[OpenMPOpt] Command Center

Note: Please feel free to comment or add anything in here, e.g., to ask for information or help!

Weekly Meeting

Meeting Notes
Tuesdays, 11:15am - 12:15pm CDT [UTC - 5]
https://bluejeans.com/280571403/3617?src=calendarLink

Collaborator List (add yourself!)

- Johannes Doerfert <jdoerfert@anl.gov> <johannesdoerfert@gmail.com> [UTC-5]
- Stefan Stipanovic <stefomeister@gmail.com> [UTC+2]
- Hamilton Tobon Mosquera <hamiltontobon77@gmail.com> [UTC-5]
- Giorgis Georgakoudis <georgakoudis1@llnl.gov> <ggeorgakoudis@gmail.com> [UTC-7]
- Joseph Huber <huberjn@ornl.gov> [UTC-6]
- Jose M Monsalve Diaz <josem@udel.edu> [UTC-5]
- Abid Malik <amalik@bnl.gov>

Tasklist (ever growing!, descriptions below the list)

OLD TASKLIST

ID	Name	Difficulty	Depends on	Assigned	Status	References
1	Add missing runtime functions	easy		Joseph	Done	
2	Add more attributes and attribute classes to runtime functions	easy - medium		Joseph	Done	
3	Deduplicate more runtime functions	easy				
3a	Deduplicate compatible `omp for` runtime calls	medium		Malik	Patch submitted. Working on tests submissions	

					and comments	
3b	Deduplicate runtime call pairs	medium				
4	Parallel region expansion	medium - hard		Giorgis	Under review	
5	ICV tracking	medium - hard		Stefan	In progress	
6	Barrier optimization	hard		Contact Giorgis		
7	Memory transfer latency hiding	medium - hard		Hamilton		
8	Interprocedural code motion	hard				
9	Heterogenous LLVM-IR modules	medium - hard		Shilei		
10	TRegion SPMD optimization	medium		Johannes?		
11	Utilize memory spaces	medium - hard				
12	Target region expansion and splitting	hard	#4 and #9	Shilei		
13	Track device memory mapping	hard		Prithayan		
14	Provide user and tool feedback	easy - medium		Joseph		
15	Guide heuristics, `omp loop`, `schedule(auto)`, …	easy - hard				
16	Modify the number of teams/threads	easy - hard				

17	Identify manually performed optimizations	medium				
18	Choose the reduction implementation	medium - hard		Fady?		
19	Utilize domain knowledge	medium				
20	Fix deduplication alloca interaction	easy		JD		link
21	Optimize local variable globalization in GPU code	mixed		Jose?		
22	OpenMP in the LLVM-Test suite	mixed				

Task Descriptions

1) Add missing runtime functions

In OMPKinds.def we define ~45 runtime functions with the `__OMP_RTL` macro. OpenMP and the OpenMP runtime have many more. We should eventually include all user facing OpenMP functions and the runtime functions generated by Clang. Once we do the latter we can replace the OpenMPRTLFunction enum in CGOpenMPRuntime.cpp as well as the createRuntimeFunction in the same file.

2) Add more attributes and attribute classes to runtime functions

In OMPKinds.def we define LLVM-IR attributes for the OpenMP runtime functions with the `__OMP_RTL_ATTRS` macro. We already define some important attributes for the runtime functions we list in this file but we could do better. As more information about runtime functions is crucial for later analysis and optimization, we need to ensure all existing and applicable attributes are set and new attributes are created if the need arises. As already done in the file,

we can group attributes into sets to simplify specifying them for various runtime functions at the same time.

3) Deduplicate more runtime functions

We need to extend and improve `deduplicateRuntimeCalls` in OpenMPOpt.cpp. That is, more runtime functions need to be listed if we can deduplicate them and we need to extend the logic to more than simple getter functions. See also #3a and #3b.

3a) Deduplicate compatible `omp for` runtime calls

Two consecutive `omp for` directives with a static schedule and the same number of iterations will cause the same two loop invariant runtime calls that will basically compute the same thing, namely the chunk for the executing thread. We can reuse the result instead of computing it again. As a consequence, two runtime calls become obsolete, less stack space is used, and, probably most importantly, we enable regular loop fusion as there are no more runtime calls between the loops and the loops bounds are trivially equal. To determine if the optimization can be performed is slightly tricky and rewriting the IR is as well. Once it works for two loops at a time we have to apply it until no more runtime calls can be removed.

3b) Deduplicate runtime call pairs

An extension of #3a is to apply similar logic to other runtime call pairs. Every pair might need some specific handling but parts should overlap, e.g., the control dependence check. As such, we should have a generic handling with specializations for runtime calls, e.g,. via template specialization based on the runtime function enum. An example that comes to mind outside of OpenMP is raja visitor/listener callbacks placed before and after a parallel region. We can deduplicate them as described in #3a to enable parallel region expansion #4 without the need for extra guard code.

4) Parallel region expansion

Please see Section 5 in https://compilers.cs.uni-saarland.de/people/doerfert/par_opt18.pdf and https://github.com/jdoerfert/llvm-project/tree/feature/openmp_opt

5) ICV tracking

Internal control variables (ICVs) are used to describe OpenMP semantics. This conceptual state may or may not have a corresponding state in the runtime. Either way, tracking the values of the ICVs would allow us to do various things including: (1) replace runtime calls with their known result, (2) improve heuristics, (3) emit warnings for the user. To track ICVs we need to define the

effects of runtime calls on their value as well as their default state. We then perform a fixpoint data-flow analysis to determine their state at every (interesting) program point.

6) Barrier optimization

(Explicit) barriers are not necessarily common in HPC code but they can occur. Especially after parallel region expansion #4 they happen to emerge. As barriers are “known” to be expensive, it is important to lower their cost. Different ways exist, some below.

6a) Barrier elimination

Please see Section 6 in https://compilers.cs.uni-saarland.de/people/doerfert/par_opt18.pdf.

6b) Barrier movement

Barriers can sometimes be hoisted/sunken out of loops, especially after inlining or as inter-procedural optimization.

6c) Barrier replacement

Barriers can often be replaced with different (synchronization) schemes, e.g. atomics or privatized variables & post processing, explicit dependences, ...

7) Memory transfer latency hiding

From the GSoC proposal by Hamilton:
Given the increasing number of use cases for massively parallel devices (GPUs), solving the problems they bring have become an important research field. One of the main problems that needs to be solved is the long time (latency) that it takes to move data from the computer’s main memory to the device’s memory. Therefore, using the LLVM compiler infrastructure, the proposed solution consists of adding a new functionality to the current OpenMP interprocedural optimization pass, OpenMPOpt, such that the OpenMP runtime calls that involve host to device memory transfers are split into “issue” and “wait” functions as follows.

- *The “issue” function will contain the code necessary to transfer the data from the host to the device in an asynchronous manner (extending the functionality of the analogous _nowait runtime calls), returning a handle in which the “wait” function will wait for completion.*
- *After splitting the runtime call, the call to the “issue” function will be moved upwards in the code until it is illegal to do so. In the same way, the call to the “wait” function will be moved downwards in the code until it is illegal to do so. Determining whether a movement is legal or not will be done using data flow techniques and the control flow graph of the function where the original runtime call is issued.*

- *Doing this, the instructions between the "issue" and the "wait" can be executed, while separately doing the data transfer to the device, hence, reducing the time the process is blocked waiting for the transfer to finish.*
- *In addition to this, given the case when a "wait" call is waiting on a transfer X to be finished, and immediately after the "wait" there is an "issue" to move X back to where it came from, then there is no need to "issue" and "wait" in both cases, they can be eliminated.*

8) Interprocedural code motion

Please see Section 7 in https://compilers.cs.uni-saarland.de/people/doerfert/par_opt18.pdf

9) Heterogenous LLVM-IR modules

In order to perform host-device code optimizations in a reasonable way we need to see the code for both at the same time, preferably in the same IR module.
TODO: Paste from Kaushik's GSoC proposal

10) TRegion SPMD optimization

Please see and https://link.springer.com/chapter/10.1007/978-3-030-28596-8_11
http://parallel.auckland.ac.nz/iwomp2019/slides_TRegion.pdf

11) Utilize memory spaces

If a device offers different memory spaces it can be beneficial to utilize them. The most straightforward example is a constant sized (at compile time) stack allocations inside a teams regions on a GPU. Such memory can be moved into "shared local memory" of the device instead of residing in global memory. This particular transformation is hard to perform until we employ TRegions #10 as Clang otherwise globalizes such stack allocations early. We can also explore the usage of constant memory for readonly arrays. An extension to the existing API is required for the latter.

12) Target region expansion and splitting

The parallel region expansion logic #4 is also applicable to target regions (aka. kernels). In addition, kernels might actually benefit from splitting, or more general, rearranging of the kernel scopes to avoid high register pressure. This depends on #4 and #9.

13) Track device memory mapping

If the memory mapping is (partially) known we can remove pairs of map instructions or simplify single map instructions. We can also provide the user feedback and warnings. This should follow the same scheme and share infrastructure with #5 with the complication of aliasing.

14) Provide user and tool feedback

Whenever we analyze or transform the code we should emit remarks to the user or tools that want to follow the reasoning and decisions. We need an infrastructure that makes this simple for OpenMPOpt developers, which utilizes additional information, e.g., the source locations in the ident_t struct, and which works well with the existing OpenMP profiling tools. In a related effort we need to determine how to preserve source information when transformations are performed and how tools generally deal with the difference between source OpenMP and executed OpenMP. One outcome could be an “OpenMP advisor” that informs about good and bad code, provides feedback and explanations, determines OpenMP related UB, … We should report various things, e.g., hard-coded thread counts are often a bad idea. It is always a bad idea to assume the thread count is reached, ...

15) Guide heuristics, `omp loop`, `schedule(auto)`, …

We need to utilize code analysis to guide existing and upcoming heuristics. The way to implement `omp loop` heavily depends on the context. The runtime does only know about the history, thus what was executed so far, not what is going to be executed next (guaranteed, likely, potentially, for sure not, ..). PGO and value (range) information can also be used here. We can adapt the chunk size of dynamic loops, make them

16) Modify the number of teams/threads

We want to serialize parallelism that is (likely) not beneficial. We want to decrease/increase parallelism when it is clearly beneficial, e.g., when an application is ported or loop bounds are low. This is only legal if the maximal number is not (implicitly) used.

17) Identify manually performed optimizations

Advanced OpenMP codes often perform manual optimizations a compiler can and should do. Some codes (I think in the NAS benchmarks) have expanded parallel regions #4 in which code is executed redundantly and via worksharing loops. Similarly, we can scout code and talk to developers to identify opportunities for advanced (OpenMP specific) optimizations.

18) Choose the reduction implementation

Reductions, especially on GPUs, can be performed in various ways. Depending on the device, memory involved, and other factors, the performance varies significantly. (In a 2018 CGO paper different reduction schemes were benchmarked on various problems.) Similar to TRegions #10, we need to expose the reduction implementation choices in the runtime API so we can pick after code analysis, potentially even at runtime.

19) Utilize domain knowledge

OpenMP comes with various restrictions that could be exploited, potentially after the user provides certain information, e.g., that the thread limit is always > 1. Especially alias information could often benefit in the presence of map clauses or actual concurrent execution (due to the absence of races).

20) Fix deduplication alloca interaction

We should move all allocas before the deduplicated runtime calls.

21) Optimize local variable globalization in GPU code

A motivating example is https://clang.godbolt.org/z/EobPfr
Optimizations:

1) The globalization code (roughly 116 - 128) is not dead but “needed”. We should however have it in a runtime function that spits out a memory location to be used.
2) Once 1) is done, we can run AANoCapture on the return value of the above call, I guess we could even do it w/o the call. Anyway, AANoCapture will tell us if the return value is eventually captured. If it is not, we can replace the call with the alloca as there is no need for globalization.
3) If 2) fails, so we keep the call/code for globalization, we should determine if we are in a parallel region or not. If we are, use the alloca (mainline has no support for nested parallel regions that are not sequentialized).
4) If 3) fails, determine if the allocation is “small” and use shared memory instead. I think the runtime function has even a flag for that, not sure if it works or not though.

22) OpenMP in the LLVM Test Suite

Meeting Notes

Date: Tuesday, August 25th
Agenda:

- Weekly Updates:
  - Hamilton:
  - Stefan:
- Patches:
  - To be committed:
  - Need review:
    - https://reviews.llvm.org/D86300
    - https://reviews.llvm.org/D86474

- Questions & Misc:
  - Slides: https://docs.google.com/presentation/d/1ELDTEzySXWULqs6eQ1XL25u195OidDw_UkySa87XrUw/edit?usp=sharing

Date: Tuesday, August 18th
Agenda:

- Weekly Updates:
  - Hamilton:
  - Stefan:
    - ICV tracking for calls with multiple AAs (should be ready now)

        - Default attributes for intrinsics TableGen + actually making them default
        - UpdateTestChecks check “attributes #”
- Patches:
    - To be committed:
    - Need review:

- Questions & Misc:
    - Slides: https://docs.google.com/presentation/d/1ELDTEzySXWULqs6eQ1XL25u195OidDw_UkySa87XrUw/edit?usp=sharing

Date: Tuesday, August 11th
Agenda:
- Weekly Updates:
    - Hamilton:
    - Stefan:
- Patches:
    - To be committed:
    - Need review:

- Questions & Misc:
  https://reviews.llvm.org/D85703

Date: Tuesday, August 6th
Agenda:
- Weekly Updates:
    - Hamilton:
    - Stefan:
- Patches:
    - To be committed:
    - Need review:

- Questions & Misc:

Date: Tuesday, July 14th
Agenda:
- Weekly Updates:
    - Hamilton:
    - Stefan: default attributes for intrinsics, update_test_checks (“Function Attrs”), ICV callsite information (should have a patch soon)
- Patches:
    - To be committed:

        - Need review:

    - Questions & Misc:

Date: Tuesday, July 7th
Agenda:
- Weekly Updates:
    - Hamilton:
    - Stefan: Basic ICV committed, default attributes for intrinsics, OMPIRBuilder fix
- Patches:
    - To be committed:

    - Need review:
        - D83176

- Questions & Misc:

Date: Tuesday, June 30th
Agenda:
- Weekly Updates:
    - Hamilton:
        - Latency Hiding of M2D Transfers
            - Write unit tests for getValuesInOfflArrays()
    - Stefan: More ICV tracking, Default attributes for intrinsics (opt-out list) D70365
- Patches:
    - To be committed:
        - **D80051**
    - Need review:
        - **D82719**
- Questions & Misc:

Date: Tuesday, June 23th
Agenda:
- Weekly Updates:
    - Hamilton:
        - Latency Hiding of M2D Transfers
            - Refactor getValuesInOfflArrays()
    - Stefan: ICV Tracking basic deduplication
    - Joseph:
- Patches:
    - To be committed:

    - Need review:
- Questions & Misc:

Date: Tuesday, June 16th
Agenda:

- Weekly Updates:
    - Stefan:
      - OMPInformationCache, ICV macros, Attributor initialization
    - Hamilton:
        - Latency Hiding of M2D Transfers
            - Identifying the memory regions to be offloaded by a runtime call.
            - Detecting the last instruction that can potentially modify the offloading regions.
            - Detecting the instructions associated with the “issue”. Needs to be changed.
            - Need of a structure that handles the information of the runtime call, whether it was split or not.
    - Joseph
        - OpenMP Runtime Function Attributes
            - https://reviews.llvm.org/D81031
        - Initial Analysis Remarks (Waiting on full ICV)
            - https://reviews.llvm.org/D81036
- Patches:
    - To be committed:
    - Need review:
- Questions & Misc:

Date: Tuesday, June 9th
Agenda:

- Weekly Updates:
    - Stefan:
        - ICV tacking
    - Hamilton:
        - Splitting code and interfaces
        - Needs use of MemorySSA instead of manual detection of memory definitions.
    - Joseph
        - Analysis remarks
- Patches:
    - To be committed:

        - Need review:
    - Questions & Misc:

Date: Tuesday, June 2nd
Agenda:

- Introductions
- Administrative questions
    - Meeting time slot
- Initial Tasks
- Questions & Misc