



## An Update on MILEPOST

Jeremy Bennett , Embecosm

The original MILEPOST project ran from 2006-2009 with the objective of integrating machine learning technology into compilers, particularly for embedded applications, to improve the quality of the code generated. The collaboration, supported by HiPEAC, involved INRIA and CAPS Enterprise in France, IBM in Israel and the University of Edinburgh and ARC International in the UK.

### The Challenge

Modern compilers offer a huge range of possible optimizations. For example the latest version of the GNU C compiler has over 200 different optimization passes. All compilers provide a set of option flags, so the user can control which optimizations are used.

However not all optimizations work well on call code. Compilers often have flag settings that group together options that are likely to work for most programs. Just GCC offers -Os (optimize for small code) and -O1, -O2 and -O3 (increasingly optimize for speed).

However as all programmers will know sometimes -O3 makes a program run more slowly. A large program may execute faster using -Os, because it uses the cache more efficiently.

There is a solution to this, called *iterative compilation*. The user repeatedly compiles their program, trying different sets of optimization options and finally selecting the best set. This can sometimes double performance, compared with the standard set of optimizations, but is of course hugely laborious, and infeasible for all but the most key program components.

### What is MILEPOST?

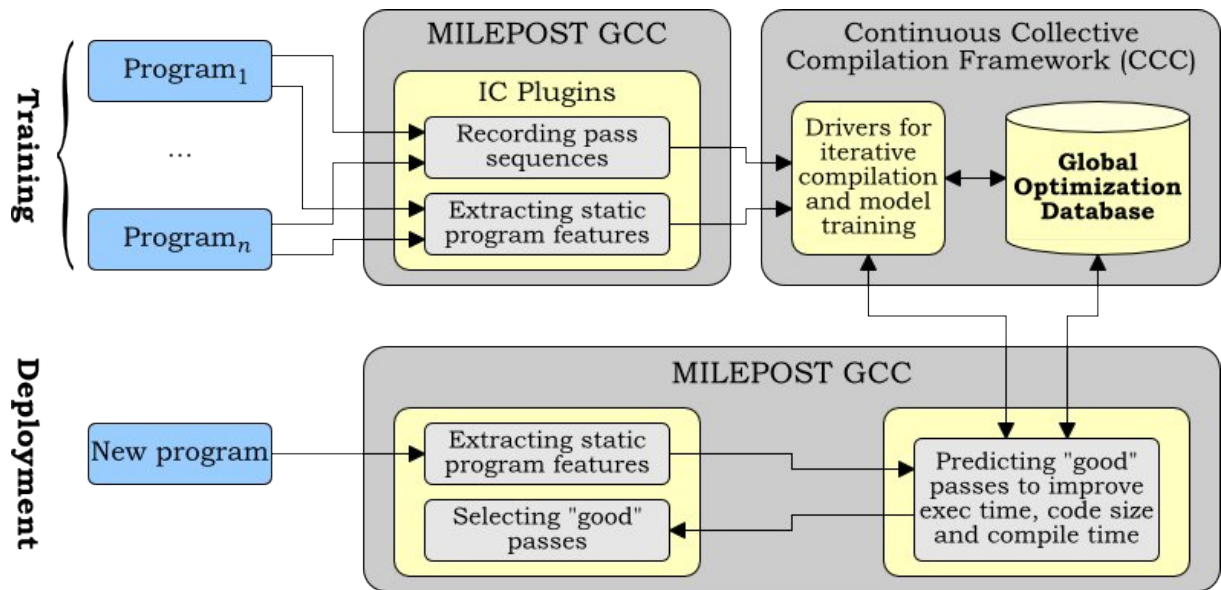
MILEPOST is based on the premise that programs which are similar will benefit from using the same set of optimization options. All that is needed is to “learn” the characteristics of programs and associate them with the correct set of optimization passes.

MILEPOST provides this infrastructure. It provides a database to hold information “learned” about which options work best with which types of programs, and uses this to select the optimization passes to be used in the compiler.

The database is first trained using a representative set of programs, which are compiled many times with different options and their performance measured, just as with iterative compilation. Standard machine learning techniques are used to construct a probabilistic model, from which predictions for new (unseen) programs can be made.

Two pieces of infrastructure are needed. The *continuous collective compilation* (CCC) framework facilitates the training of the database by running many programs with many options. The *interactive compilation interface* (ICI) allows the database to control the selection of optimizations passes in the compiler back end.

The approach is designed to be generic. However the first implementation has been an integration within GCC. The following diagram illustrates the approach.



After Fursin et al, 2008

MILEPOST GCC 4.4 was released in June 2009, with support for AMD Athlon 64, Intel Xeon, Itanium and ARC 725 processors. Early results<sup>1</sup> showed that average performance benefits ranged from 11% to 40% for the different processors, with the best programs more than doubling in speed. That is comparable with the results from iterative compilation, but with a compiler that is as easy and quick to use as standard GCC.

### Collective Tuning and Recent Work on MILEPOST

The Collective Tuning website ([www.ctuning.org](http://www.ctuning.org)) was established as a repository for continuing research on this approach to compilation. Since the completion of the original project, it has acted as the repository for the MILEPOST technology.

Over the past year, Jörn Rennecke of Embecosm has been working with INRIA to update MILEPOST, supported by HiPEAC. This has been addressing a key outstanding issue with the project.

MILEPOST was developed as a generic solution, to work with many different compilers, although eventually only GCC was fully supported. It required a plug-in infrastructure to allow robust development of the ICI and CCC.

This work occurred at the same time as GCC was exploring its own options for a plugin infrastructure. In the end, the approach adopted by GCC was radically different from that used by MILEPOST. Thus MILEPOST GCC 4.4 requires substantial modifications to the heart of mainline GCC.

For MILEPOST to be widely adopted, we need it to use standard features of compilers such as GCC. This will involve both changes to MILEPOST and changes to standard GCC.

We considered using the GCC plugin approach, but it lacks many of the necessary features, and we feel it will be difficult to get the changes required into the mainline code.

However the GCC *target hook* infrastructure, seems to offer features that are just as suitable for MILEPOST. At present adding a new target hook is a very laborious task, so GCC needs to be updated to make target hook implementation simpler.

Jörn Rennecke achieved two goals during the past year.

<sup>1</sup> Fursin et al. *MILEPOST GCC: machine learning based research compiler*. GCC Summit, Ottawa, Canada, 2008.



1. MILEPOST GCC was updated to GCC 4.5.
2. The basic improvements to the target hook infrastructure were accepted into GCC mainline.

### **The Future: MILEPOST II**

The target hook improvements to GCC are only a first step. For MILEPOST to be fully compatible, mainline GCC needs a robust multi-target infrastructure. This in turn requires changes that will affect the 30+ architectures that are incorporated within mainline GCC. Having achieved all this, we must modify MILEPOST to use the new infrastructure.

However there is a second, completely different area, that requires work if MILEPOST GCC is to become widely used. It is currently a research project. To set up a MILEPOST system is complex, it relies on a single learning database running at INRIA, reporting errors requires access to the full database. Most importantly there is no tutorial material for beginners and no guide for those wishing to port to other architectures.

We must make MILEPOST as easy to install as standard GCC. We must allow users to build their own databases (the database provides an insight into the code you are developing, which could be valuable to competitors). We must allow bugs to be reported, without requiring the entire database to be shipped. Finally we must provide tutorial material and porting documentation so adoption is easy.

We have outlined a three phase project to achieve these goals.

1. Phase One will develop the new infrastructure needed in GCC and modify MILEPOST to use it. This is a six month full-time effort, which may be spread over a slightly longer period to accommodate the needs of the partners.
2. Phase Two will push those changes through the GCC adoption process. This is a part-time effort taking up to two years.
3. Phase Three will run concurrently with Phases One and Two making the modifications to MILEPOST to use private databases and report errors simply, creating the tutorial and porting documentation and demonstrating their effectiveness by porting MILEPOST to new targets. This is a six month effort, but spread over a year.

This is a much smaller project than the original collaboration. Embecosm is able to provide the resource to do the work, and will underwrite a proportion of the cost of that resource. However we need a number of technology partners, processor manufacturers who will support the project financially.

We need a *lead technology partner*, whose processor will act as the reference processor throughout all the tutorial material and documentation. We expect MILEPOST GCC to offer the fastest compiled code of any general compiler available, and be widely used. So it will be useful if this reference processor is one that is widely known. The investment required would be up to €70k over the 2-3 year lifespan of the project.

We will also need a number of other *technology partners* during Phase Three, whose processors will be used to demonstrate the portability of the technology. The investment required would be up to €30k during Phase Three of the project.

We are in discussions with HiPEAC and INRIA over support for this work. Some form of match funding would allow us to reduce the contribution needed from the partners.

A formal announcement about this project will be circulated through HiPEAC and other channels in the coming weeks. In the meantime, companies and other organizations may express their interest by contacting Dr Jeremy Bennett at Embecosm, [jeremy.bennett@embecosm.com](mailto:jeremy.bennett@embecosm.com).



## About the Author

**Dr Jeremy Bennett** is Chief Executive of Embecosm Limited. Embecosm ([www.embecosm.com](http://www.embecosm.com)) provides open source services, tools and models to facilitate embedded software development with complex systems-on-chip. Contact him at [jeremy.bennett@embecosm.com](mailto:jeremy.bennett@embecosm.com).

This work is licensed under the Creative Commons Attribution 2.0 UK: England & Wales License. To view a copy of this license, visit [creativecommons.org/licenses/by/2.0/uk/](http://creativecommons.org/licenses/by/2.0/uk/) or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

This license means you are free:

- to copy, distribute, display, and perform the work
- to make derivative works

under the following conditions:

- *Attribution.* You must give the original author, Jeremy Bennett, credit;
- For any reuse or distribution, you must make clear to others the license terms of this work;
- Any of these conditions can be waived if you get permission from the copyright holder, Embecosm; and
- Nothing in this license impairs or restricts the author's moral rights.