Lab Course: Efficient Programming of Multicore Processors and Supercomputers

Jonas Mayer, Paul Preißner, Konrad Pröll

May 27, 2017

3.2 Automatic Parallelization

3.2.1 Try the automatic parallelization feature of icc

ICC's automatic parallelization feature is switched on by the compiler flag -parallel. Thus, we added this flag to the flags in the Makefile.

3.2.2 The compiler might parallelize all the loops. With the "-opt-report-phase=par" option you will receive a report explaining the reasons why parallelization was not possible.

Initially, the compiler suggests there is a parallel dependence between iterations of the outer loop and thus does not parallelize it. Obviously, unew, diff and sum are being reused between different iterations, so we might need to fix this by inserting the corresponding pragma.

3.2.3 Try to improve the parallelization by rewriting the code and inserting pragmas (no OpenMP pragmas). Explain why it worked or did not work.

By putting "#pragma parallel private (unew, diff, sum)" ahead of the outer loop, we suggest to the compiler that this loop might get parallelized and avoid the issue of variables being written into accross different threads. However, because the loopsize cannot be determined at compiletime, icc will not parallelize the llop due to "insufficient computational work".
If we change the pragma to "#pragma parallel always private (unew, diff, sum)", the loop will be parallelized. You can also force the compiler to parallelize by adding "-par-threshold0" to the compiler flags. This obviously does not affect specific loops, but changes the global setting in your program.
The flag -guide-par gives you hints on sections of the code that might get parallelized.

```
#pragma parallel always private (unew, diff, sum)
  for( i=1; i<sizey-1; i++) {
    int ii=i*sizex;
    int iim1=(i-1)*sizex
    int iip1=(i+1)*sizex

#pragma ivdep
    for( j=1; j<sizex-1; j++) {
      //loop implementation
    }
  }
```