Automating Data Layout Conversion in a Large Cosmological Simulation Code

Michele MARTONE, Luigi IAPICHINO, Nicolay HAMMER, Julia LAWALL*
Leibniz Supercomputing Centre, Garching bei München, Germany
*INRIA/Laboratoire Informatique Paris-6, Paris, France

Intro: The Gadget cosmological code

- Large-scale structure formation (galaxies and clusters)
- Widely available cosmological TreePM N-body - SPH code
- MPI/OpenMP-parallel, up to O(10k) Xeon cores [SuperMUCLRZ]
- Several teams and several versions [-200 kLoC each]

Figure 1: Galaxies simulated with Gadget – courtesy Magnetum

http://magneticum.org

Optimization study of Gkernel

- Gkernel is:
  - isolated representative Gadget code kernel (‘halo finder’)
  - stand-alone application, avoids simulation overhead
- Node-level optimization study [2] in the frame of the IPCC [I]
- Target computing systems:
  - Knights Corner (KNC), Ivy Bridge (IBV)
  - Haswell (HSW), Broadwell (BDW)
  - Knights Landing (KNL)
- Main changes:
  - Data layout optimization
  - Better threading parallelism

Figure 2: Tests on one-socket Xeon systems: 240 threads (4 thr./node) for KNC, 128 threads (2 thr./node) for KNL. Performance improvement: up to 10x faster on XENL, 13.6x on KNC, ca. for 5S-5h on Xeon.

Figure 3: A structure of Arrays: struct definition introduction example. Only a dozen arrays have been introduced this way in Gkernel. Gadget would need over a hundred.

Perspective for Gadget

- Obtained and validated performance guidelines for Gadget
  - Limitations of Gstudy:
    - single kernel only, no MPI
    - relying on AoS => SoA at each kernel call
  - Left open questions:
    - what to do with members like Pos[3] or DV[3][3]?
    - ... and anonymous unions?
    - ... and other kernels/quantities?
  - How to backport >200 kLoC to Gadget?

Discussion: Possible AoS => SoA breakdown

- DeSoaStructs: define struct type with arrays instead of scalars – each member might depend on #ifdef ...
- DeSoaFuncs: glue functions for SoA
  - like -allocate, -free, -convert, etc.
  - repetitive, time consuming

Figure 4: AoS to SoA transition sample. Impacts a few hundreds statements in Gkernel. Would impact ‘many thousands’ in full Gadget. Notice that 1 could not be any expression.

Solution: Semantic patching

Coccinelle [2] “...a program matching and transformation engine ... for specifying desired matches and transformations in C code”

- specification language inspired from UNIX patch (SmPL)
- SmPL can express steps in Fig. 5 enable Fig. 7 work style
- originally for collateral evolutions in the Linux kernel
- arbitrary C language transformations possible, model checking

Figure 5: Transformation steps, ordered so that their introduction does not break code. The Gkernel study has performed this on a subset of Gadget. [2]

- DecSoaFuncs: define SoA functions for SoA
- CallSoaFuncs: call glue functions
- DefSoaVars: declare SoA variables
- AosToSoa: replace relevant AoS accesses with SoA accesses
- thousands of lines impacted (error prone and repetitive)
- regular expressions help, but still error-prone and sloppy

Figure 7: Tool-assisted automatic data layout transformation would greatly accelerate cycles of development and empirical performance optimization.

A more desirable approach

- fine-grained AoS => SoA transition mechanism needed!
  - transitory patch: generating AoS => SoA converter functions
  - exclusion: inclusion of structure members based on
    - usage / occurrence in files
    - specific type
    - identifier whitelists

Perspective for Gadget

- Obtained and validated performance guidelines for Gadget
  - Limitations of Gstudy:
    - single kernel only, no MPI
    - relying on AoS => SoA at each kernel call
  - Left open questions:
    - what to do with members like Pos[3] or DV[3][3]?
    - ... and anonymous unions?
    - ... and other kernels/quantities?
  - How to backport >200 kLoC to Gadget?

Figure 6: DeSoaFuncs: Transition in SoA in Gadget would require one allocation per quantity. Manual introduction of this and further glue functions (e.g. AoS => SoA copy functions) is error-prone and repetitive. Manual SoA accesses introduction (Fig. 6) can be worse.

- DefSoaVars: declare SoA structure variables
- CallSoaFuncs: call glue functions
- AosToSoa: replace relevant AoS accesses with SoA accesses
- thousands of lines impacted (error prone and repetitive)
- regular expressions help, but still error-prone and sloppy

Figure 6: DefSoaFuncs: 4 rules for OpenMP optimizations backport.

- DecSoaFuncs: define SoA functions for SoA
- CallSoaFuncs: call glue functions
- DefSoaVars: declare SoA variables
- AosToSoa: replace relevant AoS accesses with SoA accesses
- thousands of lines impacted (error prone and repetitive)
- regular expressions help, but still error-prone and sloppy

Figure 8: Sample semantic patch for DeSoaStructs. As of Fig. 5 lines 2-3 specify semantic elements of the C language. Lines 5 and 9 specify matching context, line 8 what to add, line 7 what to remove.

References