Leo Korinth
leo.korinth@gmail.com
https://lkorinth.github.io
Löjtnantsgatan 3
115 50 STOCKHOLM

Principal Member of Technical Staff at Oracle

2017–2026 C++

Garbage Collection developer and reviewer on Open JDK. Reworked *all* parallelisation code in ParallelGC to use the same parallelisation abstractions used in the other GCs, with *significant* performance improvements and reduction in code. Removed ParallelGC centric code with GC generic oop_iterate code improving maintainability. Simplified the reference processing parallelization framework.

Improved the make system with regards to incremental warning improvements, test selection and test runner bugs. Improved GC CI code to not run duplicate tasks and also to be able to run subsets of tests that use different test cases and VM options. This improvement meant that we could easily reduce testing time by CPU weeks for many common test tasks.

Removed several VM flags; removed the Concurrent Mark and Sweep collector.

Engineering Consultant at Altran

2015–2017 Java

Developed job-matching software for the Swedish Public Employment Service.

2013–2014 Java

Adjusted lost-property software to nationwide use after reorganisation of the Swedish Police.

2013–2013 C

Created a cross-platform eye tracker library for Linux and Windows for Tobii. Transferred high bandwidth/low latency image streams created from eye tracking hardware over USB3.

Software developer at Ericsson

2011–2013 C FPL

Developed a Border Gateway Function — a kind of firewall. Programmed C on PowerPC with OSE as OS, and low-level programming of accelerated network processor from LSI.

2011–2011 C++

Developed *Component Based Architecture*, a high availability platform on top of Linux. CBA is based on OpenSAF which is standardised by Service Availability Forum.

2010–2011 C++

I was part of the product and knowledge transfer of AXE's regional processor platform from USA to Europe and Asia. The platform is based on OSE on top of PowerPC.

2008–2010 C++

I maintained and optimised the JIT compiler of the AXE platform. The compiler translated machine code of the central processor to machine code of modern CPU architectures. The JIT compiler is the base of the modern AXE telephone exchanges made by Ericsson.

2007–2008 Java

Developed Ericsson's IPTV system in JavaEE.

2004–2007 Java C++

Developed Ericsson NRG (an OSA/Parlay gateway). Built on the TSP platform with soft real time characteristics, CORBA IPC and replicated in-memory database.

Education at Royal Institute of Technology, Sweden

1998–2003 C Fortran Java Scheme Matlab Maple

Master of Science in Engineering, Degree Programme in Computer Science and Engineering.

I have specialised in computer architecture and theoretical computer science; thus chosen courses in the areas of algorithms, computer architecture, parallel computer architecture, operating systems and theoretical computer science.

In connection with my education, I also worked for Framfab 1999 – 2000, BanqIT in 2000 and Lorentzen & Wettre in 2004 before starting at Ericsson.

Interests

Computer Science

I am interested in algorithms, optimisation and different programming paradigms; I especially appreciate functional programming and static typing.

Low-level

Computer architecture is fun! That is, processors, memory subsystems, compilers, operating systems and other low-level related topics.

Engineering

I would love to work with advanced technologies, preferably in connection with physics, biology or medicine. I do appreciate working with complex problems and trying to find simple, elegant solutions.

Leo Korinth
leo.korinth@gmail.com
https://lkorinth.github.io
Löjtnantsgatan 3
115 50 STOCKHOLM

Computer Languages

C

Computer hardware interests me and I have *very good* knowledge of C. I know about trickier areas such as memory padding, restrict-qualifiers, sequence points, aliasing and what kind of things are specific to the system ABI or processor. I know how C maps to object files and machine code, a basic knowledge of ELF and DWARF, and how to inspect object files with nm and objdump, how to rename symbols with objcopy, using linker scripts etc. I also have knowledge of using profilers such as OProfile and gprof (how they are implemented, and when to use which), debuggers such as gdb and thread-and-memory debugging tools such as valgrind. I have used C on many operating systems such as many Unixes, OSE, TSP, and even Windows. I have also worked with many proprietary C compilers on Solaris, AIX, Tru64, as well as Microsoft's C/C++ compilers. I have most experience working with the excellent GNU tools: GCC and Binutils.

C++

I am less knowledgeable about C++, though I have more than a decade professional experience using it. I have a good understanding of how the language maps to C (name mangling, virtual dispatch with v-tables, global constructors and to a lesser degree exception handling using EH frames). I still constantly have to check reference material and I still learn new stuff all the time. I lack much experience of the STL, though I do have a good understanding of the container types and the algo part. The reason for this is that much of my C++ experience has been from APZVM and OpenJDK, disallowing the use of STL and exceptions.

I like to stay updated, and have a basic understanding of the new stuff in modern"C++ (move semantics, type inference, lambdas, the new loop, enum classes etc).

Java

As with C, I am a very good Java programmer with a good understanding of the core language. I have experience of JavaEE, mostly on JBoss. I have worked 9 years on OpenJDK which gives me some different perspectives on Java than most *users* of Java.

Scala

I am a beginner in Scala, and I know only the basics. I have taken the Coursera course *Functional Programming Principles in Scala*.

Haskell

Although I am a novice Haskell programmer, Haskell is my favourite language by far. It has all *I want* from a high level language — strong static typing with great type inference, purely functional with equational reasoning and a concise syntax. I annoyingly like to talk about how great Haskell is. I have taken two massive online courses in Haskell.

Scheme

Maybe the most elegant language in that Scheme is so simple, yet so powerful. I do find s-expressions beautiful and although I despise dynamic typing, it does enable a very simple syntax. I have only basic skills (some university studies and a 3D modeller I started that used guile as extension language). *Structure and Interpretation of Computer Programs* might be my favourite programming book.

Other

I have a basic understanding of many languages to a lesser degree. I have done smaller things in C#, Python, Visual Basic, shell script and also done some numeric analyses in Matlab and even touched Maple in a mechanics class. The types of languages that challenge my thinking are functional and logical languages; I never did learn to think in Prolog though I find the language most beautiful, in the same way I fancy Make.

Operating systems

Unix-like

I have a very good understanding of Unix and I have used many different brands of Unix on most CPU architectures, and even created meta-distribution using Yocto. At home, I used to use Debian GNU/Linux (and Hurd!), I now use GNU Guix. I feel at home with Unix and I love it as a development platform as well as my home platform and media centre. I have only basic character device kernel hacking skills though I have looked a bit into other subsystems.

OSE Delta
Dicos
Windows

I have done network programming in C, in kernel space, for OSE Delta. I have developed Java and C++ for Dicos. I have experience of Java, C#, C++ and VB on Windows.

Languages

Languages

Native Swedish, advanced English and very basic Spanish.