Making Fortran Legacy Code More Functional

Using the BGS* Geomagnetic Field Modelling System as an Example

Hans-Nikolai Vießmann
Heriot-Watt University
hv15@hw.ac.uk

Sven-Bodo Scholz
Heriot-Watt University
s.scholz@hw.ac.uk

Artjoms Šinkarovs
Heriot-Watt University
a.sinkarovs@hw.ac.uk

Brian Bainbridge
British Geological Survey
bba@bgs.ac.uk

Brian Hamilton
British Geological Survey
bham@bgs.ac.uk

Simon Flower
British Geological Survey
smf@bgs.ac.uk

ABSTRACT

This paper presents an application case study of the British Geological Survey's (BGS) Geomagnetic Field Modelling System code. The program consists of roughly 20 000 lines of highly-tuned FORTRAN MPI code that has a runtime of about 12 hours for a signal execution cycle on a cluster utilising approximately 100 CPU cores. The program contains a sequential bottleneck that executes on a single node of the cluster and takes up to 50% of the overall runtime. We describe an experiment in which we rewrote the bottleneck FORTRAN code in SAC, to make use of auto-parallelisation provided by the SAC compiler. The paper also presents an implementation of a foreign-function interface, to link the SAC kernel with the FORTRAN application. Our initial performance measurements compare the SAC kernel performance with the FORTRAN bottleneck code; we also present results using an OPENMP Fortran implementation. Our figures show that the SAC-based implementation achieves roughly a 12.5% runtime improvement, and outperforms the OPENMP implementation.

CCS Concepts

•**Applied computing** → *Environmental sciences;* Mathematics and statistics; •**Computing methodologies** → *Parallel programming languages;* •**Software and its engineering** → *General programming languages; Compilers;* •**Computer systems organization** → Single instruction, multiple data; Multicore architectures;

Keywords

Fortran; Single-Assignment C; High-Performance Computing; Functional Programming; Eigensystem; Foreign Function Interface;

*British Geological Survey — is the world's oldest geological survey. A member of the United Kingdom's public funding body, the Natural Environment Research Council, the BGS is the custodian of much of the country's geoscientific information as well as a source of geoscience expertise. More information can be found at www.bgs.ac.uk.

IFL '15, September 14-16, 2015, Koblenz, Germany

DOI: http://dx.doi.org/10.1145/2897336.2897348

1. INTRODUCTION

The lambda calculus' Church-Rosser property suggests that functional programming languages should offer opportunities for efficient parallel execution. Research over the last four decades has produced excellent results that demonstrate how this conceptual advantage can be turned into real speedups on parallel hardware [15, 18, 26, 27, 32, 35]. Despite these advances, functional programming has not yet made it into mainstream high-performance computing (HPC).

There are many reasons for this, but the key issue is that the HPC industry has invested heavily into imperative programming languages, such as legacy FORTRAN codes. In particular, FORTRAN has many features that make it the language of choice for performance-hungry applications. It has long been available, and its design is amenable to compiler optimisation. Most programs consist of procedures containing nested loop constructs that iterate over arrays. The use of FORTRAN for HPC applications and the design of the language, consequently, has driven research into advanced compiler optimisations which, in turn, has further improved the performance of generated code. Alongside the continuous improvement of FORTRAN compilers, hand-tuning of these imperative applications has given them highly competitive levels of performance. Given this scenario, it is not very surprising that other programming languages, including those from the functional domain, struggle to take hold in the HPC domain.

FORTRAN's limitations with respect to auto-parallelisation inspired the development of HPF (High-Performance FORTRAN), were alleviated by the manual adaptation of codes with explicit parallelism, using MPI, OPENMP, or both.

Several prominent examples show that compiler-generated codes generated from highly specialised, typically functional, domain-specific languages (DSLs), can outperform legacy codes, in both parallel and sequential settings. Examples of this approach are the Spiral project [30] for signal processing and the Diderot project [21] for image processing. While this is an indicator for the validity of the initial claim, i.e. the conceptual advantages of the functional setting for parallel executions, it only covers a few rather specific application areas. Furthermore, a DSL-based approach typically requires re-writing large parts of legacy code, if not entire applications.

In this paper, we investigate a different approach towards improving legacy FORTRAN HPC applications. Starting from a parallel, commercial FORTRAN code base we identify its sequential bottleneck, re-implement this bottleneck code in SAC and call the compiled SAC code from the FORTRAN context. The most important aspect of this approach is that it is applicable to a wide range of existing FORTRAN applications, without requiring either costly re-