Shared Memory Based JITLink Memory Manager

Name: Anubhab Ghosh
GitHub Username: argentite
Email: anubhabghosh.me@gmail.com
LLVM Discord Username: argentite#0791
LLVM Discourse Username: argentite

Description

When a separate executor process is used with LLVM JIT, the generated code needs to be transferred to the executor process. This is done by the JITLinkMemoryManager. The current implementation uses ExecutorProcessControl API (an RPC scheme) to send the generated code which goes through pipes or network sockets.

The goal of the project is to transfer it through an operating system provided shared memory regions for better performance, when both the JIT process and the executor process are sharing the same underlying physical memory. It should be done by allocating large chunks of memory and distributing it to reduce memory allocation overheads and inter process communication.

Mentors

- Vassil Vassilev
- Lang Hames

Size of Project

Large

Expected Deliverables

- A generic shared memory allocation (and deallocation) API for using in LLVM
- A slab based memory allocator that uses the shared memory APIs for allocating shared memory space in large chunks to avoid overhead of small allocations
- Implementation of the JITLinkMemoryManager using this slab allocator

Timeframe of Participation

13th June to 19th September 2022 according to the GSoC timeline.

Project Timeline

The expected timeline of the project is as follows:

Week 1

Implement a MemoryMapper interface and a shared-memory based implementation. The memory mapper will be responsible for reserving memory, recording and applying deallocation actions, and deallocating memory.
Deliverable: API for interprocess memory allocations and deallocation

Week 2

Continue the MemoryMapper implementation. The memory mapper should also be able to apply protections, run finalization actions.
Deliverable: Complete MemoryMapper implementation with finalization and memory protection

Week 3

Implement a JITLinkMemoryManager that uses the MemoryMapper API naively by reserving memory for each individual allocation.

Week 4

Continue the JITLinkMemoryManager implementation and buffer time for debugging any problems
Deliverable: Ability to run JIT code with the new memory manager

Week 5

Implement a HeapManager interface and at least a naive heap-management implementation.
Deliverable: A HeapManager implementation

Week 6

Extend the JITLinkMemoryManager to use the MemoryMapper API more efficiently by reserving larger regions up-front (on first allocation), and using the HeapManager to perform the actual allocations within the reserved region.
Deliverable: JITLinkMemoryManager with heap based allocations

Phase 1 evaluation

Week 7

Use this week as a buffer for any issues or other delayed tasks

Week 8

Implement the previous shared memory mapper interface for Windows
Deliverable: Windows implementation of the new memory mapper

Week 9

Continue last week's work. The llvm-jitlink tool (and executor) needs to be fixed to work

under Windows
Deliverable: Ability to run JIT code on Windows

Week 10-11

Test and benchmark with different types of code to find and fix any issues and get the performance impact.
Deliverable: Benchmarks results

Week 12

Documenting the work
Deliverable: a blog post

Personal Details

- I am a 3rd year student at Indian Institute of Information Technology, Kalyani. I am interested in compiler design and development. I have taken two courses in compiler design and automata theory.
- I have done an internship under INRIA, Paris where I worked on an OCaml lex-yacc based tool that parsed (incomplete) portions of C code extracted from git diffs (of the Linux kernel source) and analyzed them for multiple occurrences of similar changes. I also wrote another tool to approximately convert python code to a C-like syntax to be used by the former parser.
- I have had another internship under INRIA where I performed some experimental modifications on the Linux kernel scheduler introducing an atomic flag to avoid race conditions when multiple processes wake up at the same time causing them to end up at the run queue of the same core instead of other free ones.

Have you had any prior contributions to LLVM? If yes, please provide links to these contributions.

I have submitted a patch here: https://reviews.llvm.org/D123902
I have recently gained some familiarity with the relevant code while hacking together a quick and dirty proof of concept shared memory based allocator over the existing EPC based inter process allocator, as part of mentors' evaluation task.

Technical Skills

- C/C++
- GNU Debugger
- Git

Availability & Other Commitments

Our next semester classes are expected to begin approximately from the start of August but the workload is very low in the beginning.