SIMD PROGRAMMING WITH OPENMP*

Presenter: Georg Zitzlsberger

Date: 30-09-2015

Agenda

- **SIMD with OpenMP* 4.0 – In a Nutshell**
- Lab Time!
- SIMD with OpenMP* 4.1
- Summary

OpenMP* 4.0

- OpenMP* 4.0 ratified July 2013
- Specifications: http://openmp.org/wp/openmp-specifications/
- Well established in HPC – parallelism is critical there
- Extension to C/C++ & Fortran
- New features with 4.0:
  - Target Constructs: Accelerator support
  - Distribute Constructs/Teams: Better hierarchical assignment of workers
  - **SIMD (Data Level Parallelism!)**
  - Task Groups/Dependencies: Runtime task dependencies & synchronization
  - Affinity: Pinning workers to cores/HW threads
  - Cancelation Points/Cancel: Defined abort locations for workers
  - User Defined Reductions: Create own reductions

OpenMP* 4.0 SIMD

- Provides a portable high-level mechanism to specify SIMD parallelism
- Two main new directives:
  - To SIMDize (aka. vectorize) existing loops
    ⇨ **Simd Construct**
  - Create SIMD-enabled functions
    ⇨ **Declare Simd Construct**

SIMD Construct

- SIMD Construct (such as Loop SIMD Construct)
  The simd construct can be applied to a loop to indicate that the loop can be transformed into a SIMD loop (that is, multiple iterations of the loop can be executed concurrently using SIMD instructions).
  [OpenMP* 4.0 API: 2.8.1]

- Syntax:
  - C/C++:

```
#pragma omp simd [clause [[,]clause]…]
  for-loops
```

  - Fortran:

```
!$omp simd [clause [[,]clause]…]
  do-loops
[!$omp end simd]
```

SIMD Construct - Requirements

- C/C++:
  For-loop has to be in “canonical loop form” (see OpenMP 4.0 API:2.6)
  - Random access iterators required for induction variable
    (integer types or pointers for C++)
  - Limited test and in-/decrement for induction variable
  - Iteration count known before execution of loop
  - …

- Fortran:
  Do-loops must be do-constructs as defined by the Fortran standard

SIMD Construct - Clauses

- **`safelen(n1[,n2] …)`**
  `n1`, **`n2`**, … must be power of 2: The compiler can assume a vectorization for a vector length of **`n1`**, **`n2`**, … to be safe
- **`private(v1, v2, …)`**: Variables private to each iteration
  - **`lastprivate(…)`**: last value is copied out from the last iteration instance
  - Where is **`firstprivate(…)`**? There's a reason…
- **`linear(v1:step1, v2:step2, …)`**
  For every iteration of original scalar loop **`v1`** is incremented by **`step1`**, etc., hence incremented by **`step1 * vector length`** for the vectorized loop.
- **`reduction(operator:v1, v2, …)`**
  Variables **`v1`**, **`v2`**, … etc. are reduction variables for operation **`operator`**
- **`collapse(n)`**: Combine nested loops – collapse them
- **`aligned(v1:base, v2:base, …)`**: Tell variables **`v1`**, **`v2`**, … are aligned; (default is architecture specific alignment)

SIMD Construct - Example

```
void vec1(float *a, float *b, int off, int len)
{
#pragma omp simd safelen(32) aligned(a:64, b:64)
    for(int i = 0; i < len; i++)
    {
        a[i] = (a[i] > 1.0) ?
            a[i] * b[i] :
            a[i + off] * b[i];
    }
}
```

```
LOOP BEGIN at simd.cpp(4,5)
   remark #15388: vectorization support: reference a has aligned access   [ simd.cpp(6,9) ]
   remark #15388: vectorization support: reference b has aligned access   [ simd.cpp(6,9) ]
   …
   remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
   …
LOOP END
```

SIMD Construct - Compositions

SIMD Construct can be combined with:

- Loop SIMD
- Taskloop SIMD (new with OpenMP* 4.1)
- Distribute SIMD & Teams distribute SIMD
- Parallel loop SIMD & Distribute parallel loop SIMD
- Target SIMD, Target parallel loop SIMD & Target teams distribute SIMD
- Teams distribute parllel loop SIMD & Target teams distribute parllel loop SIMD

Worth noting:
Distributions will be done first - SIMD will be done on each set resulting from the distribution!

Declare SIMD Construct

- Declare SIMD Construct (aka. SIMD-Enabled Function):
  The declare simd construct can be applied to a function [...] or a subroutine [...] to enable the creation of one or more versions that can process multiple arguments using SIMD instructions from a single invocation from a SIMD loop. The declare simd directive is a declarative directive.
  [OpenMP* 4.0 API: 2.8.2]

- Syntax:
  - C/C++:

```
#pragma omp declare simd [clause [[,] clause]…]
[#pragma omp declare simd [clause [[,] clause]…]]
[…]
  function definition or declaration
```

  - Fortran:

```
!$omp declare simd(proc-name) [clause[[,] clause] ...]
```

Declare SIMD Construct - Semantic

- Intent:
  Express work as scalar operations (kernel) and let compiler create one or more vector versions of it. It also always creates the scalar version.
  The size of vectors can be specified at compile time (128 bit, 256 bit, 512bit) which makes it portable!
  It requires calling of SIMD-enabled function within a SIMD loop!
  ⇨ Express SIMD operations w/o need of knowing the final target

- Recommendation:
  Function declared as such should be inlined for best results.
  W/o inlining, call overhead is noticeable!

- Remember for C/C++:
  Both the function definition as well as the function declaration (header file) need to be specified!

Declare SIMD Construct - Clauses

- **`simdlen(len)`**
  `len` must be power of 2: Allow as many elements per argument (default is implementation specific)
- **`linear(v1:step1, v2:step2, …)`**
  Defines **`v1`**, **`v2`**, … to be private to SIMD lane and to have linear (**`step1`**, **`step2`**, …) relationship when used in context of a loop
- **`uniform(a1, a2, …)`**
  Arguments **`a1`**, **`a2`**, … etc. are not treated as vectors (constant values across SIMD lanes)
- **`inbranch, notinbranch`**: SIMD-enabled function called only inside branches or never; if none is used, a function can be called everywhere
- **`aligned(a1:base, a2:base, …)`**: Tell arguments **`a1`**, **`a2`**, … are aligned; (default is architecture specific alignment)

Declare SIMD Construct - Example

```
#pragma omp declare simd simdlen(16) notinbranch uniform(a, b, off)
float work(float *a, float *b, int i, int off)
{
    return (a[i] > 1.0) ?  a[i] * b[i] : a[i + off] * b[i];
}

void vec2(float *a, float *b, int off, int len)
{
#pragma omp simd safelen(64) aligned(a:64, b:64)
    for(int i = 0; i < len; i++)
    {
        a[i] = work(a, b, i, off);
    }
}
```

```
INLINE REPORT: (vec2(float *, float *, int, int)) [4/9=44.4%] simd.cpp(8,1)
  -> INLINE: (12,16) work(float *, float *, int, int) (isz = 18) (sz = 31)

LOOP BEGIN at simd.cpp(10,5)
   remark #15388: vectorization support: reference a has aligned access  [ simd.cpp(4,20) ]
   remark #15388: vectorization support: reference b has aligned access  [ simd.cpp(4,20) ]
   …
   remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
   …
LOOP END
```

Compilers Supporting OpenMP* 4.0 SIMD

- This is quite outdated: http://openmp.org/wp/openmp-compilers/ :-/
- Here's the latest compiler support* for SIMD:
  - Intel® C++ Compiler 14.0+ & Intel® Fortran Compiler 14.0+
    (for additional Fortran 2003 support of OpenMP 4.0, 15.0+; user defined reductions 16.0+)
  - GNU* GCC 4.9+ (4.9.1+ for Fortran): https://gcc.gnu.org/wiki/openmp
    GNU* GCC 5.0 added support for offloading, see:
    https://gcc.gnu.org/wiki/Offloading
  - clang/LLVM 3.5 via branch:
    http://openmp.llvm.org/, http://clang-omp.github.io/
    They use Intel® OpenMP* Runtime Library which is open source:
    https://www.openmprtl.org/
- SIMD extensions require at least **`-fopenmp-simd`** (or **`-fopenmp`**)!
  OpenMP* runtime is not needed, though.

*: Covers mostly syntax support; implementation quality might be different!

Agenda

- SIMD with OpenMP* 4.0 - In a Nutshell
- **Lab Time!**
- SIMD with OpenMP* 4.1
- Summary

Lab Time!

- Exercises:
  1. Identify and apply the right clauses to a loop SIMD construct (15 min.)
  2. Translate an exercise to use the declare SIMD construct (15 min.)
     Bonus: Combine it with a user defined reduction

- Both exercises are available for C/C+ & Fortran - pick your favorite one ☺
- Installed is Intel® C++ Compiler 16.0 and Intel® Fortan Compiler 16.0

Agenda

- SIMD with OpenMP* 4.0 - In a Nutshell
- Lab Time!
- **SIMD with OpenMP* 4.1**
- Summary

Future: OpenMP* 4.1 SIMD

- Ordered SIMD loop:
  Selectively disable SIMD within a SIMD loop via clause **`ordered`** (just use one SIMD lane), e.g.:
  `#pragma omp simd ordered` or **`!$omp simd ordered`**

- Added **`simdlen`** to SIMD loop clauses:
  Differentiation:
  - **`safelen`**: What is the the maximum size of a SIMD chunk?
  - **`simdlen`**: What SIMD chunk size should be used?

- Array sections now allowed for **`reduction`** clauses:
  All elements described by an array section are treated as separate reductions, e.g. an add-reduction on first 5 elements of **`a`**:
  `#pragma omp simd reduction(+:a[0:5])`

Ratification of OpenMP* 4.1 expected for SC'15 (November 2015).

Future OpenMP* 4.1: Status for Intel® C++ Compiler & Intel® Fortran Compiler

- Version 16.0:
  - OpenMP* 4.1 Technical Report 3 support:
    - Non-structured data allocation
    - Asynchronous offload
    - Dependence (signal)
    - Map clause extensions
  - Taskloops
  - Support of Intel® Transactional Synchronization Extensions (Intel® TSX)

- Future version 17.0 (2016):
  Initial support of OpenMP* 4.1 SIMD extensions planned

Agenda

- SIMD with OpenMP* 4.0 - In a Nutshell
- Lab Time!
- SIMD with OpenMP* 4.1
- **Summary**

Summary

- Standards for expressing SIMD are mandatory and important
- OpenMP* 4.0 provided a good start
- Future improvements for SIMD in the pipeline
- Call to action: Your feedback is needed!
- Get the latest specifications here: http://openmp.org/wp/openmp-specifications/

THANK YOU!

Legal Disclaimer & Optimization Notice

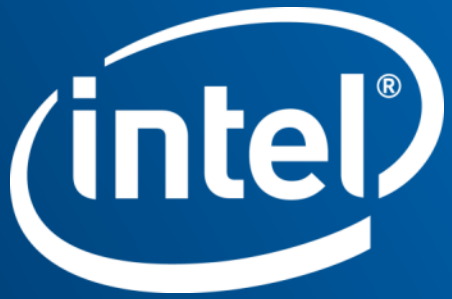
INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING 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


intel