NEW ERA FOR OPENMP*: BEYOND TRADITIONAL SHARED MEMORY PARALLEL PROGRAMMING

Xinmin Tian, Senior Principal Engineer
Mobile Computing and Compiler, DPD/SSG, Intel Corporation
April 12th, 2016
Parallel + SIMD is the Path Forward
Intel® Xeon® and Intel® Xeon Phi™ Product Families are both going parallel

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>128</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>18</td>
<td>28</td>
<td>61</td>
<td>70+</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>128</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>18</td>
<td>28</td>
<td>244</td>
<td>280+</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>18</td>
<td>28</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>18</td>
<td>28</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>18</td>
<td>28</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>18</td>
<td>28</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>18</td>
<td>28</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>18</td>
<td>28</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>18</td>
<td>28</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>18</td>
<td>28</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>18</td>
<td>28</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>18</td>
<td>28</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>18</td>
<td>28</td>
<td>256</td>
<td>512</td>
</tr>
</tbody>
</table>

More cores ⇒ More Threads ⇒ Wider vectors
OpenMP* is one of most important vehicles for the parallel + SIMD path forward

*Product specification for launched and shipped products available on ark.intel.com. 1. Not launched or in planning.

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Programming Models Used at NERSC

MPI dominates

40% of projects use OpenMP*

(Courtesy of Yun (Helen) He, Alice Koniges, et. al., (NERSC) at OpenMPCon'2015)
What is X if Use MPI+X at NERSC

OpenMP is about 50%, out of all choices of X

Courtesy of Yun (Helen) He, Alice Koniges, et. al., (NERSC) at OpenMPCon'2015
Agenda

OpenMP* Programming Model's New Era

Programming Model Overview

OpenMP* Target (or Offload) Extensions
- Target Constructs and Usage Examples

OpenMP* SIMD Extensions: Putting Explicit SIMD Programming to Work
- SIMD Constructs and Usage Examples

OpenMP* Depend Extensions (Deal with cross-iteration/task dependencies)
- Depend Clauses and Usage Examples

Future Extensions and Summary
What is OpenMP*?

De-facto standard Application Programming Interface (API) to write shared memory parallel applications in C, C++, and Fortran

Consists of Compiler Directives, Runtime routines and Environment variables

Specification maintained by the OpenMP Architecture Review Board (http://www.openmp.org)

New OpenMP* ARB mission statement:

- “The OpenMP ARB mission is to standardize directive-based multi-language high-level parallelism that is performant, productive and portable.”

OpenMP* Specification Version 4.5 was launched at SC'2015
New Era - OpenMP* Programming Model

CPUs and All forms of accelerators/coprocessors, GPU, APU, GPGPU, FPGA, and DSP

Heterogeneous consumer devices

- Kitchen appliances, drones, signal processors, medical imaging, auto, telecom, automation, not just graphics engines**

**Courtesy of Michael Wong (IBM), et.al at LLVM Developer Conference Oct. 2015
OpenMP is widely supported by the industry, as well as the academic community.
OpenMP* Programming Model

Master thread spawns a team of threads / a league of thread teams as needed.

Parallelism is added incrementally until desired performance is achieved: i.e. the sequential program evolves into a parallel program.

- Outer 3-way parallelism
- Inner 9-way parallelism
- Outer 3-way parallelism
Teams + Parallel for: SAXPY – Accelerator Code

```c
int main(int argc, const char* argv[]) {
    float *x = (float*) malloc(n * sizeof(float));
    float *y = (float*) malloc(n * sizeof(float));
    // Define scalars n, a, b & initialize x, y

#pragma omp target data map(to:x[0:n])
#pragma omp target map(tofrom:y)
#pragma omp teams num_teams(num_blocks) num_threads(bsize)

    #pragma omp distribute
    for (int i = 0; i < n; i += num_blocks) {
        workshare (w/o barrier)
    }

#pragma omp parallel for
    for (int i = i; i < i + num_blocks; i++) {
        workshare (w/ barrier)
        y[i] = a * x[i] + y[i];
    }
}
free(x); free(y); return 0;
```
Data sharing/mapping: shared or distributed memory

Shared memory

- Threads have access to a shared memory
  - for shared data
  - each thread can have a temporary view of the shared memory (e.g., registers, cache, etc.) between synchronization barriers.

Threads have private memory

- for private data
- Each thread has a stack for data local to each task it executes

Distributed memory

- The corresponding variable in the device data environment may share storage with the original variable
- Writes to the corresponding variable may alter the value of the original variable
OpenMP* Components

<table>
<thead>
<tr>
<th>Directives (with Clauses)</th>
<th>Environment variables</th>
<th>Runtime functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Parallel / Teams</td>
<td>✓ Thread Settings</td>
<td>✓ Thread Management</td>
</tr>
<tr>
<td>✓ Worksharing</td>
<td>✓ Thread Controls</td>
<td>✓ Work Scheduling</td>
</tr>
<tr>
<td>✓ SIMD</td>
<td>✓ Work Scheduling</td>
<td>✓ Tasking</td>
</tr>
<tr>
<td>✓ Tasking</td>
<td>✓ Affinity</td>
<td>✓ Affinity</td>
</tr>
<tr>
<td>✓ Affinity</td>
<td>✓ Devices</td>
<td>✓ Devices</td>
</tr>
<tr>
<td>✓ Devices</td>
<td>✓ Cancellation</td>
<td>✓ Cancellation</td>
</tr>
<tr>
<td>✓ Cancellation</td>
<td>✓ Operational</td>
<td>✓ Cancellation</td>
</tr>
<tr>
<td>✓ Synchronization</td>
<td>✓ Stack size</td>
<td>✓ Locking</td>
</tr>
<tr>
<td>✓ ...</td>
<td>✓ ...</td>
<td>✓ ...</td>
</tr>
</tbody>
</table>
TARGET (OR OFFLOAD) EXTENSIONS FOR GPUs, COPROCESSORS AND SOCs
Emerging Heterogeneous Hardware Targets

OpenMP* 4.0 and 4.5 extensions for heterogeneous systems

Target device model:

- One host device and
- One or more target devices
OpenMP 4.0 / 4.5 Target Extensions

Offload code to run on a target device
- `omp target [clause[, clause],...]`
  - structured-block
- `omp declare target`
  - [function-definitions-or-declarations]

Map variables to a target device
- `map ([map-type:] list) // map clause`
  - map-type := alloc | tofrom | to | from
- `omp target data [clause[, clause],...]`
  - structured-block
- `omp target update [clause[, clause],...]`
- `omp declare target`
  - [variable-definitions-or-declarations]

Worksharing for acceleration
- `omp teams [clause[, clause],...]`
  - structured-block
- `omp distribute [clause[, clause],...]`
  - for-loops

Runtime support routines
- `void omp_set_default_device(int dev_num)`
- `int omp_get_default_device(void)`
- `int omp_get_num_devices(void)`
- `int omp_get_num_teams(void)`
- `int omp_get_team_num(void)`
- `int omp_is_initial_device(void)`

Environment variable
- Control default device through `OMP_DEFAULT_DEVICE`
- Accepts a non-negative integer value
Offloading and Device Data Mapping

Use target construct to

- Transfer control from the host to the target device
- Map variables between the host and target device data environments

Host thread spawns target (or offloaded) task

- Sync offloading (Thread waits for target task)
- Async offloading (Thread can continue without waiting for target task)

The map clauses determine how an original variable in a data environment is mapped to a corresponding variable in a device data environment.

```
#pragma omp target
map(alloc: ...) \ 
map(to: ...) \ 
map(from: ...) 
{ ... }
```
**Target + Map Usage Example**

```
define N 1000
#pragma omp declare target
float p[N], v1[N], v2[N];
#pragma omp end declare target
extern void init(float *, float *, int);
extern void output(float *, int);
void vec_mult()
{
    int i;
    init(v1, v2, N);
    #pragma omp target update to(v1, v2)
    #pragma omp target
    #pragma omp parallel for simd
    for (i=0; i<N; i++)
    {
        p[i] = v1[i] * v2[i];
        #pragma omp target update from(p)
    }
    output(p, N);
}
```

Indicate the parallel for simd loop is offloaded to coprocessor

Indicate that global variables are mapped to a device data environment for the whole program

Use target update to maintain consistency between host and device
Comparing OpenMP* with OpenACC*

OpenMP 4.0 / 4.5 – accelerating parallel for loop among teams and threads

```c
#pragma omp target teams map(X[0:N]) num_teams(numblocks)
#pragma omp distribute parallel for
  for (i=0; i<N; ++i) {
    X[i] += sin(X[i]);
  }
```

OpenACC 2.0 / 2.5 – accelerating a for loop among gangs + workers

```c
#pragma acc parallel copy(X[0:N]) num_gangs(numblocks)
#pragma acc loop gang worker
  for (i=0; i<N; ++i) {
    X[i] += sin(X[i]);
  }
```
PUTTING EXPLICIT SIMD PROGRAMMING TO WORK
Why SIMD Extensions? In a Time Before OpenMP* 4.0

Support required vendor-specific extensions

- Programming models (e.g., Intel® Cilk Plus)
- Compiler pragmas (e.g., #pragma vector)
- Low-level constructs (e.g., _mm_add_pd())

```c
#pragma omp parallel for
#pragma vector always
#pragma ivdep
for (int i = 0; i < N; i++) {
    a[i] = b[i] + ...;
}
```

Programmers need to rely on and trust the compiler to do the “right” thing.
Program Factors Impact on Vectorization

Loop-carried dependencies

```
DO I = 2, N
  A(I) = A(I-1) + B(I)
ENDDO
```

**Function calls**

```
for (i = 1; i < nx; i++) {
  x = x0 + i * h;
  sumx = sumx + func(x, y, xp);
}
```

**Pointer aliasing**

```
void scale(int *a, int *b)
{
  for (int i = 0; i < 1000; i++)
    b[i] = z * a[i];
}
```

**Unknown loop iteration count**

```
struct _x { int d; int bound; }; 
void doit(int *a, struct _x *x)
{
  for(int i = 0; i < x->bound; i++)
    a[i] = 0;
}
```

**Indirect memory access**

```
DO i=1, N
  A(B(i)) = A(B(i)) + C(i)*D(i)
ENDDO
```

**Outer loops**

```
DO I = 1, MAX
  DO J = 1, MAX
    D(I,J) = D(I,J) + 1;
  ENDDO
ENDDO
```
Vectorize Loop with Carried Dependencies

Dependencies may occur across loop iterations (a.k.a Loop-carried lexical forward / backward dependency)

The code below has a loop-carried lexical backward dependency. A loop iteration has to complete before the next iteration can run

```c
void lcd_ex(float* a, float* b, size_t n, int m, float c1, float c2) {
    size_t i;
    #pragma omp simd safelen(17) // programmer knows m >= 17
    for (i = m; i < n; i++) {
        a[i] = c1 * a[i - m] + c2 * b[i];
    }
}
```

- Simple verifying trick: can you perform the loop reversal w/o getting wrong results?
Vector code generation has become a more difficult problem increasing need for user guided explicit vectorization that maps concurrent execution to simd hardware.

Two fundamental problems:
- Data divergence
- Control divergence
SIMD Construct

Vectorize a loop

- Paritition loop into chunks that fit a SIMD vector register
- No parallelization of the loop body

Syntax (C/C++)
#pragma omp simd [clause[, clause],…]
  for-loops

Syntax (Fortran)
!$omp simd [clause[, clause],…]
  do-loops
[$omp end simd]
**SIMD Clauses**

- **safelen(length)**
  - Maximum number of iterations that can run concurrently without breaking a dependence

- **simdlen(length)**
  - Specify preferred length of SIMD registers used
  - Must be less or equal to safelen if both are present

- **linear(list[:linear-step])**
  - The variable's value depends on the iteration number ($x_i = x_{orig} + i \times \text{linear-step}$)

- **Reduction(operator: list)**
  - Eliminate loop-carried dependencies by doing partial computation and finalize the result
    
    $x = x + c \Rightarrow v_{priv\_x} = v_{priv\_x} + c; \quad \text{vec\_x} = \text{vec\_x} + v_{priv\_x}; \quad x = \text{horizontal\_vector\_add} (\text{vec\_x})$

- **aligned (list[:alignment])**
  - Specifies that the list items have a given alignment
  - Default is alignment for the architecture

- **collapse (n)**
Parallel for + SIMD Usage Example

```c
void sprod(float *a, float *b, int n) {
    float sum = 0.0f;
    #pragma omp parallel for simd reduction(+:sum)
    for (int k=0; k<n; k++)
        sum += a[k] * b[k];
    return sum;
}
```
SIMD Modifier for Loop Scheduling

```c
void sprod(float *a, float *b, int n) {
    float sum = 0.0f;
    #pragma omp parallel for simd reduction(+:sum) schedule(simd:static,5)
    for (int k=0; k<n; k++)
        sum += a[k] * b[k];
    return sum;
}
```

The new simd modifier makes the compiler and runtime to adjust the chunk size to match it with the length of the SIMD register.

- New chunk size becomes \( [\text{chunk\_size}/\text{simdlen}] \times \text{simdlen} \)
- AVX2: new chunk size may be the power of 2 and \( \geq 8 \)
- SSE: new chunk size may be the power of 2 and \( \geq 8 \)
Vortex Code: Outer Loop Vectorization

#pragma omp simd  // simd pragma for outer-loop at call-site of SIMD-function
for (int i = beg*16; i < end*16; ++i)
particleVelocity_block(px[i], py[i], pz[i],
destvx + i, destvy + i, destvz + i, vel_block_start, vel_block_end);

#pragma omp declare simd linear(velx, vely, velz) uniform(start, end) aligned(velx:64, vely:64, velz:64)
static void particleVelocity_block(const float posx, const float posy, const float posz,
float *velx, float *vely, float *velz, int start, int end) {
for (int j = start; j < end; ++j) {
const float del_p_x = posx - px[j];
const float del_p_y = posy - py[j];
const float del_p_z = posz - pz[j];
const float dxn = del_p_x * del_p_x + del_p_y * del_p_y + del_p_z * del_p_z + pa[j] * pa[j];
const float dxctaui = del_p_y * ty[j] - ty[j] * del_p_z;
const float dyctaui = del_p_z * tx[j] - tx[j] * del_p_x;
const float dzctaui = del_p_x * tx[j] - tx[j] * del_p_y;
const float dst = 1.0f/std::sqrt(dxn);
const float dst3 = dst*dst*dst;
*velx -= dxctaui * dst3;
*vely -= dyctaui * dst3;
*velz -= dzctaui * dst3;
}
}

KNC performance improvement
over 2X going
from inner to outer-loop vectorization
SIMD Function Vectorization

Declare functions to be compiled for calls from a SIMD loop

Syntax (C/C++):

- `#pragma omp declare simd [clause[[]], clause]....`
- `[#pragma omp declare simd [clause[[]], clause]....]`
- `[#pragma omp declare simd [clause[[]], clause]....]`
- `[...]`
- `function-definition-or-declaration`

Syntax (Fortran):

- `!$omp declare simd (proc-name-list)`

```c
float min(float a, float b) {
    return a < b ? a : b;
}
```

```c
float distsq(float x, float y) {
    return (x - y) * (x - y);
}
```

```c
void example() {
    #pragma omp parallel for simd
    for (i=0; i<N; i++) {
        d[i] = min(distsq(a[i], b[i]), c[i]);
    }
}
```

```c
vec8 min_vec(vec8 a, vec8 b) {
    return a < b ? a : b;
}
```

```c
vec8 distsq_vec(vec8 x, vec8 y) {
    return (x - y) * (x - y);
}
```

```c
vd = min_vec(distsq_vec(va, vb), vc)
```
SIMD Function Vectorization

#pragma omp declare simd
float sfoo(float x)
{
    … …
}

Scalar C function
sfoo(x0)->r0
sfoo(x1)->r1
sfoo(x2)->r2
sfoo(x3)->r3
sfoo(x4)->r4
… …
Scalar execution

Compiler created
__m128 vecfoo(__m128 vx)
{
    …
}

Vector C function

vecfoo(x0...x3)->r0...r3
vecfoo(X4...X7)->r4...r7
… …

Vector execution

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
ICC Optimization Report Improvements

Significant improvement in variable names and memory references reporting

- 16.0: remark #15346: vector dependence: assumed ANTI dependence between line 108 and line 116
- 17.0: remark #15346: vector dependence: assumed ANTI dependence between *(s1) (108:2) and *(r+4) (116:2)

More precise non-vectorization reasons

- E.g.: “exception handling for function call prevents vectorization”

Gather and partial scalarization reasons reporting (-qopt-report:5)

- 16.0: remark #15328: vectorization support: gather was emulated for the variable xyBase: indirect access [scalar_dslash_fused.cpp(334,27)]
- 17.0: remark #15328: vectorization support: gather was emulated for the variable <xyBase[xbOffset][c][s][1]>, indirect access, part of index is conditional [scalar_dslash_fused.cpp(334,27)]

- Other reasons are:
  - read from memory
  - nonlinearly computed
  - is result of a call to function
  - is linear but may overflow ← either in unsigned indexing or in address computation
  - is private ← memory privatization in explicit vectorization or serialized computation
OPENMP* **DEPEND** EXTENSIONS: DEAL WITH CROSS-ITERATION (OR TASK) DEPENDENCIES
Tasking with Dependences

The below code allows for more parallelism, as there now can be two tasks per k-th iteration.

```c
void task_in_parallel(float *a) {
    #pragma omp taskloop
    for (int k = 0; k < N; ++k) {
        #pragma omp task depend(out: a[k:1]) // Task-A
        a[k] = prepare_data(k...);
        #pragma omp task depend(in: a[k:1]) // Task-B
        do_work1_with_data(a[k]...);
        #pragma omp task depend(in: a[k:1]) // Task-C
        do_work2_with_data(a[k]...);
    }
}
```
Do-across Loop Parallelization

Dependencies may occur across loop iterations

The code below has a loop-carried backward dependency

```c
void lcd_ex(float* a, float* b, size_t n, int m, float c1, float c2) {
    size_t K;
    #pragma omp parallel for ordered(1)
    for (K = 17; K < n; K++) {
        #pragma omp ordered depend(sink: K-17)
        a[K] = c1 * a[K - 17] + c2 * b[K];
        #pragma omp ordered depend(source)
    }
}
```
FUTURE WORK AND SUMMARY
Mandelbrot: ~2698x Speedup on Xeon Phi™ -- Isn’t it Cool?

```c
#pragma omp declare simd uniform(max_iter), simdlen(32)
uint32_t mandel(fcomplex c, uint32_t max_iter)
{
    uint32_t count = 1; fcomplex z = c;
    while ((cabsf(z) < 2.0f) && (count < max_iter)) {
        z = z * z + c; count++;
    }
    return count;
}

#pragma omp parallel for schedule(guided)
for (int32_t y = 0; y < ImageHeight; ++y) {
    float c_im = max_imag - y * imag_factor;
#pragma omp simd simdlen(32)
for (int32_t x = 0; x < ImageWidth; ++x) {
    fcomplex in_vals_tmp = (min_real + x * real_factor) + (c_im * 1.0f);
    count[y][x] = mandel(in_vals_tmp, max_iter);
}

#pragma omp declare simd uniform(max_iter), simdlen(32)
uint32_t mandel(fcomplex c, uint32_t max_iter)
{   uint32_t count = 1; fcomplex z = c;
    while ((cabsf(z) < 2.0f) && (count < max_iter)) {
        z = z * z + c; count++;
    }
    return count;
}
```

Mandelbrot Normalized Speedup with OMP PAR+SIMD on Xeon Phi(TM)

Intel Xeon Phi™ system, Linux64, 64 cores running 256 threads at 1.30GHz, 32 KB L1, 1024 KB L2 per core. Intel C/C++ Compiler 16.0 Update 2 build.
Future Work

Fast memory (such as KNL HBM and GFX SLM) support
Struct deep-copy for offloading
Concurrent data mapping for offloading
Device(archtype: subtype, ID)
SIMD extensions for function pointer and virtual function
Task reduction
Conflict/Expand/Compress support
AOS -> SOA data layout annotation and conversion for Effective SIMD
- Intel has provided SIMD Data Layout Template Library

... ... ... ...
Summary

The reality:

- There is no one single solution that would make all programmers happy after decades of trying.
- There is no free lunch for effectively utilizing SIMD HW, multicore CPUs, accelerators and GPUs.
- There are many emerging programming models for multicore CPUs, accelerators and GPUs.
- Programming languages and compilers are driven by hardware and application
- The incremental approach of applying the learnings from HPC and graphics is working

OpenMP* Explicit Target, Parallel and SIMD Programming paves the way to achieve “close to metal performance”
Resources

OpenMP Information

- Putting Vector Programming to work with OpenMP* SIMD
  - software.intel.com/sites/default/files/managed/77/e7/parallel_mag_issue22.pdf
- OpenMP* API Version 4.5: A Standard Evolves
- From Knights Corner to Knights Landing: Prepare for the Next Generation of Intel® Xeon Phi™ Technology
  - software.intel.com/sites/default/files/managed/4c/1c/parallel_mag_issue20.pdf

Code Modernization Links

- Modern Code Developer Community
  - software.intel.com/modern-code
- Intel Code Modernization Enablement Program
  - software.intel.com/code-modernization-enablement
- Intel Parallel Computing Centers
  - software.intel.com/ipcc
- Technical Webinar Series Registration
- Intel Parallel Universe Magazine
  - software.intel.com/intel-parallel-universe-magazine
THANKS & QUESTIONS?
Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY. RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2016, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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