```c
#include <x86intrin.h>
#define UNROLL (4)
#define BLOCKSIZE 32
void do_block (int n, int si, int sj, int sk,
               double *A, double *B, double *C)
{
  for ( int i = si; i < si+BLOCKSIZE; i+=UNROLL*8 )
    for ( int j = sj; j < sj+BLOCKSIZE; j++ ) {
      __m512d c[UNROLL];
      for (int r=0;r<UNROLL;r++)
        c[r] =  _mm512_load_pd(C+i+r*8+j*n); //[ UNROLL];

      for( int k = sk; k < sk+BLOCKSIZE; k++ )
      {
        __m512d bb = _mm512_broadcastsd_pd(_mm_load_sd(B+j*n+k));
        for (int r=0;r<UNROLL;r++)
          c[r] = _mm512_fmadd_pd(_mm512_load_pd(A+n*k+r*8+i), bb, c[r]);
       }

     for (int r=0;r<UNROLL;r++)
       _mm512_store_pd(C+i+r*8+j*n, c[r]);
     }
  }

void dgemm (int n, double* A, double* B, double* C)
{
  for ( int sj = 0; sj < n; sj += BLOCKSIZE )
    for ( int si = 0; si < n; si += BLOCKSIZE )
      for ( int sk = 0; sk < n; sk += BLOCKSIZE )
        do_block(n, si, sj, sk, A, B, C);
}
```

FIGURE 5.48 Optimized C version of DGEMM from Figure 4.80 using cache blocking. These changes are the same ones found in Figure 5.21. The assembly language produced by the compiler for the do_block function is nearly identical to Figure 4.81. Once again, there is no overhead to call the do_block because the compiler inlines the function call.

Copyright © 2021 Elsevier Inc. All rights reserved