PIC Vectorization with OpenMP

Viktor K. Decyk, UCLA, USA, and Ricardo Fonseca, ISCTE, Portugal

In this part of the research project, we investigated the use of vectorization with Intel processors. We began with the OpenMP 2D electrostatic (ES) code (mpic2) and wrote the 5 most important procedures using Intel SSE2 vector instrinsics. For better performance on the SSE, we transposed the particle array so that x locations for all the particles are stored together, followed by the y locations, etc. In addition, the loop which first assigned data to the particle array was performed in parallel. Memory is distributed among nodes in recent NUMA Intel architectures, and the assignment is made when first written. Since subsequent processing of particles used the same parallel loop structure, this avoids subsequent parallel loops where threads process data which resides on a remote node.

The results with the Intel compiler on the 2.67 GHz Nehalem i7 showed speedups as follows:

One processor:
Push, 1.8x, Deposit, 1.9x, Sort = 0.35x, Solver 2.1x, and FFT, 1.03x, with an overall speedup for the entire code of 1.7x.
Twelve processors:
Push, 1.9x, Deposit, 3.1x, Sort = 0.48x, Solver 1.06x, and FFT, 1.3x, with an overall speedup for the entire code of 1.9x.

The best one expects for the SSE2 is 4. One can see that we achieve a performance improvement of somewhat less than 2. This is largely due to fact that sorting does not vectorize well, and part of the sorting is included in the Push procedure. Also, note that the SSE2 version of the remaining part of the sorting procedure was substantially worse than the serial version. This was because that part of the sorting prefers the particle data not be transposed, whereas the particle calculations prefer the transposed structure. Overall the transposed structure is better, since the amount of time spent sorting is small.

We then attempted to use compiler directives and code rewrites to see how close we could come to the hand-coded results. We followed the same procedure as described in the document VectorPIC.pdf in the directory vpic2, where one breaks up long loops into multiple subloops with vectorizable parts and parts which run better in scalar mode, storing results into temporary arrays. With the Intel compiler, the flag -axSSE2 improved performance. With OpenMP 3.1, setting the environment variable OMP_PROC_BIND=true was also beneficial. The results with the Intel compiler on the 2.67 GHz Nehalem i7 showed speedups as follows:

One processor:
Push, 1.4x, Deposit, 1.5x, Sort = 0.40x, Solver 1.9x, and FFT, 1.0x, with an overall speedup for the entire code of 1.3x.
Twelve processors:
Push, 1.4x, Deposit, 2.7x, Sort = 0.50x, Solver 1.7x, and FFT, 1.04x, with an overall speedup for the entire code of 1.24x.

The conclusion was that rewriting our loops and using compiler directives results gives less than half the speedups obtained by hand-coded SSE2 code in most cases. Benchmark results for the particle processing are shown in Figure 1 below.

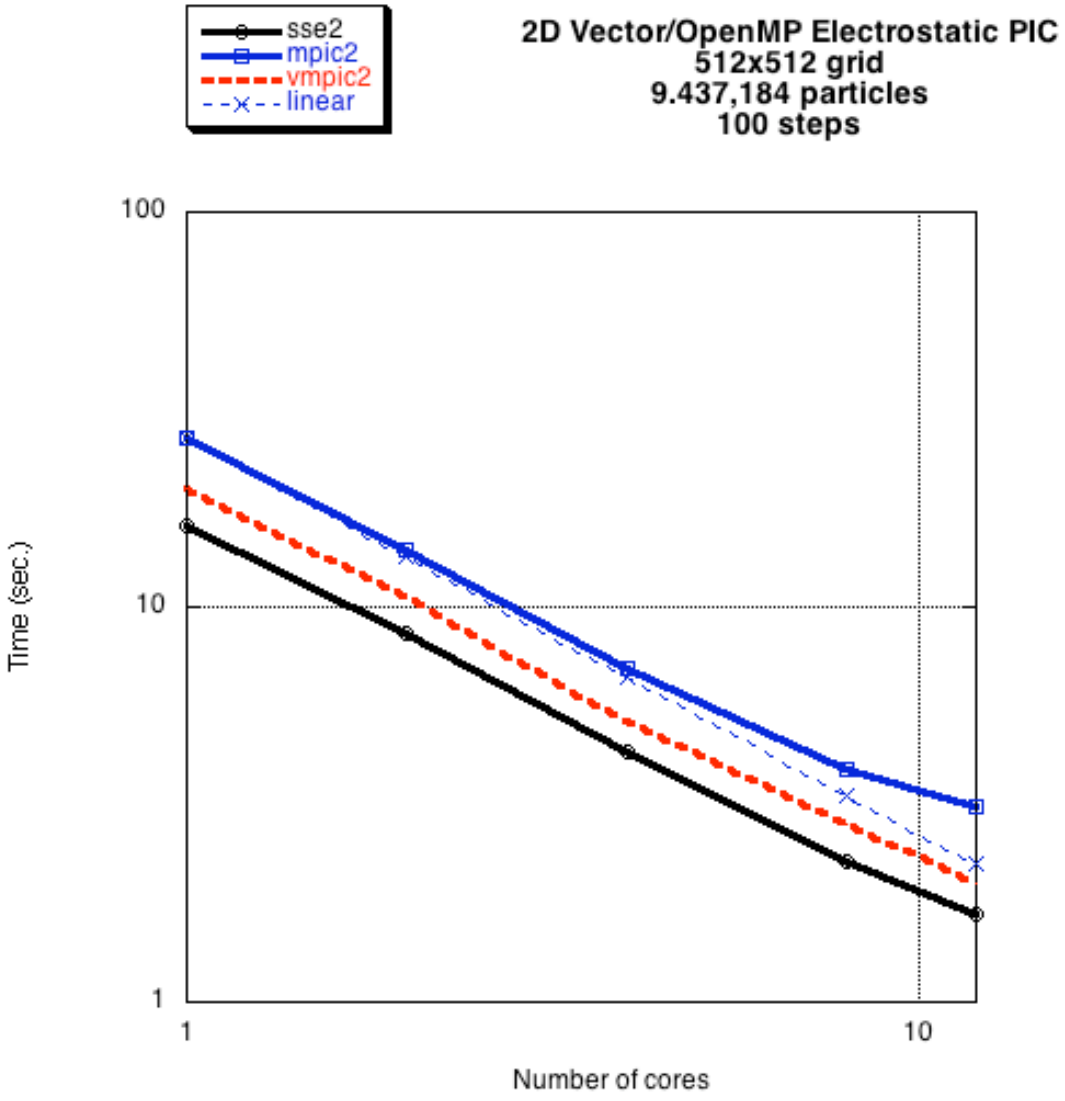

Figure 1. Overall performance comparing the original code (mpic2) with the vectorized version (vmpic2) and the SSE2/OpenMP version. For small number of processors, the performance of vmpic2 is between that of the SSE2 and mpic2. First assignment of particle data is performed in parallel. The performance with 12 cores compared to 1 core is typically 9-10x.

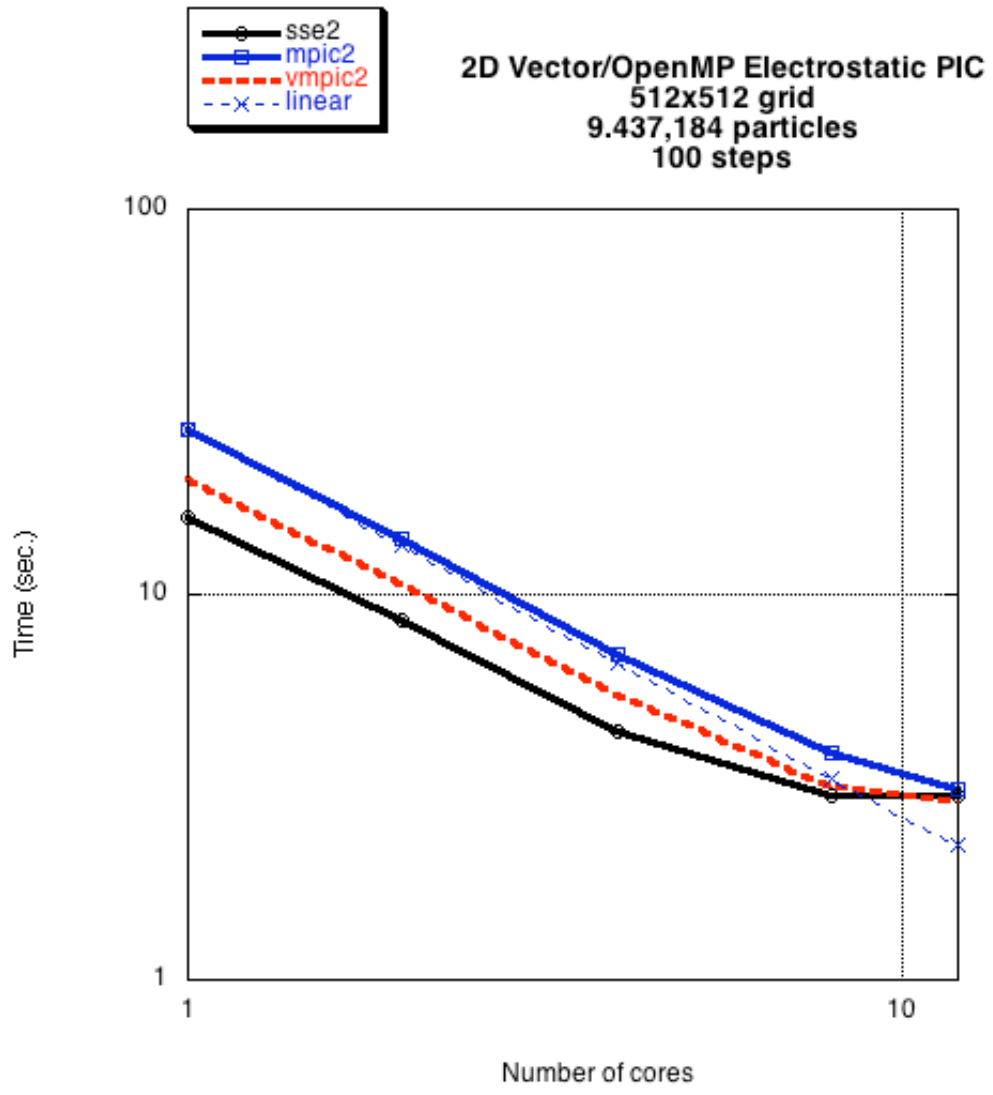

Figure 2. Overall performance comparing the original code (mpic2) with the vectorized version (vmpic2) and the SSE2/OpenMP version. For small number of processors, the performance of vmpic2 is between that of the SSE2 and mpic2. First assignment of particle data is not performed in parallel, resulting in some threads accessing memory located on remote nodes.

We then continued with the 2-1/2D electromagnetic (EM) code (mbpic2) and wrote the 6 most important procedures using Intel SSE2 vector instrinsics. With the Intel compiler, the compiler flag -axSSE2 improved performance. With OpenMP 3.1, setting the environment variable OMP_PROC_BIND=true was also beneficial. The relativistic results on the 2.67 GHz Nehalem i7 showed speedups as follows:

One processor:
Push, 2.6x, Charge Deposit, 1.9x, Current Deposit, 2.1x, Sort = 0.56x, Solver 1.2x, and FFT, 1.14x, with an overall speedup for the entire code of 2.2x.

Twelve processors:
Push, 2.6x, Charge Deposit, 3.8x, Current Deposit, 1.8x, Sort = 0.90x, Solver 0.70x, and FFT, 1.25x, with an overall speedup for the entire code of 2.2x.

The best one expects for the SSE2 is 4. One can see that we achieve a performance improvement of somewhat better than 2. Since the SSE2 vector length is 4, we set the 4th element to zero, the best result for interpolation would only give a speedup of 3x. We then attempted to use compiler directives and code rewrites to see how close we could come to the hand-coded results. The results with the Intel compiler on the 2.67 GHz Nehalem i7 showed speedups as follows:

One processor:
Push, 1.6x, Charge Deposit, 1.6x, Current Deposit, 1.2x, Sort = 0.51x, Solver 0.8x, and FFT, 1.03x, with an overall speedup for the entire code of 1.5x.
Twelve processors:
Push, 1.6x, Charge Deposit, 3.2x, Current Deposit, 1.4x, Sort = 0.90x, Solver 0.70x, and FFT, 0.96x, with an overall speedup for the entire code of 1.6x.

The conclusion was that rewriting our loops and using compiler directives results gives about half the speedups obtained by hand-coded SSE2 code in most cases. Benchmark results for the particle processing are shown in Figure 3 below.

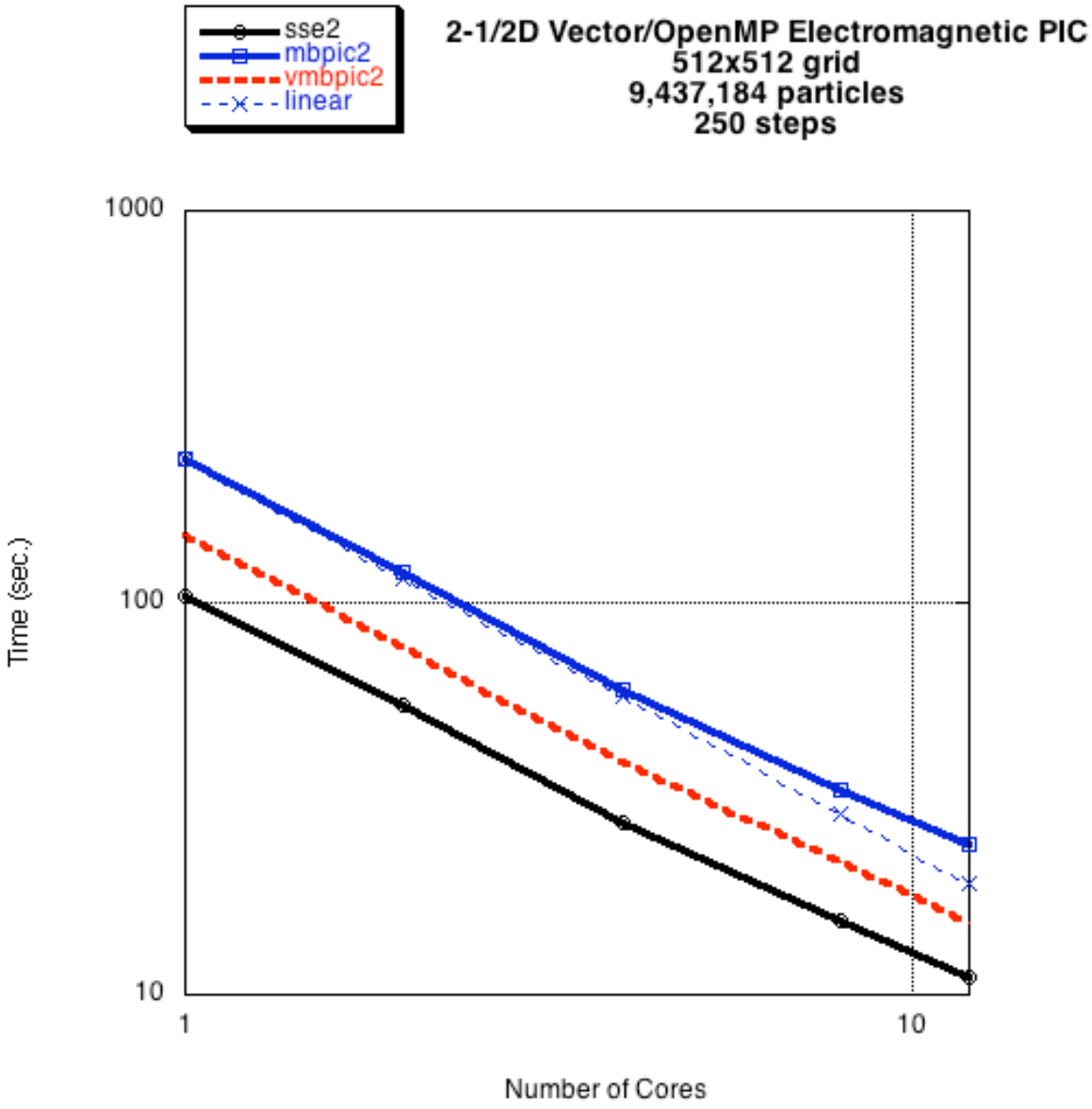

Figure 3. Overall performance comparing the original code (mbpic2) with the vectorized version (vmbpic2) and the SSE2/OpenMP version. For small number of processors, the performance of vmbpic2 is between that of the SSE2 and mbpic2. First assignment of particle data is performed in parallel. The performance with 12 cores compared to 1 core is typically 9-10x.

The benchmark codes are available at: https://idre.ucla.edu/hpc/parallel-plasma-pic-codes.