OpenMP* Support in Clang/LLVM: Status Update and Future Directions

2014 LLVM Developers' Meeting

Alexey Bataev, Zinovy Nis

Intel

Agenda

1. Intro
2. Status
3. Runtime Library
4. Coprocessor/Accelerator Support
5. SIMD Features & Status

What is OpenMP*?

http://www.openmp.org/

- Industry-wide standard for shared memory multiprocessing programming in C/C++ and Fortran
- Vendor-neutral, platform-neutral, portable, pragma based, managed by an independent consortium
- Very important in High Performance Computing (HPC)
- OpenMP Version 3.1 (July 2011)
  - Implemented in GCC*, ICC, Oracle Solaris Studio*, PGI* …
- OpenMP Version 4.0 (July 2013)
  - Adds support for coprocessors/accelerators, SIMD, error handling, thread affinity, user defined reductions
  - Implemented in GCC*, ICC …

“OpenMP* in Clang/LLVM” Team

Driving collaboration and implementation

- Implementation of OpenMP in Clang/LLVM
- Coprocessor/accelerator support
- Test coverage
- Code reviews

Current participation from

AMD*,

Argonne National Laboratory,

IBM*,

Intel,

Micron*,

Texas Instruments*,

University of Houston

Talk to us if you want to be involved!

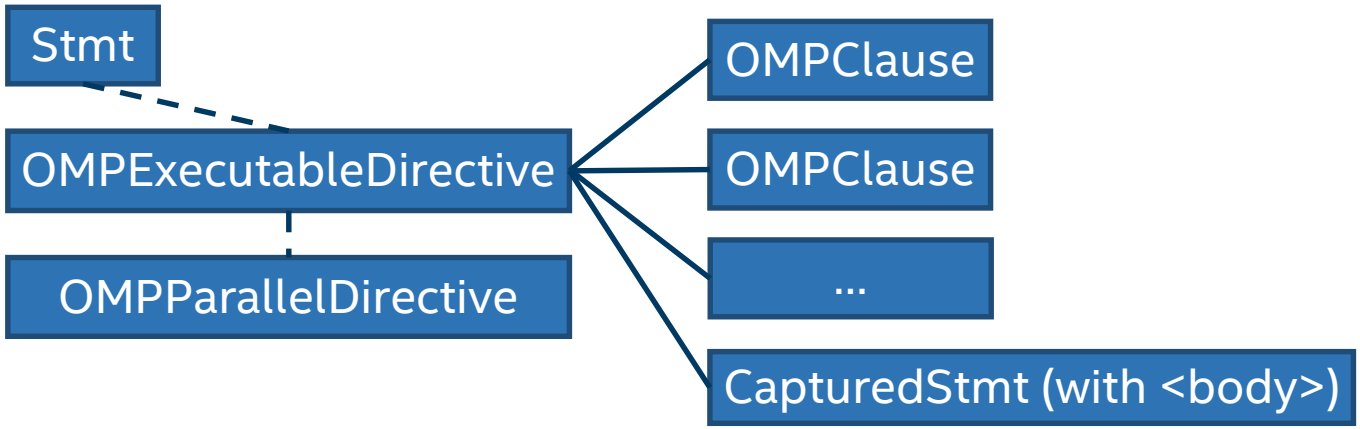
OpenMP* Support in Clang

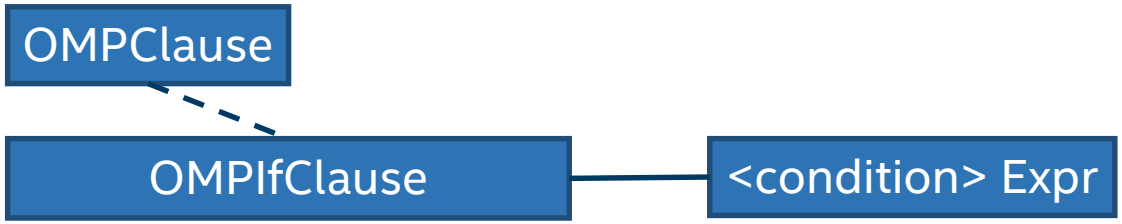
- Uses “early” outlining
  - Parallel and task regions are outlined as functions, all other regions are emitted as is
- Pragmas are just regular AST nodes with associated structured code block
  - Executable directives are Stmts
  - Declarative directives are Decls
- CodeGen follows regular rules for Stmts and Decls nodes
  - Uses **libiomp5** runtime library, targets Intel OpenMP* API
  - Calls to runtime functions are generated in frontend
- No LLVM IR extensions are required

Support in Clang/LLVM is under development

OpenMP Constructs Representation

```
#pragma omp parallel if (<condition>)
<body>
```





Current Status

- Clang/LLVM 3.5 Release
  - Parsing and semantic analysis for OpenMP* 3.1 (except for 'ordered' and 'atomic' directives)
  - CodeGen for 'parallel' and 'simd' directives, 'safelen' clause
  - First release with **libiomp5** OpenMP runtime library!
- Clang/LLVM Trunk
  - Complete parsing and semantic analysis for OpenMP 3.1 and partial for OpenMP 4.0
  - CodeGen for 'critical' directive, 'if', 'num_threads', 'firstprivate', 'private', 'aligned' and 'collapse' clauses
- No support in driver yet, use -Xclang -fopenmp=libiomp5
- Planning full OpenMP 3.1 support in Clang/LLVM 3.6 Release, most of OpenMP 4.0 in Clang/LLVM 3.7 Release

Depends on the speed of code review!

Current Status: clang-omp Repo

http://clang-omp.github.io/

- Full OpenMP* 3.1 + part of OpenMP 4.0 support
  - Coprocessor/accelerator support is under development
- OpenMP Validation Suite from OpenUH* test suite
  - Passed 119 tests of 123
- Supported on Linux* and Mac OS X*
  - x86, x86-64, PowerPC*, ARM*
- Based on Clang/LLVM 3.5 Release
  - Includes trunk-based version (maintained by Hal Finkel)
- Use -fopenmp driver option

You can try it right now!

Current Status: Summary

Features	Parsing/Sema (clang-omp)	CodeGen (clang-omp)	Parsing/Sema (Trunk)	CodeGen (Trunk)
Parallel	Yes	Yes	Yes	Partially
Worksharing	Yes	Yes	Yes	No
Tasking	Yes	Yes	Yes	No
Other (OpenMP* 3.1)	Yes	Yes	Yes	No
SIMD (OpenMP 4.0)	Yes	Yes	Partially	Partially
Coprocessor/ Accelerator (OpenMP 4.0)	Yes	Partially	Partially	No
Other (OpenMP 4.0)	Yes	Yes	No	No

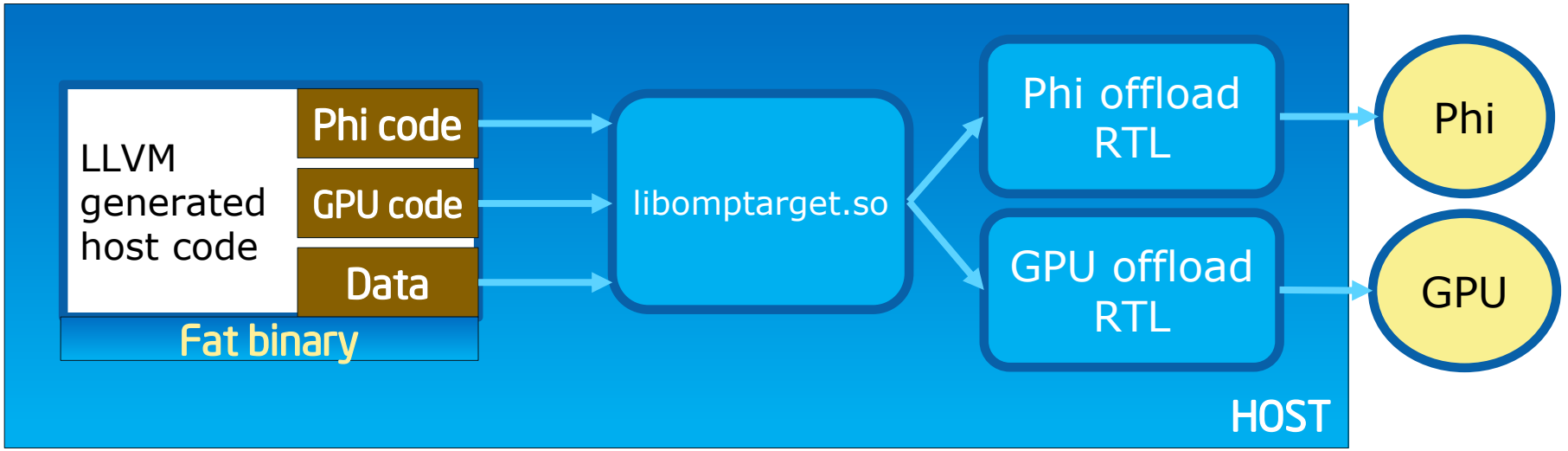
OpenMP* Runtime Library libiomp5

http://openmp.llvm.org

- Supports OpenMP 4.0 (though not all features are yet in Clang)
- Supported on Windows*, Linux* and Mac OS X*
  - x86, x86-64, PowerPC*, ARM*
- Can be built by Clang, GCC*, ICC
  - Two build bots configured (libiomp5-clang-x86_64-linux-debian, libiomp5-gcc-x86_64-linux-debian)

OpenMP* Coprocessor/Accelerator Support Library

- Required for support of OpenMP 4.0 target constructs
- Under development by Intel, IBM*, TI*, etc. in clang-omp repository
- Plan to support x86, x86-64, PowerPC* and ARM* as hosts , multiple targets (Intel® Xeon Phi™ coprocessor, GPUs, FPGAs, …)

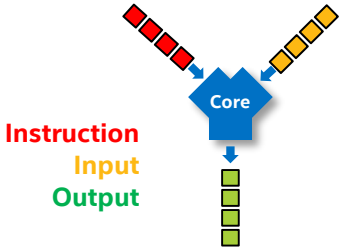


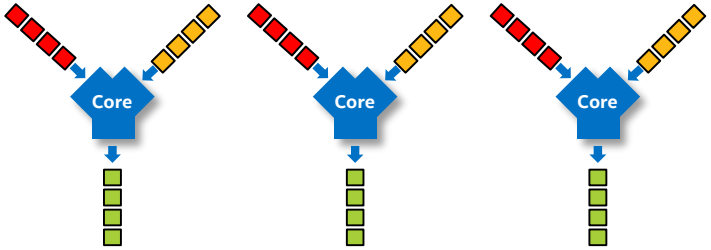
OpenMP*: Levels of Parallelism

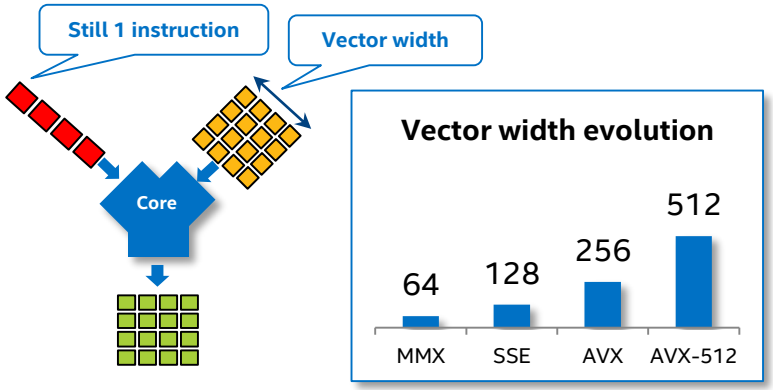
- Coprocessors/Accelerators
  - **brand-new** #pragma omp target
- Threads
  - **good old** #pragma omp parallel for
- SIMD
  - **OpenMP with SIMD flavor** #pragma omp simd

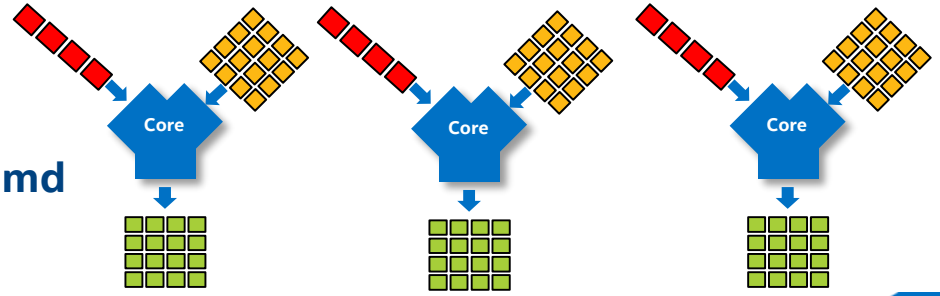
SIMD: What is It?

- How to run faster?
  - Run instructions simultaneously
    - Thread-level parallelism
    - **#pragma omp parallel for**
  - Handle the data simultaneously
    - Data-level parallelism
    - **SIMD**, Single Instruction Multiple Data, vector instructions
    - **#pragma omp simd**
  - Do Both
    - **#pragma omp parallel for simd**









SIMD: How to Use

- High and transparent portability
- High and transparent scalability
- No development cost
- Unpredictable performance

- Expresses desired vectorization
- High and transparent portability
- High and transparent scalability
- Low development cost
- Predictable performance

- Max performance
- Low portability
- High development cost
- Low scalability


```
...
float *A, *B, *C;
...
for(int i...)
  C[i] = A[i] + B[i];
$> clang –O2 ...
```

```
...
float *A, *B, *C;
...
#pragma omp simd
for(int i...)
  C[i] = A[i] + B[i];
$> clang –O2 –Xclang – fopenmp=...
```

```
#include "xmmintrin.h“
...
float *A, *B, *C;
__m128 a, b, c;
...
for(int i...)
{
 a = _mm_load_ps(A + i*4);
 b = _mm_load_ps(B + i*4);
 c = _mm_add_ps(a, b);
  _mm_store_ps(C+i * 4, c);
}
$> clang ...
```

SIMD: Autovectorization vs Pragmas

- Autovectorization in LLVM fails if:
  - cost heuristic says "no"
  - non-trivial data dependencies
  - overhead is high for run-time checks
  - loop is not innermost
- OpenMP* SIMD pragmas:
  - vectorize, don't check!

```
// RT pointer aliasing checks: float*
void doThings(float *a, float *b,
              float *c, float *d, float *e,
              unsigned N, unsigned K)
{
 for(int i = 0; i < N; ++i) // not innermost
  for(int j = 0; j < N; ++j)
   // non-trivial data-deps due to unknown K
   a[i + j] = b[i + K] + c[i + K] +
              d[i + K] + e[i + K];
}
```

```
$> clang -mllvm -debug-only=loop-vectorize ...
LV: Checking a loop in "doThings" from test.c:4:3
...
LV: Found a runtime check ptr:  %arrayidx18.us = ...
LV: We need to do 4 pointer comparisons
LV: We can't vectorize because we can't find the array bounds
LV: Can't vectorize due to memory conflicts
```

```
// the only change
#pragma omp simd collapse(2)
 for(int i = 0; i < N; ++i)
  for(int j = 0; j < N; ++j)
    a[i + j] = ...
...
```

```
$> clang -mllvm -debug-only=loop-vectorize –Xclang –fopenmp=...
LV: Checking a loop in "doThings" from test.c:4:3
LV: Loop hints: force=enabled width=0 unroll=0
LV: A loop annotated parallel, ignore memory dependency checks.
LV: We can vectorize this loop!
```

OpenMP* SIMD: Status in Clang

- OpenMP SIMD requires support from both frontend and backend
- Frontend parses OpenMP SIMD pragmas and creates LLVM IR **metadata hints**, which are then used by vectorizer

```
#pragma omp simd safelen(4)
  for(int i = 0; i < N; ++i) ...
```

```
LV: Checking a loop in ...
LV: Loop hints: force=enabled width=4 unroll=0
...
!42 = metadata !{metadata !"llvm.loop.vectorize.enable", i1 true}
!43 = metadata !{metadata !"llvm.loop.vectorize.width", i32 4}
```

- In case of *collapse* clause frontend creates a new loop, *collapsing* the existing loop nest into single loop

OpenMP* SIMD: Status in LLVM backend

- LLVM loop vectorizer recognizes loop vectorizing metadata
- ***Collapse***, ***safelen*** and ***aligned*** clauses are ready to use
- Other clauses and constructs are under development
- TODO:
  - Vectorizing metadata must be propagated through all the passes prior to the vectorizer
  - Compile-time memory checks must be disabled in the presence of OpenMP SIMD pragmas

Questions?

OpenMP* Support in Clang/LLVM: Status Update and Future Directions

Alexey Bataev, Zinovy Nis

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.

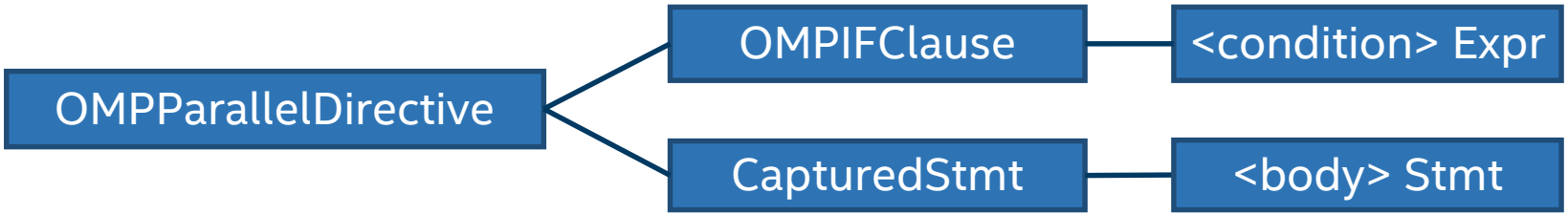
Optimization Notice

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.

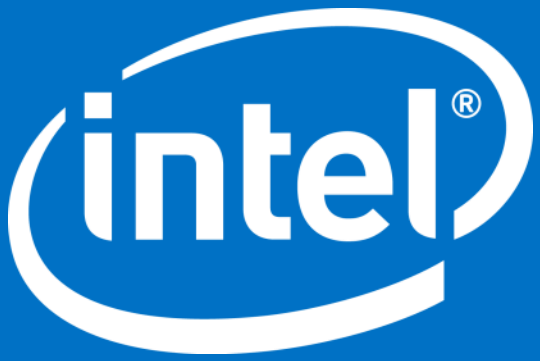
Notice revision #20110804

Backup

OpenMP Constructs Representation: Continued




```
br i1 <condition>, label %omp.if.then, label%omp.if.else
omp.if.else:
 <body>
br label %omp.if.end
omp.if.then:
  %3 = bitcast %struct.anon* %agg.captured to i8*
  call void @__kmpc_fork_call(<loc>, i32 1, void (i32*, i32*, ...)* bitcast (void (i32*,
i32*, i8*)* @.omp_microtask. to void (i32*, i32*, ...)*), i8* <captured_vars>)
  br label %omp.if.end
omp.if.end:
...
define internal void @.omp_microtask.(i32*, i32*, i8*) #0 {
  %.gtid. = load i32* %0
  <body>
  call i32 @__kmpc_cancel_barrier(<loc>, i32 %.gtid.)
}
```


intel®