Combining Locality, Thread and SIMD Parallelism:

The following code excerpt is representative of a common signal processing technique called convolution. Convolution combines two signals to form a third signal. In this example, we slide a small (32x32) signal around in a larger (4128x4128) signal to look for regions where the two signals have the most overlap.

```c
for (l=0; l<N; l++) {
    for (k=0; k<N; k++) {
        C[k][l] = 0.0;
        for (j=0; j<W; j++) {
            for (i=0; i<W; i++) {
                C[k][l] += A[k+i][l+j]*B[i][j];
            }
        }
    }
}
```
Programming Assignment 2, cont.

• Your goal is to take the code as written, and by either changing the code or changing the way you invoke the compiler, improve its performance.

• You will need to use an Intel platform that has the "icc" compiler installed.
  - You can use the CADE Windows lab, or another Intel platform with the appropriate software.
  - The Intel compiler is available for free on Linux platforms for non-commercial use.
  - The version of the compiler, the specific architecture and the flags you give to the compiler will drastically impact performance.

• You can discuss strategies with other classmates, but do not copy code. Also, do not copy solutions from the web. This must be your own work.
Programming Assignment 2, cont.

• How to compile
  - OpenMP (all versions): *icc -openmp conv-assign.c*

• Measure and report performance for five versions of the code, and turn in all variants:
  - Baseline: compile and run code as provided (5 points)
  - Thread parallelism only (using OpenMP): *icc -openmp conv-omp.c* (5 points)
  - SSE-3 only: *icc -openmp -msse3 -vec-report=3 conv-sse.c* (10 points)
  - Locality only: *icc -openmp conv-loc.c* (10 points)
  - Combined: *icc -openmp -msse3 -vec-report=3 conv-all.c* (15 points)

• Explain results and observations (15 points)

• Extra credit (10 points): improve results by optimizing for register reuse