



# *Moving Forward with OpenMP\* Implementation in LLVM and Clang*

*Xinmin Tian, Alexey Bataev, Andrey S. Bokhanko, James H. Cownie, Ayal Zaks  
Intel Corporation  
November 15th, 2015*

*SC'2015 LLVM-HPC2 Workshop*



# Legal Notice and Disclaimers

By using this document, in addition to any agreements you have with Intel, you accept the terms set forth below.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

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>

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 complete information about performance and benchmark results, visit [Performance Test Disclosure](#)**

Intel does not control or audit the design or implementation of third party benchmark data or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmark data are reported and confirm whether the referenced benchmark data are accurate and reflect performance of systems available for purchase.

Intel processor numbers are not a measure of performance.

Processor numbers differentiate features within each processor family, not across different processor families: Go to: [Learn About Intel® Processor Numbers](#)

Intel® Advanced Vector Extensions (Intel® AVX)\* are designed to achieve higher throughput to certain integer and floating point 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 should consult your system manufacturer for more information.

\*Intel® Advanced Vector Extensions refers to Intel® AVX, Intel® AVX2 or Intel® AVX-512. For more information on Intel® Turbo Boost Technology 2.0, visit <http://www.intel.com/go/turbo>

## Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Copyright © 2014 Intel Corporation. All rights reserved. Intel, the Intel logo, Intel Xeon, and Intel Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries. \*Other names and brands may be claimed as the property of others.



# Agenda

OpenMP\* Programming Model's New Era

Programming Model Overview

OpenMP\* Support in Clang/LLVM: A Brief History

10000ft View: A High-Level Design for Moving Forward

- ✓ Design Guidelines
- ✓ Back-End: LLVM Prepass, Lowering and Outlining
- ✓ An Example

OpenMP\* SIMD extension support in LLVM

- ✓ Declare SIMD support
- ✓ Vectorizing Loops with math function calls

Summary

## *What is OpenMP?*

- ✓ De-facto standard Application Programming Interface (API) to write shared memory parallel applications in C, C++, and Fortran
- ✓ Consists of Compiler Directives, Runtime routines and Environment variables
- ✓ Specification maintained by the OpenMP Architecture Review Board (<http://www.openmp.org>)
- ✓ New ARB mission statement:
  - ✓ “The OpenMP ARB mission is to standardize directive-based multi-language high-level parallelism that is performant, productive and portable.”
- ✓ **OpenMP\* Specification Version 4.5 was launched in Now at SC'2015**

# OpenMP\* Programming Model's New Era

- ✓ CPUs and All forms of accelerators/coprocessors, GPU, APU, GPGPU, FPGA, and DSP
- ✓ Heterogenous consumer devices
  - ✓ Kitchen appliances, drones, signal processors, medical imaging, auto, telecom, automation, not just graphics engines – (Courtesy of Michael Wong (IBM) and Alexey Bataev (intel), et.al. LLVM Developer Conference Oct. 2105)



Search OpenMP.org

Google™ Custom Search

Search

Archives

- June 2014
- April 2014
- March 2014
- February 2014
- January 2014
- December 2013
- November 2013
- September 2013
- July 2013
- May 2013
- April 2013
- March 2013
- February 2013
- January 2013
- December 2012
- November 2012
- October 2012
- September 2012
- July 2012
- June 2012
- May 2012
- April 2012
- March 2012
- February 2012
- January 2012
- November 2011

Members

Permanent Members of the ARB:

- AMD (Dibyendu Das)
- Convey Computer (Kirby Collins)
- Cray (James Beyer/Luiz DeRose)
- Fujitsu (Eiji Yamanaka)
- HP (Sujoy Saraswati)
- IBM (Kelvin Li)
- Intel (Xinmin Tian)
- NEC (Kazuhiro Kusano)
- NVIDIA (Jeff Larkin)
- Oracle Corporation (Nawal Copty)
- Red Hat (Matt Newsome)
- ST Microelectronics (Christian Bertin)
- Texas Instruments (Andy Fritsch)

Auxiliary Members of the ARB:

- ANL (Kalyan Kumaran)
- ASC/LLNL (Bronis R. de Supinski)
- BSC (Xavier Martorell)
- cOMPunity (Barbara Chapman)
- EPCC (Mark Bull)
- LANL (David Montoya)
- NASA (Henry Jin)
- ORNL (Oscar Hernandez)
- RWTH Aachen University (Dieter an Mey)
- SNL-Sandia National Lab (Steven Oliver)
- Texas Advanced Computing Center (Kent Milfeld)
- University of Houston (Yonghong Yan/Barbara Chapman)

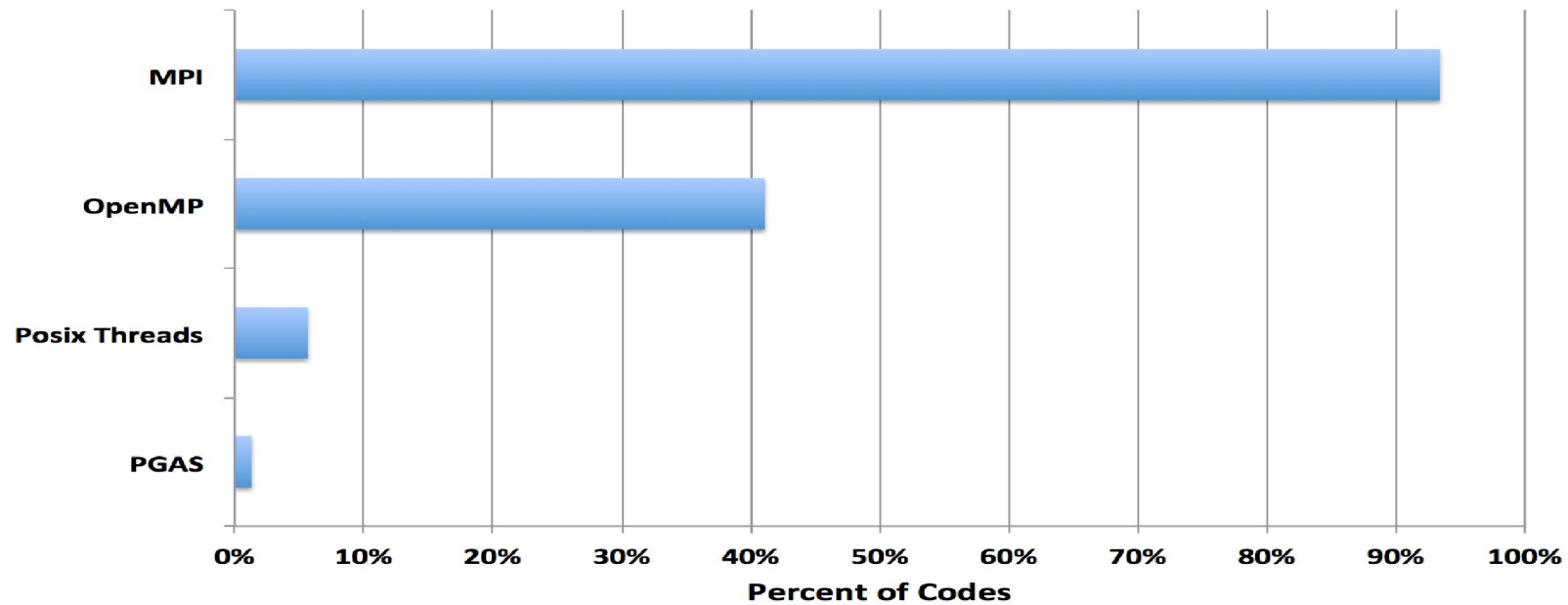
*OpenMP is widely supported by the industry, as well as the academic community*



# Programming Models Used at NERSC

- ✓ MPI dominates
- ✓ 40% of projects use OpenMP\*

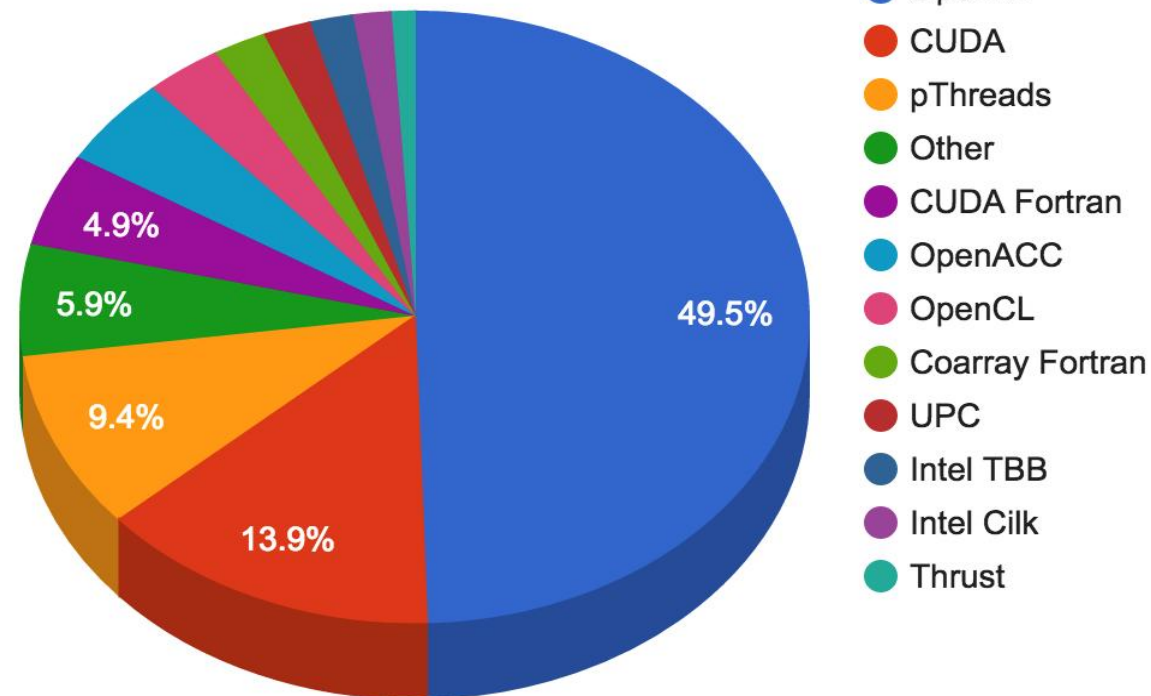
**Programming Models Used at NERSC 2015**  
(Taken from allocation request form. Sums to >100% because codes use multiple languages)



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

## What is X if Use MPI+X at NERSC

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

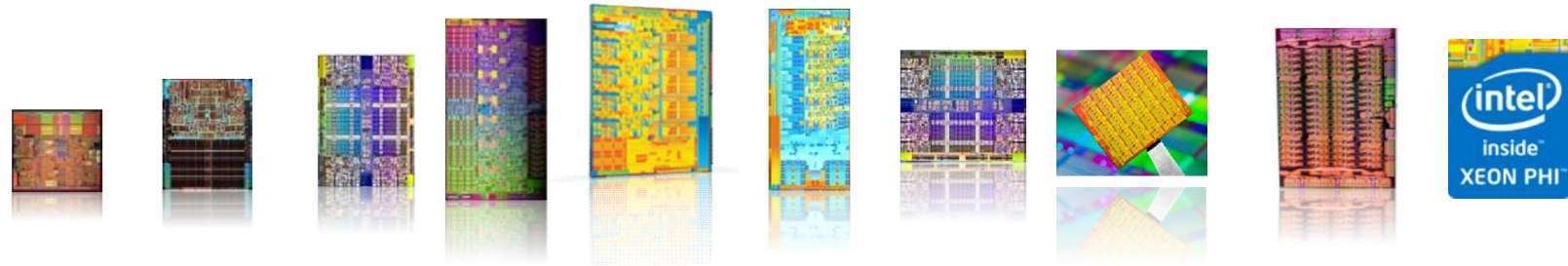


Courtesy of Yun (Helen) He, Alice Koniges, et. al., (NERSC) at OpenMPCon'2015



# Parallel + SIMD is the Path Forward

*Intel® Xeon® and Intel® Xeon Phi™ Product Families are both going parallel*



	Intel® Xeon® processor 64-bit	Intel® Xeon® processor 5100 series	Intel® Xeon® processor 5500 series	Intel® Xeon® processor 5600 series	Intel® Xeon® processor code-named Sandy Bridge EP	Intel® Xeon® processor code-named Ivy Bridge EP	Intel® Xeon® processor code-named Haswell EP	Intel® Xeon® Processor codenamed Skylake EP	Intel® Xeon Phi™ coprocessor Knights Corner	Intel® Xeon Phi™ coprocessor & coprocessor Knights Landing <sup>1</sup>
Core(s)	1	2	4	6	8	12	18	28	61	70+
Threads	2	2	8	12	16	24	36	56	244	280+
SIMD Width	128	128	128	128	256	256	256	512	512	512

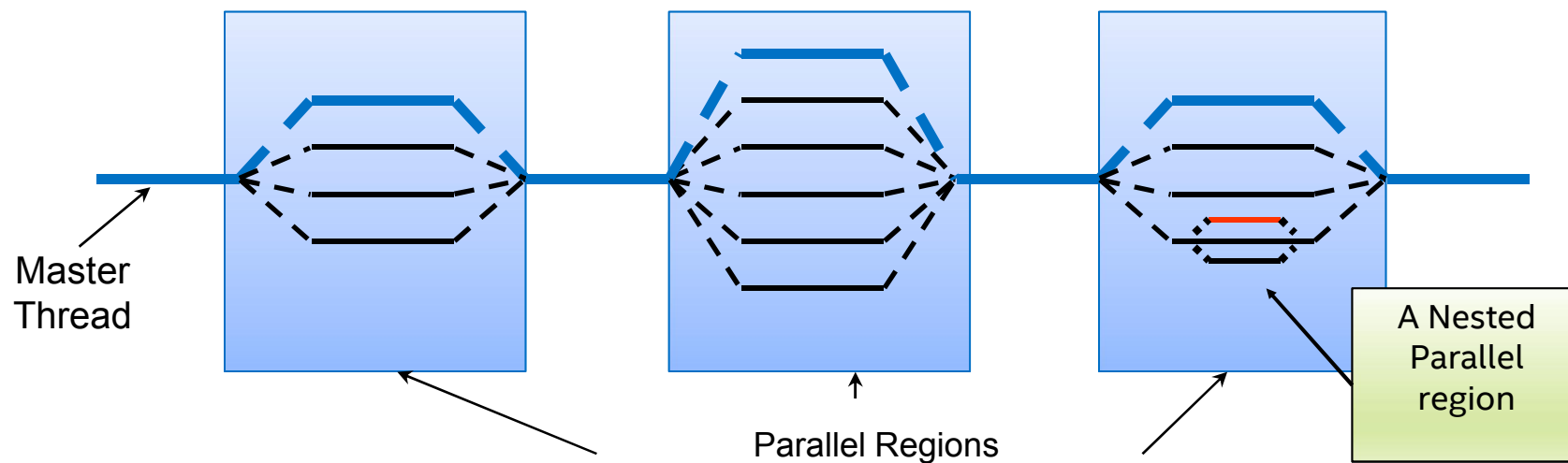
More cores → More Threads → Wider vectors

OpenMP\* is one of most important vehicles for the parallel + SIMD path forward

\*Product specification for launched and shipped products available on [ark.intel.com](http://ark.intel.com). 1. Not launched or in planning.




## OpenMP\* Programming Model

- ✓ **Master thread** spawns a **team of threads** / a **league of thread teams** as needed.
- ✓ Parallelism is added incrementally until desired performance is achieved: i.e. the sequential program evolves into a parallel program.



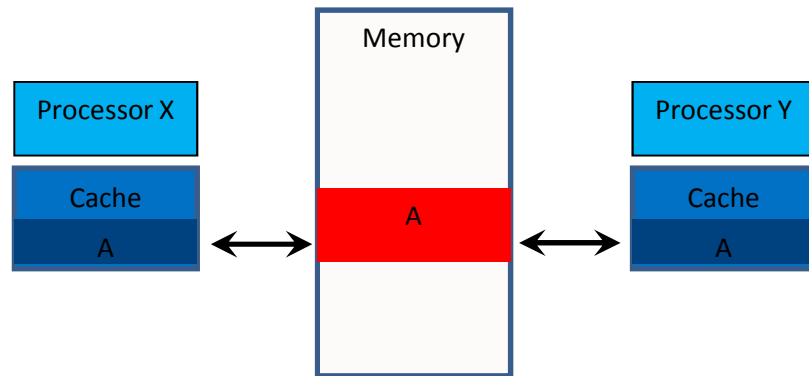
# SAXPY: Coprocessor/Accelerator

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

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

#pragma omp distribute
    for (int i = 0; i < n; i += num blocks) {

#pragma omp parallel for
        for (int j = i; j < i + num blocks; j++) {

            y[j] = a*x[j] + y[j];
        }
    }
} free(x); free(y); return 0; }
```

# Data sharing / mapping: shared or distributed memory

## Shared memory



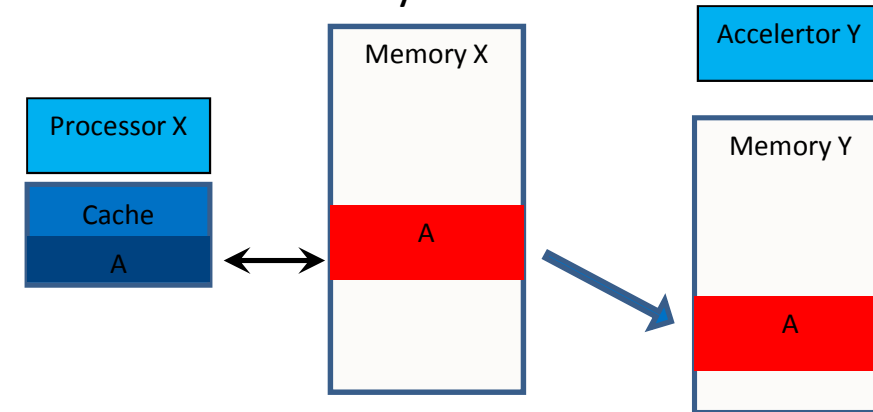
Threads have access to a *shared* memory

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

Threads have *private* memory

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

## Distributed memory



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

## *OpenMP\* in Clang/LLVM: A Brief History*

2H 2012: Several proposals with LLVM IR extensions and late outlining

- ✓ From Intel, Hal Finkel, others
- ✓ All of them involve changes to LLVM IR and thus, require modifications of LLVM phases
- ✓ None of them got enough support in the community

October 2012: OpenMP\* in Clang project

- ✓ Started by AMD\*, continued by Intel
- ✓ Early FE lowering and outlining
- ✓ OpenMP RTL calls generated in Clang

October 2015: OpenMP\* 4.0 Target (Device) model supported in Clang FE

- ✓ Initial implementation available at [https://github.com/clang-omp/clang\\_trunk](https://github.com/clang-omp/clang_trunk)  
([Joint work by AMD, IBM, Intel and TI](#))

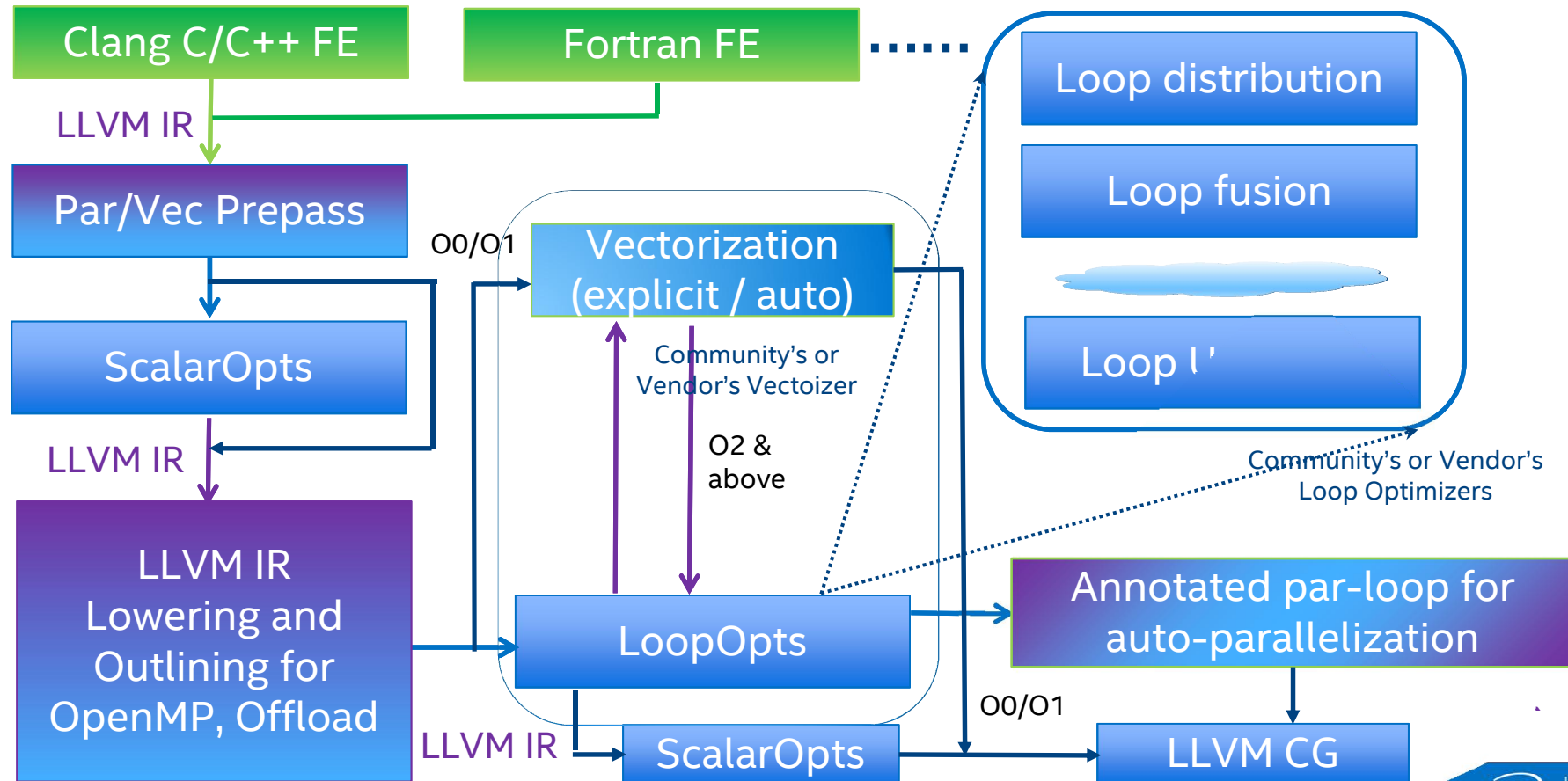
\* Other brands and names are the property of their respective owners.

Copyright © 2014, Intel Corporation. All rights reserved. \*Other names and brands may be claimed as the property of others.

Optimization Notice



# 10000ft View: High-Level Design for Moving Forward





## LLVM OpenMP\* Compiler Design Guideline

- ✓ *Correctness: easy to achieve and maintain compiler intermediate states and consistency*
- ✓ *Competitive performance: compile-time and runtime performance*
- ✓ *Competitive code size: generated code size and adds-on module size from Intel implementation.*
- ✓ *Composite ability: analysis module and individual loop opts can composed flexibility to achieve high performance*
- ✓ *Debug ability: generate sufficient debug information*
- ✓ *Programmer friendly diagnostic and report messages*

## LLVM IR Intrinsic Definitions

```
def int_directive : Intrinsic<[], [llvm_metadata_ty],  
                    [IntrReadWriteArgMem], "llvm.intel.directive">;  
def int_directive_qual : Intrinsic<[], [llvm_metadata_ty],  
                    [IntrReadWriteArgMem], "llvm.intel.directive.qual">;  
def int_directive_qual_opnd : Intrinsic<[], [llvm_metadata_ty, llvm_any_ty],  
                    [IntrReadWriteArgMem], "llvm.intel.directive.qual.opnd">;  
def int_directive_qual_opndlist : Intrinsic<[], [llvm_metadata_ty, llvm_vararg_ty],  
                    [IntrReadWriteArgMem], "llvm.intel.directive.qual.opndlist">;
```

Each C++ obj is represented with four arguments: Value, default constructor, copy constructor and destructor.

\* References to constructors / destructors are required to correctly create and destroy private copies of variables.

Array section is represented with (2 + # of DIMs x 3) arguments: Value, # of DIMs, lower0, length0, stride0, lower1, length1, stride1, ...

# LLVM IR Prepass

- ✓ CFG Restructuring
- ✓ Transform parallel sections / worksharing sections to parallel loops / worksharing loops
- ✓ Pre-privatization renaming
- ✓ Multi-versioning for different targets (Host, GPU and Coprocessors)
- ✓ ... ..

```
float parloop(float *a) {  
  int k = 0;  
  float x = 108.8f;  
}
```

```
float x_temp = x;  
@llvm.directive(DIR.OMP.PARALLEL.LOOP);  
@llvm.directive.qual(DIR.QUAL.IS.OMP.SIMD);  
@llvm.directive.qual.opnd.list(DIR.QUAL.REDUCTION.ADD,  
                               DIR.QUAL.OPND.VALUE, &x_temp);  
@directive_qual_opnd_list(DIR_QUAL_SHARED,  
                           DIR_QUAL_OPND_VALUE, a, "opnd.end");  
{ int priv_k;      // #pragma omp parallel for simd reduction(+: x) shared(a)  
  for (priv_k=0; priv_k<10000; priv_k++) {  
    x_temp = x_temp + a[priv_k] + 100.08f;  
  }  
}  
@llvm.directive(DIR.OMP.END.PARALLEL.LOOP);  
x = x_temp;  
return x;  
}
```

```
float parloop(float *a) {  
  int k = 0;  
  float x = 108.8f;  
  #pragma omp parallel for simd reduction(+: x) shared(a) private(k)  
  for (k=0; k<10000; k++) {  
    x = x + a[k] + 100.08f;  
  }  
  return x;  
}
```

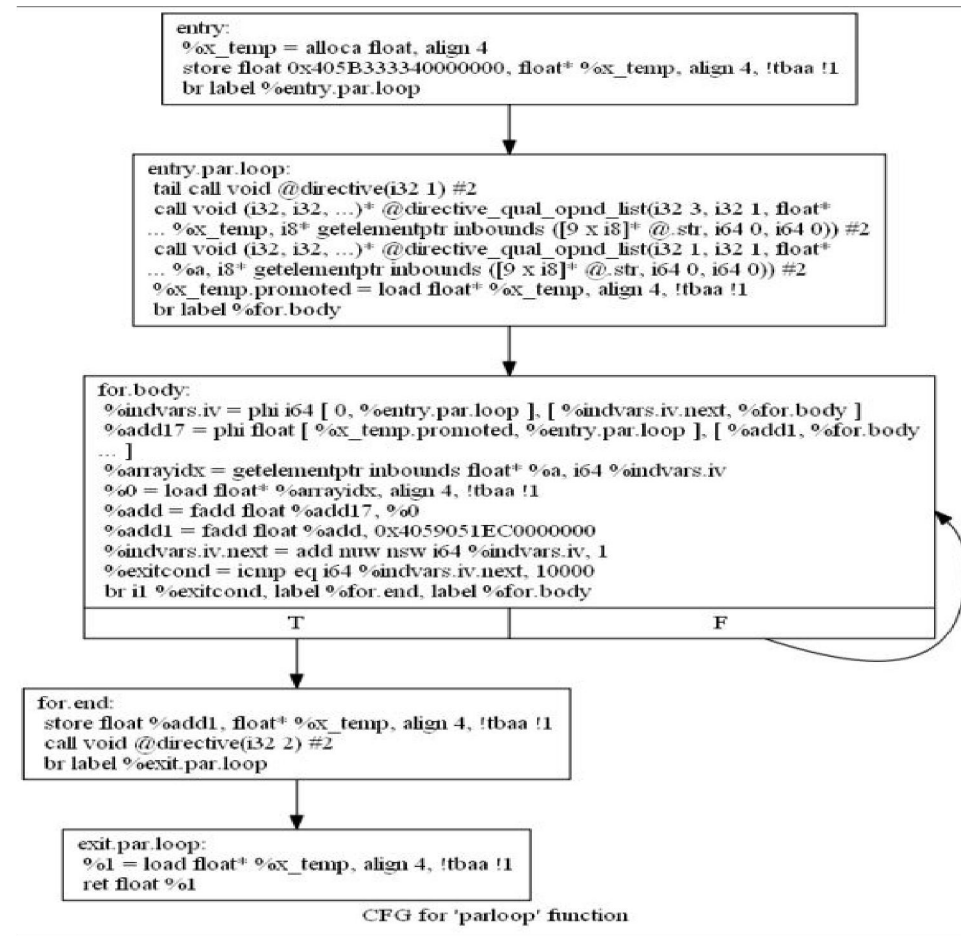
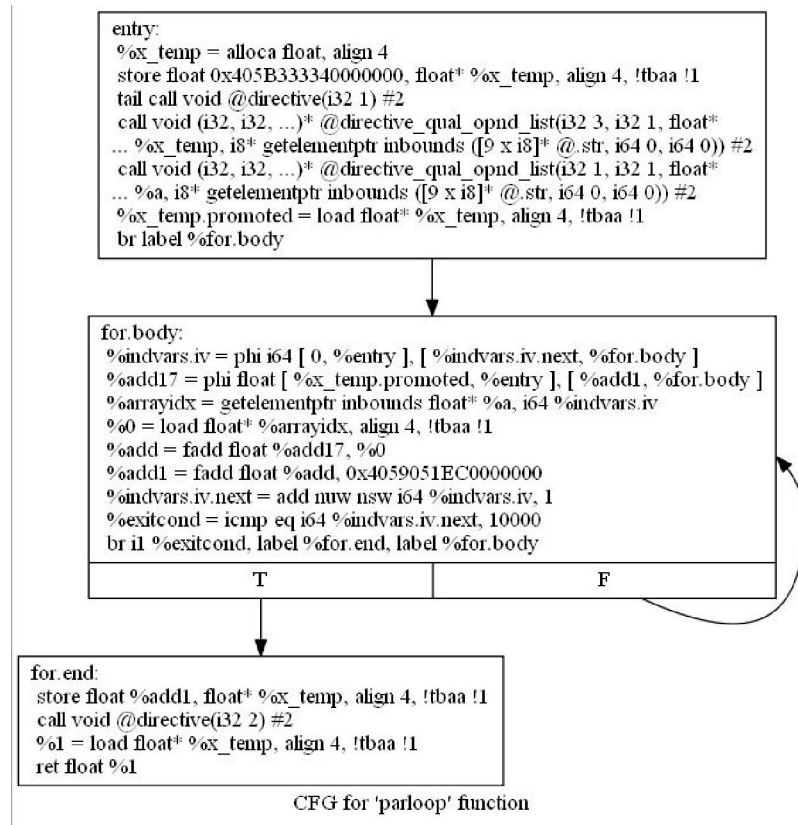
# LLVM IR with Directive Intrinsics

```
@.str = private unnamed_addr constant [9 x i8] c"opnd.end\00", align 1
; Function Attrs: nounwind uwtable
define float @parloop(float* %a) #0 {
entry:
  %x_temp = alloca float, align 4
  store float 0x405B333340000000, float* %x_temp, align 4, !tbaa !1
  call void @directive(i32 1) #2
  call void @directive.qual(i32 28) #2
  call void (i32, i32, ...)* @directive.qual.opndlist(i32 3, i32 1, float* %x_temp, i8* getelementptr inbounds ([9 x i8]* @.str, i64 0, i64 0)) #2
  call void (i32, i32, ...)* @directive.qual.opndlist(i32 1, i32 1, float* %a, i8* getelementptr inbounds ([9 x i8]* @.str, i64 0, i64 0)) #2
  %x_temp.promoted = load float* %x_temp, align 4, !tbaa !1
  br label %for.body

for.body:                                ; preds = %for.body, %entry
  %indvars.iv = phi i64 [ 0, %entry ], [ %indvars.iv.next, %for.body ]
  %add17 = phi float [ %x_temp.promoted, %entry ], [ %add1, %for.body ]
  %arrayidx = getelementptr inbounds float* %a, i64 %indvars.iv
  %0 = load float* %arrayidx, align 4, !tbaa !1
  %add = fadd float %add17, %0
  %add1 = fadd float %add, 0x4059051EC0000000
  %indvars.iv.next = add nuw nsw i64 %indvars.iv, 1
  %exitcond = icmp eq i64 %indvars.iv.next, 10000
  br i1 %exitcond, label %for.end, label %for.body

for.end:                                  ; preds = %for.body
  store float %add1, float* %x_temp, align 4, !tbaa !1
  call void @directive(i32 2) #2
  %1 = load float* %x_temp, align 4, !tbaa !1
  ret float %1
}
```

# LLVM CFG Restructuring



## LLVM IR Lowering and Outlining Passes

### ✓ Lowering

- ✓ Loop transformation (e.g. loop collapsing)
- ✓ Generate code for atomic, critical, single, master, ..., constructs (i.e. these constructs do not need outlining)
- ✓ Loop partitioning based on schedule type and chunk-size
- ✓ Generation code for reduction, lastprivate, firstprivate, copyprivate, ... threadprivate, etc.
- ✓ Generate debug info and opt-reports

### ✓ Outlining

- ✓ Parallel regions/loops/sections/Tasks (use OpenMP runtime)
- ✓ Affinity setting
- ✓ Generate runtime control code
- ✓ Target regions (use Offload runtime)
- ✓ Cilk for loops (use Cilk runtime)
- ✓ Packing / Unpacking arguments
- ✓ Generate debug info and opt-reports



## Debugging Support

- ✓ Associate “privatized” LLVM VALUES (variables) to original LLVM VALUES (variables)
- ✓ Associate “argument” VALUES of outlined function to original VALUES
- ✓ LLVM debug info generation framework has similar functionalities for preserving debug info for privatization and data-sharing transformation.

## A Lowering and Outlining Example (Pseudo Code)

```
typedef struct { float *x; float *a; } PARMLIST;

void __outlined_parloop(
  int g_lower, int g_upper, int g_stride, PARMLIST *parms) {
  int x_reduction; float *a; float *x;
  int t0 = 0, lower = 0, t1 = 0, upper = 0, stride = 1;
  int tid = __kmpc_global_get_thread_num();
  a = parms->a; x = parms->x;
  __kmpc_static_init(tid, &lower, &upper,
    &stride, g_lower, g_upper, g_stride);
  x_red = 0.0f; t0 = lower; t1 = upper;
  @llvm.directive(DIR.OMP.SIMD);
  @llvm.directive.qual.opndlist(DIR.QUAL.REDUCTION.ADD,
    DIR.QUAL.OPND.VALUE, &x_red);
  for (int k=t0; k<t1; k++) {
    x_red = x_red + a[k] + 100.8f;
  }
  @llvm.directive(DIR.OMP.END.SIMD);
  __kmpc_critical(tid);
  *x = *x + x_red;
  __kmpc_end_critical(tid);
  return;
}
```

```
float parloop(float *a) {
  float x = 108.8f;
  PARMLIST thunk;

  float x_temp = x;

  thunk.x = &x_temp;
  thunk.a = a;

  if (__kmpc_ok_fork()) {
    __kmpc_fork_call(outlined_parloop, 0, 10000, 1, &thunk);
  }
  else {
    __outlined_parloop(0, 10000, 1, &thunk);
  }
  x = x_temp;
  return x;
}
```

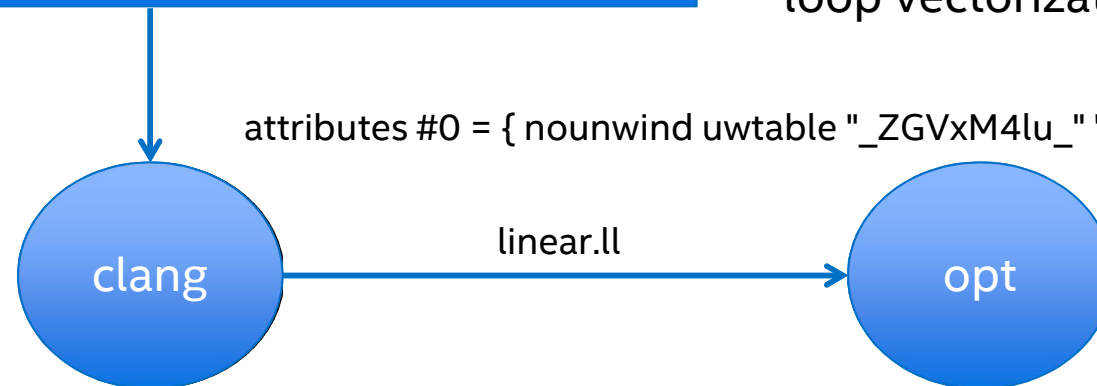
## DECLARED SIMD: Function to Loop Conversion

```
#pragma omp declare simd linear(i:1) uniform(x)
int foo(int i, int x) {
    return (x + i);
}
```

linear.c

### Key Ideas:

- Produces correct scalar code for the function
- Treat function vectorization as loop vectorization



`clang -c -emit-llvm -S -fopenmp-simd linear.c`

`opt -simd-function-cloning linear.ll -o linear.bc`

```
define __stdcall <4 x i32> @_ZGVbN4lu_foo(i32 %i, i32 %x) #0
```

```
entry:
%i.addr = alloca i32, align 4
%vec_retval = alloca <4 x i32>
%x.addr = alloca i32, align 4
store i32 %i, i32* %i.addr, align 4
%veccast = bitcast <4 x i32>* %vec_retval to i32*
```

```
entry:
%i.addr = alloca i32, align 4
%x.addr = alloca i32, align 4
store i32 %i, i32* %i.addr, align 4
store i32 %x, i32* %x.addr, align 4
%0 = load i32, i32* %x.addr, align 4
%1 = load i32, i32* %i.addr, align 4
%add = add nsw i32 %0, %1
ret i32 %add
```

**ORIGINAL BODY**

```
return:
%cast = bitcast i32* %veccast to <4 x i32>*
%vec_ret = load <4 x i32>, <4 x i32>* %cast
ret <4 x i32> %vec_ret
```

```
simd.begin.region:
call void @llvm.directive(metadata !7)
call void @llvm.directive.qual.opnd.i32(metadata !8, i32 4)
call void (metadata, ...) @llvm.directive.qual.opndlist(metadata !9, i32 %i)
call void @llvm.directive.qual(metadata !10)
```

```
simd.loop:
%index = phi i32 [ 0, %simd.begin.region ], [ %indvar, %simd.loop.exit ]
store i32 %x, i32* %x.addr, align 4
%0 = load i32, i32* %x.addr, align 4
%1 = load i32, i32* %i.addr, align 4
%mul = mul i32 1, %index
%add.1 = add i32 %1, %mul
%add = add nsw i32 %0, %add.1
%vec_gep = getelementptr i32, i32* %veccast, i32 %index
store i32 %add, i32* %vec_gep
```

```
simd.loop.exit:
%indvar = add nuw i32 1, %index
%vlcond = icmp ult i32 %indvar, 4
```

```
simd.end.region:
call void @llvm.directive(metadata !11)
```

# Vectorizing for Loop with Math Function Calls

```
icc sinf.c -fimf-max-error=0.6 -fimf-precision=high
```

```
#pragma omp simd  
for ( i = 0; i < 1000; i++) {  
    array[i] = sinf(i);  
}
```

*Before Vectorization:*

```
((F32) t0(F32)) = sinf{ic=SINF}.imf_attrs(max-error=0.6 domain-exclusion=0 valid-status-bits=false precision=high)(  
F32) (EXPR_CONV_SI32.F32(i.219_V$3(SI32)))(F32); [CALL_CONVENTION_UNIX_ABI]
```

*After Vectorization:*

```
((MS128) t107(MS128)) = __svml_sinf4{ic=VX_VMLS_SIN4}.imf_attrs(max-error=0.6 domain-exclusion=0 valid-  
status-bits=false precision=high)((MS128) t108(MS128)); [CALL_CONVENTION_UNIX_ABI]
```

*Assembly:*

```
..B1.2:          # Preds ..B1.8 ..B1.7  
                # Execution count [5.56e+00]  
                cvtdq2ps %xmm8, %xmm0          #36.3  
                call    __svml_sinf4_ha        #36.3    # LOE rbx r12 r13 r14 r15 xmm0 xmm8 xmm9
```

# Prototype Implementation

## Before Vectorization:

```
%call = call float @sinf(float %div) #4, !dbg !22
```

Adding a Clang FE patch would be something like:

```
%call = call float @llvm.sin.f32(float %div) #4, !dbg !22
```

## After Vectorization:

```
%4 = call <4 x float> @llvm.sin.v4f32(<4 x float> %3), !dbg !27, !imf-precision !10, !imf-max-error !11
```

```
!10 = !{"imf-precision=high"}
```

```
!11 = !{"imf-max-error=0.6"}
```

## After SVML translation pass:

```
%3 = call <4 x float> @__svml_sinf4_ha(<4 x float> %2)
```



## Prototype Implementation with XMM

```
vector.body:                                ; preds = %vector.body, %entry
  %index = phi i64 [ 0, %entry ], [ %index.next, %vector.body ], !dbg !2
  %0 = trunc i64 %index to i32, !dbg !7
  %broadcast.splatinsert6 = insertelement <8 x i32> undef, i32 %0, i32 0, !dbg !7
  %broadcast.splat7 = shufflevector <8 x i32> %broadcast.splatinsert6, <8 x i32> undef, <8 x i32> zeroinitializer, !dbg !7
  %induction8 = add <8 x i32> %broadcast.splat7, <i32 0, i32 1, i32 2, i32 3, i32 4, i32 5, i32 6, i32 7>, !dbg !7
  %1 = sitofp <8 x i32> %induction8 to <8 x float>, !dbg !7
  %shuffle = shufflevector <8 x float> %1, <8 x float> undef, <4 x i32> <i32 0, i32 1, i32 2, i32 3>
  %vcall = call <4 x float> @__svml_sinf4_ha(<4 x float> %shuffle)
  %shuffle.1 = shufflevector <8 x float> %1, <8 x float> undef, <4 x i32> <i32 4, i32 5, i32 6, i32 7>
  %vcall.2 = call <4 x float> @__svml_sinf4_ha(<4 x float> %shuffle.1)
  %shufflecomb = shufflevector <4 x float> %vcall, <4 x float> %vcall.2, <8 x i32> <i32 0, i32 1, i32 2, i32 3, i32 4, i32 5, i32 6, i32 7>
  %2 = getelementptr inbounds float, float* %array, i64 %index, !dbg !8
  %3 = bitcast float* %2 to <8 x float>*, !dbg !9
  store <8 x float> %shufflecomb, <8 x float>* %3, align 4, !dbg !9, !tbaa !10
  %index.next = add i64 %index, 8, !dbg !2
  %4 = icmp eq i64 %index.next, 1000, !dbg !2
  br i1 %4, label %for.end, label %vector.body, !dbg !2, !llvm.loop !14
```

```
#pragma omp simd simdlen(8)
for ( i = 0; i < 1000; i++) {
    array[i] = sinf(i);
}
```

## Summary

- ✓ OpenMP is evolving with a set of new features that needs scalar optimization, vectorization and loop optimizations to be seamlessly integrated with parallelization (privatization, lowering, outlining, ... etc.)
- ✓ Multiple languages support with effective engineering and maintaining cost
- ✓ Path-finding efforts to study feasibility of the Back-End solution
  - ✓ Minimal extensions for LLVM IR
  - ✓ Minimal Impact on LLVM infrastructure and optimizations
  - ✓ Getting optimal threaded code to leverage target HW potential
  - ✓ Targeting modern CPUs, Coprocessors, GPUs, DSP, FPGA, ... etc.

*Thanks & Questions?*

# Legal Disclaimer & Optimization Notice

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

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

Copyright © 2014, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

## Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

