CAPTURE TRACKING IMPROVEMENTS

ABSTRACT

Capture tracking analysis is an analysis pass used to determine which pointers are "captured". This means that a function has made a copy of a pointer that may potentially outlive the function that called it. This information is useful during the optimisation process; this is used to improve memory management via aliasing.

The capture tracking analysis is currently inefficient and inaccurate in cases due to the fact that it returns false positives and expensive functions are unnecessarily repeated. This is where I would make improvements. I would aim to reduce the number of false positives without increasing the compile time significantly. It could be improved in a number of ways, as mentioned by Philip Reames on the mailing list. I would like to use this opportunity to take my previous experience within LLVM and apply it to other areas of LLVM.

BENEFITS

Improvements to pointer capture tracking would be greatly beneficial. Changes such as minimising the number of false positives as well as the reduction to the cost through the introduction of caching will be made. Caching the results of certain functions, storing the result, will cause performance increases, should the same data be requested again as the result will already be stored. As a result of this, it could become a more valuable tool to have within the LLVM suite. This would also cause improvements to the code optimisation process and potentially increase the quality of code compiled with LLVM. It should also use up fewer resources during the analysis pass.

DELIVERABLES

- There are cases where "potentially captured" is returned incorrectly. As a starter task I would gather a list of all of the known and unknown cases that cause this to occur and work on removing them. This would serve as a good introduction to LLVM compiler analysis and would be achievable in a short time span.
  Approximately 5 weeks.
- By making changes to the current design, this could be made to be more cost effective than it currently is. I would do this by caching the results as previously suggested. This will be used to invalidate results when required.
  Approximately 6 weeks.

- The analysis could be made more accurate in order to recognise object sub-graphs which do not escape.
  Approximately 5 weeks.

This work will link in with other ongoing work within LLVM and will assist other developers working on compiler analysis. I believe that I may not be able to complete all of these tasks within the given time, however I will aim to.

BIOGRAPHICAL INFORMATION

I have been working on LLVM for the past year as an industrial placement student within a compiler team and am keen to do more LLVM work. I am a BSc Computer Science student at the University and will begin my final year of study in September. In the past I have worked on a University project which required me to be inventive with data structures in order to efficiently solve the given problem. I thoroughly enjoyed this and would be grateful for another opportunity to exercise my design skills.

Due to other commitments with my current industrial placement, unfortunately I will be unable to work on this project during working hours (9-5 GMT) until the first week of June. However, I am more than happy to make up this time as the summer progresses.