Eight Key Policies to Modernize Code on Multi-Core and Many-Core Platforms

Zhe Wang – Software Application Engineer, Intel Corporation
Shan Zhou – Software Application Engineer, Intel Corporation

SFTS003
Agenda

• Why Modernization Code

• Methodology

• Modernization

• Summary
Agenda

• Why Modernization Code

• Methodology

• Modernization

• Summary
Performance and Programmability for Highly-Parallel Processing Now

How do we attain extremely high compute density for parallel workloads AND maintain the robust programming models and tools that developers crave?

<table>
<thead>
<tr>
<th>Core(s)</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>12</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threads</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>24</td>
<td>36</td>
</tr>
<tr>
<td>SIMD Width</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>256</td>
<td>256</td>
<td>256</td>
</tr>
</tbody>
</table>

2006

More cores >> More Threads >> Wider vectors

2015
Future Architecture Analysis

- More cores and more threads
- Wider vector instructions
- Higher memory bandwidth
- Higher integration and complexity in one chip and node
- Common instructions, languages, directives, libraries & tools
How Can I Achieve High Performance?

How to get benefit from Exascale with your code in the future?

Lot of performance is being left on the table.

Modernization of your code is the solution.
Agenda

• Why Modernization Code

• Methodology

• Modernization

• Summary
Methodology: A Cycle Model for Modernization Code

- Analysis feature on Intel® Architecture by tools
  - Hotspots for thread, task, memory, I/O or process on single node
  - Hotspots for scale-out on multi nodes
  - Match between algorithm and architecture of application
- Design and optimize code for future architectures

Question assumptions using repeatable and representative benchmarks
Optimization: A Top-down Approach

System Configuration
- Network I/O
- Disk I/O
- Database Tuning
- OS

App Design
- App Server Tuning
- Driver Tuning
- Parallelization
- Hiding Data transfer

Cache-Based Tuning
- Low Level tuning by Intel Intrinsic

System

Application

Processor
Modernization Code With 4D

- Choose Proper compiler option
- Use Optimized Library (Intel® Math Kernel Library)
- Choose right precision ......
- Remove IO bottleneck
- Remove unnecessary computing

- Choose the proper parallelization method
- Load balance
- Synchronization overhead
- Thread Binding

- Auto-vectorization
- Intel® Cilk™ Plus Array Notations
- Elemental functions
- Vector class
- Intrinsics

- Data Alignment
- Prefetch
- Cache Blocking
- Data restructure: aos2soa
- Streaming Store

- Serial and Scalar
- Parallelism
- Vectorization
- Memory Access

IDF15
Agenda

• Why Modernization Code

• Methodology

• Modernization

• Summary
Build a Solid Foundation for Modernize Code
by Serial and Scalar optimization

#1 Policy - Get benefit from Intel® Tools
  • Intel Compiler Tools such as Intel® Parallel Studio
  • Optimized library such as Intel® Math Kernel Library, Intel® Threading Building Blocks

#2 Policy - Remove data transfer bottleneck
  • File read | write
  • Data transfer
Deep Learning

Use Intel® Math Kernel Library random generator instead of Glibc rand(), further increasing parallelism.

```c
#ifdef USE_MKL_RAND
    vs1NewStream(4Randomstream, /*VSL_BRNG_MT19937*/VSL_BRNG_SFM319937, RANDSEED);
#endif

#ifdef USE_MKL_RAND
    vsRngUniform (VSL RNG METHOD UNIFORM_STD, Randomstream, samples * J, rand_array, 0.0f, 1.0f);
#else
    for (s = 0; s < samples; s++)
    {
        for (j = 0; j < J; j++)
        {
            #ifndef OPT7
                rand_array[s * J + j] = (float)rand() / RAND_MAX;
            #else
                rand_array[s * J + j] = (float)rand() * rcp_randmax;
            #endif
        }
    }
#endif
```

1.63X
Case Study: Reduce I/O Bottleneck

Reduce I/O bottleneck by using double buffer even multi buffer

Original Code:

1.3X

Double Buffer:

1.08X

Multi Buffer:
Build a Structure of Modernize Code based on Future Arch by Parallelism

#3 Policy - Choose the proper parallelism method according algorithm
- Automatically Parallelism by Tools. Such as Intel® Integrated Performance Primitives/Intel® Math Kernel Library, Compiler
- Multi-threads (OpenMP®, Pthread, Intel® Cilk™ Plus, Intel® Threading Building Blocks)
- Multi-processes (MPI)
- Hybrid (Process + Threads)

#4 Policy - Hiding data transfer

#5 Policy - Balance between calculate, communication, system call
- Tune Load Balance
- Remove or reduce system cost such as locks, waits, barrier overhead or launch overhead
How To: Parallelism

Given a large scale workload, key factors for good scalability:

- Computation Granularity
- Load Balance
- Communication and I/O Hiding
- Synchronization Overhead
- Parallelization Mode

Good Scalability

General parallelization mode:

- Multi-task
  - Intel® Threading Building Blocks
  - Intel® Cilk™ Plus
  - OpenMP®
  - Pthread
- Multi-thread

Hybrid

Given a large scale workload, key factors for good scalability:
Increase Parallelism for your code

- 2 Level MPI Network Architecture to reduce overhead
- Communication benefit while using huge MPI processes

Total $N = M_1 \times M_2$ MPI Process

**Parent process**
- call `kernal_computer()`
- call `transfer_data()`

**Son process**
- Spawn $M_2$ Son Process for each Parent Process
- call `get_data_from_parent()`
- call `kernal_computer()`
- call `update_data_to_parent()`

Total $M_1$ MPI Process

Total $M_1 \times M_2$ MPI Process

**Total $N = M_1 \times M_2$ MPI Process**

**Total $M_1$ MPI Process**

**Total $M_1 \times M_2$ MPI Process**
Case Study: Pipeline to Hide I/O latency

PSTM

- Pipeline to hiding communication between Intel® Xeon™ and Intel® Xeon Phi™ processors
- Unibite Binary code for Optimized kernel code with Intel® Compiler

```cpp
PstmKernel::run(){
    Input data Loop: {
        Get Input data Buffer
        if( Worker on Xeon Phi){
            COIBufferWrite(coi_buffers[Worker id], 0, (void *)buff, BUFF_FLOAT_len,
            COI_COPY_USE_DMA, 0, NULL, NULL);
            COIPipelineRunFunction(pipelines[worker id], pstm_kernel, 6,
            coi_buffers+_thread_Index*6, coi_flags, 0,
            NULL, NULL, 0, NULL, 0, &cmplt[_thread_Index]); // Level 3 parallel
            COIEventWait(1, &cmplt[worker id], -1, true, NULL, NULL);
        } else{ /// worker on Xeon
            #pragma omp parallel default(shared) num_threads(pstm_data_cpu_int[0]) // Level 3 parallel
                {
                    pstm_kernel
                }
        }
    } /// No Input Data.
}
```
Load balance between Coprocessor and Host

Fine Grained Parallelism and move some unnecessary model from MIC to Host

```c
If(myid.eq.MICID) then ! execute on Xeon Phi.
Offload input data to MIC
{
  call phase
  call amplenowave
  call mixing
}
else ! execute on Xeon
{
  get Input data of Xeon
  call phase
  call amplenowave
  call mixing
}
end if
merge result of CPU and Xeon Phi
```
Reasonable data structure for Modernize Code by Memory Access Optimized

#6 Policy - Choose suitable Memory access model
- Streaming Store: using non temporal pragma and compiler option “-opt-streaming-stores always” to improve memory bandwidth
- Reduce memory access by using different data types
- Huge Page Setting for MIC to improve TLB hit ratio
- Data restructure: AOS→SOA

#7 Policy - Improve Cache efficiency
- Cache blocking, improves data locality to reduce cache miss
- Pre-fetch: by compiler option, pragma prefetch or _mm_prefetch intrinsic
- Loop fusion and Loop Split: to reduce memory traffic by increasing locality if possible
Cache Blocking to improve cache efficiency

- Blocking 2D Matrix Data Instruct and get 1.08X improved

```c++
for (int iX=x0; iX<x1; ++iX) {
for (int iY=y0; iY<y1; ++iY) {
    grid[iX][iY].collide(getStatistics());
    lbHelpers<T,Lattice>::swapAndStream2D(grid, iX, iY);
}
}
```
Data Restructure

- Minimize memory access and replace them with temporary variables if possible

\[
\text{do } i=1, N \\
\quad \ldots.. \\
\quad f(i)=\ldots.. \\
\quad v(i)=\ldots.. f(i) \\
\quad g(i)=f(i)\times v(i) \\
\quad \ldots.. \\
\text{enddo}
\]

- Replace 2D struct with 1D array via array of pointer

\[
\text{for } (iy=0; iy<nz; iy++) \\
\quad y = y0 + dy*iy; \\
\quad \ldots.. \\
\quad nz = sqrt(z^2+hs^2*hs); \\
\quad ttt[iy] = 0.5 + dz/dt; \\
\quad for (iz=0; iz<nz; iz++) \\
\quad \quad itt[iz] = ttt[iy]; \\
\quad \quad for (iz=0; iz<nz; iz++) \\
\quad \quad \quad if (itt[iz] < nc) \\
\quad \quad \quad \quad \quad modl[iy] += data[iy][itt[iz]]; \\
\quad \quad \quad \text{modl[0:nz] := data[tvp_iy_x_nt + itt[0:nz]];}
\]

\[
\text{do } i=1, N \\
\quad \ldots.. \\
\quad f(i)=\ldots.. \\
\quad v = \ldots.. f(i) \\
\quad g(i)=f(i)\times v \\
\quad \ldots.. \\
\text{enddo}
\]
High efficiency binary code for Modernize Code by Vectorization

#8 Policy – Vectorization and vectorization
- Make full use of wide vector
- Remove data dependency
- Add compiler option
- Add pragma to help compiler auto-vectorization
- Avoid non-contiguous memory access
- Choose the proper method to deep dive vectorization, for example, by intrinsic

Important for high instructor/data width of core.
More VPU on next gen Intel® Xeon Phi™ Processor

- Up to 72 new Intel® Architecture cores
- 36MB shared L2 cache
- Full Intel® Xeon™ processor ISA compatibility through Intel® Advanced Vector Extensions 2
- Extending Intel® Advanced Vector Extensions architecture to 512b (AVX-512)
- Based on Silvermont microarchitecture:
  - 4 threads/core
  - Dual 512b Vector units/core
- 6 channels of DDR4 2400 up to 384GB
- 36 lanes PCI Express® (PCIe®) Gen 3
- 8GB/16GB of extremely high bandwidth on package memory
- Up to 3x single thread performance improvement over prior gen \(^1,2\)
- Up to 3x more power efficient than prior gen \(^1,2\)

1. As projected based on early product definition and as compared to prior generation Intel® Xeon Phi™ Coprocessors.
2. Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.
Example - Common Used Skills

Compiler Option

Add compiler option to vectorize code

- "-O2" or higher to default vectorization
- "-no-vec" to disable vectorization
- "-qopt-report=n" for detailed information
- -ansi-alias : assert lack of type casts for type disambiguation
- -fno-alias : assert no function argument aliasing

Use pragma

Add pragma to vectorize the loops

- #pragma ivdep
- #pragma simd
- #pragma vector align asserts that data within the following loop is aligned
- #pragma novector, disable vectorization for small loops, such as loop count<8 for DP or <16 for SP on Intel® Xeon Phi™ coprocessor

Other Skills

Vectorize the code manually

- Loop interchange can help for vectorization sometimes
- Try to use gather/scatter intrinsic if you can't make sure contiguous memory access
Restructure Source code to auto vectorization by Compiler

The code can’t be auto-vectorized because of branch in the inner loop. But it can be resolved by pre-calculating the range of inner loop to achieve auto-vectorization by compiler.
# Case Study: Loop Interchange to Achieve SIMD

## Example: Typical Matrix Multiplication

```c
void matmul_slow(float *a[], float *b[], float *c[])
{
    int N = 100;
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            for (int k = 0; k < N; k++)
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
}
```

## Example: After interchange

```c
void matmul_fast(float *a[], float *b[], float *c[])
{
    int N = 100;
    for (int i = 0; i < N; i++)
        for (int k = 0; k < N; k++)
            for (int j = 0; j < N; j++)
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
}
```

### MLFMA: small matrix multiply
Achieve vectorization by data restructure and loop Interchange

#### original code, matrix multiplication:

```c
for(int n=0;n<len;n++)
for(int i=0;i<ilen;i++)
for(int j=0;j<jlen;j++)
for(int k=0;k<klen;k++)
c[n][i][j]=c[n][i][j]+a[n][i][k] * b[n][k][j];
```

#### Step 1: memory copy, data restructure

```c
for(int i=0;i<ilen;i++)
    for(int k=0;k<klen;k++)
        aaa[i][k][n]=a[n][i][k];

for(int n=0;n<len;n++)
    for(int k=0;k<klen;k++)
        bbb[k][j][n]=b[n][k][j];
```

#### Step 2: compute

```c
for(int i=0;i<ilen;i++)
    for(int j=0;j<jlen;j++)
        for(int n=0;n<len;n++)
            ccc[i][j][n]=0;

for(int i=0;i<ilen;i++)
    for(int j=0;j<jlen;j++)
        for(int k=0;k<klen;k++)
            for(int n=0;n<len;n++)
                ccc[i][j][n]+=aaa[i][k][n]*bbb[k][j][n];
```

#### Step 3: memory copy back

```c
for(int i=0;i<len<3;i++)
for(int j=0;j<len;j++)
    c[n][i][j]=ccc[i][j][n];
```
Case Study: Other Vectorization Method

Merge the similar operations from different array together and vectorize

```c
#define KIC_
__m512d tmp1;
tmp1 = mm512_loadupsd_pd(tmp1, &cell[1]);
tmp1 = mm512_loadupsd_pd(tmp1, &cell[1]+6);
__m512d tmp_omega = mm512_set_pd(omega_0, omega_1, omega_2, omega_3, omega_4, omega_5, omega_6, omega_7);
__m512d tmp_omega1 = mm512_set_pd(omega_0, omega_1, omega_2, omega_3, omega_4, omega_5, omega_6, omega_7);
__m512d tmp_omega2 = mm512_set_pd(omega_0, omega_1, omega_2, omega_3, omega_4, omega_5, omega_6, omega_7);
__m512d tmp_omega3 = mm512_set_pd(omega_0, omega_1, omega_2, omega_3, omega_4, omega_5, omega_6, omega_7);

__m512d tmp_1 = mm512_mul_pd(tmp1, tmp_omega);
__m512d tmp_0 = mm512_add_pd(tmp1, mm512_sub_pd(tmp_1, mm512_add_pd(tmp_1, tmp_omega)));

__m512d tmp_2 = mm512_mul_pd(tmp1, tmp_omega1);
__m512d tmp_3 = mm512_add_pd(tmp1, mm512_sub_pd(tmp_2, mm512_add_pd(tmp_2, tmp_omega1)));

__m512d tmp_4 = mm512_mul_pd(tmp1, tmp_omega2);
__m512d tmp_5 = mm512_add_pd(tmp1, mm512_sub_pd(tmp_4, mm512_add_pd(tmp_4, tmp_omega2)));

__m512d tmp_6 = mm512_mul_pd(tmp1, tmp_omega3);
__m512d tmp_7 = mm512_add_pd(tmp1, mm512_sub_pd(tmp_6, mm512_add_pd(tmp_6, tmp_omega3)));

__m512d tmp_8 = mm512_mul_pd(tmp1, tmp_omega);
__m512d tmp_9 = mm512_add_pd(tmp1, mm512_sub_pd(tmp_8, mm512_add_pd(tmp_8, tmp_omega)));

__m512d tmp_10 = mm512_mul_pd(tmp1, tmp_omega1);
__m512d tmp_11 = mm512_add_pd(tmp1, mm512_sub_pd(tmp_10, mm512_add_pd(tmp_10, tmp_omega1)));

__m512d tmp_12 = mm512_mul_pd(tmp1, tmp_omega2);
__m512d tmp_13 = mm512_add_pd(tmp1, mm512_sub_pd(tmp_12, mm512_add_pd(tmp_12, tmp_omega2)));

__m512d tmp_14 = mm512_mul_pd(tmp1, tmp_omega3);
__m512d tmp_15 = mm512_add_pd(tmp1, mm512_sub_pd(tmp_14, mm512_add_pd(tmp_14, tmp_omega3)));

__m512d tmp_16 = mm512_mul_pd(tmp1, tmp_omega);
__m512d tmp_17 = mm512_add_pd(tmp1, mm512_sub_pd(tmp_16, mm512_add_pd(tmp_16, tmp_omega)));

__m512d tmp_18 = mm512_mul_pd(tmp1, tmp_omega1);
__m512d tmp_19 = mm512_add_pd(tmp1, mm512_sub_pd(tmp_18, mm512_add_pd(tmp_18, tmp_omega1)));

__m512d tmp_20 = mm512_mul_pd(tmp1, tmp_omega2);
__m512d tmp_21 = mm512_add_pd(tmp1, mm512_sub_pd(tmp_20, mm512_add_pd(tmp_20, tmp_omega2)));

__m512d tmp_22 = mm512_mul_pd(tmp1, tmp_omega3);
__m512d tmp_23 = mm512_add_pd(tmp1, mm512_sub_pd(tmp_22, mm512_add_pd(tmp_22, tmp_omega3)));

__m512d tmp_24 = mm512_mul_pd(tmp1, tmp_omega);
__m512d tmp_25 = mm512_add_pd(tmp1, mm512_sub_pd(tmp_24, mm512_add_pd(tmp_24, tmp_omega)));

__m512d tmp_26 = mm512_mul_pd(tmp1, tmp_omega1);
__m512d tmp_27 = mm512_add_pd(tmp1, mm512_sub_pd(tmp_26, mm512_add_pd(tmp_26, tmp_omega1)));

__m512d tmp_28 = mm512_mul_pd(tmp1, tmp_omega2);
__m512d tmp_29 = mm512_add_pd(tmp1, mm512_sub_pd(tmp_28, mm512_add_pd(tmp_28, tmp_omega2)));

__m512d tmp_30 = mm512_mul_pd(tmp1, tmp_omega3);
__m512d tmp_31 = mm512_add_pd(tmp1, mm512_sub_pd(tmp_30, mm512_add_pd(tmp_30, tmp_omega3)));

```
Agenda

• Why Modernization Code

• Methodology

• Modernization

• Summary
Summary

- More cores
- More Threads
- Wider vectors
- Higher Memory Bandwidth

Intel® Architecture

+  

High Performance

- Parallelization
- Vectorization
- Memory Access Efficiency
- Remove I/O Bottleneck

Modernization of Your Code
Next Steps

• Try these policies to modernize your code on Intel® Xeon™ Processor and Intel® Xeon Phi™ Coprocessor

• Experience the hands-on lab of software optimization methodology tomorrow

<table>
<thead>
<tr>
<th>Session ID</th>
<th>Title</th>
<th>Day</th>
<th>Time</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFTL002</td>
<td>Hands-on Lab: Software Optimization Methodology for Multi-core and Many-core Platforms</td>
<td>Thurs</td>
<td>13:15</td>
<td>Lab Song 2</td>
</tr>
</tbody>
</table>
Additional Sources of Information

• A PDF of this presentation is available from our Technical Session Catalog: www.intel.com/idfsessionsSZ. This URL is also printed on the top of Session Agenda Pages in the Pocket Guide.

Legal Notices and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

Statements in this document that refer to Intel's plans and expectations for the quarter, the year, and the future, are forward-looking statements that involve a number of risks and uncertainties. A detailed discussion of the factors that could affect Intel's results and plans is included in Intel's SEC filings, including the annual report on Form 10-K.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Intel, Xeon, Xeon Phi, Cilk, Core and the Intel logo are trademarks of Intel Corporation in the United States and other countries.

*Other names and brands may be claimed as the property of others.

© 2015 Intel Corporation.
Legal Disclaimer

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 SSE3 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
Risk Factors

The above statements and any others in this document that refer to plans and expectations for the first quarter, the year and the future are forward-looking statements that involve a number of risks and uncertainties. Words such as "anticipates," "expects," "intends," "plans," "believes," "seeks," "estimates," "may," "will," "should" and their variations identify forward-looking statements. Statements that refer to or are based on projections, uncertain events or assumptions also identify forward-looking statements. Many factors could affect Intel's actual results, and variances from Intel's current expectations regarding such factors could cause actual results to differ materially from those expressed in these forward-looking statements. Intel presently considers the following to be important factors that could cause actual results to differ materially from the company's expectations.

Demand for Intel's products is highly variable and could differ from expectations due to factors including changes in the business and economic conditions; consumer confidence or income levels; customer acceptance of Intel's and competitors' products; competitive and pricing pressures, including actions taken by competitors; supply constraints and other disruptions affecting customers; changes in customer order patterns including order cancellations; and changes in the level of inventory at customers. Intel's gross margin percentage could vary significantly from expectations based on capacity utilization; variations in inventory valuation, including variations related to the timing of qualifying products for sale; changes in revenue levels; segment product mix; the timing and execution of the manufacturing ramp and associated costs; excess or obsolete inventory; changes in unit costs; defects or disruptions in the supply of materials or resources; and product manufacturing quality/yields. Variations in gross margin may also be caused by the timing of Intel product introductions and related expenses, including marketing expenses, and Intel's ability to respond quickly to technological developments and to introduce new features into existing products, which may result in restructuring and asset impairment charges. Intel's results could be affected by adverse economic, social, political and physical/infrastructure conditions in countries where Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Results may also be affected by the formal or informal imposition by countries of new or revised export and/or import and doing-business regulations, which could be changed without prior notice. Intel operates in highly competitive industries and its operations have high costs that are either fixed or difficult to reduce in the short term. The amount, timing and execution of Intel's stock repurchase program and dividend program could be affected by changes in Intel's priorities for the use of cash, such as operational spending, capital spending, acquisitions, and as a result of changes to Intel's cash flows and changes in tax laws. Product defects or errata (deviations from published specifications) may adversely impact our expenses, revenues and reputation. Intel's results could be affected by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust, disclosure and other issues. An unfavorable ruling could include monetary damages or an injunction prohibiting Intel from manufacturing or selling one or more products, precluding particular business practices, impacting Intel's ability to design its products, or requiring other remedies such as compulsory licensing of intellectual property. Intel's results may be affected by the timing of closing of acquisitions, divestitures and other significant transactions. A detailed discussion of these and other factors that could affect Intel's results is included in Intel's SEC filings, including the company's most recent reports on Form 10-Q, Form 10-K and earnings release.