PIC Vectorization with Intel KNC MIC

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 PHI co-processors. The co-processor we are testing is the 1.1 GHz Xeon Phi Coprocessor 5110P (Knights Corner) MIC. We began with the Intel 16.0.2 Fortran compiler and compiled the Fortran version of the serial 3D electrostatic (ES) code (pic3.f90) without vectorization -O3 -mmic -no-vec) and executed on a single core of the KNC MIC to establish a baseline performance benchmark.

We then wrote the 4 most important procedures using the Intel KNC MIC C/C++ vector intrinsics. For better performance on the KNC MIC, we transposed the particle array so that x locations for all the particles are stored together, followed by the y locations, etc. The results with the Intel 16.0.2 C/C++ compiler on a single core showed speedups compared to the baseline as follows: Push, 2.4x, Deposit, 2.4x, Solver 1.8x, and FFT, 2.4x, with an overall speedup for the entire code of 2.3x. The best one expects for the KNC MIC is 16, so the results showed modest improvements. We then attempted to use compiler directives and code rewrites to see how close we could come to the hand-coded results.

We first compiled the code pic3.f90 with vectorization enabled (-O3 -mmic). The initial results were very disappointing. The FFT vectorized, with a speedup of about 1.5x. However, the push routine slowed down by a factor of 7, and the other 2 procedures had no substantial change in performance. However, if we transposed the particle array and made small simplifications in addressing, the results were much better. The Push procedure achieved a speedup of 1.6x. The Deposit did not vectorize, but nevertheless had a speedup of 1.8x. (Evidently, the transposed particle array is better for the deposit even for scalar code.) The entire code had a speedup of 1.7x.

The Deposit procedure did not vectorize because of a well known data collision: two particles could be attempting to deposit to the same memory location simultaneously. One solution to this problem used on the Cray vector computers made use of the fact that each particle always deposits its own weights to distinct addresses. The conflicts occur from different particles. If one makes the number of distinct weights the inner loop, it could be safely vectorized. For the 3D electrostatic deposit, there are 8 distinct weights, which is half the vector length for KNC MIC. Thus we process particles in blocks and break up the inner loop into two loops. The first loop processes 32 particles in a block, calculates the addresses and weights, and stores them in small temporary arrays. The second loop reads the stored weights and addresses and deposits the charge, with a short inner loop of 8. The compiler vectorized the first loop, and chose not to vectorize the short loop, but the result was a speedup of only 1.6x, slightly worse than the unvectorized code.

In conclusion, the Fortran compiler based vectorization achieved about 75% of the performance achieved by using vector intrinsics. However, the performance was far from 16x improvement desired.

We continued with the Intel C compiler. The C version of the serial code pic3.c executed about 14% faster without vectorization than the Fortran version overall. We then tested the vectorization by compiling the code pic3.c with vectorization enabled (-O3 -mmic). The results showed no change in performance from the non-vectorized version.

We then transposed the particle array and made small simplifications in addressing as we had in the Fortran version, but in this case the results were worse by 42% overall. The particle push was dominant reason. It did not vectorize and was slower by nearly 60%. (Evidently, the transposed particle array is worse for the push for scalar code, unlike the case for the deposit.)

To encourage the compiler to vectorize, we rewrote the Push procedure to process particles in blocks and isolate the part of the code with indirect addressing into a short loop, as we had done for the deposit. This was better but still worse by 30% than the original scaler baseline push..

In conclusion, the C compiler based vectorization achieved a slowdown of 1.3x compared to the original baseline code versus a speedup of 2.0x by using vector intrinsics. Thus compiler vectorization with the C compiler is seriously deficient.

We then continued with the 3D electromagnetic (EM) code. We began with the Intel 16.0.2 Fortran compiler and compiled the Fortran version of the serial 3D electromagnetic (EM) code (npic3.f90) without vectorization -O3 -mmic -no-vec) and executed on a single core of the KNC MIC to establish a baseline performance benchmark.

We then wrote the 5 most important procedures using the Intel KNC MIC C/C++ vector intrinsics. For better performance on the KNC MIC, we transposed the particle array so that x locations for all the particles are stored together, followed by the y locations, etc. The relativistic results results with the Intel 16.0.2 C/C++ compiler on a single core showed speedups compared to the baseline as follows: Push, 2.8x, Charge Deposit, 2.5x, Current Deposit, 1.5x, Solver 2.7x, and FFT, 2.5x, with an overall speedup for the entire code of 2.2x. The best one expects for the KNC MIC is 16, so the results showed modest improvements. We then attempted to use compiler directives and code rewrites to see how close we could come to the hand-coded results.

We first compiled the code bpic3.f90 with vectorization enabled (-O3 -mmic). The initial results were very disappointing. The FFT vectorized, with a speedup of about 1.3x. However, the push routine slowed down by a factor of 12x, and the other 3 procedures had no substantial change in performance. However, if we transposed the particle array and made small simplifications in addressing, the results were much better. The Push procedure achieved a speedup of 2.2x. The Charge Deposit did not vectorize, but nevertheless had a speedup of 1.8x. (Evidently, the transposed particle array is better for the deposit even for scalar code.) The Current Deposit slowed down by a factor of 1.7x. The entire code had a speedup of 1.2x.

To encourage the compiler to vectorize the current, we rewrote the current deposit procedure to process particles in blocks and isolate the part of the code with indirect addressing into a short loop, as we had done for the charge deposit. This gave a speedup of 1.3 compared to the original scaler baseline push. The entire code speedup by a factor of 1.6x.

We continued with the Intel C compiler. The C version of the serial code bpic3.c executed about 10% slower without vectorization than the Fortran version overall. We then tested the vectorization by compiling the code bpic3.c with vectorization enabled (-O3 -mmic). The results showed no change in performance from the non-vectorized version.

We then transposed the particle array and made small simplifications in addressing as we had in the Fortran version, but in this case the results were worse by 17% overall. The current deposit was dominant reason. It was slower by nearly 40%.

To encourage the compiler to vectorize, we rewrote the Current deposit procedure to process particles in blocks and isolate the part of the code with indirect addressing into a short loop, as we had done for the charge deposit. This was better but still worse by 14% than the original scaler baseline push.

In conclusion, the C compiler based vectorization achieved a speedup of 1.1x compared to the original baseline code versus a speedup of 2.4x by using vector intrinsics. Thus compiler vectorization with the C compiler is quite deficient.

The benchmark programs (Table I) had a grid of 128x128x128with 56, 623,104 particles (27 particles per cell) and was run on one core of the 1.1 GHz Xeon Phi Coprocessor 5110P (Knights Corner) MIC. Intel Fortran 16.0.2 is used with -O3. Time reported is per particle per time step. A plasma in thermal equilibrium was simulated for 100 steps in the ES case and 250 steps in the EM case.

Electrostatic Code, with dt = 0.1

| | Scalar | Vectorized | KNC MIC |
|---|---|---|---|
| Push | 283.3 ns. | 147.6 ns. | 117.1 ns. |
| Deposit | 163.0 ns. | 90.5 ns. | 68.1 ns. |
| Reorder | 9.0 ns. | 24.4 ns. | 13.2 ns. |
| Total Particle | 455.4 ns. | 262.6 ns. | 198.5 ns. |

Total particle speedup was about 1.7x for the Vectorized code, 2.3x for the KNC MIC code.

Electromagnetic Code, with dt = 0.04, c/vth = 10

| | Scalar | Vectorized | KNC MIC |
|---|---|---|---|
| Push | 571.2 ns. | 256.8 ns. | 201.0 ns. |
| Deposit | 450.8 ns. | 324.3 ns. | 258.5 ns. |
| Reorder | 9.2 ns. | 22.2 ns. | 9.4 ns. |
| Total Particle | 1031.3 ns. | 603.3 ns. | 468.9 ns. |

Total particle speedup was about 1.7x for the Vectorized code, 2.2x for the KNC MIC code.

Table I: Benchmarks for particle processing of Vectorized codes using compiler directives and KNC MIC codes using vector intrinsics.

The benchmark codes are available at: http://picksc.idre.ucla.edu/.