

### Agenda

- Introduction to HPC
- Silicium first:
  - CPU
  - Coprocessor
- Application point of view:
  - Perf measurement
  - How to make it faster
  - Are the results valid ?

The slide contains an agenda list on the right side. On the left side, there is a small icon of a robot with a colorful screen on its chest. The background of the slide is a grayscale image of a desert landscape with several people in the distance, some with their arms raised in celebration. The foreground shows a textured, rippled surface, possibly sand or a dry lake bed.

## Glossary, "High performance computing"

- Peak Flops = nb of floating points operations per cycle \* frequency

$$Peak = (Flops / cycle) * (cycle / sec) = Flops / sec$$

Efficiency = % of the peak performance

For Flops => Peak flops / Achieved flops

For BW => Peak GB/s versus Achieved GB/s

Both depend on Hardware and Software

*First question : what is the peak perf of your laptop ?*

3 / 75

Software & Services Group, Energy Engineering Team



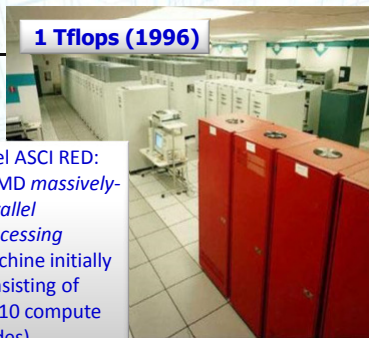
## A very brief history



The **CM-2** is SIMD  
(16 k processors)



The **CM-5** is MIMD (Multiple Instructions Multiple Data) using commodity SPARC processors using a "fat tree" interconnect



**1 Tflops (1996)**

Intel ASCI RED:  
MIMD *massively-parallel processing* machine initially consisting of 4,510 compute nodes)

**RISC  
Multithreading  
Superscalar  
Vector machine  
MPP machine**

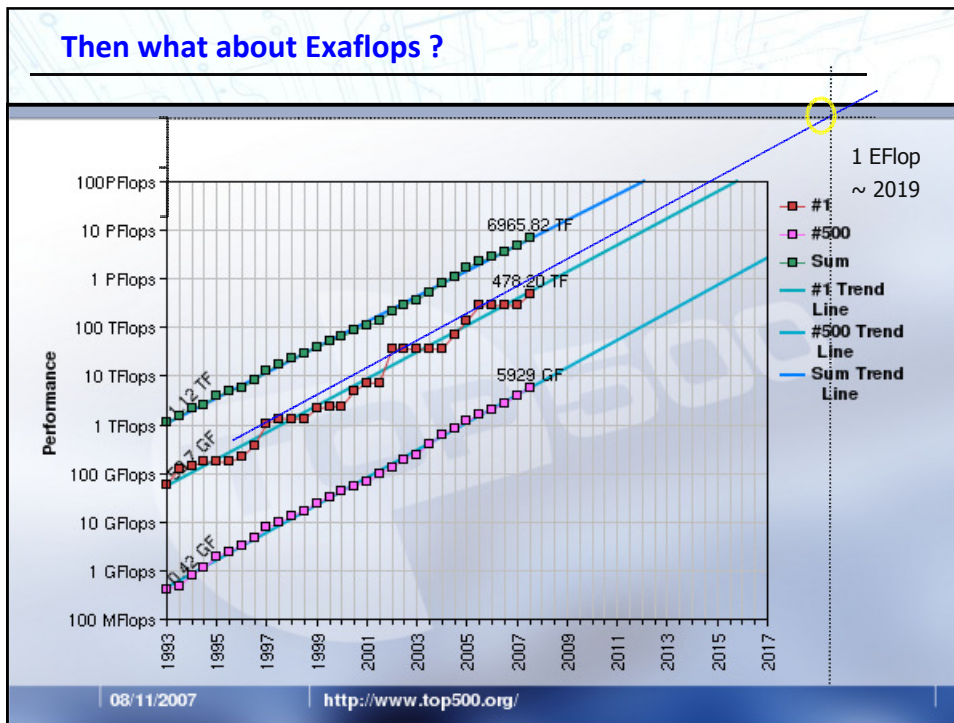
The **Intel Paragon** was a series of massively parallel supercomputers produced by Intel. The Paragon series was based around the Intel i860 RISC microprocessor. Up to 2048 (later, up to 4000) i860s were connected in a 2D grid

60's      70's      80's      90's      00's

4 / 75

Software & Services Group, Energy Engineering Team





### SuperComputer power challenges

x 5. 10<sup>6</sup>  
(40 M cores)

Would gives ~ 1,5 GW for 1 Exaflops Peak

Today CPU:  
200 GF-DP  
150 W

French Nuclear plants ~63 GW:  
34 reactors of 900 MW  
20 reactors of 1300 MW  
4 reactors of 1450 MW  
+ EPR 1 600 MW

[20:40] MW would be a reasonable target

## SuperComputer power challenges

- At multi-rack system level
  - **Super Computer Today:** 10 PF, 12 MW → **1200 pJ / Ops**
  - **Exaflop (lower lim.):** 1000 PF, 20 MW → **20 pJ / Ops**
  - **Exaflop (upper lim.):** 1000 PF, 40 MW → **40 pJ / Ops**
- Processor portion needs to reduce to 10 pJ/ Ops
- **Needs huge improvement (30 to 60x) in all system components**

## Micro architecture Performance Parameters

- Core capability (IPC)
- Vector Flop density (SIMD)
- Core count
- On-die Interconnect
- Frequency
- Mem Latency
- Mem BW & Size

$$\text{Perf} = \text{Frequency} * \text{IPC}$$

$$\text{Power} \sim C * \text{Voltage}^2 * \text{Freq}$$

*frequency + voltage reduction : Cubic reduction of power*

## First approximation: HPC is only a matter of Freq and BW

- Frequency is « not an issue »
- Data movement is the issue

The model : "Total Elapsed time = T\_CPU + T\_mem + ..." is fine but ...

T\_CPU and T\_mem are strongly correlated

```

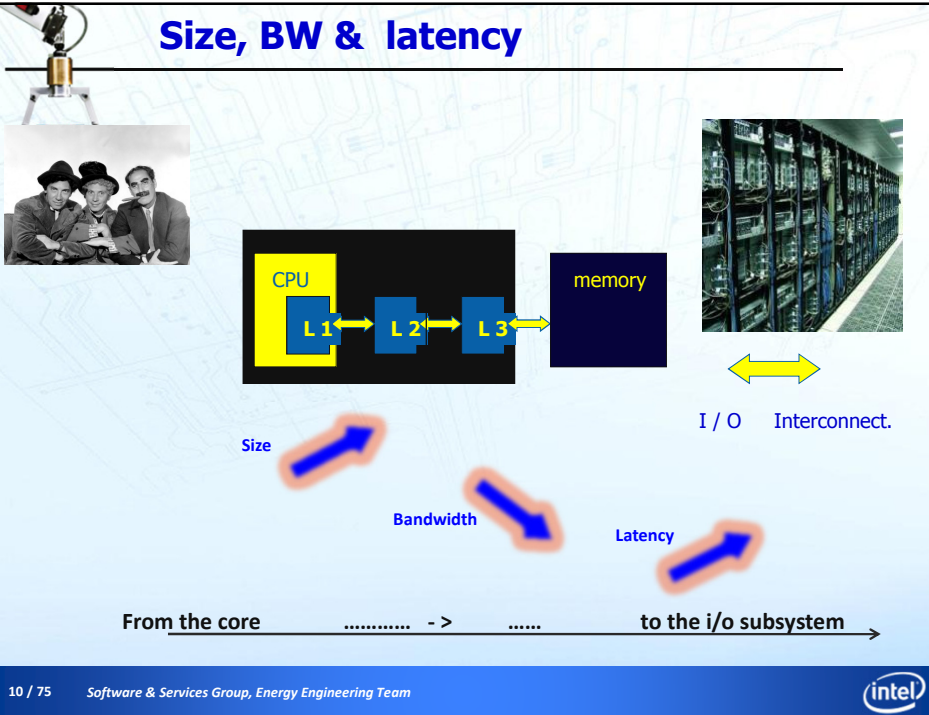
for (i=0; i<=MAX; i++)
    c[i] = a[i] + b[i] * d[i];
    
```

store    load    load    load  
add    mul

Achieved Flops/s won't be high enough if load / store are not fast enough



## Size, BW & latency



### How to deal with that on the application side ?

---

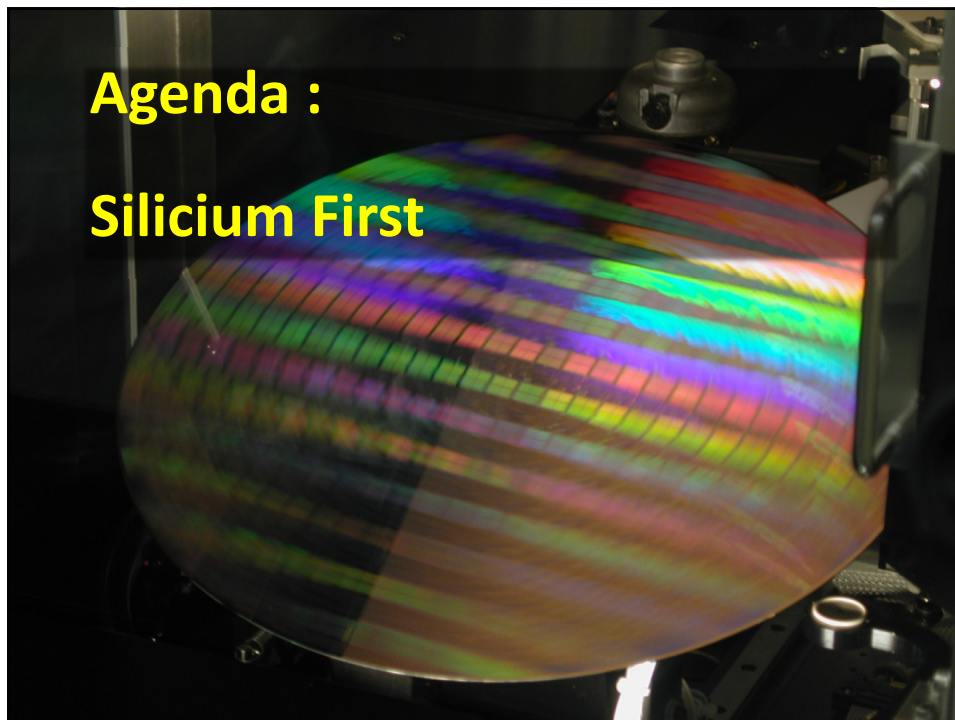
**CPU bound.**  
"HPL"
**Real world applications**
**Memory bound.**  
"Stream"

Part of the code  
doing ops

Part of the code  
doing memory  
access

$$\text{CPU.ops\%} * (\text{cycle / inst.}) + \text{Mem.ops\%} * [ (\text{LL\_success\%} * \text{cycle / inst}) + \text{LL\_misses\%} * \text{cycle/inst} ]$$

11 / 75    Software & Services Group, Energy Engineering Team

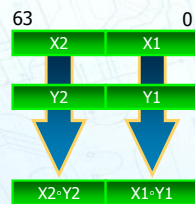


## Vectorization

- **Single Instruction Multiple Data (SIMD):**
  - Processing vector with a single operation
  - Provides data level parallelism (DLP)
  - Because of DLP more efficient than scalar processing
- **Vector:**
  - Consists of more than one element
  - Elements are of same scalar data types (e.g. floats, integers, ...)



## SIMD Types for Intel® Architecture I



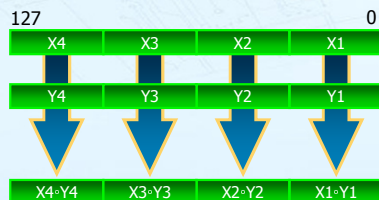
### MMX™

Vector size: **64 bit**

Data types:

•8, 16 and 32 bit integer

VL: 2, 4, 8



### SSE

Vector size: **128 bit**

Data types:

•8, 16, 32, 64 bit integer

•32 and 64 bit float

VL: 2, 4, 8, 16

Illustrations:  $X_i$ ,  $Y_i$  & results 32 bit integer

### SIMD Types for Intel® Architecture II

255  
X8 X7 X6 X5 X4 X3 X2 X1  
Y8 Y7 Y6 Y5 Y4 Y3 Y2 Y1  
X8·Y8 X7·Y7 X6·Y6 X5·Y5 X4·Y4 X3·Y3 X2·Y2 X1·Y1

511  
X16 ... X4 X3 X2 X1  
Y16 ... Y2 Y1  
X16·Y16 ... X8·Y8 X7·Y7 X6·Y6 X5·Y5 X4·Y4 X3·Y3 X2·Y2 X1·Y1

**AVX**  
Vector size: **256 bit**  
Data types:  
• 32 and 64 bit float  
VL: 4, 8, 16

**Intel® MIC**  
Vector size: **512 bit**  
Data types:  
• 32 and 64 bit integer  
• 32 and 64 bit float  
VL: 8,16

Illustrations: Xi, Yi & results 32 bit float

15 / 75    Software & Services Group, Energy Engineering Team   

### Reminder about the peak Flops Intel® Sandy Bridge micro-u

Scheduler (Port names as used by Intel® Architecture Code Analyzer \*\*\*)

Port 0	Port 1	Port 5	Port 2	Port 3	Port 4
ALU VI* MUL SSE MUL DIV** AVX FP MUL AVX FP Blend	ALU VI* ADD SSE ADD AVX FP ADD	ALU/JMP VI* ADD SSE Shuf AVX FP Shuf AVX FP Bool AVX FP Blend	Load Store Address	Load Store Address	Store Data

**6 instructions / cycle:**

- 3 memory ops
- 3 computational operations

**Nehalem : Two 128 bits SIMD per cycle**

4 MUL (32b) and 4 ADD (32b): **8 Single Precision Flops / cycle**

2 MUL (64b) and 2 ADD (64b): **4 Double Precision Flops / cycle**

**Sandy bridge : Two 256 bits SIMD per cycle**

8 MUL (32b) and 8 ADD (32b): **16 Single Precision Flops / cycle**

4 MUL (64b) and 4ADD (64b): **8 Double Precision Flops / cycle**

16 / 75    Software & Services Group, Energy Engineering Team



## Performance at a glance : What SNB brings

The total benefit (at node level) is given by a combination of factors

- **Benefit from micro-u optimization (IPC)**

from 1.1x to 1.3x

- **Benefit from the nb of cores**

up to 1.33x

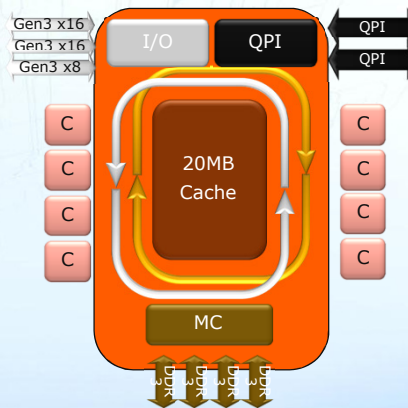
- **Benefit from AVX**

up to 2x

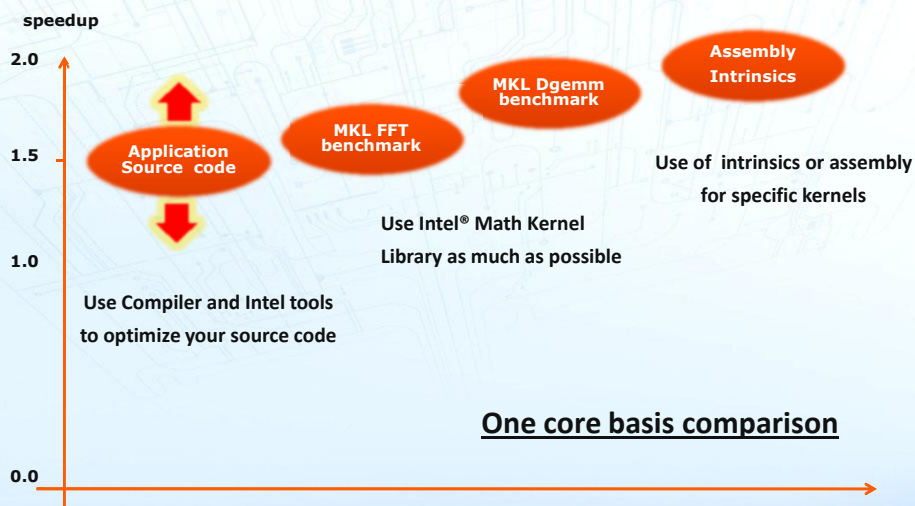
- **Benefit from Memory bandwidth**

up to 1.42x per core

up to 1.9x on node basis



## How to take advantage of AVX for more Flops / cycles



## Memory Bandwidth: a key point

Basical rules for theoretical memory BW [Bytes / second] :

$$8 \text{ [Bytes / channel]} * \text{Mem freq [Gcycles/sec]} * \text{nb of channels} * \text{nb of sockets}$$

For Westmere DP server with 1333 Mhz DDR3:

$$8 * 1.333 * 3 * 2 = 63.984 \text{ GB/s where Stream triad gives } \sim 42 \text{ GB/s}$$

For WSM / Nehalem-EX platform: 4 channels , 4 sockets and 1066 MHz memory

$$8 * 1.066 * 4 * 4 = 136.484 \text{ GB/s peak (ST : 102 GB/s)}$$

For Sandy Bridge EP platform: 4 channels , 2 sockets and 1600 MHz memory

$$8 * 1.600 * 4 * 2 = 102.4 \text{ GB/s peak (ST : 80 GB/s)}$$

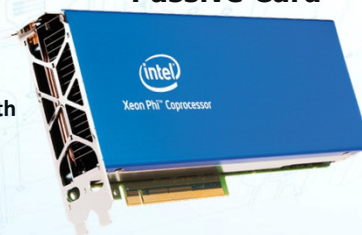
\*Software and workloads used in performance tests may have been optimized for performance only on Intel® microprocessors. Performance tests are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. 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.



## Intel® Xeon Phi™ Product Family



### Passive Card



- Up to 61 IA cores/1.1 GHz/ 244 Threads
- Up to 8GB memory with up to 352 GB/s bandwidth
- 512-bit SIMD instructions
- Open Source Linux operating system
- IP addressable
- Standard programming languages, tools, clustering
- 22 nm process

### Active Card



<http://software.intel.com/en-us/mic-developer>

INTEL® DEVELOPER ZONE: **Intel® Xeon Phi™ Coprocessor** Optimize your software for data center, cloud, and high performance computing

### Parallel Processing

Architecture for Discovery



OVERVIEW TOOLS & DOWNLOADS PROGRAMMING TRAINING CASE STUDIES ARTICLES / FORUMS / BLOGS

GET SUPPORT

Intel® Many Integrated Core Architecture Forum  
Parallel Programming Forum

VISIT RELATED ZONES

Server  
Parallel Programming

SOFTWARE DEVELOPMENT PRODUCTS

Intel® Cluster Studio XE  
Intel® Parallel Studio XE

PLATFORMS

Productivity via architecture innovation coupled with familiar software.

Intel® Xeon Phi™ coprocessor:

- Extends hardware support to higher degrees of parallelism with power savings
- Uses familiar and standard programming models to preserve investments
- Shares parallel programming with general purpose processor

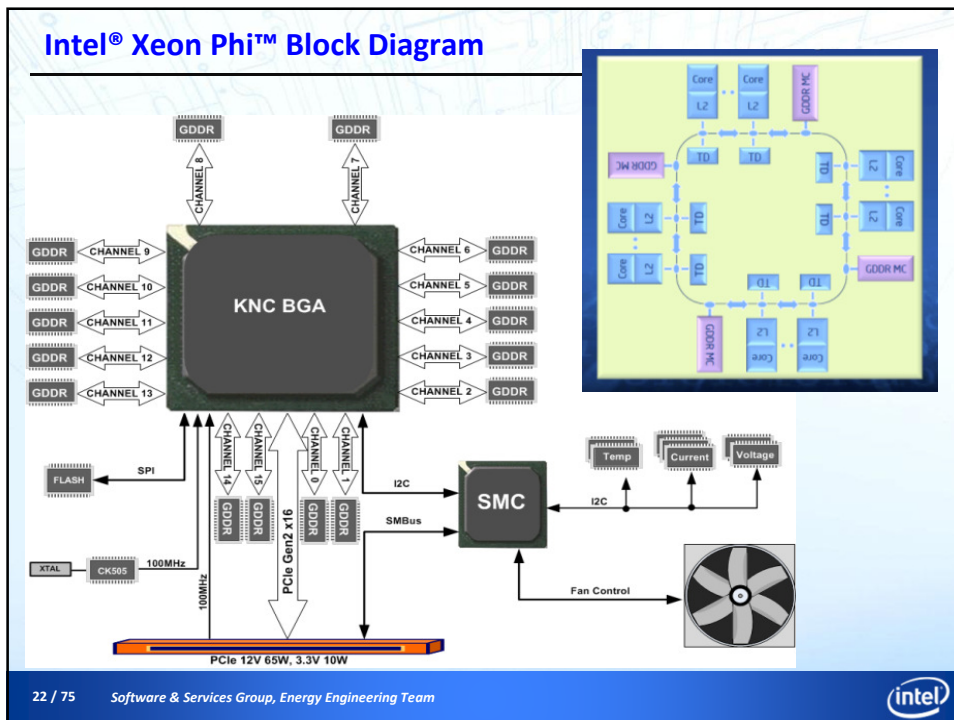
Learn how your applications can benefit from Intel Xeon Phi coprocessors

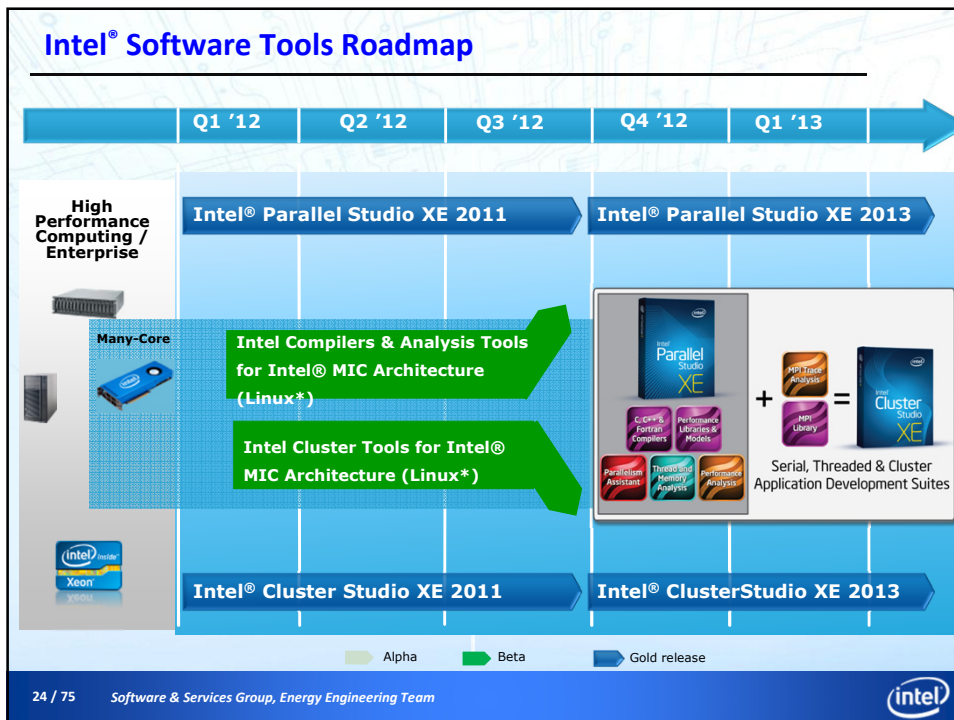
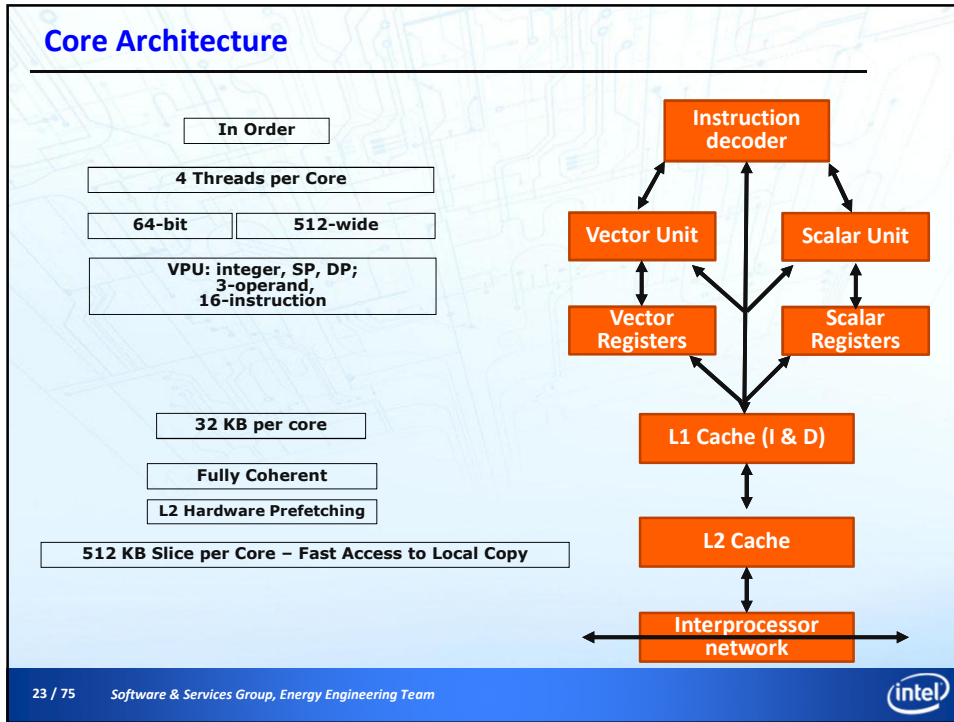


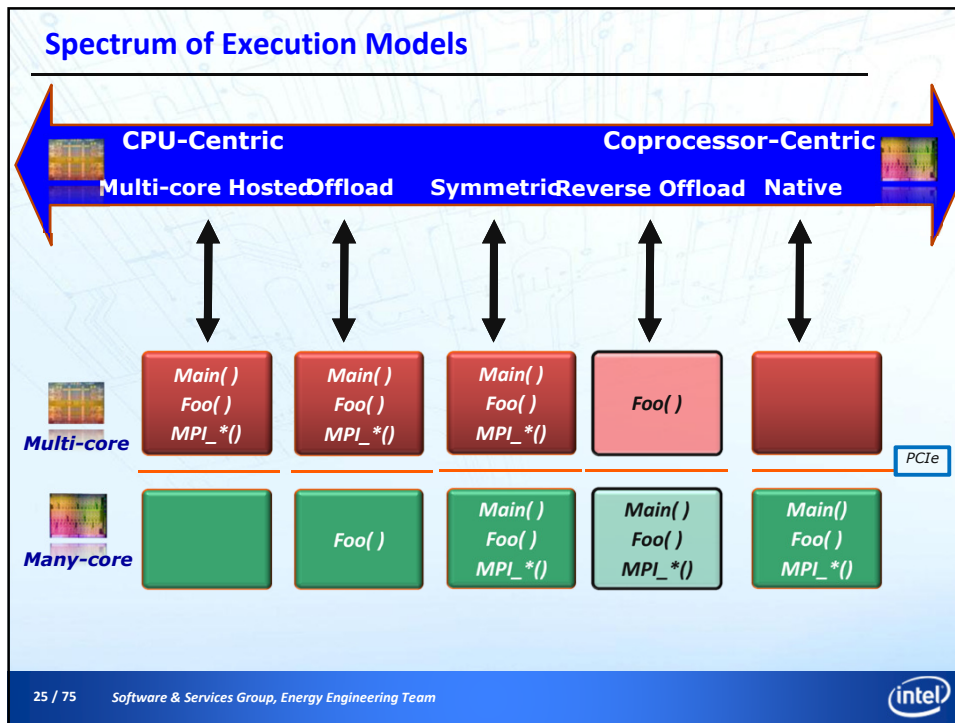
- Intel® Xeon Phi™ Coprocessor Architecture
- An Overview of Programming for Intel® Xeon® processors and Intel® Xeon Phi™ coprocessors
- Intel® Xeon Phi™ Coprocessor Developer's Quick Start Guide
- Intel® Xeon Phi™ Coprocessor Instruction Set Architecture Reference Manual

21 / 75 Software & Services Group, Energy Engineering Team 

**Intel® Xeon Phi™ Block Diagram**







### Spectrum of Execution Models (Offload / Native / Symmetric)

**Offload:**  
 Workload is run on host, and highly parallel phases on Coprocessor

```

!dir$ omp offload target(mic)
  !$omp parallel do
    do i=1,10
      A(i) = B(i) * C(i)
    enddo
  !$omp end parallel
    
```

The diagram shows a host with a CPU and Data, connected to a Network. The host sends Data to a Coprocessor via Offload, and the Coprocessor returns Data to the host via MPI.

**MPI Example on Host with offload to coprocessor**

26 / 75 Software & Services Group, Energy Engineering Team

Spectrum of Execution Models (Offload / **Native** / Symmetric)

**Native (Coprocesor-only model):**

Workload is run solely on coprocessor

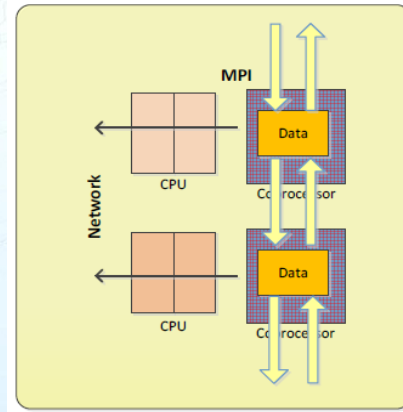
```
icc -mmic ... ./bin_mic
```

Then

```
ssh mic0
./bin_mic
```

Or start it from host

```
micnaticeloadex ./bin_mic
```



MPI example on Coprocessor only

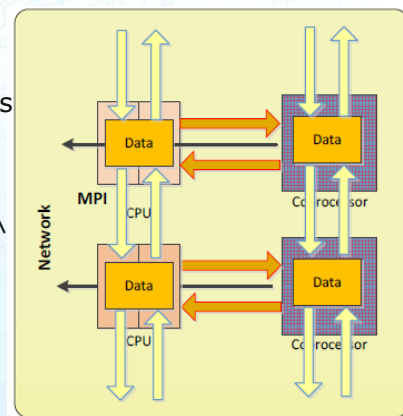


Spectrum of Execution Models (Offload / Native / **Symmetric**)

**Symmetric:**

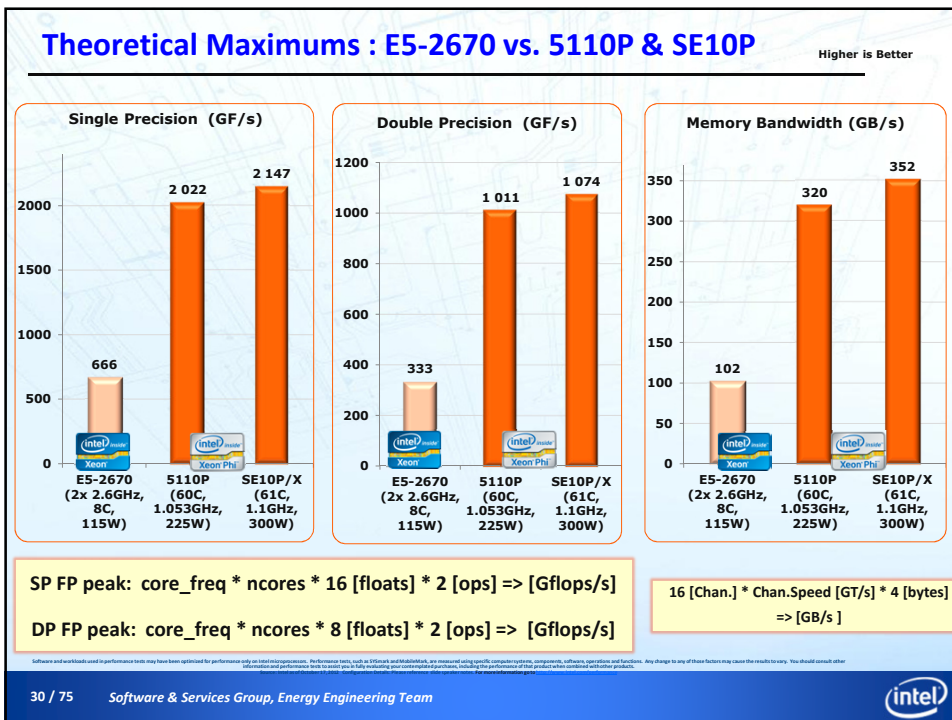
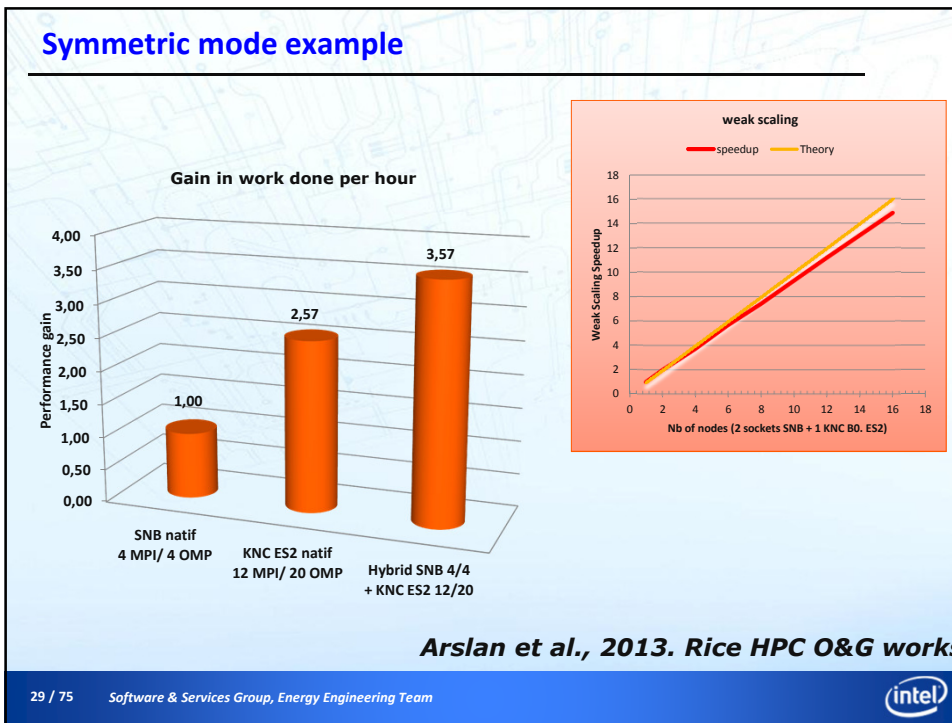
Workload runs on Host & Coprocessors

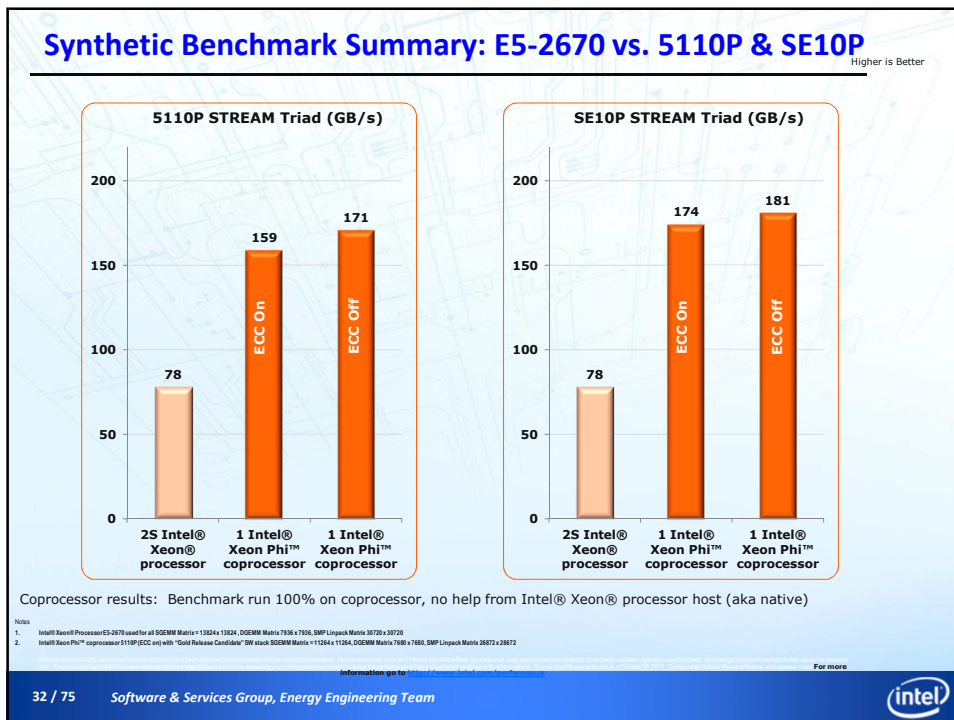
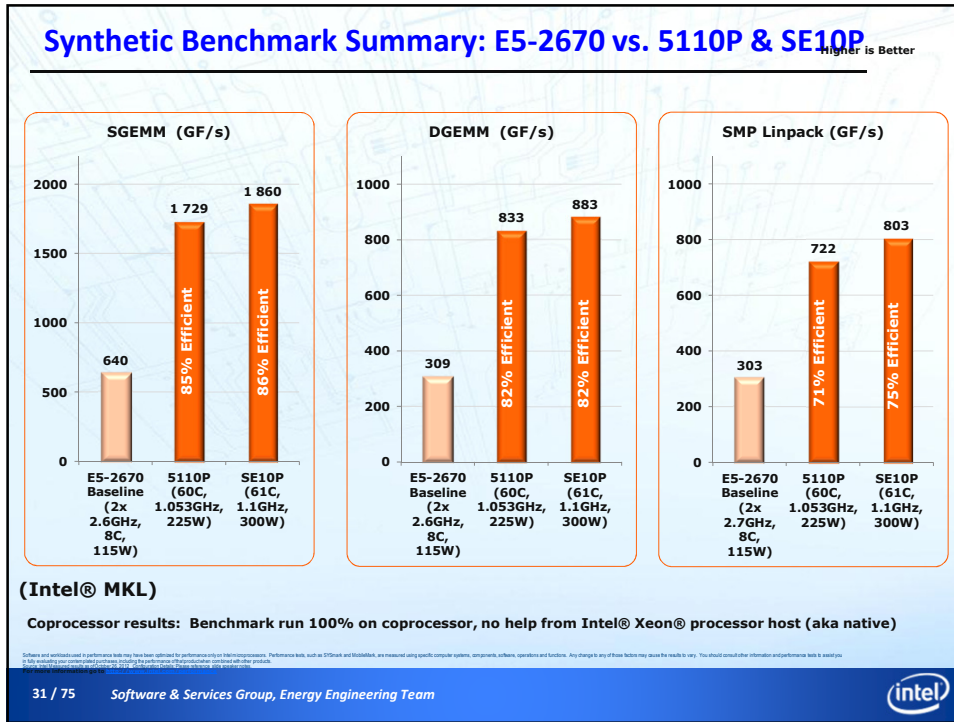
```
Mpiexec -host $host -np XX ... \
-env OMP_NUM_THREADS YY ./mpi_bin_xeon \
: -host mic0 -np VV ... \
-env OMP_NUM_THREADS WW ./mpi_bin_phi
```



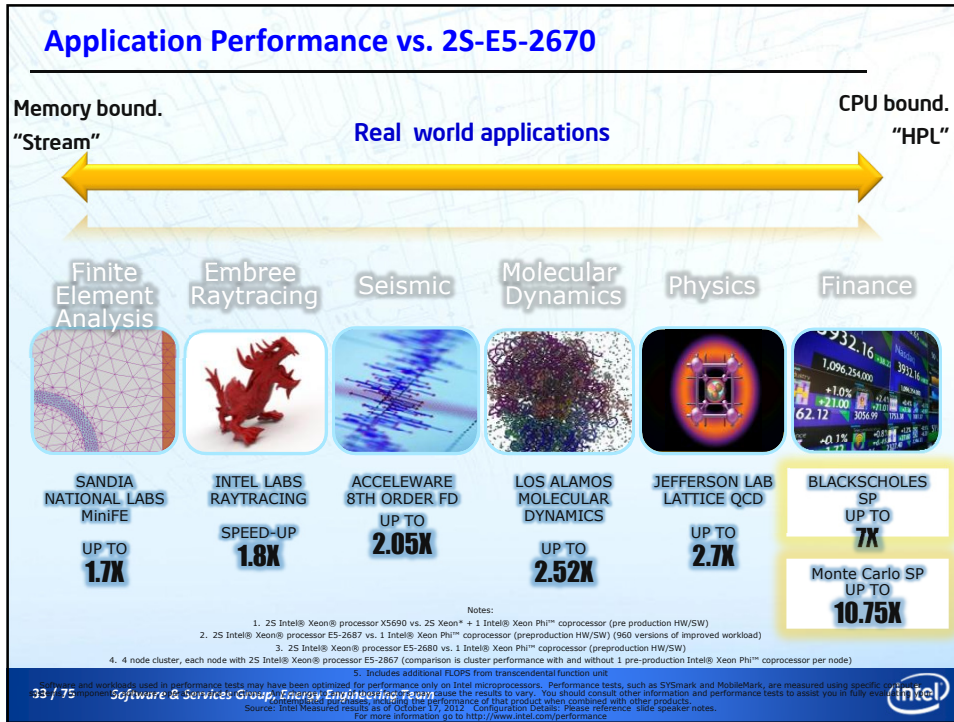
MPI example On Host and Coprocessor











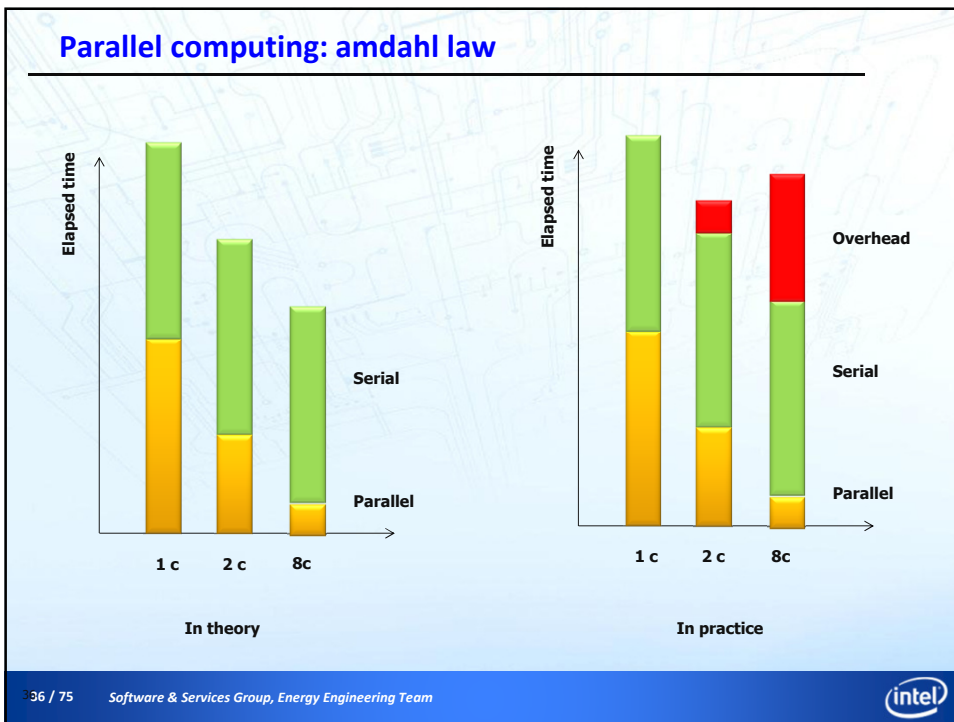
### End users point of view : Physics and computing needs increase

**Increase problem size**  
(at constant time and physics)

**Increase physics complexity**  
(at constant elapsed time and size)

**Decrease elapsed time.**  
(at constant size and physics)

35 / 75    Software & Services Group, Energy Engineering Team    intel



## Speedup and efficiency

$T(n)$  : elapsed time on  $n$  cores

$S(n)$  : speedup

$E(n)$  : efficiency

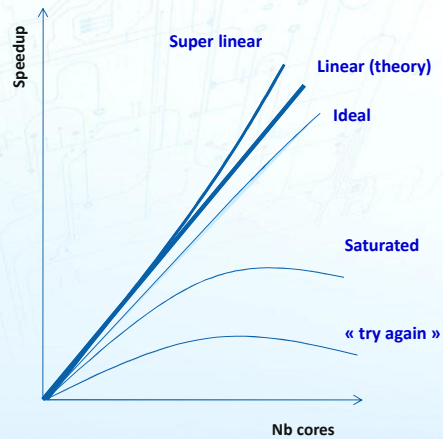
$$S(n) = T(1) / T(n)$$

$$E(n) = S(n) / n$$

**Ideal case:**

$$S(n) = T(1) / T(n) = n * T(n) / T(n) = n$$

$$E(n) = S(n) / n = 1$$



(remember to draw this graph from node to cluster level)

## Why applications don't scale

- |                            |   |
|----------------------------|---|
| • <b>Amdahl's law</b>      | <b>The serial bottleneck</b>                  |
| • <b>Software</b>          | <b>Compiler, OpenMP, MPI</b>                  |
| • <b>Data Locality</b>     | <b>Cache misses, latency</b>                  |
| • <b>Load Balancing</b>    | <b>Affect the parallel efficiency/scaling</b> |
| • <b>Parallel Overhead</b> | <b>Affect the parallel efficiency/scaling</b> |
| • <b>Operating System</b>  | <b>Scheduling, Placement, I/O,...</b>         |
| • <b>Hardware</b>          | <b>CPU MHz, Latency, BW, Topology, I/O</b>    |

**"it's the bandwidth, stupid !"**

### Parallel computing: HowTo ?

**0 - Serial Optimization:**

**1 - Work on the overhead: strong HDW /SFTW relationship**

**2 - Take advantage of parallelism to run bigger workloads ? (weak versus strong scaling)**

39 / 75 Software & Services Group, Energy Engineering Team

### First at all : do it serial

Elapsed time

Elapsed time

1 c

1 c

Profiling + HDW counters  
 Compiler option  
 Help the compiler  
 SIMD (sse)  
 Check memory management  
 Are all the Flops useful

40 / 75 Software & Services Group, Energy Engineering Team

### 10 000 feet view of application measurements

Starting time

Total number of Instructions / sec (hdw counter)

Total elapsed time = nb of CPU CYCLES / Frequency

**Whole application**

- i/o
- Comm
- Parallel overhead
- « other instructions »

**Computational kernel**

- Flops
- Int
- Read/write Dram

by hands + profiling tools

by hands to get max theoretical Perf

Get approximate values using HDW counters

41 / 75
Software & Services Group, Energy Engineering Team

### 3D Finite Difference Stencil and Time Derivative updates

```

do t = 1, NT // time steps
  do z = zmin, zmax // SPACE STEPS
    do y = ymin, ymax
      do x = xmin, xmax
        [ Finite Differences ]
        [ Time Derivatives ]
      end do
    end do
  end do
end do
    
```

```

sum = c(0)* u(i1, i2, i3) + &
c(1)*(u(i1+1, i2, i3) + u(i1-1, i2, i3) + u(i1, i2+1, i3) + &
+ u(i1, i2-1, i3) + u(i1, i2, i3+1) + u(i1, i2, i3-1)) + &
c(2)*(u(i1+2, i2, i3) + u(i1-2, i2, i3) + u(i1, i2+2, i3) + &
+ u(i1, i2-2, i3) + u(i1, i2, i3+2) + u(i1, i2, i3-2)) + &
c(3)*(u(i1+3, i2, i3) + u(i1-3, i2, i3) + u(i1, i2+3, i3) + &
+ u(i1, i2-3, i3) + u(i1, i2, i3+3) + u(i1, i2, i3-3)) + &
c(4)*(u(i1+4, i2, i3) + u(i1-4, i2, i3) + u(i1, i2+4, i3) + &
+ u(i1, i2-4, i3) + u(i1, i2, i3+4) + u(i1, i2, i3-4))
    
```

curr\_P, old\_P

z

y

x

3D FD

Time Derivative

curr\_P old\_P

new\_P

42 / 75
Software & Services Group, Energy Engineering Team

## What results to look for ?

Gflops /sec: Are we far from the peak of the algorithm & of the machine ?

(Mpts/s) -> (Flops/pts) -> (Flops/s)

Gbytes/sec : Are we bandwidth limited or latency limited ?

Flops/ joule: Can we see the impact of any given implementation ?

By hands to get max theoretical Perf

Computational kernel

- Flops / Int
- Read/write Dram

```
Total nb Ops per point per iteration 292
Total nb of point 2229969960.00000
Total (sec) = 9.598
Total/ite (sec) = 0.479
Speed 1: 232.313 M points / sec
Speed 2: 6.968 E-002 Points / cycle
Speed 3: 67.835 Gflops / sec
=> 42.390 % of peak, 1 threads, 6 MPIs
```

Get approximate values using HDW counters

**Needs to collect more than only the Elapsed time !**

43 / 75 Software & Services Group, Energy Engineering Team



## Real numbers when we can't count FLOPS by hands

Flops/s : Collect FP related counters

(FP\_COMP\_OPS\_EXE.x87 + FP\_COMP\_OPS\_EXE.SSE\_DOUBLE\_PRECISION + FP\_COMP\_OPS\_EXE.SSE\_SINGLE\_PRECISION  
+ FP\_COMP\_OPS\_EXE.SSE\_FP\_SCALAR) \*1E-09 / Elapsed\_time

GB/s : Collect 'UNCore read and write events'

(UNC\_IMC\_WRITES for each socket + UNC\_IMC\_READS for each socket) \*1E-09 \* Cache line size / E\_time

+ WATT and Energy as a function of time

**Hardware counters remain the fastest way to measure performance & efficiency**

**=> Gives Flops/s , memory demand and then Bytes/flops**

*where Elapsed time = CPU\_CLK\_UNHALTED / Processor Frequency / Nb of Cores*

44 / 75 Software & Services Group, Energy Engineering Team



### Max theoretical Flops/s by hands (isotropic 3DFD kernel)

K : half stencil length

```

sum = c(0)* u(i1, i2,i3) + &
c(1)*(u(i1+1, i2, i3) + u(i1-1, i2, i3) + u(i1, i2+1, i3) &
+ u(i1, i2-1, i3) + u(i1, i2, i3+1) + u(i1, i2, i3-1)) + &
c(2)*(u(i1+2, i2, i3) + u(i1-2, i2, i3) + u(i1, i2+2, i3) &
+ u(i1, i2-2, i3) + u(i1, i2, i3+2) + u(i1, i2, i3-2)) + &
c(3)*(u(i1+3, i2, i3) + u(i1-3, i2, i3) + u(i1, i2+3, i3) &
+ u(i1, i2-3, i3) + u(i1, i2, i3+3) + u(i1, i2, i3-3)) + &
c(4)*(u(i1+4, i2, i3) + u(i1-4, i2, i3) + u(i1, i2+4, i3) &
+ u(i1, i2-4, i3) + u(i1, i2, i3+4) + u(i1, i2, i3-4))

v(i1, i2, i3) = 2.*u(i1, i2, i3) - v(i1, i2, i3) + sum * w(i1, i2, i3)
    
```

**Total number of flops per point:**  $nF = 7K+5$ ,

made of ADD:  $nA = 6K + 2$  and MUL:  $nM = K + 3$

-> max (nA,nM) is the limiting factor

**Achievable peak** (with infinite bandwidth) is the ratio of nF by the max(nA, nM)

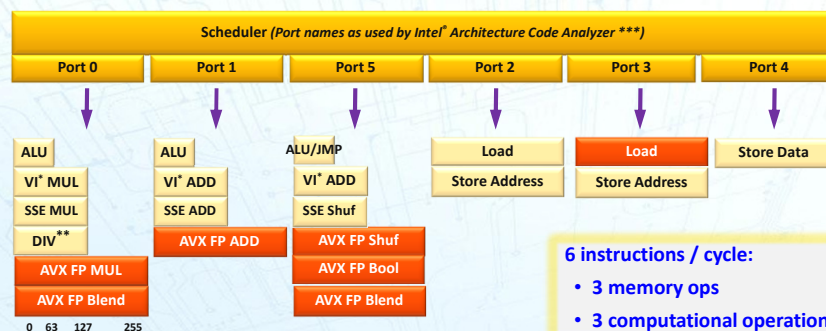
$$\%Peak(K) = 100.0 * [ nF / 2 * \max (nA, nM) ]$$

=> Give the kernel efficiency



### Reminder about the peak Flops

Intel® Sandy Bridge micro-u



**Nehalem : Two 128 bits SIMD per cycle**

- 4 MUL (32b) and 4 ADD (32b): 8 Single Precision Flops / cycle
- 2 MUL (64b) and 2 ADD (64b): 4 Double Precision Flops / cycle

**Sandy bridge : Two 256 bits SIMD per cycle**

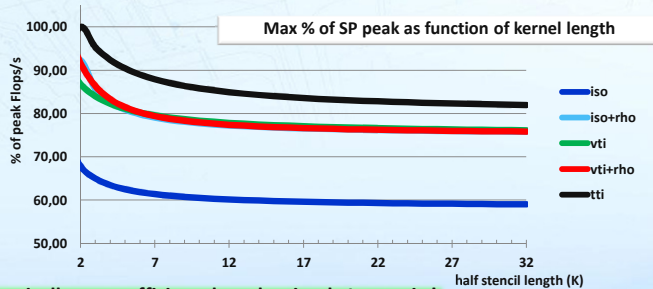
- 8 MUL (32b) and 8 ADD (32b): 16 Single Precision Flops / cycle
- 4 MUL (64b) and 4ADD (64b): 8 Double Precision Flops / cycle

$$Peak = (Flops / cycle) * (cycle / sec) = Flops / sec$$



### Flops/s Wall : FP operations

	Nb of tables	implementation	ADD	MUL	Total flops
Iso	3	$\frac{\partial^2 P}{\partial t^2} = \rho \frac{\partial^2 P}{\partial x^2}$	6*K + 2	K+3	7*K+5
Iso, rho	6	$\frac{1}{\rho c^2} \frac{\partial^2 P}{\partial t^2} = \nabla \cdot \left( \frac{\nabla P}{\rho} \right)$	36*K - 95	19 *K - 46	56*K - 141
VTI	7	Duveneck et al, 2011	6*K + 7	3*K + 8	9*K + 15
VTI, rho	10	Duveneck et al, 2011	36*K - 90	19 *K - 42	56*K - 132
TTI	24	Fletcher et al., 2009	30*K + 38	18*K + 62	48*K+100
Visco	n/a	Komatish et al., 2009	n/a	n/a	n/a



TTI kernel is theoretically more efficient than the simple Isotropic !

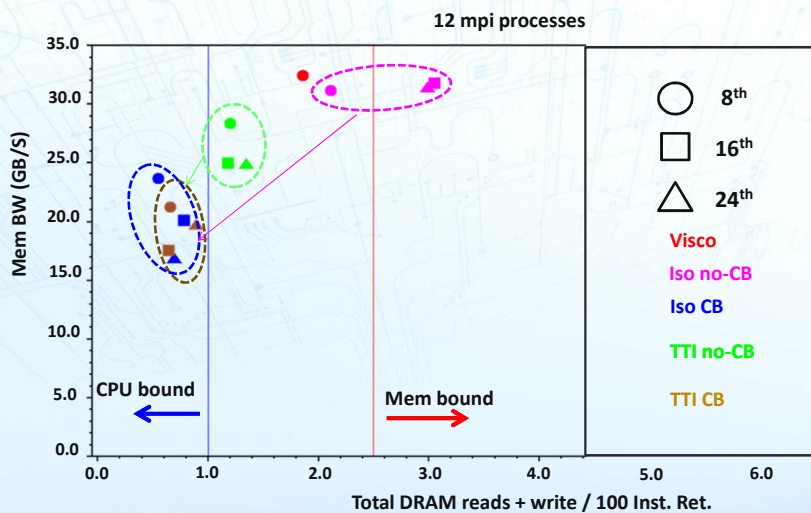
47 / 75

Software & Services Group, Energy Engineering Team

$$\% \text{Peak}(K) = 100.0 * [ nF / 2 * \max(nA, nM) ]$$



### DRAM BW vs Total DRAM Reads + Writes



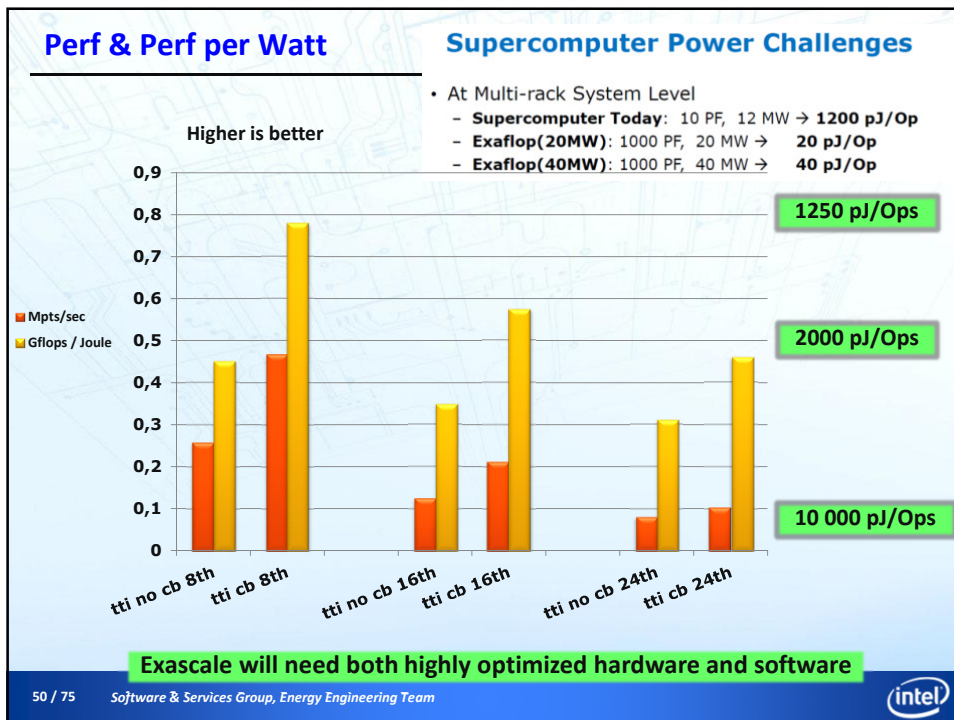
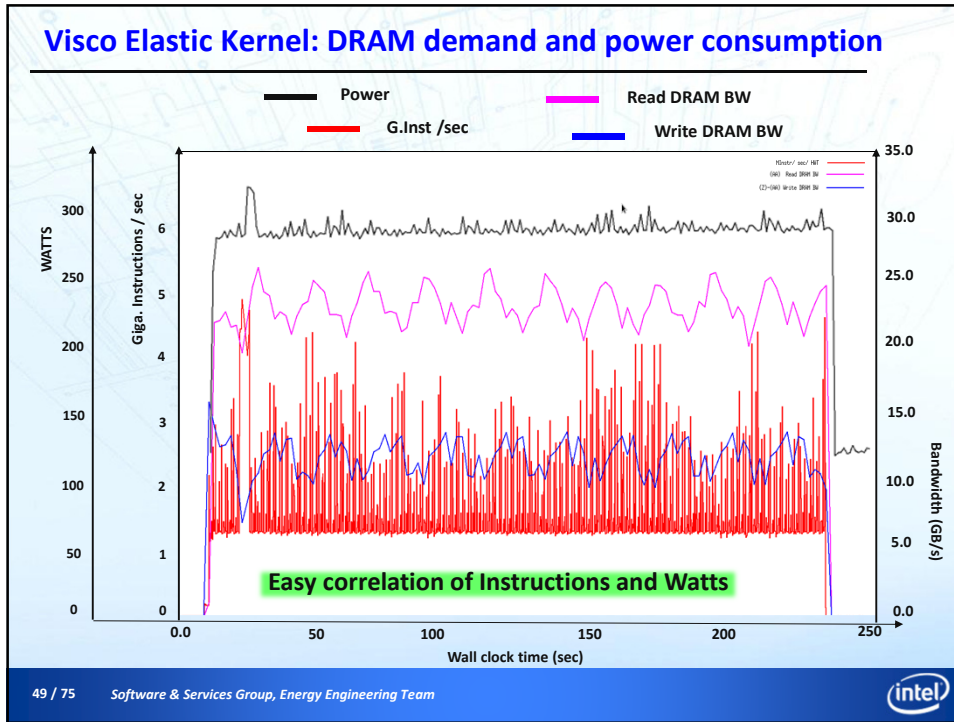
TTI - 3DFD is definitively not bandwidth bound

48 / 75

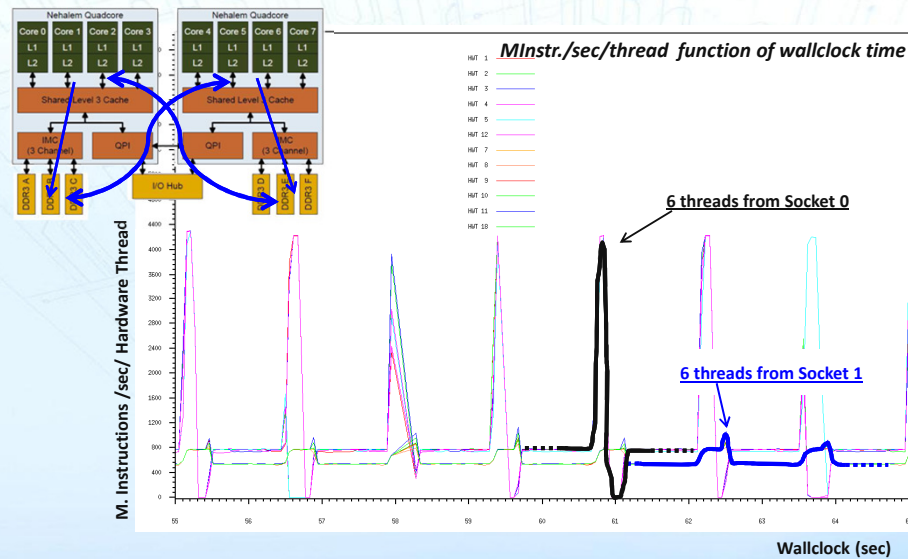
Software & Services Group, Energy Engineering Team







## Example of Non Uniform Memory Access (NUMA)



51 / 75 Software & Services Group, Energy Engineering Team

SEG 2011

51



## FD (and RTM) BKM: what may work .. or not

- Compiler option ( -xSSE4.2,AVX, -ftz, -ipo) + specific loop directives
- SSE/AVX implementation
  - difficult when you have many approximations with several stencils lengths
- Data layout: valuable in any case
- Cache Blocking:
  - works fine on kernel , less impact on whole application
  - impossible with variable stencil lengths and independent partial derivatives
- Loop splitting for complex equation
  - valuable for independant partial derivatives
  - impossible for variable stencil lengths
- Domain decomposition / Data decomposition
  - MPI domain decomposition useful to save memory per node (to avoid i/o for example)
  - batch scheduler distribution of 1 shot / node : no MPI needed
- NUMA-aware Data Placement
  - mandatory for Hybrid MPI + OMP
- Dedicated optimization for MIC

52 / 75 Software & Services Group, Energy Engineering Team



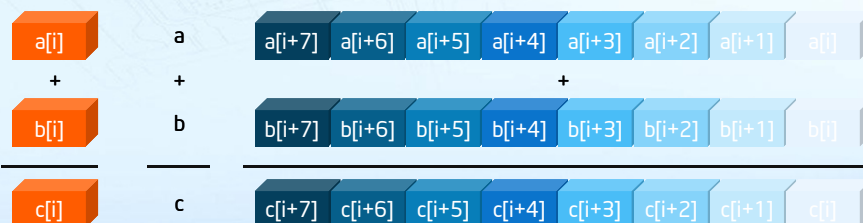
## Agenda :

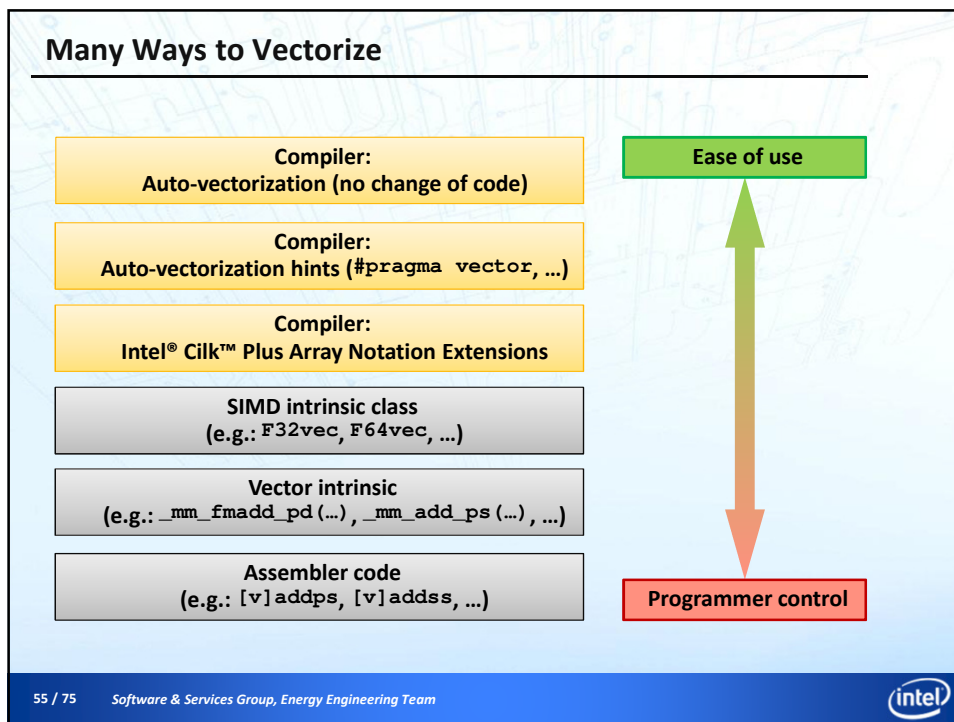
## Application Point of View Just Make it

### Vectorization of Code

- Transform sequential code to exploit vector processing capabilities (SIMD)
  - Manually by explicit syntax
  - Automatically by tools like a compiler

```
for(i = 0; i <= MAX; i++)
  c[i] = a[i] + b[i];
```





## Control Vectorization !

Provides details on vectorization success & failure:  
Linux\*, Mac OS\* X: **-vec-report<n>**, Windows\*: **/Qvec-report<n>**

n	Diagnostic Messages
0	Tells the vectorizer to report no diagnostic information. Useful for turning off reporting in case it was enabled on command line earlier.
1	Tells the vectorizer to report on vectorized loops. [default if n missing]
2	Tells the vectorizer to report on vectorized and non-vectorized loops.
3	Tells the vectorizer to report on vectorized and non-vectorized loops and any proven or assumed data dependences.
4	Tells the vectorizer to report on non-vectorized loops.
5	Tells the vectorizer to report on non-vectorized loops and the reason why they were not vectorized.
6*	Tells the vectorizer to use greater detail when reporting on vectorized and non-vectorized loops and any proven or assumed data dependences.

\*: First available with Intel® Composer XE 2013

56 / 75 Software & Services Group, Energy Engineering Team 1/31/2013

## Vectorization Report II

```

35:  subroutine fd( y )
36:  integer :: i
37:  real, dimension(10), intent(inout) :: y
38:  do i=2,10
39:      y(i) = y(i-1) + 1
40:  end do
41:  end subroutine fd

```

```

novec.f90(38): (col. 3) remark: loop was not vectorized: existence
of vector dependence.
novec.f90(39): (col. 5) remark: vector dependence: proven FLOW
dependence between y line 39, and y line 39.
novec.f90(38:3-38:3):VEC:MAIN_: loop was not vectorized:
existence of vector dependence

```

### Note:

In case inter-procedural optimization (`-ipo` or `/Qipo`) is activated and compilation and linking are separate compiler invocations, the switch to enable reporting needs to be added to the link step!



## Reasons for Vectorization Fails & How to Succeed

- Most frequent reason is **Dependence**:  
Minimize dependencies among iterations by design!
- **Alignment**: Align your arrays/data structures
- **Function calls in loop body**: Use aggressive in-lining (IPO)
- **Complex control flow/conditional branches**:  
Avoid them in loops by creating multiple versions of loops
- **Unsupported loop structure**: Use loop invariant expressions
- **Not inner loop**: Manual loop interchange possible?
- **Mixed data types**: Avoid type conversions
- **Non-unit stride between elements**: Possible to change algorithm to allow linear/consecutive access?
- **Loop body too complex reports**: Try splitting up the loops!
- **Vectorization seems inefficient reports**: Enforce vectorization, benchmark !



## IVDEP vs. SIMD Pragma/Directives

Differences between IVDEP & SIMD pragmas/directives:

- `#pragma ivdep` (C/C++) or `!DIR$ IVDEP` (Fortran)

- Ignore vector dependencies (IVDEP):

- Compiler ignores assumed but not proven dependencies for a loop

- Example:

```
void foo(int *a, int k, int c, int m)
{
    #pragma ivdep
    for (int i = 0; i < m; i++)
        a[i] = a[i + k] * c;
}
```

- `#pragma simd` (C/C++) or `!DIR$ SIMD` (Fortran):

- Aggressive version of IVDEP: Ignores all dependencies inside a loop

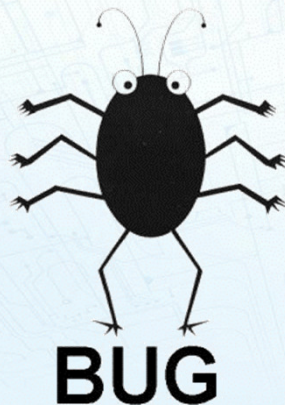
- It's an imperative that forces the compiler try everything to vectorize

- Efficiency heuristic is ignored

- **Attention: This can break semantically correct code!**

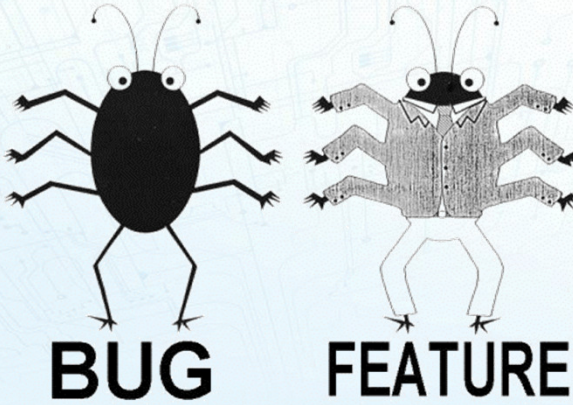
However, it can vectorize code legally in some cases that wouldn't be possible otherwise!

## Value Safety



*"It may be very difficult to understand that two simulations of the same process with the same code and the same parameters on the same computer give different results"*

## Value Safety



**BUG**      **FEATURE**

*“It may be very difficult to understand that two simulations of the same process with the same code and the same parameters on the same computer give different results”*

61 / 75      Software & Services Group, Energy Engineering Team      intel

## Floating Point (FP) Programming Objectives

- **Accuracy**
  - Produce results that are “close” to the correct value
    - ✓ Measured in relative error, possibly in ulp
- **Reproducibility**
  - Produce consistent results
    - ✓ From one run to the next
    - ✓ From one set of build options to another
    - ✓ From one compiler to another
    - ✓ From one platform to another
- **Performance**
  - Produce the most efficient code possible

These options usually conflict!  
Judicious use of compiler options lets you control the tradeoffs.  
Different compilers have different defaults.

62 / 75      Software & Services Group, Energy Engineering Team      intel

## Users are Interested in Consistent Numerical Results

- **Root cause for variations in results**
  - floating-point numbers → order of computation matters!
  - Single precision arithmetic example  $(a+b)+c \neq a+(b+c)$

$$2^{26} - 2^{26} + 1 = 1 \quad (\text{infinitely precise result})$$
$$(2^{26} - 2^{26}) + 1 = 1 \quad (\text{correct IEEE single precision result})$$
$$2^{26} - (2^{26} - 1) = 0 \quad (\text{correct IEEE single precision result})$$

- **Conditions that affect the order of computations**

- Different code branches ( e.g. SSE2 versus AVX )
- Memory alignment ( scalar or vector code )
- Dynamic parallel task / thread / rank scheduling

- **Bitwise repeatable/reproducible results**

**repeatable** = results the same as last run (same conditions)

**reproducible** = results the same as results in other environments

Environments = OS / architecture / # threads / CPU /

```
4.012345678901111
4.012345678902222
4.012345678902222
4.012345678901111
4.012345678902222
4.012345678901111
4.012345678901111
4.012345678901111
4.012345678902222
4.012345678902222
4.012345678901111
4.012345678902222
4.012345678901111
4.012345678902222
4.012345678902222
4.012345678901111
...
```



## The -fp-model switch

- **-fp-model**
  - fast [=1] allows value-unsafe optimizations (default)
  - fast=2 allows additional approximations (very unsafe)
  - precise value-safe optimizations only (also source, double, extended)
  - except enable floating point exception semantics
  - strict precise + except + disable fma + don't assume default floating-point environment
- Replaces old switches -mp, -fp-port, etc (don't use!)
- **-fp-model precise -fp-model source**
  - recommended for ANSI/ IEEE standards compliance, C++ & Fortran
  - "source" is default with "precise" on Intel 64 Linux





## Value Safety

ANSI/ IEEE standards compliance C++ & Fortran:

**-fp-model source** or **-fp-model precise**

- Prevents vectorization of reductions
- No use of “fast” division or square root

Ensures ‘Value Safety’ by disallowing:

$x / x \Leftrightarrow 1.0$	x could be 0.0, $\infty$ , or NaN
$x - y \Leftrightarrow -(y - x)$	If x equals y, $x - y$ is +0.0 while $-(y - x)$ is -0.0
$x - x \Leftrightarrow 0.0$	x could be $\infty$ or NaN
$x * 0.0 \Leftrightarrow 0.0$	x could be -0.0, $\infty$ , or NaN
$x + 0.0 \Leftrightarrow x$	x could be -0.0
$(x + y) + z \Leftrightarrow x + (y + z)$	General reassociation is not value safe
$(x == x) \Leftrightarrow \text{true}$	x could be NaN

## Value Safety

Affected Optimizations include:

- Reassociation
- Flush-to-zero
- Expression Evaluation, various mathematical simplifications
- Math library approximations
- Approximate divide and sqrt

**[-no]-prec-div /Qprec-div[-]**

- Enables[disables] various divide optimizations
  - $x / y \Leftrightarrow x * (1.0 / y)$
  - Approximate divide and reciprocal

**[-no]-prec-sqrt /Qprec-sqrt[-]**

- Enables[disables] approximate sqrt and reciprocal sqrt

## Math Libraries – known issues

- Differences could potentially arise between:
  - ✓ Different compiler releases, due to algorithm improvements
    - ✓ Use `-fimf-precision`
  - ✓ Different platforms, due to different algorithms or different code paths at runtime
    - ✓ Libraries detect run-time processor internally
    - ✓ Independent of compiler switches
    - ✓ use `-fimf-arch-consistency=true`
  - ✓ Expected accuracy is maintained
    - ✓ 0.55 ulp for libimf
    - ✓ < 4 ulp for libsvml (default for vectorized loops)

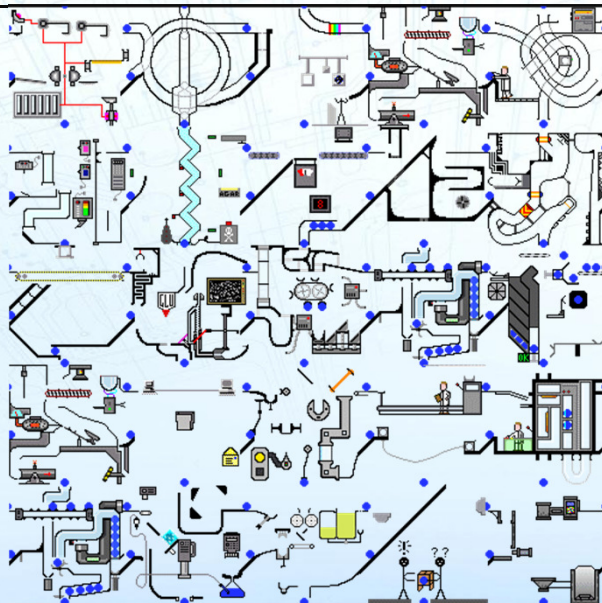
## Intel® Math Kernel Library

- Linear algebra, FFTs, sparse solvers, statistical, ...
  - Highly optimized, vectorized
  - Threaded internally using OpenMP\*
  - Repeated runs may not give identical results
- **Conditional BitWise Reproducibility**
  - Repeated runs give identical results under certain conditions:
    - Same number of threads
    - `OMP_SCHEDULE=static` (the default)
    - Same OS and architecture (e.g. Intel 64)
    - Same microarchitecture, or specify a minimum microarchitecture
    - Consistent data alignment
  - Call `mkl_bwr_set(...)`

## Reproducibility of Reductions in OpenMP\*

- Each thread has its own partial sum
  - Breakdown, & hence results, depend on number of threads
  - Partial sums are summed at end of loop
  - Order of partial sums is undefined (OpenMP standard)
    - First come, first served
    - Result may vary from run to run (even for same # of threads)
    - For both gcc and icc
    - Can be more accurate than serial sum
  - For icc, option to define the order of partial sums (tree)
    - Makes results reproducible from run to run
    - `export KMP_FORCE_REDUCTION=tree` (may change!)
      - ✓ May also help accuracy
      - ✓ Possible slight performance impact, depends on context
      - ✓ Requires static scheduling, fixed number of threads
      - ✓ currently undocumented ("black belt", at your own risk)
      - ✓ See example

## Conclusions



## Conclusions 1

- Amdahl is back (or never died) in 2D
  - Application must be 100% // and 100% SIMD
- Only your application can tell
  - How far are you from the peak flops and peak BW ?
  - What is the current impact of vectorization ?
  - How parallel it is
- What execution model (offload, native, symmetric) ?
  - Keep advantage of all CPU cores + Coprocessor
    - Power efficiency
    - Portable workload



71 / 75

Software &amp; Services Group, Energy Engineering Team



## Conclusion 2 : Perf measurement

- Use of the right metrics for performance measurements
- Know the max theoretical performance of your implementation
- Don't forget system configuration and its impact on measurements
- Simple projections are usefull
- Final goal of perf modelling must be clearly defined
  - Short term optimization with current kernel and hwd
  - Short / mid term extrapolation for future hardware
  - Long term extrapolation with future kernels and future hardwares

72 / 75

Software &amp; Services Group, Energy Engineering Team

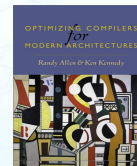


## Conclusion 3

- Keep portability in mind
- There is no free lunch with magic hardware , be sure you take the most from the actual one
- Validate your results
  - As usual on the physical side with analytical solution when possible
  - Keep in mind « aggressive optim » and validate with secured options

## References

- [1] Aart Bik: "The Software Vectorization Handbook"  
[http://www.intel.com/intelpress/sum\\_vmmx.htm](http://www.intel.com/intelpress/sum_vmmx.htm)
- [2] Randy Allen, Ken Kennedy: "Optimizing Compilers for Modern Architectures: A Dependence-based Approach"
- [3] Steven S. Muchnik, "Advanced Compiler Design and Implementation"
- [4] Intel Software Forums, Knowledge Base, White Papers, Tools Support (see <http://software.intel.com>)  
Sample Articles:
  - <http://software.intel.com/en-us/articles/a-guide-to-auto-vectorization-with-intel-c-compilers/>
  - <http://software.intel.com/en-us/articles/requirements-for-vectorizable-loops/>
  - <http://software.intel.com/en-us/articles/performance-tools-for-software-developers-intel-compiler-options-for-sse-generation-and-processor-specific-optimizations/>
- The Intel® C++ and Fortran Compiler Documentation, "Floating Point Operations"
- "Consistency of Floating-Point Results using the Intel® Compiler" <http://software.intel.com/en-us/articles/consistency-of-floating-point-results-using-the-intel-compiler/>
- Goldberg, David: "What Every Computer Scientist Should Know About Floating-Point Arithmetic" *Computing Surveys*, March 1991, pg. 203
- the new Intel® BWR features – see this [article](#) for more details
- We need your feedback on missing, failing or suboptimal compiler functionality
- Please file a Premier case or post your findings/wishes to the compiler user forum



## Questions



*"Prediction is very difficult, especially about the future"*  
 by Niels Bohr, Physicist, 1885-1962

 Software & Services Group, Energy Engineering Team

## Optimization Notice

### Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel® Compiler User and Reference Guides" under "Compiler Options." Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools 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 Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® 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.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101

## Legal Disclaimer

---

### Legal Disclaimer

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.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference [www.intel.com/software/products](http://www.intel.com/software/products).

Intel 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.

Copyright © 2010. Intel Corporation.