OpenMP 4.0 implementation in GCC

Jakub Jelínek
Consulting Engineer, Platform Tools Engineering, Red Hat

OpenMP 4.0 implementation in GCC

- Work started in April 2013, C/C++ support with host fallback only finished in October 2013 (~9 man-months)
  - Released in April 2014 in GCC 4.9.0
- Work on Fortran support started in April 2014, finished in June 2014 (~2 man-months)
  - Released in July 2014 in GCC 4.9.1
- Accelerator device offloading:
  - Intel XeonPhi (Knights Landing) merged in November 2014 for upcoming GCC 5
  - Nvidia PTX / Cuda offloading for OpenACC 2.0 being currently reviewed
  - Porting of libgomp (runtime library) to NVPTX being considered now
- About 230 new tests written for OpenMP 4.0 impl

SIMD support

- `#pragma omp simd` loops lowered into normal loops in the IL, with safelen recorded as an optimization hint on the loop structure
- SIMD lane privatized variables lowered into arrays indexed by GOMP_SIMD_LANE () magic function
- The vectorizer then bypasses some data ref analysis if safelen hints vectorization is safe, SIMD lane arrays vectorized into vector accesses; if vectorization fails, SIMD lane arrays changed into length 1 arrays and optimized later on
- `#pragma omp declare simd` results in cloning of functions, with an outer loop later vectorized; for i386/x86_64 SSE2, AVX and AVX2 variants, using slightly modified Intel Cilk+ mangling ABI

Depend, taskgroup, proc_bind

- Depend clauses lowered into an array of addresses of first bytes of variables / array sections (and initial two elements containing total number of depend addresses and how many of them are out), passed to GOMP_task, which then uses a hash table to determine inter-task dependencies
- Taskgroups implemented as GOMP_taskgroup_start / GOMP_taskgroup_end entrypoints, and extending library task scheduling algorithm
- For proc_bind changed #pragma omp parallel to use new GOMP_parallel call where even the initial thread has outlined parallel region called through callback, and flags passed
- OpenMP 4.0 affinity implemented using Linux pthread_setattr_affinity_np/pthread_setaffinity_np

Cancellation

- New library entry points: GOMP_cancel, GOMP_cancellation_point
- For workshare cancellation, just branch out (with running destructors of course) if those return true
- For parallel added to explicit and implicit barrier functions alternate entrypoints like GOMP_barrier_cancel which return if parallel has been cancelled; branch out if they return non-zero; alternate entrypoints used only in regions that can be cancelled
- Task cancellation handled similarly

User defined reductions

- Mostly done in the language front-ends, for multiple types in the same `#pragma omp declare reduction` the same tokens parsed multiple times with different type of the magic omp_out / omp_in / omp_priv / omp_orig artificial vars injected into scope
- During front-end finalization of OpenMP clauses, UDRs looked up and IL for merge and init operations added to the clauses
- The middle-end just emits those operations where needed

Offloading

- `#pragma omp target` region outlined into separate function
- Map clauses (and to / from clauses) lowered into quadruplets: host address, size, kind, alignment
- The quadruplets passed using 3 separate arrays to GOMP_target, GOMP_target_data and GOMP_target_update functions: one array holding host addresses (usually non-constant), another one the sizes (often but not always constant) and another one holding kind and alignment together (always constant)
- To the standard map clause kinds added two special ones, one is “pointer” - allocate with pointer translation – size is implicit, size array field holds pointer bias and “pointer set” for Fortran array descriptors – like “to”, but causes all following “pointer” kinds falling into the range not to allocate and just store the translated pointer

Offloading (continued)

- `#pragma omp declare target` functions, variables and outlined `#pragma omp target` regions marked with special attributes / flags
- If any accelerator targets configured, during compilation of host code the GCC GIMPLE IL for the flagged functions and variables is streamed as LTO bytecode into specially named data sections
- All the flagged variables stored also into special array, containing tuples: host address, size
- All the outlined `#pragma omp target region` functions stored into special array, containing the host addresses of the outlined region functions

Offloading (continued)

- During linking of host code, when the special data sections are found, linker plugin invokes separate compiler (e.g. for Intel XeonPhi) where the LTO bytecode is compiled into accelerator device code (for XeonPhi into shared libraries, one for each shared library or binary containing offloading code or variables) and also array with offloading target addresses
- GOMP_target then uses the arrays to populate mapping splay trees, map clauses also result in additions and later removals from the splay trees
- Offloading region not identified by names, but by translating host address of the offloaded region using splay tree to offloading device address to invoke

Upstream community OpenMP plans (tentative)

- Try to support OpenMP offloading for PTX target – as OpenACC PTX support will be there for GCC 5, it is mainly about porting the runtime library to the PTX dynamic parallelism and synchronization primitives so that target regions can use OpenMP directives
- HSA support for GCC is being worked on, likely to be merged for GCC 6
- Further vectorization improvements for SIMD regions, use thunks and wrappers instead of full functions for each of SSE2/AVX/AVX2 variants if desirable
- Support OpenMP 4.1 and OpenMP 5.0 when they are released