NERSC Threading Workshop

TCG Micro
SSG DPD
NERSC, March 2015

Outline

Part 1

- Introduction
  - Review of hardware & parallel programming models
  - NERSC NECAP
- Principles of High Performance Parallel Programming (HPPP)
- EMGeo: basic
- EMGeo: intermediate

Part 2

- Know MIC and programming model
- Multi-level parallelism: Nested OpenMP

Part 3

- PARSEC
- EMGeo: advanced
- Conclusions

About the Presenter: Jeongnim Kim, PhD

- Sr. HPC application Engineer at Joe Curley's MICRO (MIC Ramp Organization) group; working for code modernization and optimization on Xeon and Xeon Phi™
- Has been active in computational materials science and HPC since 1993
  - Used most of parallel computing platforms at DOE and NSF HPC centers: Intel Paragon, Cray T3D/T3E, SGI Origin 2000, Intel Itanium, IBM Power 3-7, Cray XT/XE/XK/XC, and IBM Blue Gene Q
  - Distributed programming on Intel Paragon (1994); OpenMP programming on SGI Origin (1998)
- Prior to joining Intel in April 2014
  - Worked for Oak Ridge National Laboratory (ORNL) and National Center for Supercomputing Applications and Materials Computation Center, University of Illinois, Urbana-Champaign
  - Developed QMCPACK and led Quantum Monte Carlo collaboration between ORNL, ANL, LLNL, Sandia and UI
- PhD in condensed matter theory from the Ohio State University, USA, and a BS in Physics from Korea Advanced Institute of Science and Technology, Korea

How to exploit OpenMP* for high-performance parallel applications

Someone said

- Shared-memory programming models on multi- and many-core processors are critical. You must hybridize your application!
- OpenMP* is so easy. All you have to do is to find loops and put `OMP parallel do` over the loops.
- OpenMP* 4.x let you express your intention of vectorization of the loops and compilers can vectorize them.
- MKL comes with threaded numerical libraries. Use threads with GEMM or FFT.

Then, you are thinking

"I tried OpenMP but the performance is much worse than MPI. Where is the performance?"

Distributed-shared-memory programming (a.k.a., hybrid programming)

"I tried OpenMP but the performance is much worse than MPI. Why bother?"

A good question! But,

- The laws of physics say otherwise: finite electron velocity, limited parallel channels, multiple hops, ….
- Just ask how many instructions are needed to execute a put or get. E.g., a simple send/recv = memory-> [MPI buffer]^P -> memory.
- Moving data with MPI must be more expensive than memory to cache.

So, what is going on?

This workshop aims to

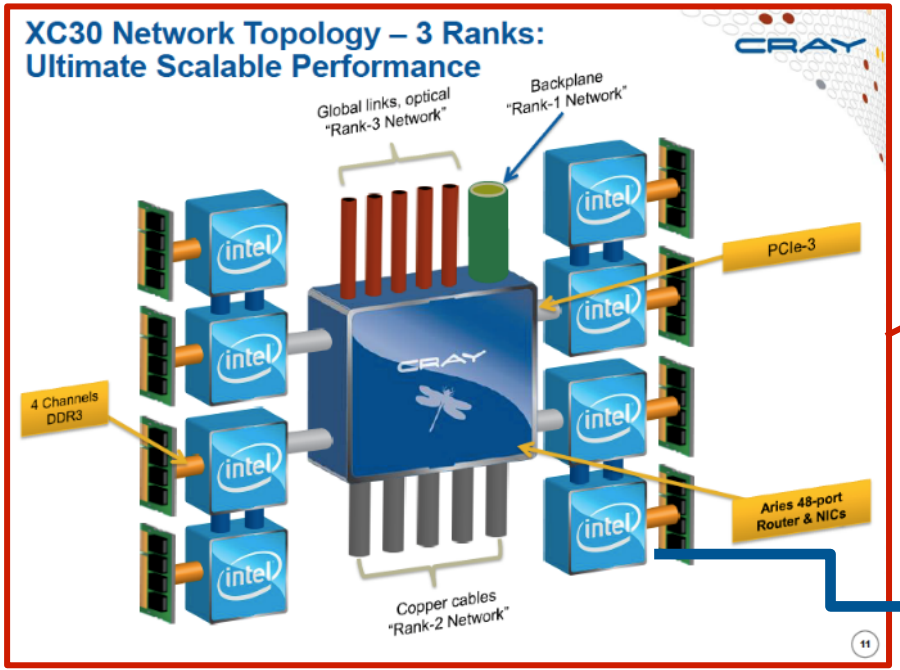
- Refresh your knowledge of hardware, software and parallel programming
- Remind you of Parallel computing 101
- Use NESAP codes to discuss *processes* to exploit modern hardware
- Introduce advanced OpenMP* concepts and techniques
- Promote code design and thinking out of box

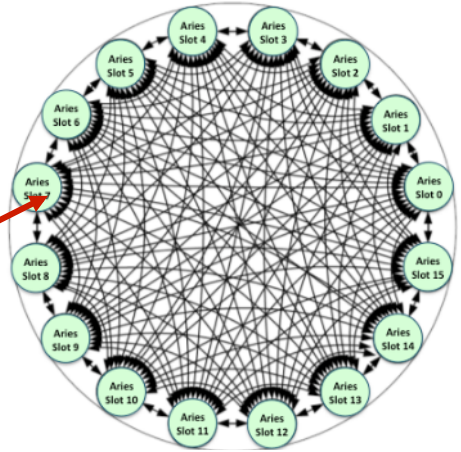
Disclaimers

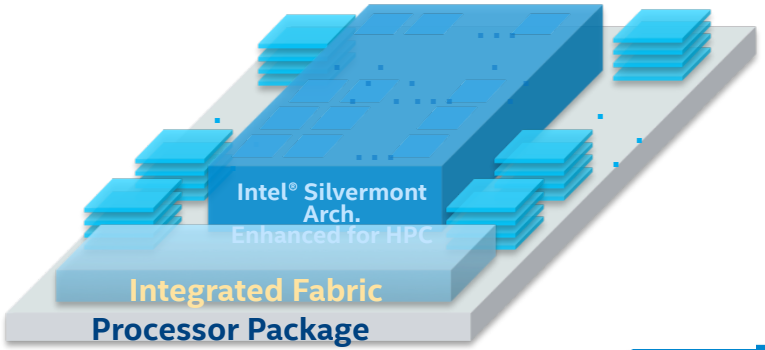
- OpenMP* (MPI) is selected as the de-facto standard for shared (distributed) parallel programming model.
- Processes based on the experiences with numerous HPC applications.
- Materials using MPI/Fortran applications chosen by NERSC
- Each process will be marked by the target developers

Dev0	New member of the team; cannot find code documentations (or hidden) and everyone is busy.
Dev1	Computer scientist or engineer; know nothing about the application (science); have to work with the “domain scientists”.
Dev2	Designed and wrote the code and “invented” the algorithms.
Dev3	Jeongnim Kim (instructor)
Dev4	Balint Joo or work at MICRO and PCL

Cray XC30: a distributed shared-memory cluster







Trends in Parallel Machines: clusters of SMPs

Top10 systems in November 2014 : clusters of SMPs using specialized interconnects

- Tianhe-2 : Xeon + Xeon Phi
- Titan : Opteron + Tesla
- Sequoia : Blue Gene Q
- K Computer : MIPS
- XC30 : Xeon

Canonical HPC systems: clusters of SMPs using commodity interconnects

Your desktops and laptops: a SMP node with multi/many cores

Each system is an optimized solution of high performance and low cost (manufacturing, building, power, support)

Why Parallel Computing?

We have parallel computers. Need to use them well!

Parallel computing uses multiple computing units in parallel to

- solve problems more quickly than a single processor ("strong scaling")
- solve larger problems in the same time as a single processor ("weak scaling")
- solve problems with higher fidelity

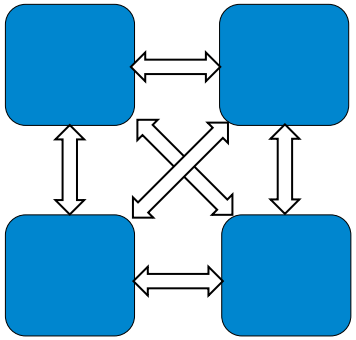
Enables computational simulations for breakthrough discovery and prediction.

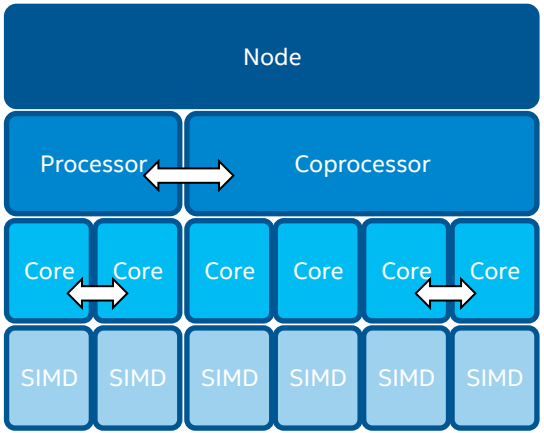
High-performance parallel computing is hard and requires

- Finding enough parallelism
- Deciding the optimal granularity, locality and load balance
- Coordination and synchronization

Real-world applications/algorithms are complex and often hierarchical; monolithic programming model is limited; no silver bullets

Parallel Programming for Performance





- Numerical and system libraries
- Distributed-memory parallel programming: MPI, PGAS
  - Map on to a set of memory domains, e.g. nodes, sockets, cores
  - Explicit and implicit data exchanges and synchronization
- Shared-memory parallel programming: OpenMP*, Pthreads, TBB, Cilk™ Plus, OpenCL*
- Vector programming: auto-vectorization, OpenMP* 4.0

Cori Applications: NERSC 44 NESAP

http://www.hpcwire.com/2014/09/03/nersc-reveals-44-nesap-code-teams/

Benchmark	Parallelism		Language			2014		2017
	MPI	THREADS	Fortran	C	C++	BGQ	KNC	Coral
MILC/CHROMA +	x	x		x	x	O	O	
Nuclear QMC	x	x		x	x	O		
BerkeleyGW/NWCHEM(PW)/ QE/VASP +	x	*	x			P	P	
NWCHEM/CP2K +	x	*	x	x		P	P	
GTC-P/GTCP-C	x	x	x	x		P/O	P	Y
QBOX	x	x			x	O		Y
LAMMPS/NAMD +	x	x			x	O	O	Y
HACC	x	x			x	O		Y
AMG-2013	x	x		x			P	Y

BGQ & KNC: **O**ptimized and **P**orted
Red: non-DOE applications

Why can't we just stick to MPI*?

- We have clusters of SMPs.
  - Each node has 10-100 of cores and multiple threads per core.
  - Some hardware claims to support millions and soon billions of concurrency.
  - Multiple memory & cache levels with various sharing modes: L1 shared by 4 HT on KNC
- Cannot wait for a magic MPI implementation which does all.
- Applications can use the large memory available per SMP node
  - Eliminate/reduce data replications: only one copy of shared constant data is needed.
  - No extra data copies with put/get
- Consider MPI* time and resource use at scale
  - Scaling of collectives: O(C log C) *vs* O(N log N), C=(1-1000)N
  - *Serialization* of point-to-point communications
  - Data for MPI abstractions and communications

Evolution in computation, memory and communication

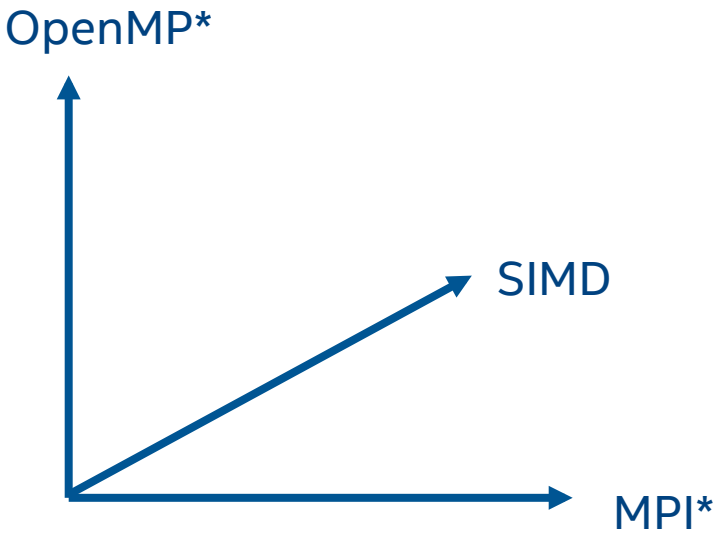
	Cray T3E-1350 [1]	Cray XC30 (Edison@NERSC)	XC30/T3E		Cori/T3E	Cori/Edison
			Per SMP	Per Core		
Processor Clock	675 MHz	2.4GHz	3.64			
SMP	1 CPU	2x12 cores	24	1		
Peak GF/s	1.350 /CPU	460.8 /SMP 19.2 /core	**341**	**14.2**	> 2000	> 6
Peak Memory BW	1.2 GB/s/CPU	89 GB/s/SMP*	**74**	**3**	> 370**	>5**
Memory	256 MB/CPU	64 GB/SMP 2.67 GB/core	256	10.4		
Peak bisection BW	166 GB/s (512 CPUs)	11 GB/s/node	**34**	**1.4**	34	1
MPI Latency (µsec)	6	0.25-3.7	**3**	**0.125***	3	1

- Assume serialized MPI zero-message point-to-point communications.

** Depend on DDR4 or On-package Memory

[1] http://www.filibeto.org/~aduritz/truetrue/supercomputing/cray/datasheets/t3e.pdf

High-performance parallel computing



Moving data is expensive!

- Node-node
- Socket-Socket; Processor-(co)processor
- Core-core
- SIMD lanes

At each parallel level

- Find enough parallelism
- Decide the optimal granularity
- Optimize locality and data movement
- Ensure load balance
- Reduce the impact of coordination and synchronization

All the parallel units have to be coordinated with maximum overlap of data movement and computing.

Set the goals and priorities (Dev*)

- Define performance and your performance goal
  - Strong scaling: reduced time-to-solution at any cost
  - Weak scaling: “constant” time-to-solution with increasing resources
  - Both at a sustained high performance
- Set your priorities
  - Performance, Performance, Performance
  - Optimize (performance, portability, maintenance, ….)
- Know your type, your team and ecosystem
  - Incremental development from the bottom (evolutionary)
  - Transformative development (revolutionary)
  - Iterative process of using both

High-performance Hybrid Programming 101 (Dev*)

- Apply computing 101: const, restrict, C99, alignment, remove branching ….
- Map the data and algorithms to the hierarchical memory and communication hardware and the parallel programming models
- Maximize the shared memory use: eliminate/reduce data replications.
  - Remember only one copy of shared constant data per task is needed!
- Maximize the distributed memory use: localize the data and do not share
  - Think what is needed for high-performance MPI applications
  - Use private data and thread-local storage
- Consider cost of OpenMP* or any thread-based (parallel programming) methods
  - Creating/destroying a team of threads is not FREE!
  - Implicit synchronization and barriers
  - Cache coherency
  - False sharing and write/read conflicts.

EMGeo: Part 1 for Dev0/Dev1

Know your application

Design experiments

Bottom-up transformation

Know EMGeo

Excerpts from README.md

- “EMGeo is a Fortran 90 pure MPI code“
- While the code is somewhat complex, the good news is that the 220 line `qmr` subroutine found in `**krysolver**.f90` takes up over 90 % of the wall-clock run time under typical configurations. Further, this QMR solver routine spends a significant portion of time in **ELLPACK-format** *sparse matrix-vector multiply operation* appearing within the main loop (lines 243-255 of `krysolver.f90`).”
- “a finite difference (FD) code for electromagnetic imaging in geophysical exploration”
- “uses two levels of parallelism: FD method and multiple FD problems”
- “The FD problem domains are decomposed on an I x J x K grid of MPI ranks (inner level)”
- “**Please** refer questions to Scott before attempting to contact Michael.”

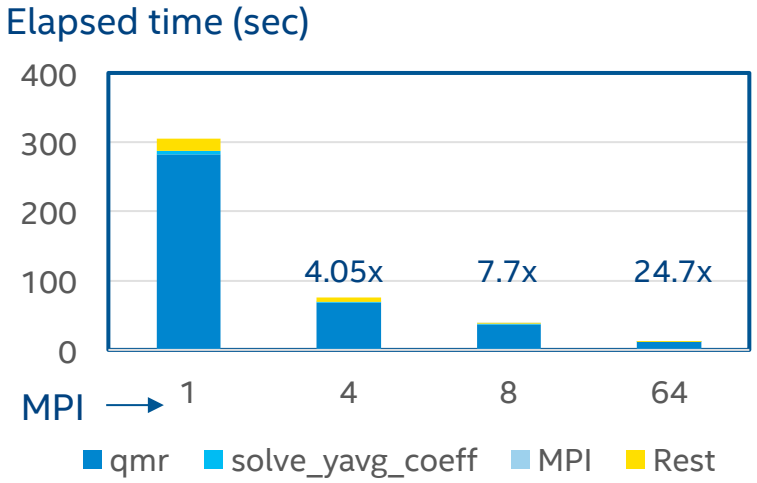
Set the goal(s) and design experiments

Goal: Transform EMGeo to attain *sustained performance* with any combination of MPI tasks and OpenMP threads

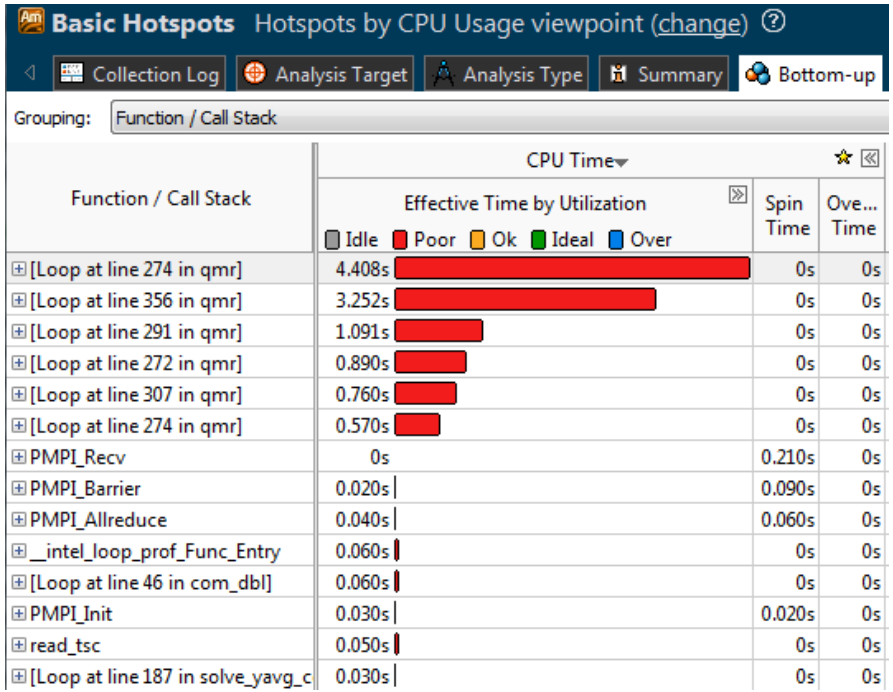
- Workload in run directory: p##_IxJxKxd1 where I*J*K=*MPI tasks*
  - p*01*_1x1x1xd1 p*04*_2x2x1xd1 p*08*_2x2x2xd1 p*64*_4x4x4xd1
- Establishing the baseline
  - Strong scaling with respect to MPI task
  - Hotspots analysis on a quad-socket HWS: p04_2x2x1xd1 and p08_2x2x2xd1
- Bottom-up transformation
- Results

Baseline performance on HSW-EX (quad 18-core)

- I_MPI_PIN_DOMAIN
  - 4 = socket ; 8 = auto:9 ; 64 =core
- Just confirmed README.md
  - QMR is *the* hotspot
  - Domain-decomposition with boundary exchanges: constant total memory footprint
- Super-scaling from 1 to 4 task!
- Excellent strong scaling and all the parts scale well.
- 10% in MPI at 64 tasks: allreduce, send/recv



Hotspot analysis: Loops and functions



- 92.6% in qmr in krysolver.f90

```
do i=1,n
enddo
MPI_ALLREDUCE
do i=1, n
enddo
MPI_ALLREDUCE
…..
```

- All the loops are the same size and LHS (lvalue) is linearly accessed.
- Go ahead and put OMP pragma
- Just careful about **`allreduce`** and make local variables private

QMR* in krysolver

* quasi-minimum-residue, a Krylov space solver

```
do i=1,n
  ay = ay + cpnt(i)*dconjg(cpnt
  csum = czero
  mcole=lshift(rshift(mcol(i),1
  do j=1,mcole,2
     csum = csum + mtx(j,i)*cgs
     csum = csum + mtx(j+1,i)*(
  enddo
  if(mcole.ne.mcol(i))csum=csum
  cvk1(i) = csum
enddo
call MPI_ALLREDUCE(ay,dsum,1,MP

beta = czero

do i=1,n
  cvk1(i) = cpnt(i) - cvk1(i)
  crk(i) = cvk1(i)
  beta = beta + cvk1(i)*cvk1(i)
enddo
call MPI_ALLREDUCE(beta,csum,1,MPI_DOUBLE_COMPLEX, ...)
```

```
!$OMP PARALLEL DO REDUCTION(+:ay) PRIVATE(csum,mcole,j)
do i=1,n
  ay = ay + cpnt(i)*dconjg(cpnt(i))
  csum = czero
  mcole=lshift(rshift(mcol(i),1),1)
  do j=1,mcole,2
    csum = csum + mtx(j,i)*cgs(index(j,i))
    csum = csum + mtx(j+1,i)*cgs(index(j+1,i))
  enddo
  if(mcole.ne.mcol(i)) csum=csum+mtx(mcol(i),i)*cgs(index(mcol(i),i))
  cvk1(i) = csum
enddo
!$OMP END PARALLEL DO
call MPI_ALLREDUCE(ay,dsum,1,MPI_DOUBLE_PRECISION,MPI_SUM, ..)
```

SpMV: sparse matrix-vector multiplication

QMR* in krysolver

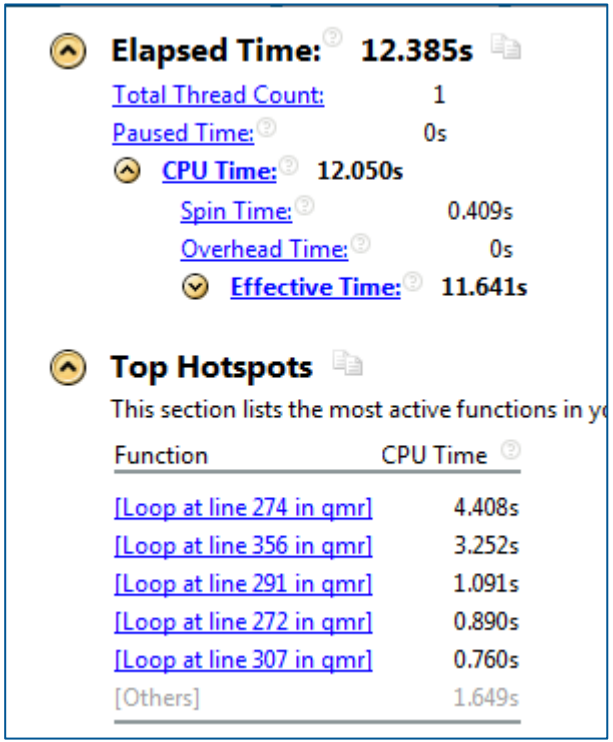
```
do i=1,n
  ay = ay + cpnt(i)*dconjg(cpnt(i))
  csum = czero
  mcole=lshift(rshift(mcol(i),1),1)
  do j=1,mcole,2
     csum = csum + mtx(j,i)*cgs(index(j,
     csum = csum + mtx(j+1,i)*cgs(index(
  enddo
  if(mcole.ne.mcol(i))csum=csum+mtx(mcol(
  cvk1(i) = csum
enddo
call MPI_ALLREDUCE(ay,dsum,1,MPI_DOUBLE_

beta = czero
```

```
do i=1,n
  cvk1(i) = cpnt(i) - cvk1(i)
  crk(i) = cvk1(i)
  beta = beta + cvk1(i)*cvk1(i)
enddo
call MPI_ALLREDUCE(beta,csum,1,MPI_DOUBLE_COMPLEX, ...)
```

```
beta = czero
!$OMP PARALLEL DO REDUCTION(+:beta)
do i=1,n
  cvk1(i) = cpnt(i) - cvk1(i)
  crk(i) = cvk1(i)
  beta = beta + cvk1(i)*cvk1(i)
enddo
!$OMP END PARALLEL DO
call MPI_ALLREDUCE(beta,csum,1,MPI_DOUBLE_COMPLEX, ...)
```

Results: p64 vs p04 with 8 threads

Summary of p64 (using 64 cores)



Elapsed Time: 12.385s

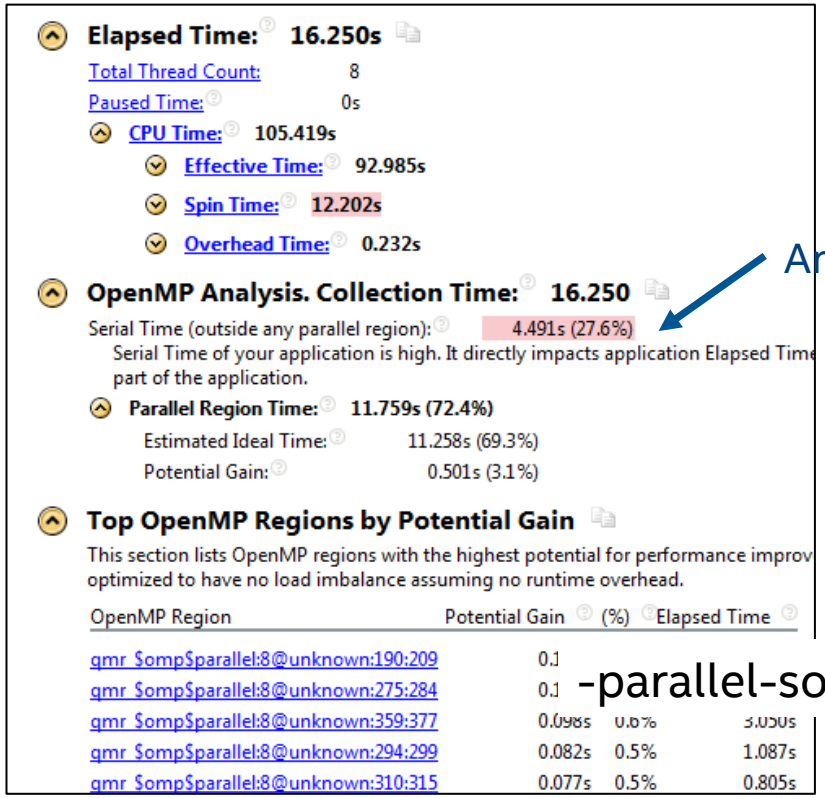
Total Thread Count: 1
Paused Time: 0s
CPU Time: 12.050s
Spin Time: 0.409s
Overhead Time: 0s
Effective Time: 11.641s

Top Hotspots

This section lists the most active functions in y

Function	CPU Time
[Loop at line 274 in qmr]	4.408s
[Loop at line 356 in qmr]	3.252s
[Loop at line 291 in qmr]	1.091s
[Loop at line 272 in qmr]	0.890s
[Loop at line 307 in qmr]	0.760s
[Others]	1.649s

Summary of p04 (using 32 cores)



Elapsed Time: 16.250s

Total Thread Count: 8
Paused Time: 0s
CPU Time: 105.419s
Effective Time: 92.985s
Spin Time: 12.202s
Overhead Time: 0.232s

OpenMP Analysis. Collection Time: 16.250

Serial Time (outside any parallel region): 4.491s (27.6%)
Serial Time of your application is high. It directly impacts application Elapsed Time
part of the application.

Parallel Region Time: 11.759s (72.4%)

Estimated Ideal Time: 11.258s (69.3%)
Potential Gain: 0.501s (3.1%)

Top OpenMP Regions by Potential Gain

This section lists OpenMP regions with the highest potential for performance improv
optimized to have no load imbalance assuming no runtime overhead.

OpenMP Region	Potential Gain	(%)	Elapsed Time
qmr ompparallel:8@unknown:190:209	0.1		
qmr ompparallel:8@unknown:275:284	0.1		
qmr ompparallel:8@unknown:359:377	0.098s	0.6%	3.050s
qmr ompparallel:8@unknown:294:299	0.082s	0.5%	1.087s
qmr ompparallel:8@unknown:310:315	0.077s	0.5%	0.805s

Amdahl's law?

-parallel-source-info=2

Conversation with Dev2

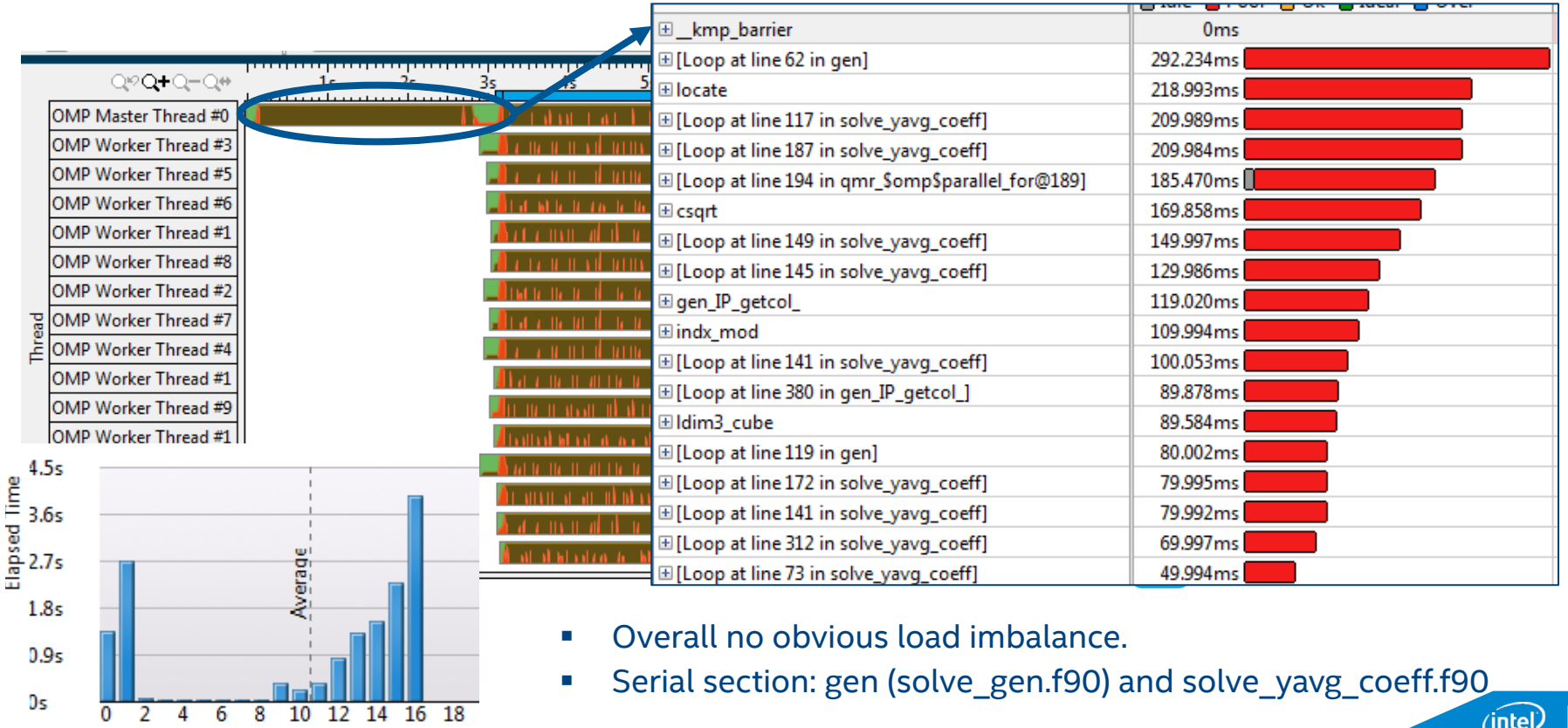
- It should be straightforward to parallelize the other parts
  - MPI can do it. Then, why not OpenMP?
- Prediction: 11.48 sec = 1.21 (p64_2x2x2x2) + 10.27 (p04_2x2x1/16 OMP)
  - 8% gain as implied by the MPI time with 64 tasks

If Dev2 says, “What is the point? All these work for few % gain?”, then stop.

If Dev2 says, “That looks interesting. But, it just shows that the physics is not violated. Show me *performance*.”

EMGeo: Part 2

OpenMP Analysis: p04_2x2x1xd1 using 16 threads



- Overall no obvious load imbalance.
- Serial section: gen (solve_gen.f90) and solve_yavg_coeff.f90

Get rid of the “serial” bottleneck: Dev3

- Apply OpenMP in solve_yang_coeff at L108 and similar loops in solve_gen.f90
  - It looks like all the temporary variables within the loop can be made private.

```
nrow = 0
nrow2= 0 ! rows on node space
do kndx=dim3_sim(5),dim3_sim(6)
  do jndx=dim3_sim(3),dim3_sim(4)
    do indx=dim3_sim(1),dim3_sim(2)
      nrow2 = nrow2 + 1
      do icomp=1,3
        nrow = nrow + 1
        .....
      enddo !icomp
    enddo !indx
  enddo !jndx
enddo !kndx
```

```
!$OMP PARALLEL DO COLLAPSE(3) PRIVATE(nrow,nrow2,...)
do kndx=dim3_sim(5),dim3_sim(6)
  do jndx=dim3_sim(3),dim3_sim(4)
    do indx=dim3_sim(1),dim3_sim(2)
      nrow2 = compute_row(indx,jndx,kndx)
      nrow = nrow2*3;nrow2=nrow2+1
      do icomp=1,3
        nrow = nrow + 1
        .....
      enddo !icomp
    enddo !indx
  enddo !jndx
enddo !kndx
!$OMP END PARALLEL DO
```

Results: disaster – NAN

What went wrong and how to proceed

- All the advertised gotchas exist: common block, hidden dependency ....

  ⇒ There are tools for that and Fortran users can fix them.

- Initialization determines the sparse-matrix storage ordering in ELLPACK-format and SpMV, need a critical look at
  - How data are ordered, allocated and initialized
  - How to facilitate SIMD optimizatioin: collapse(2) vs collapse(3)
  - How auxiliary data structures are used; how many of them are used; why they are needed.
- Many solutions exist and time for serious discussion with Dev2 for transformative code design.

EMGeo on Cori

It will work *fine* on Xeon™ Phi

- Can use multiple MPIs on a node: no problem with memory use.
- Performance improvement through MPI/OpeMP on Xeon is real.
- Enough parallelisms to exploit; load-balancing is not difficult.
- Most of the critical loops are amenable to vectorizations.
- No hard serial bottlenecks exist. Just a matter of using OpenMP correctly.

Can it work *great* on Cori and future MICs?

- All these point to *Probably* but it is time to have serious conversation with the developers for code design and reset out goals.

Code design following best practices of today

- A Core is a new Node but threads are not MPI processes.
- Similar hierarchical architectures of CPUs: socket-core-SIMD
- Microarchitectures matter
  - Xeon™ HSW != KNL
  - Memory bandwidth, NUMAness, process vs thread, cache modes, SIMD, …
  - Improved serial performance on KNL does not mean serial bottlenecks become magically uncritical.

Focus on

- Adaptive data partition and load balancing algorithms with MPI/OpenMP/SIMD
- Code pruning to facilitate compiler optimization
- Portable and performance portable code: encapsulate targeted optimization

Code Design not just Port

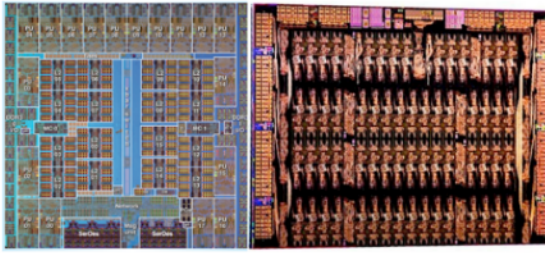
http://press3.mcs.anl.gov/salman-habib/files/2013/05/hacc_pflops.pdf

Co-Design vs. Code Design

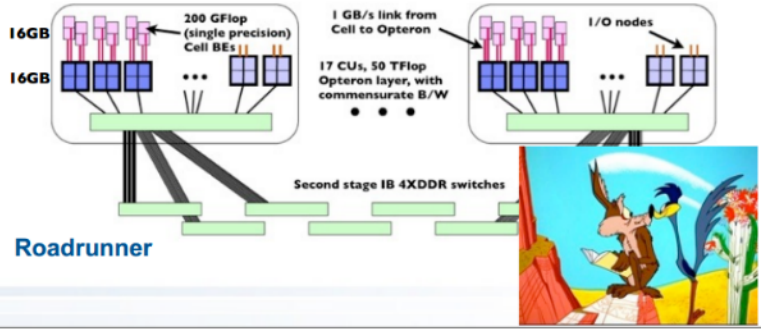
- **HPC Myths**
  - The magic compiler
  - The magic programming model/ language (DSL)
  - Special-purpose hardware
  - Co-Design?
- **Dealing with (Current) HPC Reality**
  - Follow the architecture
  - Know the boundary conditions
  - There is no such thing as a 'code port'
  - Think out of the box
  - Get the best team
  - Work together

BQC:
- 16 cores
- 205 GFlops, 16 GB
- 32 MB L2, crossbar at 400 GB/s (memory connection is 40 GB/s)
- 5-D torus at 40 GB/s

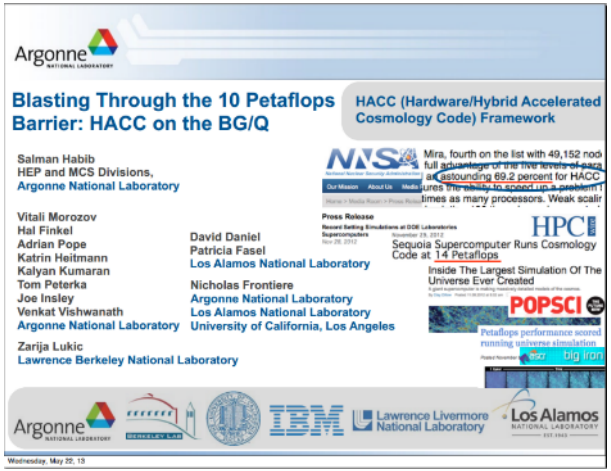
Xeon Phi:
- 60 cores
- 1 TFlops, 8 GB
- 32 MB L2, ring at 300 GB/s (connects to cores and memory)
- 8 GB/s to host CPU

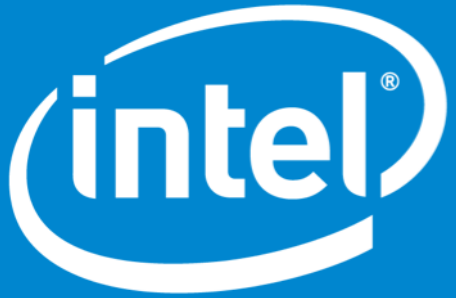


Average performance speed-up on ~10 applications codes on Titan is ~2 (ranging from 1.few to 7), but of Titan's 27 PFlops, only 2.5 PFlops are in the CPU! What is wrong with this picture?



Wednesday, May 22, 13





Legal Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document 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.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.
Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to:
http://www.intel.com/design/literature.htm

Knights Landing and other code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user

Intel, Look Inside, Xeon, Intel Xeon Phi, Pentium, Cilk, VTune and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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

Legal Disclaimers

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 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

Legal Disclaimers

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. For more information go to http://www.intel.com/performance.

Intel® Advanced Vector Extensions (Intel® AVX)* provides higher throughput to certain processor operations. Due to varying processor power characteristics, utilizing AVX instructions may cause a) some parts to operate at less than the rated frequency and b) some parts with Intel® Turbo Boost Technology 2.0 to not achieve any or maximum turbo frequencies. Performance varies depending on hardware, software, and system configuration and you can learn more at http://www.intel.com/go/turbo.

Estimated Results Benchmark Disclaimer:
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.

Software Source Code Disclaimer:
Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Legal Disclaimers

The above statements and any others in this document that refer to plans and expectations for the third 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 the important factors that could cause actual results to differ materially from the company’s expectations. Demand could be different from Intel's expectations due to factors including changes in business and economic conditions; customer acceptance of Intel’s and competitors’ products; supply constraints and other disruptions affecting customers; changes in customer order patterns including order cancellations; and changes in the level of inventory at customers. Uncertainty in global economic and financial conditions poses a risk that consumers and businesses may defer purchases in response to negative financial events, which could negatively affect product demand and other related matters. Intel operates in intensely competitive industries that are characterized by a high percentage of costs that are fixed or difficult to reduce in the short term and product demand that is highly variable and difficult to forecast. Revenue and the gross margin percentage are affected by the timing of Intel product introductions and the demand for and market acceptance of Intel's products; actions taken by Intel's competitors, including product offerings and introductions, marketing programs and pricing pressures and Intel’s response to such actions; and Intel’s ability to respond quickly to technological developments and to incorporate new features into its products. The 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; start-up costs; excess or obsolete inventory; changes in unit costs; defects or disruptions in the supply of materials or resources; product manufacturing quality/yields; and impairments of long-lived assets, including manufacturing, assembly/test and intangible assets. 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. Expenses, particularly certain marketing and compensation expenses, as well as restructuring and asset impairment charges, vary depending on the level of demand for Intel's products and the level of revenue and profits. Intel’s results could be affected by the timing of closing of acquisitions and divestitures. Intel's results could be affected by adverse effects associated with product defects and errata (deviations from published specifications), and by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust, disclosure and other issues, such as the litigation and regulatory matters described in Intel's SEC reports. 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. 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.

Rev. 7/17/13