DATA COMPRESSION LANGUAGE EXTENSIONS FOR C/C++

P. Radtke (pawel.k.radtke@durham.ac.uk), T. Weinzierl

Durham University

Introduction

Due to C/C++ language design and limitations, HPC applications suffer from:
- excessive memory usage
- boilerplate integration code, e.g. MPI mappings
- brittle, hand-crafted performance optimisations

State of the art

HPC applications often resort to manual performance optimisations for optimal performance. **Domain-Specific Languages (DSLs)** are employed to automate the process of writing error-prone, repetitive code.

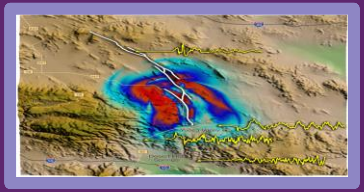
Data Compression Language Extensions (DCLE) for C/C++

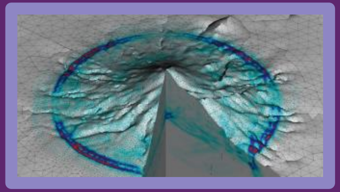
We propose a set of C/C++ language extensions – **#pragma directives and C++11 attributes** – which implement memory usage optimisation via **struct compression techniques at compiler level**, thereby eliminating the need for a fully-fledged DSL.

A user introduces the new attributes or #pragma directives to their new or existing project, and compiles their code using **our provided LLVM/Clang-based compiler** to get the optimised binary.

Where is DCLE used?

ExaHyPE (Weinzierl, 2019) is a high-performance engine for PDEs with applications in **seismology** and **astrophysics**.





From the ExaHyPE gallery (www.exahype.eu)

DCLE API

DCLE attributes work on **data structs and their fields**:

- **`clang::compress`** optimises fixed value-set type fields, e.g. bool or enum
- **`clang::compress_range(a,[b])`** optimises fixed value-range type fields, e.g. int or long
- **`clang::truncate_mantissa(a)`** optimises floating point type fields, e.g. float, double

```
struct Data {
    [[clang::compress]] bool b;
    [[clang::compress_range(1024)]] int cells[16];
    [[clang::compress_truncate_mantissa(12)]] double d;
}
```

The semantics of the code do not change if a DCLE-unaware compiler is used.

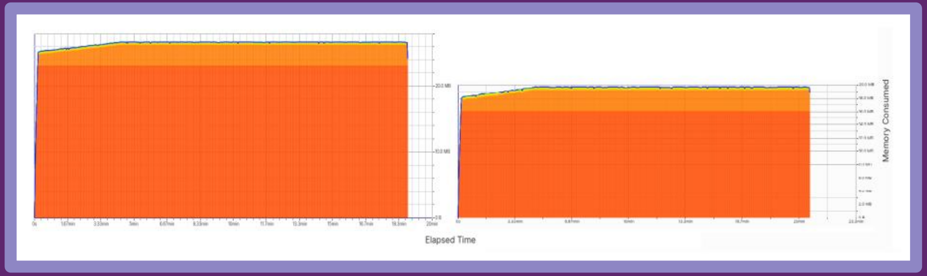
DCLE in ExaHyPE

Objective: **reduce the memory footprint of a numerical simulation without sacrificing performance**

Setup: ExaHyPE-based smoothed-particle hydrodynamics (SPH) simulation with DCLE attributes applied

During compilation, our DCLE-aware compiler:
- Produces optimised implementations of the compressed data structures
- Substitutes references to the original data structures with the optimised counterparts
- Does so transparently to the user and with negligible increase of the compilation times

(Very) early results
- **Memory consumption reduced by 30%**
- **Impact on runtime performance ≤ 5%**



Memory consumption over time before (left) and after (right) applying the DCLE attributes. Dark orange represents the memory occupied by the data structures subject to optimisation

Compression methods

The approach to compression changes depending on the datatype:
- For datatypes with **finite value sets**, such as booleans or enumeration types, we **automatically inter** the amount of information stored
- For **integer** datatypes, users can specify **upper/lower bounds** which serve as basis for compression
- For **floating point** datatypes, users can specify how many **bits of precision** should be preserved

Compressed bits are stored into and retrieved from a constant-width array of bytes, which becomes the only field in the optimised data structure.

Discussion

DCLE-based compression works most optimally if the optimised data structures occupy the majority of a program's memory footprint.

The impact on runtime strongly **depends on the memory-boundedness** or the program, as well as **access patterns** to the compressed values.

Areas for future development: automatic **integration with MPI** and MPI-based nonpersistent loadbalancing libraries (Samfass et al.,2020) and large-scale testing.

References

[1] Tobias Weinzierl. *The Peano Software - Parallel, Automation-Based, Dynamically Adaptive Grid Traversals.* TOMS, 45(2), 2019

[2] Samfass et al. *Lightweight task offloading exploiting MPI wait times for parallel adaptive mesh refinement.* Concurrency and computation: practice and experience, 32(24), 2020.