Title

Proto-Crunch: EBProto Implementation of Crunch Reaction Network on GPUs



Authors

Trebotich, David

Paolini, Christopher

Publication Date

2023-12-13

Peer reviewed

# Proto-Crunch: EBProto Implementation of Crunch Reaction Network on GPUs

**Christopher Paolini, San Diego State University**

**David Trebotich, Lawrence Berkeley National Laboratory**

**November 2, 2020**

# Executive Summary

The ECP Subsurface application code development project uses a performance portability strategy for Chombo-Crunch based on the Proto portability and productivity layer. A new C++ version of CrunchFlow has been developed and invokable as a single point function call in EBProto::forall. Baseline CPU vs. GPU performance is reported.

# 1. Outline of Execution Plan

1. Port CrunchFlow Lite to GPUs
    a. Convert CrunchFlow Lite from Chombo Fortran to C++
    b. Use Proto::ShapeArray for vector length arrays
    c. Develop single point function call, invokable as device code in Proto::forall
    d. Implement Proto-Crunch driver and test
    e. Optimize Proto-Crunch on GPUs

# 2. TECHNICAL WORK SCOPE, APPROACH, RESULTS

### CrunchFlow GPU Portability

Chombo-Crunch couples flow and coservative transport solvers in Chombo with the geochemical reaction network in CrunchFlow. The interface was previously based on the Chombo FORTRAN DSL which seamlessly integrates CrunchFlow, a FORTRAN 90/77 code. As we are making use of the Proto C++ library for performance portability of Chombo-Crunch to GPUs, and in doing so replacing Chombo FORTRAN, we have converted CrunchFlow FORTRAN 90/77 code to C++. We have also reduced the Chombo-Crunch interface to a single point function call invokable as device code in Proto::forall for GPU portability. Several areas of development have occurred in this process:

- C++ implementation of CrunchFlow
    - The standard C language implementation uses a column-wise representation for all arrays to achieve storage agreement between C and FORTRAN to facilitate continued FORTRAN/C interoperability, and aid in backward compatibility for regression testing
        - We developed an offset library in C to reference array elements using FORTRAN-style indexing
        - We implemented a regression testing framework to compare simulation results between the C implementation and the original FORTRAN implementation.
    - The initial C implementation used variable-length arrays (VLAs), an ISO/IEC 9899:1999 (i.e. C99) standard, to pass multidimensional arrays to functions. This implementation has been committed to the LBNL Subversion code repository.
    - Use of VLAs promotes the object-oriented design principle of *data encapsulation* to avoid referencing arrays declared to have global scope, a liability found in the original FORTRAN 90/77 implementation
    - However, support for VLAs was not accepted for inclusion into the C++ standard. Many suggest using the std::vector mechanism in place of VLAs, however use of std::vector is not recommended for GPU acceleration. We therefore transitioned to using the available *Proto::ShapeArray* API, which permits a developer to use standard C array indexing syntax, given a pointer to the first element of a multidimensional array.
- Global data
    - Previous FORTRAN multidimensional arrays were global (in COMMON block storage) which presents non thread-safe code that may not be reentrant:
        - Data encapsulation: we refactored the original FORTRAN 90/77 implementation to remove the dependence on global storage (common blocks, module files) using VLA and data encapsulation. We encapsulated data within functions where

possible. For example, functions that solve a linear system will locally store a matrix *A* and RHS vector *b*.
        - Data hiding keeps data from being exposed and therefore vulnerable to unintentional *side effects*.
        - FORTRAN 90 module files were converted to prototype C++ header files
- Proto-Crunch
    - Developed single point function call to function *os3d_newton*, which implements a Newton-Raphson iterative method
    - invokable as device code in Proto::forall
    - Created standalone test code driver that isolates os3dNewton iterative solver in CrunchFlow
        - Break before os3dNewton in CrunchFlow to dump array values as initial values for Proto::Crunch test code
        - Hardcode 79 pointers to data using Proto::ShapeArray
            - call by reference, not by value
            - sum of all arguments is under 2^10 bytes: 1024/79=12.96 < 16
    - Converted to LinPack
        - CUDA optimized BLAS and LinPack (future)
    - Performance times:

| Optimization / Architecture | Host time [ns]<br>Intel Xeon E5-2620v4 2.10GHz | GPU time [ns]<br>Nvidia V100 PCIe 16GB |
|---|---|---|
| Baseline | 2,285,781 | 12,664,179 |
| matrix assembly on device | | |

    - Further optimizations
        - push matrix assembly onto device
        - push entire reaction network on device, pass device pointer to solution (future work)

## 3. CONCLUSIONS AND FUTURE WORK

The primary goal of future work would be to push the entire Crunch reaction network computation onto the GPU. A device pointer to the solution of reaction components would need to be passed into Crunch.

Another focus of future research pertaining to reaction geochemistry will be to accelerate CrunchFlow on the NVIDIA A100 architecture. We will also investigate how we can best make use of Fused Multiply-Add (FMA) operations on A100 Tensor Cores that can perform mixed 16-bit floating point (FP16) and 32-bit floating point (FP32) matrix operations on 4×4 block matrices in one clock cycle. We will explore how we can use the new TF32 precision provided by the A100, which is capable of 20x more FLOPS in codes that use FP32. SDSU is purchasing an nVidia DGX cluster with 8x A100 GPUs, and we will port the accelerated CrunchFlow code over to the A100 architecture and evaluate speedup. A second

focus of future research will be to use the newly released Xilinx Vitis Accelerated Math Libraries to accelerate general matrix multiply (GEMM) operations in ChomboCrunch on a Xilinx Alveo U280 (FPGA) data center accelerator card. We will acquire two U280s in November 2020 dedicated to this effort, funded through our recent NSF Office of Advanced Cyberinfrastructure (OAC) CC* Compute Grant 2019194 ($399,328) *CC* Compute: Central Computing with Advanced Implementation at San Diego State University*. The use of hardware accelerators will allow us to configure detailed kinetic mechanisms involving a greater number of silicate and carbonate minerals and aqueous electrolytes in formation water, and perform simulations at multiple timescales.