```
1  #define BLOCKSIZE 32
2  void do_block (int n, int si, int sj, int sk, double *A, double
3  *B, double *C)
4  {
5     for (int i = si; i < si+BLOCKSIZE; ++i)
6        for (int j = sj; j < sj+BLOCKSIZE; ++j)
7           {
8              double cij = C[i+j*n];/* cij = C[i][j] */
9              for( int k = sk; k < sk+BLOCKSIZE; k++ )
10                cij += A[i+k*n] * B[k+j*n];/* cij+=A[i][k]*B[k][j] */
11             C[i+j*n] = cij;/* C[i][j] = cij */
12          }
13 }
14 void dgemm (int n, double* A, double* B, double* C)
15 {
16    for ( int sj = 0; sj < n; sj += BLOCKSIZE )
17       for ( int si = 0; si < n; si += BLOCKSIZE )
18          for ( int sk = 0; sk < n; sk += BLOCKSIZE )
19             do_block(n, si, sj, sk, A, B, C);
20 }
```

FIGURE 5.21 Cache blocked version of DGEMM in Figure 3.21. Assume C is initialized to zero. The do_block function is basically DGEMM from Chapter 2 with new parameters to specify the starting positions of the submatrices of BLOCKSIZE. The gcc optimizer can remove the function overhead instructions by inlining the do_block function.