lelism of the latter algorithm. To illustrate this, suppose $n = 4$ and group the six subproblems into three *rotation sets* as follows:

$$\begin{aligned} rot.set(1) &= \{(1,2),(3,4)\} \\ rot.set(2) &= \{(1,3),(2,4)\} \\ rot.set(3) &= \{(1,4),(2,3)\} \end{aligned}$$

Note that all the rotations within each of the three rotation sets are "nonconflicting." That is, subproblems (1,2) and (3,4) can be carried out in parallel. Likewise the (1,3) and (2,4) subproblems can be executed in parallel as can subproblems (1,4) and (2,3). In general, we say that

$$(i_1, j_1), (i_2, j_2), \ldots, (i_N, j_N) \qquad N = (n-1)n/2$$

is a *parallel ordering* of the set $\{(i,j) \mid 1 \le i < j \le n\}$ if for $s = 1{:}n-1$ the rotation set $rot.set(s) = \{ (i_r, j_r) : r = 1 + n(s-1)/2{:}ns/2 \}$ consists of nonconflicting rotations. This requires n to be even, which we assume throughout this section. (The odd n case can be handled by bordering A with a row and column of zeros and being careful when solving the subproblems that involve these augmented zeros.)

A good way to generate a parallel ordering is to visualize a chess tournament with n players in which everybody must play everybody else exactly once. In the $n = 8$ case this entails 7 "rounds." During round one we have the following four games:

1	3	5	7
2	4	6	8

$$rot.set(1) = \{ (1,2),(3,4),(5,6),(7,8) \}$$

i.e., 1 plays 2, 3 plays 4, etc. To set up rounds 2 through 7, player 1 stays put and players 2 through 8 embark on a merry-go-round:

1	2	3	5
4	6	8	7

$$rot.set(2) = \{(1,4),(2,6),(3,8),(5,7)\}$$

1	4	2	3
6	8	7	5

$$rot.set(3) = \{(1,6),(4,8),(2,7),(3,5)\}$$

1	6	4	2
8	7	5	3

$$rot.set(4) = \{(1,8),(6,7),(4,5),(2,3)\}$$

1	8	6	4
7	5	3	2

$$rot.set(5) = \{(1,7),(5,8),(3,6),(2,4)\}$$

1	7	8	6
5	3	2	4

$$rot.set(6) = \{(1,5),(3,7),(2,8),(4,6)\}$$

1	5	7	8
3	2	4	6

$$rot.set(7) = \{(1,3),(2,5),(4,7),(6,8)\}$$

We can encode these operations in a pair of integer vectors $top(1{:}n/2)$ and $bot(1{:}n/2)$. During a given round $top(k)$ plays $bot(k)$, $k = 1{:}n/2$. The pairings for the next round is obtained by updating *top* and *bot* as follows:

```
function: [new.top, new.bot] = music(top, bot, n)
    m = n/2
    for k = 1:m
        if k = 1
            new.top(1) = 1
        else if k = 2
            new.top(k) = bot(1)
        elseif k > 2
            new.top(k) = top(k − 1)
        end
        if k = m
            new.bot(k) = top(k)
        else
            new.bot(k) = bot(k + 1)
        end
    end
```

Using **music** we obtain the following parallel order Jacobi procedure.

Algorithm 8.4.4 (Parallel Order Jacobi) Given a symmetric $A \in \mathbb{R}^{n\times n}$ and a tolerance $tol > 0$, this algorithm overwrites A with V^TAV where V is orthogonal and $\text{off}(V^TAV) \le tol\| A \|_F$. It is assumed that n is even.

```
V = I_n
eps = tol|| A ||_F
top = 1:2:n; bot = 2:2:n
while off(A) > eps
    for set = 1:n − 1
        for k = 1:n/2
            p = min(top(k), bot(k))
            q = max(top(k), bot(k))
            (c , s) = sym.schur2(A, p, q)
            A = J(p, q, θ)^T A J(p, q, θ)
            V = V J(p, q, θ)
        end
        [top, bot] = music(top, bot, n)
    end
end
```