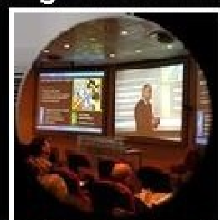
Optimisation of weather applications on Power and x86 architectures

with a focus on reproducibility

IBM Montpellier, France

Infrastructure Solutions Center Activities

The Center of Infrastructure Solutions partners with clients to meet their IT infrastructure goals and improve their overall business by demonstrating the capabilities of the IBM server, storage and software portfolio. This is accomplished through a comprehensive approach of designing, developing, benchmarking and validating solutions.

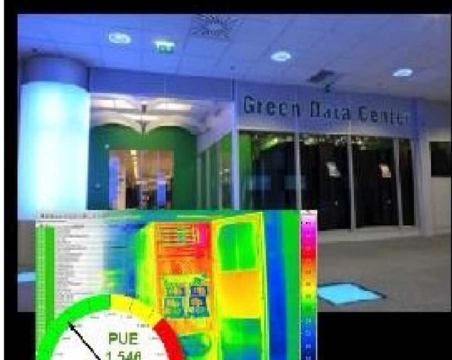


Talk & Teach

Design

Prove

Technology Innovation
First Green Data Center in Europe

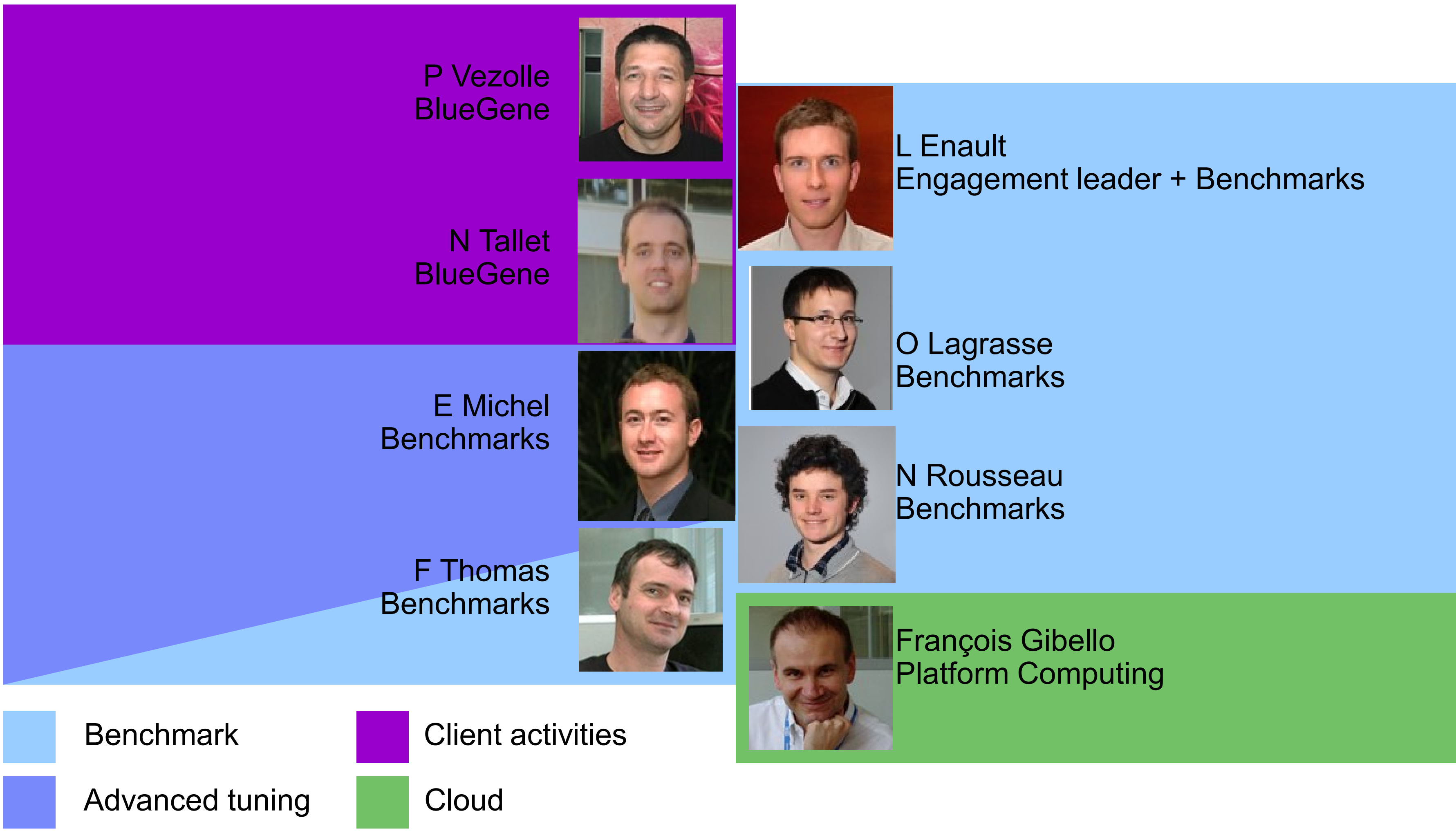


In 2011, 7000 visitors representing companies from 70 countries participated in over 3100 client engagements at our center:

311 Briefings & Conferences, 260 Smarter Computing & Design Workshops, 197 ISV Solutions, 379 z New Technology Assessment, 324 Benchmarks & Testing, 1700 Demos & zTEC Proof of Technology

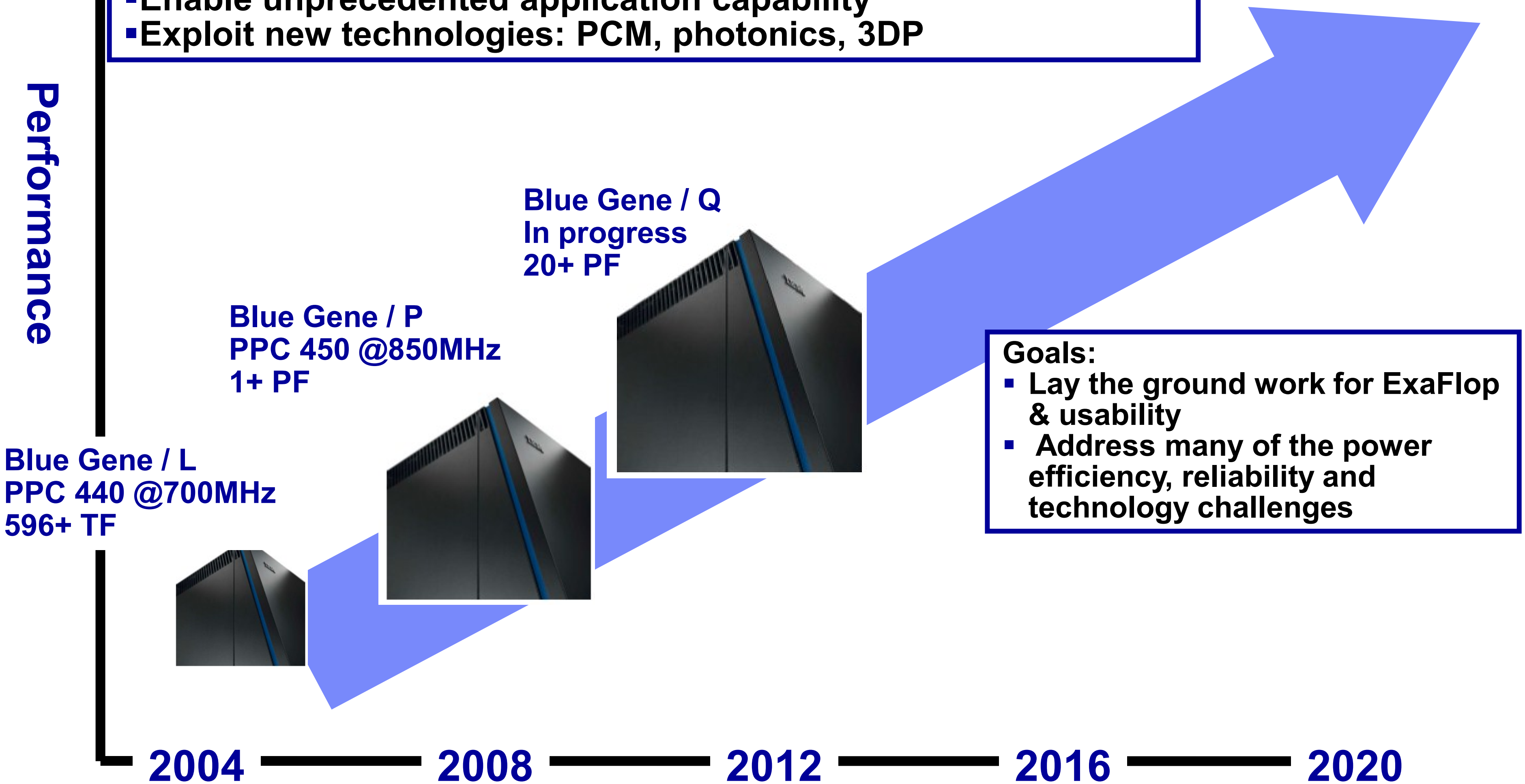
Provisioning & Virtualization live in our Cloud Data Center

IBM Montpellier HPC team



Blue Gene

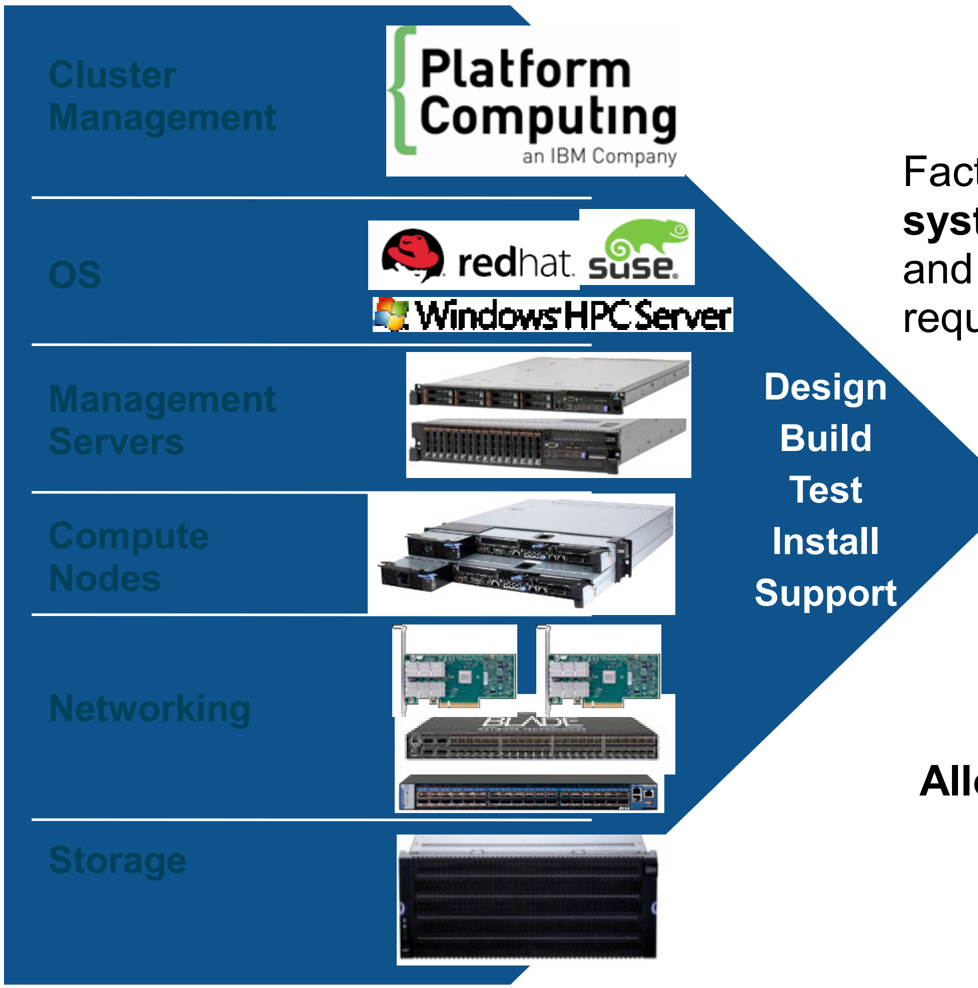
Goals:
- **Three orders of magnitude performance in 10 years**
- **Push state of the art in Power efficiency, scalability, & reliability**
- **Enable unprecedented application capability**
- **Exploit new technologies: PCM, photonics, 3DP**



IBM Intelligent Cluster – it's about faster time-to-solution

Take the time and risk out Technical Computing deployment

Building Blocks: Industry-leading IBM and 3rd Party components

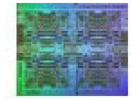


IBM Intelligent Cluster

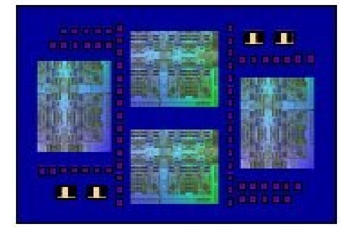
Factory-integrated, interoperability-tested **system** with compute, storage, networking and cluster management tailored to your requirements and supported as a solution!

Allows clients to focus on their business not their IT – that is backed by IBM

POWER7-IH system hierarchy



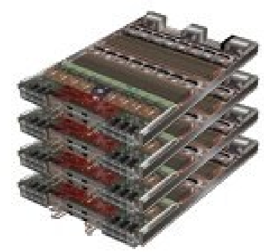
- **POWER7 processor**
  - 8 Cores



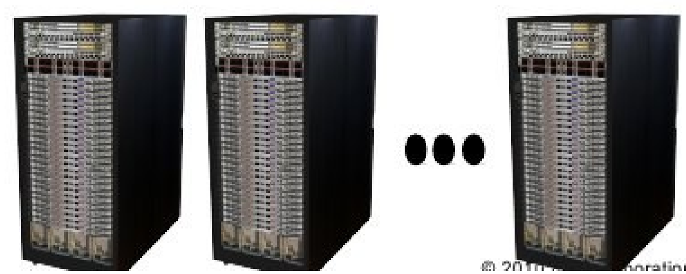
Hub Chip

- **POWER7 QCM & Hub Chips**
  - QCM: 4 POWER7 processors
    - 32 core SMP Image
  - Hub Chip: 1 per QCM
    - Interconnects QCM, Nodes, and Super Nodes

- **POWER7 IH Node**
  - 2U drawer / Node
  - 8 QCMs
    - 256 Cores



- **POWER7 Super Node**
  - 4 drawers / Nodes
    - 1024 Cores
- **Full System**
  - Up to 512 Super Nodes
  - 512K Cores



WW HPC Benchmark Centers

Agenda

Live experiments

Reproducibility requirements

Why is that computation not reproducible ?

What can we do ?

Agenda

Live experiments

Reproducibility requirements

Why is that computation not reproducible ?

What can we do ?

Let's run the same code on a few systems...

```
$ for sys in power7 x86 sparc
do
  ssh $sys a.out
done
42.00000000
42.00000001
41.99999998
```

Oh well, let's stick to x86 only...

```
$ for sys in oldxeon wsm snb snb-mic0
do
  ssh $sys a.out
done
42.00000000
42.00000001
42.00000000
42.00000002
```

Sandy Bridge is what really matters today...

```
[snb]$ for mpi*omp in 48*8 96*4 192*2
do
  a.out
done
42.00000000
42.00000001
42.00000000
```

At least, “that” should work...

```
[snb]$ for repetitions in 1..10
do
  a.out
done
42.00000000
42.00000000
42.00000000
42.00000001
42.00000000
```

Agenda

Live experiments

Reproducibility requirements

Why is that computation not reproducible ?

What can we do ?

Maybe you can live without reproducibility ?

- All floating point computations are wrong anyway
- Initial conditions can be very uncertain too (ensemble)
- Numerical schemes should be made insensitive to tiny errors
- That's what Intel compilers think at least...

Or maybe you need reproducibility after all ?

- Regulatory requirement
  - Nuclear reactors design
  - Automotive crash simulation
  - Aircraft engines
- Weather and climate studies
- Software QA
  - Bit wise reproducibility is a great debugging aid !
- Can only be harder to achieve in the future :-(

Various flavours of non-reproducibility

- From **run to run**
  - Nothing changed
  - Fixed mpi*omp or even a sequential program
  - On the same machine
- From **mpi*omp to mpi'*omp'**
- From one set of compiler options to another (**Debug vs Release**)
- From one **architecture** to another

Agenda

Live experiments

Reproducibility requirements

Why is that computation not reproducible ?

What can we do ?

How can we get non-reproducible results from run to run ?

- It takes a combination of
  - Code sensitive to the order of computations
    - « **reduction** » operations (DDOT, DGEMM)
  - A **SIMD** instruction set (SSE, AVX, VSX)
  - Non deterministic memory **alignment**
  - In your code or in someone else's (**MKL**, ESSL)

- **malloc()/allocate()** do not always return 16 bytes (SSE/VSX) or 32 (AVX) aligned data

- Heap and stack alignment can **vary** due to
  - varying run time conditions (date, directory, pid, ...)
  - ASLR (Address Space Layout Randomization)
    - check /proc/sys/kernel/randomize_va_space

- The compiler will process loops with a **prologue** (scalar) up to the first aligned index, the loop **body** (SIMD) and an **epilogue** (scalar).

Why can we get non-reproducible results from run to run ?

```
double ddot(int n,
  double *a,double *b)
{
  double  sum=0.0;
  int i;
  for(i=0;i<n;i++) {
    sum+=a[i]*b[i];
  }
  return sum;
}
$ icc -O2/-O3 -xAVX -S ddot.c
```

```
… loop prologue

..B1.12:
        vmovupd   (%rsi,%r8,8), %xmm2
        vmovupd   96(%rsi,%r8,8), %xmm10
        vmulpd    (%rdx,%r8,8), %ymm3,
    %ymm4
        vaddpd    %ymm4, %ymm1, %ymm8
… unrolled by 4
        je ..B1.12

… loop epilogue
```

vmulpd packed double (AVX)

The result depends on alignment of a and b

Why can we get non-reproducible results from run to run ?

```
double ddot(int n,
  double *a,double *b)
{
  double  sum=0.0;
  int i;
  for(i=0;i<n;i++) {
    sum+=a[i]*b[i];
  }
  return sum;
}
$ gcc -O3 -mavx -ftree-vectorize -S ddot.c
```

```
L4:
        vmovsd    8(%rsi,%r8,8),%xmm0
        vmulsd    8(%rdx,%r8,8),%xmm0,
    %xmm4
        vaddsd    %xmm4, %xmm3, %xmm0
        jb        L4
```

`vmulsd scalar double (scalar AVX)`

Why can we get non-reproducible results from run to run ?

```
double ddot(int n,
  double *a,double *b)
{
  double  sum=0.0;
  int i;
  for(i=0;i<n;i++) {
    sum+=a[i]*b[i];
  }
  return sum;
}
```

```
..B1.4:
        vmovsd    8(%rsi,%r8,8),%xmm0
        vmulsd    8(%rdx,%r8,8),%xmm0,
    %xmm4
        vaddsd    %xmm4, %xmm3, %xmm0
        jb        ..B1.4
```

vmulsd scalar double (scalar AVX)

```
$ icc -O2/-O3 -xAVX -fp-model precise
```

Agenda

Live experiments

Reproducibility requirements

Why is that computation not reproducible ?

What can we do ?

What can you do ?

- Don't use **SIMD** at all
- Don't use SIMD for **reductions**
- Don't use reductions
  - Cast a DGEMM in terms of DAXPYs rather than DDOTs
- Use « safe » **compiler options**
- Don't use **MKL**, it might do bad things without warning you
  - ESSL's DGEMM is reproducible and alignment safe
- Wrap your memory allocations so that they return **consistently aligned addresses**

A nice feature of the GNU linker : wrap

```
$ cat wrap_malloc.c
#include <stdlib.h>
void *__wrap_malloc(size_t bytes)
{
   void *p;
   if ( (posix_memalign(&p,128,bytes) != 0)) { // 128=SIMD length
      p=(void *)0;
   }
   return p;
}
$ gcc -Wl,-wrap,malloc -o a.out main.o objects.o wrap_malloc.o
```

All references to malloc() will be resolved in our __wrap_malloc() routine

(Sort of) Safe Intel compiler options

- -O3 -xAVX -fp-model precise -assume protect_parens -prec-div -prec-sqrt -no-ftz -nolib-inline
  - **-fp-model precise** : Won't SIMDize reductions
  - **-assume protect_parens** : Comply with parentheses
  - **-prec-div** : no fancy divide
  - **-prec-sqrt** : no fancy sqrt
  - **-no-ftz** : do not flush denormals to zero
  - **-nolib-inline** : do not use inline optimized math functions

- Performance **hit** : 10-15-20 % ?

- **-no-vec** will turn off SIMD code generation altogether

- Recover performance using the **#pragma simd/!DIR$ SIMD** directives around hot loops

Recent additions to Intel compilers and MKL to help reproducibility

- The very latest Intel Composer XE 2013 and MKL 11.0 bring features around « **CBWR** » : Conditional Bit-Wise Reproducibility

- MKL contains multiple code paths for the same function (SSE2, SSE4.2, AVX). An application can require that the same code path be followed on all platforms (**MKL_CBWR=COMPATIBLE, SSE4_2, AVX**,...)

- For OpenMP reductions, use **KMP_DETERMINISTIC_REDUCTION=yes**

OpenMP and MPI have a lot to offer in the reproducibility violation department

- **!OMP$ PARALLEL FOR REDUCTION (+:SUM)**

- **`OMP_SCHEDULE=dynamic`** or guided
  - Calling for trouble if the order of computations within the loop does matter

- How reproducible is MPI_SUM in **MPI_Allreduce** ?

- Just like OMP_SCHEDULE :

```
DO I=1,NOBS
      CALL_MPI_RECV(A,1,MPI_REAL8,MPI_SOURCE_ANY, MPI_TAG_ANY,...)
      SUM=SUM+A
ENDDO
```

What's next ?

- Bit wise reproducibility across <runs,MPI,OpenMP,what not> is **nice** (brings trust)
- It may **hurt performance**, although compilers can help by selecting critical areas of code
- Involves components that may **not be under your control** (math libraries, parallel runtime)
- Will be harder to achieve in the **future** (NTV, higher levels of parallelism, more SIMD, hybrid systems, accelerators, FPGA)
- Does not play nice with **performance** and **power consumption**
- Reproducibility is partially addressed by people studying the **resilience of HPC applications**
- A lot to be done in little time (2018 is approaching fast)
- My post-Mayascale prediction : « The weather forecast for Dec, 21st, 2018 could be different from the weather forecast for Dec, 21st, 2018, itself different from the weather forecast for Dec, 21st, 2018. »