) {
809          0 :             dst[i * dst_stride + j] = static_cast<TDST>(src[i + j * src_stride]);
810            :         }
811            :     }
812          0 : }
813            :
814            : template <typename TSRC, typename TDST>
815            : inline void transpose_16xK_kernel(TDST* dst, TSRC* src, size_t K, size_t dst_stride, size_t src_stride) {
816            :     for (size_t i = 0; i < K; i++) {
817            :         for (size_t j = 0; j < 16; j++) {
818            :             dst[i * dst_stride + j] = static_cast<TDST>(src[i + j * src_stride]);
819            :         }
820            :     }
821            : }
```

```
      822             :
      823             : #endif
      824             :
      825             : }  // namespace ov::Extensions::Cpu::XARCH
```