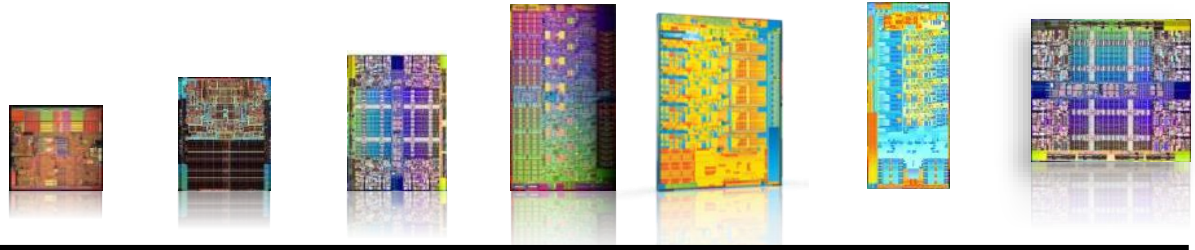
Seamless Parallelization and Vectorization Integration using OpenMP 4.0 on Intel® Architectures*

Xinmin Tian, Principal Engineer
Jim Cownie, Principal Engineer
Intel Corporation
November 18, 2014

SC'2014 OpenMP BoF

Parallel + Simd is the Path Forward

Intel® Xeon® and Intel® Xeon Phi™ Product Families are both going parallel



	Intel® Xeon® processor 64-bit	Intel® Xeon® processor 5100 series	Intel® Xeon® processor 5500 series	Intel® Xeon® processor 5600 series	Intel® Xeon® processor code-named Sandy Bridge EP	Intel® Xeon® processor code-named Ivy Bridge EP	Intel® Xeon® processor code-named Haswell EP	Intel® Xeon Phi™ coprocessor Knights Corner	Intel® Xeon Phi™ processor & coprocessor Knights Landing[1]
Core(s)	1	2	4	6	8	12	18	61	>61
Threads	2	2	8	12	16	24	36	244	>244
SIMD Width	128	128	128	128	256	256	256	512	512

*Product specification for launched and shipped products are available on ark.intel.com.

1. Not launched or in planning.

More cores → More Threads → Wider vectors

OpenMP* is one of most important vehicles for the parallel + SIMD path forward

Intel® C/C++ and Fortran Compilers for OpenMP 4.0*

- ✓ Places and thread affinity (14.0, 15.0)
- ✓ Main features of SIMD extensions (14.0, 15.0)
- ✓ Main features of Device / Accelerator extensions (14.0, 15.0)
- ✓ Combined simd, parallel, target, teams, distribute constructs (15.0)
- ✓ Taskgroup and dependent tasks (15.0)
- ✓ Sequentially consistent atomics (14.0, 15.0)
- ✓ Error handling (15.0)
- ✓ User-defined reductions (in later releases)
- ✓ Fortran 2003 support (15.0)

A Showcase: N-Body Parallelization+Vectorization

- Conceptually

  Foreach element ← Parallelize Here

  Foreach element ← Vectorize Here

  Accumulate contribution

  Store accumulated contribution

- Actual Code (locality…)

  Foreach chunk

  Foreach element ← Parallelize Here

  Foreach element in chunk ← Vectorize Here

  Accumulate contribution

  Store accumulated contribution

N-Body: Baseline Serial Code

```
#define CHUNK_SIZE  8192
int  body_start_index;
for (body_start_index = 0; body_start_index<global_number_of_bodies; body_start_index += CHUNK_SIZE)  {
      int i;
      int body_end_index = body_start_index + CHUNK_SIZE;
      for(i=starting_index; i<ending_index; i++) {
         int j;
         TYPE acc_x_0 = 0, acc_y_0 = 0, acc_z_0 = 0;
         for(j=body_start_index; j<body_end_index; j+=1) {
            TYPE delta_x_0 = Input_Position_X[(j+0)] - Input_Position_X[i];
            TYPE delta_y_0 = Input_Position_Y[(j+0)] - Input_Position_Y[i];
            TYPE delta_z_0 = Input_Position_Z[(j+0)] - Input_Position_Z[i];
            TYPE gamma_0 = delta_x_0*delta_x_0 + delta_y_0*delta_y_0 + delta_z_0*delta_z_0 + epsilon_sqr;
            TYPE s_0 = Mass[j+0]/(gamma_0 * SQRT(gamma_0));
            acc_x_0 += s_0*delta_x_0;
            acc_y_0 += s_0*delta_y_0;
            acc_z_0 += s_0*delta_z_0;
         }
         Output_Acceleration[3*(i+0)+0] += acc_x_0;
         Output_Acceleration[3*(i+0)+1] += acc_y_0;
         Output_Acceleration[3*(i+0)+2] += acc_z_0;
      }
}
```

N-Body: Parallel Code

```
#define CHUNK_SIZE  8192
int  body_start_index;
for(body_start_index = 0; body_start_index < global_number_of_bodies; body_start_index += CHUNK_SIZE)  {
   int i;
   int body_end_index = body_start_index + CHUNK_SIZE;
   #pragma omp parallel for private(i) schedule(guided)
   for(i=starting_index; i<ending_index; i++) {
      int j;
      TYPE acc_x_0 = 0, acc_y_0 = 0, acc_z_0 = 0;
      for(j=body_start_index; j<body_end_index; j+=1) {
         TYPE delta_x_0 = Input_Position_X[(j+0)] - Input_Position_X[i];
         TYPE delta_y_0 = Input_Position_Y[(j+0)] - Input_Position_Y[i];
         TYPE delta_z_0 = Input_Position_Z[(j+0)] - Input_Position_Z[i];
         TYPE gamma_0 = delta_x_0*delta_x_0 + delta_y_0*delta_y_0 + delta_z_0*delta_z_0 + epsilon_sqr;
         TYPE s_0 = Mass[j+0]/(gamma_0 * SQRT(gamma_0));
         acc_x_0 += s_0*delta_x_0;
         acc_y_0 += s_0*delta_y_0;
         acc_z_0 += s_0*delta_z_0;
      }
      Output_Acceleration[3*(i+0)+0] += acc_x_0;
      Output_Acceleration[3*(i+0)+1] += acc_y_0;
      Output_Acceleration[3*(i+0)+2] += acc_z_0;
   }
}
```

Example Timings
Host: 29.4sec (17.3x on 32 threads)
KNC: 39.4sec (on 240 threads)

N-Body: Parallel + SIMD Vector Code

```
#define CHUNK_SIZE  8192
int  body_start_index;
for(body_start_index = 0; body_start_index < global_number_of_bodies; body_start_index += CHUNK_SIZE)  {
   int i;
   int body_end_index = body_start_index + CHUNK_SIZE;
   #pragma omp parallel for private(i) schedule(guided)
   for(i=starting_index; i<ending_index; i++) {
      int j;    TYPE acc_x_0 = 0, acc_y_0 = 0, acc_z_0 = 0;
      #pragma omp simd reduction(+:acc_X_0, acc_Y_0, acc_Z_0)
      for(j=body_start_index; j<body_end_index; j+=1) {
         TYPE delta_x_0 = Input_Position_X[(j+0)] - Input_Position_X[i];
         TYPE delta_y_0 = Input_Position_Y[(j+0)] - Input_Position_Y[i];
         TYPE delta_z_0 = Input_Position_Z[(j+0)] - Input_Position_Z[i];
         TYPE gamma_0 = delta_x_0*delta_x_0 + delta_y_0*delta_y_0 + delta_z_0*delta_z_0 + epsilon_sqr;
         TYPE s_0 = Mass[j+0]/(gamma_0 * SQRT(gamma_0));
         acc_x_0 += s_0*delta_x_0;
         acc_y_0 += s_0*delta_y_0;
         acc_z_0 += s_0*delta_z_0;
      }
      Output_Acceleration[3*(i+0)+0] += acc_x_0;
      Output_Acceleration[3*(i+0)+1] += acc_y_0;
      Output_Acceleration[3*(i+0)+2] += acc_z_0;
   }
}
```

Example Timings
Host: 7.3sec/4.17sec (4x on 4-way vector, 7.1x on 8-way vector)
KNC: 2.53sec (15.6x on 16-way vector)

N-Body - Alignment Optimization

```
#define CHUNK_SIZE  8192
int  body_start_index;
for(body_start_index = 0; body_start_index < global_number_of_bodies; body_start_index += CHUNK_SIZE)  {
   int i;
   int body_end_index = body_start_index + CHUNK_SIZE;
   #pragma omp parallel for private(i) schedule(guided)
   for(i=starting_index; i<ending_index; i++) {
      int j;
      TYPE acc_x_0 = 0, acc_y_0 = 0, acc_z_0 = 0;
      #pragma omp simd reduction(+:acc_X_0, acc_Y_0, acc_Z_0) \
                              aligned(Input_Position_X,Input_Position_Y, Input_Position_Z, Mass:64)
      for(j=body_start_index; j<body_end_index; j+=1) {
         TYPE delta_x_0 = Input_Position_X[(j+0)] - Input_Position_X[i];
         TYPE delta_y_0 = Input_Position_Y[(j+0)] - Input_Position_Y[i];
         TYPE delta_z_0 = Input_Position_Z[(j+0)] - Input_Position_Z[i];
         TYPE gamma_0 = delta_x_0*delta_x_0 + delta_y_0*delta_y_0 + delta_z_0*delta_z_0 + epsilon_sqr;
         TYPE s_0 = Mass[j+0]/(gamma_0 * SQRT(gamma_0));
         acc_x_0 += s_0*delta_x_0;
         acc_y_0 += s_0*delta_y_0;
         acc_z_0 += s_0*delta_z_0;
      }
      Output_Acceleration[3*(i+0)+0] += acc_x_0;
      Output_Acceleration[3*(i+0)+1] += acc_y_0;
      Output_Acceleration[3*(i+0)+2] += acc_z_0;
   }
}
```

Example Timings
Host: 7.3sec (no gains)
KNC: 2.26sec (+12%)

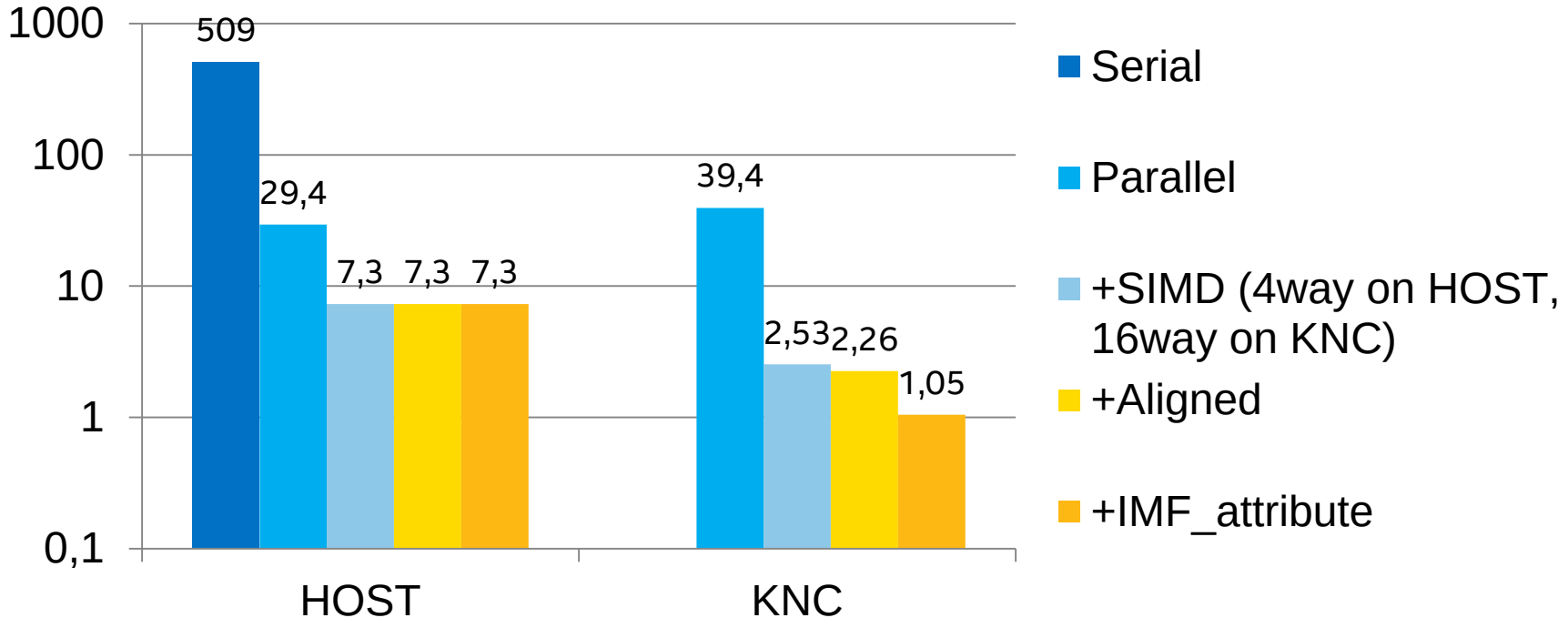
N-Body: Further Optimization

```
#pragma omp simd reduction(+:acc_X_0, acc_Y_0, acc_Z_0)
                 aligned(Input_Position_X, Input_Position_Y, Input_Position_Z, Mass:64)
for (j=body_start_index; j<body_end_index; j+=1) {
    TYPE delta_x_0 = Input_Position_X[(j+0)] - Input_Position_X[i];
    TYPE delta_y_0 = Input_Position_Y[(j+0)] - Input_Position_Y[i];
    TYPE delta_z_0 = Input_Position_Z[(j+0)] - Input_Position_Z[i];
    TYPE gamma_0 = delta_x_0*delta_x_0 + delta_y_0*delta_y_0 + delta_z_0*delta_z_0 + epsilon_sqr;
    TYPE s_0 = Mass[j+0]/(gamma_0 * SQRT(gamma_0));
    acc_x_0 += s_0*delta_x_0;
    acc_y_0 += s_0*delta_y_0;
    acc_z_0 += s_0*delta_z_0;
}
```

Example Timings
Host: 7.3sec (no gains)
KNC: 1.05sec (2.15x)

- Add these flags in the compilation:
  -fimf-domain-exclusion=15 -fimf-accuracy-bits=12
- Direct Link: http://software.intel.com/en-us/articles/advanced-optimizations-for-intel-mic-architecture-low-precision-optimizations
- Top-Level Reference: Programming and Compiling for Intel® Many Integrated Core Architecture

N-Body: Recap (Exec Time in LOG scale)



Seamless Parallelization and Vectorization Integration is essential

Still need to seek further opportunities

Conclusions

Parallelization and vectorization both matter

OpenMP* 4.0 can handle both

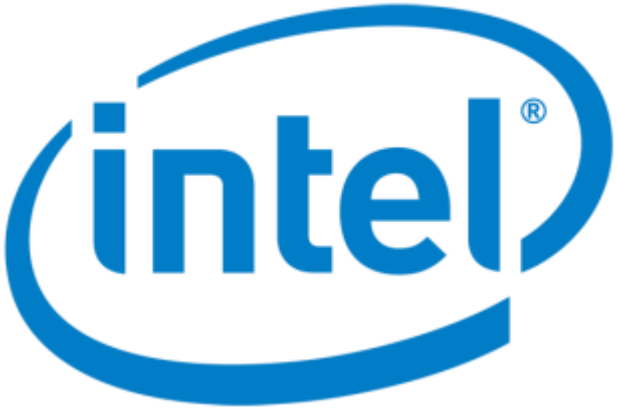
Intel® Compilers for OpenMP 4.0 are already here.

Legal Disclaimer & Optimization Notice

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804


intel