A survey on Machine Learning in Compiler

Sriyasree Swain*, Bhavani Supriya*, Nivedita Manohar, Vijaya Lakshmi R****

*Dept.of Information Technology
ACED, Alliance University, Bengaluru

**Assistant Professor, Dept.of CS/IT,
ACED, Alliance University,Bengaluru

Abstract: Optimization of a tuning hardwired compiler is the rapidly budding hardware which makes porting a compiler which will be optimizing in its feature and challenging. The approach for the such challenging compiler is with development of modular, self-optimizing and extensible compilers that adopt the best optimization heuristics based on the behavior of the platform. The contribution of the machine learning towards the development of compiler which is capable to adjust automatically with improved execution time, code size on different architectures is Machine Learning for Embedded PrOgramS optimization (MILE POST). Recursive queries technique can be utilized for effective execution plans and the resulting runtime plans can be executed on a single unified data parallel query processing engine.
High performance software development is also difficult task that requires the use of low-level, architecture specific programming models such as MPI for clusters, CUDA for GPUs, and OpenMP for CMPs. Probabilistic search of the optimization space can support to a significance speedup over the baseline compilers with the higher optimization settings, on several different processor architecture. Classifier systems are massively parallel, message-passing, rule-based systems that learn through credit assignment (the bucket brigade algorithm) and rule discovery (the genetic algorithm). They typically operate in environments that exhibit some characteristics such as -perpetually novel events accompanied by large amounts of noisy or irrelevant data, continual, often real-time, requirements for action, implicitly or inexactly defined goals, sparse payoff or reinforcement obtainable only through long action sequences. Classifier systems are designed to absorb new information continuously from such environments, devising sets of competing hypotheses (expressed as rules) without disturbing significantly capabilities already acquired.
The survey highlights the approaches taken so far, the obtained results, the fine-grain classification among different approaches and finally, the influential research in the field. The challenges are listed at end.
Index Terms- Optimization, Classifier systems, Machine Learning

I INTRODUCTION

In simple design of game software, any good player can complicate the system by throwing some twists. But in real world such twist will be more, and design of system should support to face such twists. For a machine learning system, the problem is one of constructing relevant categories from the system's primitives (pixels, features, or whatever else is taken as given). Discovery of relevant categories is only half the job; the system must also discover what kinds of action are appropriate to each category. The overall process bears a close relation to the Newell-Simon [3] problem solving paradigm, though there are differences arising from problems created by perpetual novelty, imperfect information, implicit definition

of the goals, and the typically long, coordinated action sequences required to attain goals. Environments wherein timely outside intervention is difficult or impossible. The only option then is learning or, using the more inclusive word, adaptation. In broadest terms, the object of a learning system, natural or artificial, is the expansion of its knowledge in the face of uncertainty. More directly, a learning system improves its performance by generalizing upon experience. Clearly, in the face of perpetual novelty, experience can guide future action only if there are relevant regularities in the system's environment. Human experience indicates that the real world abounds in regularities, but this does not mean that it is easy to extract and exploit them. The overall process bears a close relation to the Newell-Simon [40] problem solving paradigm, though there are differences arising from problems created by perpetual novelty, imperfect information, implicit definition of the goals, and the typically long, coordinated action sequences required to attain goals[1][2][3].

New compiler techniques must arise to support complex image processing applications without sacrificing programmability. This paper focuses on two image processing interfaces considered as DSLs, Simple Morphological Image Library(SMIL) and Framework for Embedded Image Applications (FREIA), supporting each a different set of hardware targets and providing different levels of programmability. We built a compiler to automatically generate lower-level but more portable FREIA DSL code from high-level SMIL DSL applications. We evaluate this compiler on a set of seven image processing applications. Some of the advanced compilers to support image processing are Simply Morphological Image Library(SMIL), Framework for Embedded Image Applications (FREIA), Cython which is a Python to C – Compiler[4].

To locate the bugs of program it is necessary to design new types of compilers which will support to debug the errors in server level as well as in kernel levels[5]. These all compilers are the challenges of the competitive industry these are discussed. The reference reveals that in the domain of Big data domain specific languages are necessary to extend with machine learning concept in bigdata. In such situation specific languages should support for all aspect with optimized way[6].

ScalOps is a DSL with the goal of enabling Machine Learning algorithms to run on a cloud computing environment and overcoming a limitation of the traditional MapReduce programming model: the lack of iteration. ScalOps is a textual programming DSL developed in jointly by the University of California, Irvine and Santa Cruz, and the division Yahoo! Research. The DSL also has Scala language serving as a host language, which means that ScalOps is an internal DSL. Its high-level syntax makes it a declarative language, and as the type checking happens in compilation time, ScalOps is considered a statically typed. ScalOps needs to be compiled to generate lower level code, which makes it be classified as a translated language, according to the analysis of this survey. Additionally, the language supports vector, matrix, and graph operations in both parallel and cloud computing environment. To support iterations in MapReduce, ScalOps designers introduced an enhanced version of the programming model called Map-Reduce-Update. This new version consists of three user-defined functions called map, reduce, and update. The map function receives read-only global state values and is applied to training data points in parallel. The reduce function aggregates the output of the map function. Finally, the update function receives the aggregated value and produces a new global state value for the next iteration. Alternatively, when appropriate, the update function indicates that no additional iteration is necessary [9][10][11].

Compilation can be speed up the compile processes by at least a factor of two with almost the same generated code quality on the SPEC2000 benchmark suite, and that our logistic classifier achieves the same prediction quality for non-SPEC benchmarks.

II. METHODOLOGY

The compilation algorithm can be written as shown in the Table 1. The algorithm accept the Majority Inverter Graphs (MIG) M as an input to give a Programmable Logic-in-Memory (PliM) program as an output. PliM enables logic operations on a regular Resistive Random-Access Memories (RRAM) array. It uses a single instruction RM3, which computes the three major operations in which one input is inverted [10]

Table 1: Compilation Algorithm

Input MIG M

1. Foreach leaf in M
2. Do set COMP[v]← T
3. End
4. Foreach MIG node in M
5. Do if all children of v are computed then
6. Q.enqueue(v)
7. End
8. End
9. While Q is not empty
10. Do set c ← Q.pop();
11. Set P ← P U `translate( c );`
12. Set COMP[c] ←T
13. Foreach parent of c do
14. If all children of v are computed then
15. Q.enqueue(v);
16. End
17. End

SMIL Python Code:

```
import smilPython as smil
imin = smil.Image("input.png")
imout = smil.Image(imin)
smil.dilate(imin, imout)
imout.save("output.png")
```

FREIA C Output Code:

```
#include "freia.h"
#include "smil-freia.h"
int main(int argc, char *argv[]) {
/* initializations... */
freia_data2d *imin;
imin = freia_common_create_data(/* */ );
freia_data2d *imout;
imout = freia_common_create_data(/* */ );
#define e0 SMILTOFREIA_SQUSE
#define e0_s 1
freia_cipo_dilate_generic_8c(imout, imin, e0, e0_s);
freia_common_tx_image(imout, &fdout);
freia_common_destruct_data(imout);
freia_common_destruct_data(imin);
/* shutdown... */
}
```

Figure 1 SMIL Input Code and FREIA C Code

For SMIL applications, the compiler such as smiltofreia generates directly FREIA C code from SMIL Python programs. The sample of such code is as shown figure 1.[13]

III CONCLUSION

The study throws many challenges such as requirement of optimized compiler for processing image processing languages, conversion from one languages to other many languages. The translation of languages is essential in case of multi programming development. This makes user to adopt any one language. Hardware specific optimization compilation is also required which is one of the challenging.

REFERENCES

[1] Grigori Fursin,et al, "MILEPOST GCC; machine learning based research compiler".

[2] L.B.Booker,D.E. Goldberg and J.H Holland," Classifier Systems and Genetic Algorithms", Artificial Intelligence, Elsevier Science Publisher, 1989.

[3] Newell A and Simon H A, Human Problem Solving (Prentice-Hall, Englewood Cliffs,NJ,1972).

[4] Pierre Guillou, Bnoit Pin, Fabien Coelho, Francois Irigoin, " A Dynamic to Static DSL Compiler for Image Processing Application", Processing Applications. Compilers for Parallel Computing 2016. . July 8, 2016, Valladolid, Spain.

[5] Chengnian Sun, VuLe, Qirun Zhang, Zhendong Su, " Toward Compiler Bugs in GCC and LLVM", ACM, ISSTA '16, July 18-22

[6] Ivens Portugal David R. Paulo Alencar David R. Donald Cowan David R. ," A Survey on Domain-Specific Languages for Machine Learning in Big Data",

[7] Gennady Pekhimenko , Angela Demke Brown, "Efficient Program Compilation through Machine Learning Techniques", http://sysweb.cs.toronto.edu

[8] Borkar, V., Bu, Y., Carey, M. J., Rosen, J., Polyzotis, N., Condie, T., Weimer, M., & Ramakrishnan, R. (2012). Declarative Systems for Large-Scale Machine Learning. IEEE Bulletin of the Technical Committee on Data Engineering, 35(2), 24-32.

[9] Mathias Soeken et al, "An MIG based Compiler for programmable Logic-in-Memory Archtectures

[10] Weimer, M., Condie, T., & Ramakrishnan, R. (2011). Machine learning in ScalOps, a higher order cloud computing language. In NIPS 2011 Workshop on Parallel and Large-scale Machine Learning (BigLearn) (Vol. 9, pp. 389-396).

[11] Odersky, M., Spoon, L., & Venners, B. (2005). Programming in Scala. École Polytechnique Fédérale de Lausanne.

[12] Pierrc Guilou et al " A Dynamic to Static DSL Compiler for Image Processing Application",