Vectorization with OpenMP (4.0+)

Georg Zitzlsberger

georg.zitzlsberger@vsb.cz

IT4Innovations
national supercomputing
center

5th of July 2017

Agenda

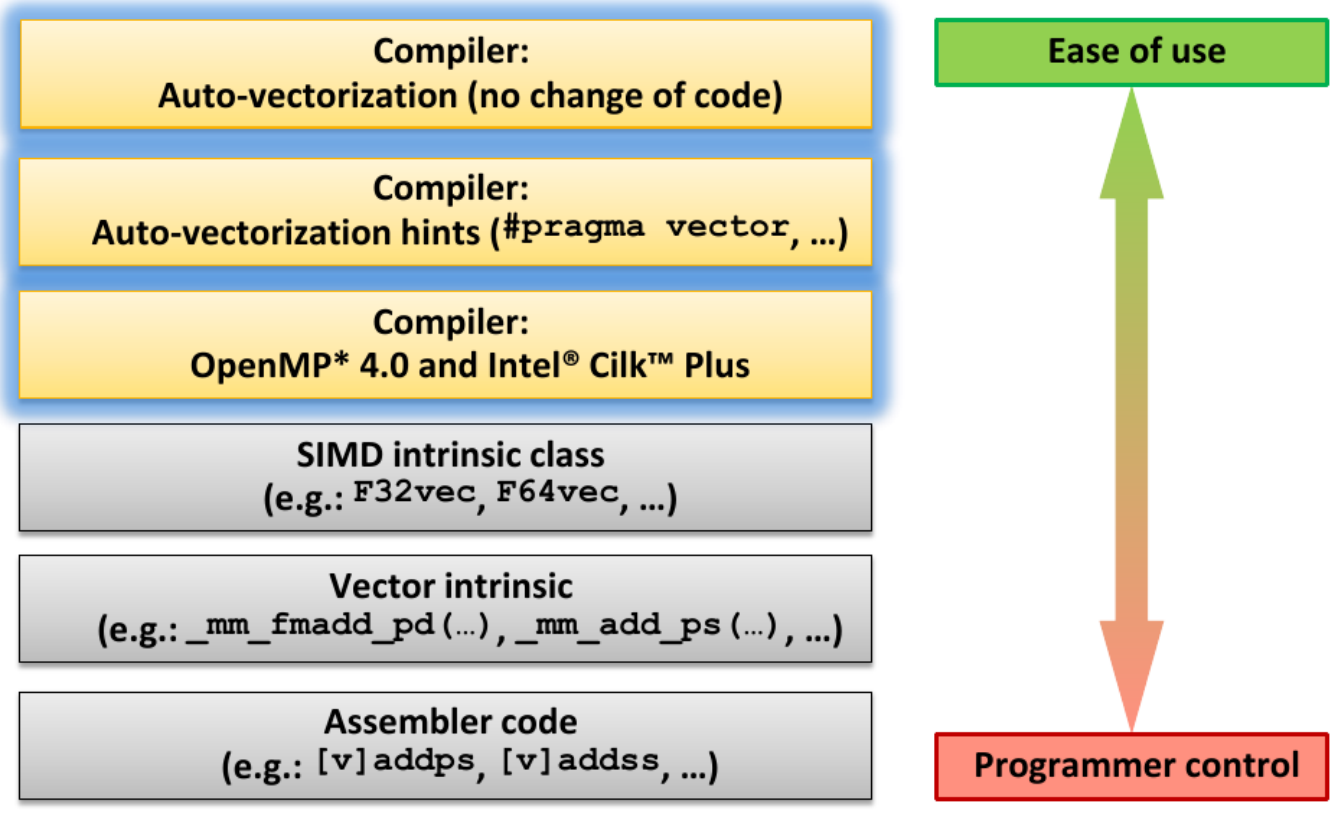
OpenMP 4.0

- OpenMP 4.0 ratified July 2013
- Specifications: here
- Well established in HPC
- 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
Enabling HPC since 1997

openmp.org

Vectorization with OpenMP 4.x



(Image: Intel)

SIMD Construct

- 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:

```
#pragma omp simd [clause [,clause]...]
    for-loop
```

- 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
  - ...

SIMD Construct Clauses

- `safelen(n1[,n2] ...)`
  `n1`, `n2`, ... must be power of 2: The compiler can assume a vectorization for a VL 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
- `linear(v1:step1, v2:step2, ...)`
  For every iteration of original scalar loop v1 is incremented by `step1`, ... Therefore it is incremented by `step1` * VL for the vectorized loop.
- `reduction(operator:v1, v2, ...)`
  Variables `v1`, `v2`, ... 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

Ignore data dependencies, indirectly mitigate control flow dependence and assert alignment:

```
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
```

(Image: Intel)

Declare SIMD Construct

- SIMD-enabled function (aka. declare simd construct):
  The declare simd construct can be applied to a function [...] 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.
  [OpenMP 4.0 API: 2.8.2]
- Syntax:

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

- Intent:
  Express work as scalar operations (kernel) and let compiler create a vector version of it. The size of vectors can be specified at compile time (SSE, AVX, ...) which makes it portable!

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
- `aligned(a1:base, a2:base, ...)`
  Tell arguments `a1`, `a2`, ... are aligned; default is architecture specific alignment

Declare SIMD Construct Example

Ignore data dependencies, indirectly mitigate control flow dependence and assert alignment:

```
#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
```

(Image: Intel)

OpenMP 4.5

OpenMP 4.5 was ratified November 2015:

- OpenMP `SIMD LINEAR` clause:
  - `val` (default):
    Value passed is linear; if passed by reference the vector of references is passed
  - `ref` (C++ and Fortran only):
    For parameters passed by reference and the underlying reference is linear
  - `uval` (C++ and Fortran only):
    The reference to the first lane is passed, other values constructed (faster)

Example:

```
#pragma omp declare simd notinbranch linear(ref(p))
// linear(val(p)) would be inefficient
int add_one(const int& p) {
    return (p + 1);
}
```

- OpenMP 4.5 `SIMD DECLARE` construct:
  Can now be C++ virtual functions.

OpenMP 4.5

- OpenMP 4.5 ordered blocks in SIMD contexts:
  Structured block in the SIMD loop/function executed in the order of loop iterations or sequence of call to SIMD functions.

```
#pragma omp simd
for (i = 0; i < N; i++)
{
    ...
#pragma omp ordered simd
    {
        a[indices[i]] += b[i]; // index conflict
    }
    ...
#pragma omp ordered simd
    {
        if (c[i] > 0)
            q[j++] = b[i];      // compress pattern
    }
    ...
#pragma omp ordered simd
    {
        lock(L)                 // atomic update
            if (x > 10) x = 0;
        unlock(L)
    }
    ...
}
```

- Support for `#pragma omp simd simdlen(n)`

OpenMP 4.5

- OpenMP 4.5 `REDUCTION` clause:
  Allows now array and pointer variables too for C/C++.
- OpenMP `SIMD PROCESSOR` clause:
  Intel extension for selecting SIMD for a processor generation.
- Other non-SIMD related extensions:
  - OpenMP 4.5 loop construct `LINEAR` clause:
    Added existing linear clause to loop construct. Initializing like `firstprivate`.
  - OpenMP 4.5 `TASKLOOP`:
    Parallelize a loop using OpenMP tasks (Fortran and C/C++).
  - OpenMP 4.5 "doacross" Loops:
    Enables loops with well structured dependencies to be parallelized now (with `DEPEND` and `ORDERED` clauses).
  - OpenMP 4.5 Hints for Locking:
    Using Intel Transactional Synchronization Extensions.
  - OpenMP 4.5 Offload:
    Added device pointer and device memory API

Support of OpenMP 4.x

The following compilers support OpenMP 4.x:

- OpenMP 4.0:
  GNU GCC 4.9 for C/C++, 4.9.1 for Fortran; GCC 5.0 added offloading support
  OpenMP 4.5:
  GNU GCC 6.1
- OpenMP 4.5:
  LLVM/clang 3.9 except offloading
- OpenMP 4.0:
  Intel C++/Fortran Compiler 15.0; full 16.0
  OpenMP 4.5:
  Intel C++/Fortran Compiler 17.0
- Full list of other compilers is here

Eanble with `-fopenmp-simd` (`-qopenmp-simd` for ICC) or `-fopenmp` (`-qopenmp` for ICC)
OpenMP runtime is not needed for just SIMD!

Summary

SIMD can be expressed using OpenMP 4.x:

- `SIMD` construct for vectorizing loops
- `DECLARE SIMD` construct for creating vectorized (aka. SIMD-enabled) functions
  Often used together with `SIMD` construct to call them inside a loop

Find lot's of examples here

Discussion forum for examples here (rather new)

Interesting article about latest OpenMP API here

Lab Time!