Author	Presentation Title	Presentation Abstract
~~Alan Modra~~	~~Maintainer vs Submitter BoF~~	CANCELLED
~~Alan Modra~~	~~Porting gold to PowerPC~~	CANCELLED
Andrew MacLeod	GCC re-architecture BOF	Discussion of a plan to move towards a better modularized GCC source base. This involves creating a proper API between all front-end and back-end communications. The most noticeable aspect of this will be formalizing gimple and providing wrapper functions for all tree accesses in the middle/back-end so they can be weaned off of trees. The proposal document will be available on the mailing lists before the Cauldron.
Benjamin Kosnik	GCC/ABI BoF	Talk about the C++ ABI defaults for 4.9 and or any transition plans
Roland McGrath	GNU C Library BoF	The GNU C Library is used as the C library in the GNU systems and most systems with the Linux kernel. The library is primarily designed to be a portable and high performance C library.It follows all relevant standards including ISO C11 and POSIX.1-2008. It is also internationalized and has one of the most complete internationalization interfaces known. This BOF aims to bring together developers of other components that have dependencies on glibc and glibc developers to talk about the following topics: * Planning for glibc 2.19 and what work needs to be done between the August -> December 2013 timeframe (2.19 development phase). * Performance? - The project calls itself a "high performance C library", but we've never had a standard performance regression testsuite - How do we measure performance? - How do we track it? - What criteria do we use to evaluate patches based on performance? - Starting with libm e.g. accuracy vs. runtime performance. * libm - Offer three libraries? - Slow high performance, middle of the road, high performance low precision. * Tuning the C library? - What do we expose? - Why? - Hardware lock elision examples. - Runtime tuning. - Environment variables. * POSIX conformance - The usual. * ISO C11, and C++11 - What's left? * IPv4 and IPv6 - getaddrinfo mess.
David Edelsohn	GNU Toolchain ecosystem on AIX	This talk will discuss the status of the GNU Toolchain on AIX and the improving resources for developers to maintain AIX configurations of Free and Open Source Software projects.
David Edelsohn	Steering Committee / Release Managers BoF	
Dehao Chen	AutoFDO: use sample based profile to drive feedback directed optimizations	In traditional FDO, instrumentation is used to collect profile. Because instrumented binary is slow, it cannot be deploy in production. Also the profile is tightly coupled with compiler IR thus is not good at tolerating source changes. We designed and implemented AutoFDO to mitigate these problems. We use sampling based approach to collect profile, and use debug info to represent the profile. In this presentation, we will introduce the design, implementation and performance tuning of AutoFDO, and lessons we have learned while making it usable in real production.

Author	Presentation Title	Presentation Abstract
Dmitry Melnik	Using the Tool for Automatic Compiler Tuning (TACT) for GCC Development	Given the complexity of GCC, its long development history, the number of supported target platforms and optimizations, its performance usually can be improved by tuning compiler optimization parameters for the specific target platform. We present a tool that automatically finds optimal GCC optimization flags for a given application, and most importantly, helps tracing the achieved performance improvement to exact GCC optimizations (including the cases when default in -O2 optimizations should be disabled with -fno-... flags). Tuning GCC for ARM on several popular applications has shown 15-35% speedup compared to -O2, and that approximately the half of GCC flags that caused speedup actually were -fno-... and --param flags. Analyzing assembly output for each of the resulting flags and comparing it with that for -O2 has helped us diagnosing performance problems with default -O2 optimizations, and develop fixes for some of them. We give an overview of the tool, the tuning results for ARM, and show how it can help in development of GCC. The main features of the tool are: - Primarily designed for tuning GCC on ARM, but can be used for any platform and compiler; - Uses Genetic Algorithm to search optimization space; - Supports parallel compilation and running on several devices, including cross-compilation; - Supports profile-guided compilation (the phases are interleaved so the compilation with other flags executed while the training phase is running); - After tuning, the tool shows 5-15 most significant options (including -fno-...) that contribute the most to the improvement, sorted according to their impact. Then, the developer can compare assembly dumps with and without each option and trace a problem to certain GCC optimization pass; - Supports multi-objective tuning (e.g. at once for performance and size, finds Pareto optimal solutions); - Reliability: checks for miscompilations, ICEs, hangups, whether CFLAGS are passed correctly through makefiles to GCC, etc; - Provides templates with user scripts and config files for placing arbitrary applications, and scripts for importing SPEC2K benchmarks; - Can give results quickly (it usually takes 1-2 days to tune one SPEC2K application with "train" data), and shows the exact list of GCC optimizations for further manual investigation.
Dmitry Melnik	Developing Interblock Combine Pass in GCC	GCC combine pass tries to mathematically substitute expression values previously set for the registers into subsequent instructions that refer to that registers, if legal CPU instruction for such substitution exists. However, the current GCC combine implementation has two limitations. First, it doesn't support substituting instructions across basic blocks. Second, it requires that destination register of the producer should not be live after consumer instruction, otherwise combine tries to issue parallel instruction that contains both producer and substituted consumer patterns, but in most cases it fails. The most notable motivation example is inability of combine optimization to substitute shift into multiple ARM ALU instructions that can have "free" shift operation on their third argument (like "add r1, r2, r3 lsl #2"). We propose new interblock combine pass that doesn't have the above limitations. We use GCC dataflow framework to find candidate instructions for combining and verify correctness of the transformation, and try to reuse parts of original GCC combiner to perform the substitution and select CPU instruction. The optimization is still work-in-progress. Currently, it decreases code size of SPEC2K INT by 0.5% for ARM, if it runs after GCC's original combine pass. We're now looking into the cases that are handled better by original combine, and fixing them within the new interblock optimization. Also, we need to test it on other major platforms. Depending on the final results of interblock combine, it may worth separate presentation, or we can briefly talk about combine after presentation about the tuning tool (this optimization actually followed from the automatic tuning results, when on crafty it shown that -fno-gcse helps getting shifts inside add instructions).
Dodji Seketeli	Address and Thread Sanitizer in GCC: State of the Onion	A the previous GNU Cauldron there was a call for help from the initial Address and Thread Sanitizer authors to get this tools included in GCC. A year later some substantial progress have been made. This talk will present the current landscape of this project as of GCC 4.8 and explore the future potential developments.
Gary Funck	GNU UPC	A quick overview of UPC and a description of the architecture of the GNU UPC compiler and runtime. With sufficient time, we could also provide a demo.
Hui Zhu	KGTP, a "bayonet" for GDB on Linux Kernel	KGTP is a flexible , lightweight and realtime Linux debugger and tracer. It makes Linux Kernel supply a GDB remote debug interface. Then GDB in current machine or remote machine can debug and trace Linux kernel and user space program through GDB tracepoint and some other functions without stopping the Linux Kernel. I will do some show about use KGTP debug Linux without stop it.

Author	Presentation Title	Presentation Abstract
James Pallister	The Impact of Different Compiler Options on Energy Consumption	This talk describes an extensive study into how compiler optimization affects the energy usage of benchmarks on different platforms. We use an fractional factorial design to explore the energy consumption of 87 optimizations GCC performs when compiling 10 benchmarks for five different embedded platforms. Hardware power measurements on each platform are taken to ensure all architectural effects on the energy are captured and that no information is lost due to inaccurate or incomplete models. We find that in the majority of cases execution time and energy consumption are highly correlated, but the effect a particular optimization may have is non-trivial due to its interactions with other optimizations. There is no one optimization that is universally positive for run-time or energy consumption, as the structure of the benchmark heavily influences the optimization's effectiveness. This talk presents the results and conclusions gain from the project we introduced last year at the previous GNU Tools Cauldron.
Jan Hubicka	LTO/IPA BOF	
Jan Hubicka	State of interprocedural optimizers in GCC	In the talk I overview current state of interprocedural optimizers in GCC and present benchmark results on their effectivity. I will explain the new implementation of inline predicates driving inliner's heuristic and explain individual inline hints. I will also review other optimization passes (pure-const, ipa-reference, profile propagation).
Jeff Law	Graphite BOF	BOF session on graphite and its future in GCC
Jeremy Bennett	MAGEEC: MAchine Guided Energy Efficient Compilation	At GNU Tools Cauldron last year, we introduced a new research project with the University of Bristol to measure the impact of compiler options on the energy consumption of compiled code. James Pallister is presenting the results of that work in a separate talk, but in summary it demonstrated that compiler options make a big difference. However for any particular piece of code, it proves hard to choose which optimizations to use, and in which order to use them. It seems that only a machine learning approach, such as that of the earlier MILEPOST project, will be suitable. The UK government has agreed to fund a joint project, MAGEEC, over 18 months between Embecosm and University of Bristol to develop a prototype open source machine learning infrastructure for compilers, aimed at minimizing energy consumption of compiled code. This will build on the work of MILEPOST, but aims to be much more general purpose. In particular it should be capable of working with any release of GCC (MILEPOST was tightly integrated to particular releases) and indeed other open source compilers. In this talk we'll outline the initial work on the project, and seek suggestions from the wider GCC community about the approach we are taking. We look forward to reporting on progress at future meetings.
Markus T Metzger	Integrating Data Race Detection into GDB	We integrated the Intel(R) compiler's data race detector into GDB with the intent of being able to debug data race bugs interactively as part of a normal debugging session. Data race detection induces a huge performance overhead. To make interactive debugging feasible, we allow the scope of the data race analysis to be restricted by the user. We used GDB's python interface to add new commands for configuring the data race analysis and to communicate with the analyzer library, which is linked to the debuggee. The presentation will describe the data race detector and the technique we used to restrict the analysis scope as well as our experiences with GDB's python interface and its limitations.
~~Martin Jambor~~	~~Overview of current IPA-CP and IPA-PROP capabilities~~	CANCELLED
David Edelsohn	PowerPC BOF	Title: PowerPC BOF Authors: Michael Meissner & Alan Modra Abstract: We will cover various things that we have done and learned in the last year in the PowerPC arena. We will cover porting GCC to cover new instructions in future generations of the machine. Well will discuss various issues we have dealt with successfully at the compiler level, as well as current roadblocks that we are still trying to address. We will welcome contributions from other people working the the powerpc space to find ways to solve current problems.

Author	Presentation Title	Presentation Abstract
Niranjan Hasabnis	GCC-plugin for light-weight bounds checking	Title: GCC-plugin for light-weight bounds checking Abstract: Buffer overflow, a sub-class of more general problem called as out-of-bound access error, is very old but still very prevalent security problem. Although the research community has proposed a number of solutions to this problem, very few are deployed within production compilers. In the context of GCC, Mudflap is a popular plug-in for detecting buffer overflows, but it has high overheads, and moreover, does not detect bounds errors that occur due to crossing of object boundaries. High overheads make it hard to deploy Mudflap as a runtime defence. We have developed LBC (Light-weight Bounds Checker) that addresses the drawbacks of Mudflap. LBC inserts guard zones between objects, thereby enabling it to detect overflows missed by Mudflap. LBC uses an innovative metadata representation that avoids additional memory accesses in the common case, thus yielding performance that is much closer to normal performance. As a result of this representation, LBC's overhead is typically a quarter of that of Mudflap, even though LBC detects a larger class of errors than Mudflap. We have already implemented LBC (available at http://www.seclab.cs.sunysb.edu/lbc/) using a C-programming language transformation framework called CIL [IEEE/ACM Code Generation and Optimization, 2012]. However, since CIL is not widely deployed, this implementation of LBC is not readily available to the wider community of C and C++ programmers. We have therefore developed a new design of LBC as a plug-in for GCC. By operating on the GIMPLE, LBC supports all input languages and target architectures that GCC supports. Using our current implementation of the plug-in, we can compile and produce SPEC 2000 and 2006 benchmark results. We plan to test the plug-in on complex packages and release it to the open-source community before the summit.
Olga Kupriianova	Metalibm	Metalibm As the current libm implementation provides a set of mathematical functions implementations in a limited set of precisions, it is actually a compromise between performance and accuracy. The current library is not flexible enough for the users needs. For example, some unnecessary steps are provided for computing the log(x) on [1,5] with only 4 fractional digits. The libm was written by different teams and contains some 20-year old codes, which make it hardly maintainable. Despite of its prevalence, the libm does not fully perform correct rounding and the execution time is not bounded. In order to avoid all these disadvantages the libm has to be rewritten. As it takes about 1 man-month to implement a function (and there is about 70 of them) we propose the metalibm prototype: an automatic parametrized code generator. The metalibm produces proven codes on-the-fly taking into account the declared accuracy, precision, domain, etc. We aim to integrate the codes to the existing glibc (compilers), as well as provide vectorizable code. The subjects for the following scientific research are the efficient evaluation of composite functions and the implementation of some "exotic" functions (i.e. Gamma-, Bessel-functions).
Ramana Radhakrishnan	ARM/AArch64 BoF	
Torvald Riegel	Accelerator BoF	In this BOF, we will discuss how we can support accelerators (e.g., GPGPUs) in GCC. This is a rather wide problem and raises questions ranging from which programming abstractions to support (e.g., OpenMP 4.0 accelerator extensions, OpenACC, and others) to which accelerator (virtual) ISAs to support -- and everything in between. Therefore, the primary goal of this BOF is to start the discussion about how to approach this problem and which initial steps we might take. Possible sub-topics for discussion include: * Front-end issues: Which programming abstractions for parallelization on accelerators at the programming-language level do we target first? OpenMP 4.0 will include accelerator abstractions; OpenACC has similar features. Are there features that are shared across several programming abstractions (e.g., ways to put constraints on regions of code so that it can indeed be executed on a particular accelerator)? * Which infrastructure do we need in the middle-end to support different language-level abstractions without a lot of duplicated code? Which GCC-internal abstractions do we need? Can we have a single shared internal representation of accelerator code? How do we deal with issues such as heterogeneous accelerator hardware or separate address spaces on accelerators? How do we represent communication between hosts and accelerators? Is the current SIMD support sufficient for accelerator code too? * Back-end questions: Which (virtual) architectures do we want to target first? Should we target low-level programming abstractions such as OpenCL, too? Which additional runtimes do we want to -- or need to -- bind to? Can/should we target existing runtimes, or do we need to build our own (glue) layers? * Testing: Do we need anything special to test all of this? * Other GNU tools: What do we need to do in other tools such as GDB?

Author	Presentation Title	Presentation Abstract
Vladimir Makarov	GCC Register Allocation BOF	Informal discussion about progress of switching GCC targets to LRA, general RA problems (IRA, LRA, regmove, dealing with register pressure) and their possible solutions.
Yao Qi	Port GDB To A New Processor Architecture: TI C6X	GNU Debugger has been ported to dozens of architectures and it has a clear interface for architecture-specific code. However it is still not easy to port GDB to a new architecture, because the sequence of porting and the interactions between different parts are unclear to engineers. This tutorials is to describe the steps of porting GDB 7.4 to a new architecture TI C6X and the interactions between architecture-specific parts and common parts in GDB. This presentation includes adding breakpoint, software single step, and prologue analysis for the new port. Finally, we will show the porting work needed for ucLinux, such as handling PLT stub and signal trampoline unwinding. This tutorial is beneficial to engineers who are to port GDB to their own architecture, but also useful to GDB developers to understand GDB deeply.