Implementing C++17 Parallel STL

(for heterogeneous and homogeneous parallel platforms!)

Using GCC's HSAIL and Offloading Support

Pekka Jääskeläinen
pekka.jaaskelainen@parmance.com

GNU Tools Cauldron 2017-09-09

Outline

- C++17 Parallel STL in a nutshell
- Heterogeneous Systems Architecture
- Implementation plan/discussion

C++17 Parallel STL (PSTL)

- Adds *execution policies* to the *algorithm* library of the Standard Template Library (STL)
  - std::parallel::seq – execute serially in the calling thread
  - std::parallel::par – execution may be parallelized (multithread)
  - std::parallel::par_unseq – execution may be vectorized and parallelized

```
sort(std::parallel::par, std::begin(input), std::end(input));
```

The TS provides parallelized versions of the following 69 algorithms from <algorithm>, <numeric> and <memory>:

Standard library algorithms for which parallelized versions are provided [Collapse]

- std::adjacent_difference
- std::adjacent_find
- std::all_of
- std::any_of
- std::copy
- std::copy_if
- std::copy_n
- std::count
- std::count_if
- std::equal
- std::fill
- std::fill_n
- std::find
- std::find_end
- std::find_first_of
- std::find_if
- std::find_if_not
- std::generate
- std::generate_n
- std::includes
- std::inner_product
- std::inplace_merge
- std::is_heap
- std::is_heap_until
- std::is_partitioned
- std::is_sorted
- std::is_sorted_until
- std::lexicographical_compare
- std::max_element
- std::merge
- std::min_element
- std::minmax_element
- std::mismatch
- std::move
- std::none_of
- std::nth_element
- std::partial_sort
- std::partial_sort_copy
- std::partition
- std::partition_copy
- std::remove
- std::remove_copy
- std::remove_copy_if
- std::remove_if
- std::replace
- std::replace_copy
- std::replace_copy_if
- std::replace_if
- std::reverse
- std::reverse_copy
- std::rotate
- std::rotate_copy
- std::search
- std::search_n
- std::set_difference
- std::set_intersection
- std::set_symmetric_difference
- std::set_union
- std::sort
- std::stable_partition
- std::stable_sort
- std::swap_ranges
- std::transform
- std::uninitialized_copy
- std::uninitialized_copy_n
- std::uninitialized_fill
- std::uninitialized_fill_n
- std::unique
- std::unique_copy

Source: http://en.cppreference.com/w/cpp/experimental/parallelism

PSTL Algorithms with Functors

- Some of the algorithms implement custom functionality via function object argument(s)
  - Custom comparison predicates
  - Custom transformation functions

```
std::transform(std::par, s.begin(), s.end(), s.begin(),
                [](unsigned char c) -> unsigned char {
                    return std::toupper(c);
               });
```

- Compile function objects to offload targets' ISA
  - Single source to multiple targets compilation

Heterogeneous Systems Architecture (HSA)

- Standard for shared virtual address space heterogeneous platforms
- Multiple specifications describing HSA-compliant systems and their programming
  - Platform System Architecture: hardware requirements for HSA compliant platforms
  - Programmer Reference Manual: kernel intermediate language HSAIL
  - Runtime Specification: host-side software layer
- Freely available: http://www.hsafoundation.com/standards/

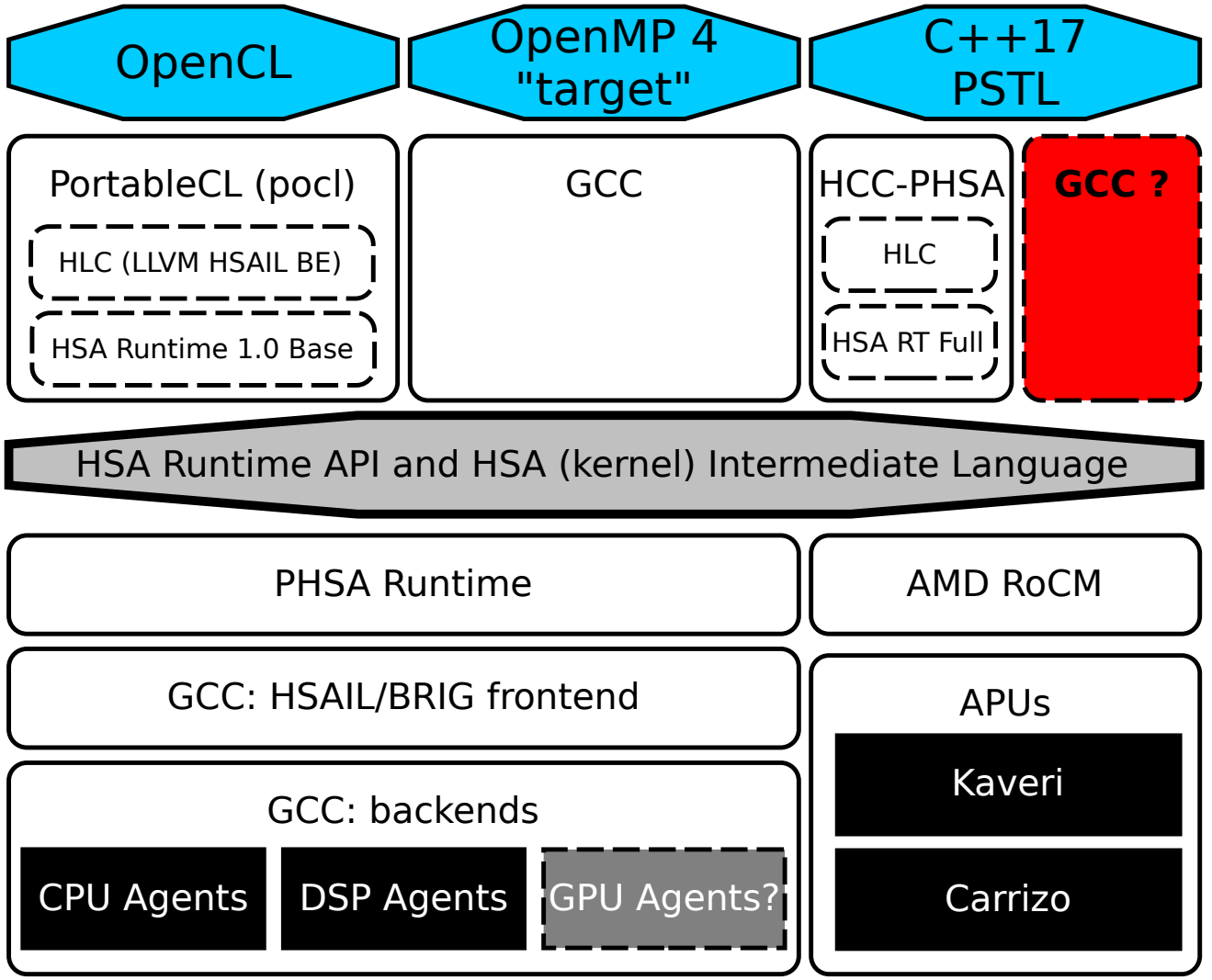
HSA's Key Benefits for C++17/PSTL in GCC

- Clearly identifies itself as a middleware on top of which higher-level higher-productivity programming languages and APIs can be implemented
- Unified coherent virtual address space a foundational feature in HSA Full profile
  - Sharing data between host and devices (Agents) using host-allocated pointers
  - Compare to OpenCL 2.0 SVM which requires an OpenCL-specific buffer allocation call that includes the buffer size
- **Good support for its intermediate language (HSAIL) in upstream GCC**
  - **HSAIL->GENERIC->... and ...->GIMPLE->HSAIL**
- Also other open source components freely available (next slide)

HSA-Based Open Source Heterogeneous Software Stack

- **GCC 7**
  - HSAIL input (finalization): 1.0 Base profile
    - Upstreamed, a *staging branch* at https://github.com/HSAFoundation/gccbrig
    - Currently optimizing the performance
  - HSAIL output: most of HSAIL features working
    - By Martin Jambor et al. for OpenMP 4 'target' offloading
- **PHSA-Runtime** – https://github.com/HSAFoundation/phsa-runtime
  - Implements the host side runtime API: 1.0 Base profile with support for most of Full profile
  - Can be used to implement CPU/DSP/GPU Agents, relies on GCC for HSAIL finalization
- **HCC-PHSA** – https://github.com/parmance/hcc
  - Fork of AMD's HCC with fixes for stock HSA runtime API and HSAIL
  - Clang/LLVM based
  - *Experimental support for Parallel STL offloading*
- **PortableCL (HSA driver)** – http://portablecl.org
  - OpenCL 1.2+ support for stock HSA (works also on PHSA)

HSA-Based Open Source Heterogeneous Software Stack



How to Implement the Outlining?

Note: Coding not started yet. This is only an initial plan to collect feedback.

- Outlining: Isolate the code that can run in another ISA
  - In this case the code is already isolated to a functor method (or a lambda), not an arbitrary code block
- Use OpenMP4 'target' pragma?
  - Shared/global region data: Do implicit data passing work now with OMP4 impl? What if data not defined in the same scope as the functor call?
  - Do indirect function calls work inside the regions currently?
  - How about OpenACC?
- Use Martin Jambor's "Direct HSA" patches?

Martin Jambor's Direct HSA Patch

- Attribute marking the HSA-compiled functions
  - Add the attribute automatically to the potential functors?
- What if we cannot analyze all possible indirect callees?
  - Add *hsa_kernel* to "every functor-looking function"?
  - Resolve/fallback to host execution at runtime if HSA version was not found
- Questions (to Martin):
  - Can *first_kernel* read non-local currently? Can we pass data via member variables or globals?
  - Does the fallback-to-CPU mechanism work already if *first_kernel* is not found in the HSAIL translated set?

```
void __attribute__((hsa_kernel))
first_kernel (int *p)
  {
    int i = *p;
    *p = i + 1;
  }

int main (int argc, char **argv)
  {
    int data = 2;
    __builtin_GOMP_hsa_direct_invoke (
        0, first_kernel, 1, 1, 0,
        0, 0, 0, &data);

    return data;
  }
```

Utilize the GCC Offload Infra for HSA Implementation?

Note: Coding not started yet. This is only an initial plan to collect feedback.

- libgomp-plugin-hsa.so? Would it make sense to implement a new one that basically mimics libcuda.so?
- HSAIL is currently not a gcc backend
  - Emits HSAIL from GIMPLE/SSA – is this a problem for the offload infra?
  - Should it be converted to a backend for cleaner integration? Any benefits?
- mkoffload-hsa:
  - Would output HSAIL/BRIG (the binary representation of HSAIL) that is fed to HSA finalizer API to produce ISA?

Performance Portability Considerations

- std::parallel::par
  - No vectorization across instances of functors allowed, but can still autovectorize inside a functor instance
- std::parallel::par_unseq
  - Can vectorize / "data parallelize" also across instances of functors to produce interleaved execution of multiple functor calls in the same thread
- Mapping functor calls to HSA execution concepts
  - *HSA Work-items*: Independent "threads" inside WGs, typically SIMD/SIMT executed in massive scale – *par_unseq*
  - *HSA Work-groups*: parallel collections of WIs typically executed with multiple independent cores/HW threads – *par*

Mapping to Different Targets

- GPUs and FPGAs favor massive number of work-items
- Multicore CPUs/DSPs/ISPs/the rest: Exploit multiple threads and SIMD instructions inside a thread
  - Split to work-groups to utilize MT (a runtime pre-launch decision in HSA)
  - Autovectorize / "SPMD vectorize" work-items across work-groups to improve SIMD instruction utilization
    - Works already with the BRIG FE + gcc autovec for simple non-barrier-using kernels – There's no barrier in C++17
    - Later on a CFG transformation can be added for handling barrier cases like we did for OpenCL in Portable Computing Language (*)

(*) http://portablecl.org

Conclusions

- C++17 added Parallel STL, which can be used to offload also to heterogeneous devices
  - Algorithms with and without custom functors
- HSA seems the best standard for implementation
  - Coherent system address space eases data sharing
  - Solid open source code base available: Can develop and test the stack using "CPU Agents"
- Implementation plan
  - Direct-HSA patch: hsa_kernel function attribute to outline potentially offloaded functions
  - Extend and utilize the current GIMPLE->HSAIL path or create a "proper" HSAIL backend?

Thanks for your interest!

Any input / advice / collaboration proposals welcome.

pekka.jaaskelainen@parmance.com

Relevant links:

HSA Foundation: http://hsafoundation.com
Portable HSA (phsa): https://github.com/HSAFoundation/phsa
GCCBRIG (BRIG FE staging repo): https://github.com/HSAFoundation/gccbrig
hcc-phsa fork: https://github.com/parmance/hcc
PortableCL (pocl): http://portablecl.org