ScopLib support for Graphite

Linking Graphite to the huge industrial and research community

April 27, 2010

1 Summary

Many polyhedral optimization tools and libraries have been developed since the 90's. Our project aims to enable Graphite to take advantage of these tools by making it able to export/import internal polyhedral representation to the scopLib file format, a standard format used for communication between polyhedral tools and libraries.

2 The project

Advanced compiler optimization projects such as Graphite need a huge amount of work, not only from the side of developers but also from the research and the industrial side. Collaborative work is crucial to the project, our proposal aims to bridge the gap and enable better collaboration between worldwide researchers/industrial users and Graphite developers, the final benefit for Graphite is to reach a better optimized, well tested and more reliable code generation.

Graphite uses the polyhedral model as a basic model for loop optimization and parallelization. Many tools such as PoCC[1], Loopo[2] and Pluto[3] rely also on this model to perform advanced loop transformations. Making graphite benefit from the algorithms that are currently implemented in these tools is a big step toward the support of a full polyhedral framework in GCC.

The goal of our project is to enable Graphite to Export/Import polyhedral internal loop representation to other polyhedral tools through the scopLib file format, a format designed to enable polyhedral tools and libraries to communicate with each other. GCC and graphite will parse loops, detect scops, and prepare structures needed for polyhedral optimization, these structures are dumped to a scop file (a text file in the scopLib format that specifies all necessary information for loop transformations), polyhedral tools will then read this file, apply polyhedral transformations and dump the result back to a scop file, Graphite will then read this file, and proceed to generate code.

3 Benefits for Graphite

- ScopLib support will help developers in the mid-term to experiment with different "real world" transformations that come from different tools and that are not yet implemented in Graphite, this is crucial for developers to take strategic decisions about futur releases. With scopLib support, Graphite will be better tested and more parts of graphite will be evaluated with a variety of test cases in order to simulate real world usage. To encourage this aspect, a big part of our work is based on testing and problem-hunting,
- By enabling the communication between Graphite and external tools, new algorithms will be tested easily outside of the complexity of GCC and the best of these algorithms will be merged to GCC. Researchers from all over the world will work side by side with the graphite community to evaluate and provide a valuable feedback about the real efficiency of the generated code,
- Since the scopLib format is human readable, it will help greatly graphite developers, especially in debugging and testing special and rare cases,
- This work is a basic work for futur support of privatization in Graphite,

[1] http://www-rocq.inria.fr/~pouchet/software/pocc/
[2] http://www.infosun.fim.uni-passau.de/cl/loopo/
[3] http://pluto-compiler.sourceforge.net/

- The project will complete the circle (Graphite ↔ research community↔ industry) : our project has a long term effect on Graphite and GCC, it will enable more people to get involved in Graphite and thus more papers will appear and more advanced algorithms will be implemented, the results will encourage more support from the industry[4] [5]. All the benefit is for Graphite and for GCC,

- Graphite will benefit from a wealth of advanced polyhedral optimization algorithms implemented by researchers from all over the world, it will be able to communicate with any polyhedral tool/library that implements the standard ScopLib format. It's a big step towoard a large usability of Graphite code outside GCC in all industrial and academic open source tools.

4 Details about the project

After scop formation in "build_poly_scop()", scops are optimized through "scop_do_block()", "scop_do_strip_mine()" and "scop_do_interchange()". After these optimizations, we export the scop (dump it) to a text file in special standard format readable by other standard polyhedral tools (and human readable also). After scop dumping, the tool reads the scop file, applies appropriate polyhedral transformations and then dumps the transformed polyhedral structures into a scop file. Graphite will then read the generated file and proceed to code generation through Gloog "scop_to_clast()".

The current internal scop representation in graphite is actually different from the simple scopLib format. In graphite, a scop is formed of a vector of polyhedral basic blocks (poly_bb or pbb for short), each pbb has its own scattering function, iteration domain, and a vector of polyhedral data references (poly_dr_p) that describe read and write accesses.

ScopLib has the same general structure, but differs in some internal details from the graphite polyhedral representation, in this section, we present a list of differences between scopLib format and the internal Graphite format :

- Statement representation : this is different between Graphite and scopLib, but as the statement section in a scop file is not needed by Pluto to perform loop transformations, it will be ignored for now and not considered as a priority,

- Iteration domain : exactly the same between Graphite and scopLib,

- Scattering functions : Graphite uses full scattering relations represented as a union of polyhedron whereas in scoplib the scattering function is a set of affine functions that does not consider union of polyhedron. We will use the extended scopLib format to handle these differences (the extended scopLib is an extension that supports scattering relations, union of scattering polyhedrons...),

- Data accesses : data accesses in graphite are represented as relations, scopLib represent accesses as functions, we will use the extended scopLib format to be able to represent graphite data accesses,

- Data dependencies : polyhedral tools need dependency information to perform optimization. Data dependencies will not be exported from Graphite, they will be calculated by the tool itself (Pluto will call candl to extract data dependencies and insert them into the .scop file).

Test plan

- Make sure that graphite is generating the correct .scop files by comparing between what graphite generates and what PoCC generates (we need to store some correct scop files generated by pocc in the testsuite and compare them with the scop files generated later by Graphite),

- Make sure that Graphite is applying correctly what it's reading from the scop file by reading the scop file and comparing the result with what should be obtained,

- Test the whole process : Compile a C program and optimize it with Pluto, a scop file will be generated, optimized by Pluto and read back again by Graphite.

We will need to add these three tests to the GCC testsuite.

[4] Diego Novillo. Keynote talk: Using GCC as a toolbox for research: GCC plugins and whole-program compilation. 2nd International Workshop on GCC Research Opportunities (GROW'10). Italy, 2010.

[5] Y.Huang, L.Peng, C.Wu, Y.Kashnikov, J.Rennecke, G.Fursin. Transforming GCC into a research-friendly environment: plugins for optimization tuning and reordering, function cloning and program instrumentation. 2nd International Workshop on GCC Research Opportunities (GROW'10). Italy, 2010.

5 Required deliverables

- Graphite must be able to Import/Export a portable polyhedral representation fully compatible with scopLib format,
- We will focus on tiling in Pluto as an example for code transformations, by the end of the project, Graphite must become able to deal with tiling through Pluto without any problem,
- Pass regression tests.

6 Nice to have

According to advancement in work on scoplib and results, we want to :

- Be able to export dependency information from Graphite to the scop file,
- Enable graphite to deal with Pluto annotations for privatization :
  - Focus on simple cases where the implementation relies on OMP,
  - Enable more advanced privatization.

Additional Info

Project wiki page : http://gcc.gnu.org/wiki/ScopLibSupportForGraphite