Dusk & Dawn -
Introduction and Overview

Summer School on Effective HPC for Climate and Weather
24.08.2020

Slides: Matthias Röthlin
Speaker: Giacomo Serafini
Contents of this Presentation

• A short history of dawn
• Current development efforts on dawn
  – Small detour: The ICON Model
  – (Some) Requirements on dawn to translate the ICON model
• dusk & dawn
  – Toolchain overview
  – Language Features
• Discussion & Conclusion
Dawn?

Dawn - Compiler toolchain to enable generation of high-level DSLs for geophysical fluid dynamics models
Dawn History

• In development since 2017
• Initially conceptualized for handling Finite Differences on Cartesian grids / the COSMO model
• Ships with a frontend called gtclang
  – embedded in C++
  – powerful enough to express all stencils in the COSMO dynamical core
• Wide array of optimization strategies
• Various backends
  – C++ (naive)
  – gridtools MC / GPU
  – cuda
Dawn History

• Dawn was used to successfully translate the complete COSMO dycore
  – advection schemes
  – diffusion
  – tridiagonal solver
  – ....

• Outperforms previous efforts of translating the COSMO dycore using DSLs, at a fraction of the lines of code
Dawn History

Paper

MeteoSwiss
Dawn Current Development Efforts

Cosmo Model is End of Life
  > Dawn needs to adapt to new models

• ICON Model
  – Hybrid Numerics on Icosahedral Triangular Mesh
  – Development efforts lead by MeteoSwiss

• FV3 Model
  – Finite Volumes on the cubed Sphere
  – Development efforts lead by Vulcan Inc.
The ICON Model - Overview

- ICOsahedral Non-Hydrostatic Model
- Joint development: Max Planck Institute (MPI) & Deutscher Wetterdienst (DWD)
- FORTRAN 90, ~370'000 lines (comments removed)
- Uses both Finite Differences as well as Finite Volumes in the Dycore
  - Finite Elements in (some configurations of) tracer flux
The ICON Model - Overview

ICON meshing procedure:

Example: 13km global resolution requires about $3 \cdot 10^6$ triangles
The ICON Model - Requirements for dawn

Stencils

\[
\text{lap}(i,j) = -4 \times u(i,j) + u(i-1,j) + u(i+1,j) + u(i,j-1) + u(i,j+1)
\]

Reductions

\[
\text{VertexField} \quad \text{lap}, \quad u \\
\text{lap} = \text{reduce} (\text{VERTEX} > \text{VERTEX} \quad u) \\
\text{lap} = \text{lap} - 4 \times u
\]
The ICON Model - Requirements for dawn

Stencils

Reductions

```plaintext
VertexField lapl, u
lapl = reduce( VERTEX>VERTEX u )
lapl = lapl - 6*u
```
The ICON Model - Requirements for dawn

Stencils

??

Reductions

```
VertexField target
CellField source
target =
  reduce(
    VERTEX>CELL>CELL, source)
```
dusk & dawn - Toolchain overview

High level View

- dusk
- dawn

- *.cpp
- gcc
- binary

- *.cu
- nvcc
- binary
dusk & dawn - Toolchain overview

High level View

- Frontend: dusk
- Compiler: dawn

ForCPP files:
- dusk -> *.cpp -> gcc -> binary

ForCU files:
- dusk -> *.cu -> nvcc -> binary
dusk & dawn - Toolchain overview

Closer View

Frontend

dusk

SIR

dawn-opt

Compiler

IIR

dawn-codegen

*.cpp

gcc

binary

*.cu

nvcc

binary
Fields with location type

#dusk code

edge_length: Field[Edge]
cell_area: Field[Cell]
node_id: Field[Vertex]

//C++ Code

float_t edge_length[numK][numEdges]
float_t cell_area[numK][numCells]
float_t node_id[numK][numVertices]

- Reserved keywords: Field, Edge, Cell, Vertex
- Fields span the full domain
- Entries are always either doubles or floats (depending on dawn configuration)
Sparse Fields / Fields with a Chain of Location Types

```c++
const size_t edgesPerCell = 3;
const size_t C_E_C_V_Size = 6;
float_t dist_cc_to_edge[numK][numCells][edgesPerCell];
float_t intp_coeffs_6[numK][numCells][C_E_C_V_Size];
```
# Sparse Fields / Fields with a Chain of Location Types

```dusk_code
dist_cc_to_edge: Field[Cell>Edge]
intp_coeffs_6: Field[Cell>Edge>Cell>Vertex]
```

```cpp
const size_t edgesPerCell = 3
const size_t C_E_C_V_Size = 6
float_t dist_cc_to_edge[numK][numCells][edgesPerCell]
float_t intp_coeffs_6[numK][numCells][C_E_C_V_Size]
```
dusk & dawn - Language Features - Instructions

Reduction

Lhs = reduce(Expr: Rhs, Str: Op, Expr: Init, Chain: Nbh)
"Simple" Reduction

#dusk code

```python
average = 1./3.*reduce(in, "+", 0., Cell > Edge)
```

#sequential pseudo code

```python
for all levels k:
    for all cells c:
        for all edge_neighbors[c] e:
            average[k, c] += 1./3.*in[k, e]
```
dusk & dawn - Language Features - Instructions

Weighted Reduction → e.g. Gradient

```python
#dusk code
gradient = reduce(psi/dual_edge_length, "+", 0., Edge > Cell, [-1, 1])
```

\[
\nabla_n \psi(e) = \frac{\psi(c_1(e)) - \psi(c_0(e))}{\hat{l}}
\n\]
dusk & dawn - Language Features - Instructions

Reductions consuming Sparse Fields → e.g. Interpolations

#dusk code

intp = reduce(f_vertex*intp_coeffs_6, "+", 0., Cell>Edge>Cell>Vertex)
dawn - Conclusions & Discussion

- dawn is able to translate most ICON stencils at a fraction of the number of lines of code
  - code is automatically parallel
  - potential improvement in maintainability
- The code emitted by dawn is able to compete with expert tuned FORTRAN / OpenACC Code
dawn - Conclusions & Discussion
dawn - Conclusions & Discussion

- dawn is able to translate most ICON stencils at a fraction of the number of lines of code
  - code is automatically parallel
  - potential improvement in maintainability
- The code emitted by dawn is able to compete with expert tuned FORTRAN / OpenACC Code
- dawn is not yet able to translate the complete ICON dycore
  - upwinding, semi-Lagrangian advection, vertical indirection
- dawn, by design, requires existing climate model code to be re-written (e.g. in dusk)