ECMWF's New Product Generation

Lessons learned from development to operations

T. Quintino, B. Raoult, M. Zink, P. Maciel

ECMWF

tiago.quintino@ecmwf.int
ECMWF’s Forecasting Systems

What do we do?

Operations – **Time Critical**
- HRES 0-10 day, 00Z+12Z
  - O1280 (9km) 137 levels
- ENS 0-15 day, 00Z+12Z
  - O640 (18km) 91 levels
- ENS extended 16-46 day, twice weekly
  - O320 (36km) 91 levels
- BC 06Z and 18Z
  - hourly post-processing 0-5 days

Research – **Non Time Critical**
- Experiments to improving our models
- Reforecasts, Climate reanalysis, etc
ECMWF’s Production Workflow

- **From** 288 million fields @ 90 TiB/day
- **To** 230 million products @ 30 TiB/day
- **Using** 144 x 10 MPI tasks
Data Growth – History and Projections

Historical Growth of Generated Products

Model Output Projected Growth
Venerable ProdGen

• Legacy code
  – Written in Fortran 77/90
  – Continuously tweaked, No configuration files
  – Performance was acceptable but not optimal
  – Aborted on first error
  – Grown organically over decades without a “fresh rethink”

• Limited Reproducibility
  – Source code versioning was inconsistent
  – Workflow / Suite was maintained manually
  – Production was not transactional, could lead to data corruption
New PGen

• **New code**
  – Written in C++ 11
  – Optimized for *minimal work*
  – Configuration driven
  – **Resilient** to errors
  – New design, Based on *past experience*
  – *I/O server* for reduced I/O impact

• **Full Reproducibility**
  – Automated system from versioning to testing to production
  – Workflow / Suite was automatically generated from source
  – *Transactional* production
Architecture: ProdGen vs PGen

Explored different designs and architectures…

… but converged on a similar, yet evolved system design
Be bold in new directions,
But respect and learn from the past
MIR - ECMWF’s New Interpolation

- **Flexible** and maintainable design
  - Rewrite in C++
  - Configuration driven

- Kernel based on linear **Interpolation Operators**

- **Programmable Pipelines**
Optimizing Interpolation

- Kernel is a Matrix-Vector multiply
- **Deterministic Interpolation** operators
  - Enabling caching of operators

- Use of highly optimized **Linear Algebra** libraries (BLAS & MKL)

\[ F_i = \sum w_{ij} G_j \]
## Performance

<table>
<thead>
<tr>
<th>Grid</th>
<th>N Points</th>
<th>Memory [GiB]</th>
<th>Wall Time [ms]</th>
<th>Speed [Mp/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>N160</td>
<td>204 k</td>
<td>1.7</td>
<td>28.4</td>
<td>7.2</td>
</tr>
<tr>
<td>N256</td>
<td>524 k</td>
<td>1.8</td>
<td>33.0</td>
<td>15.9</td>
</tr>
<tr>
<td>N512</td>
<td>2097 k</td>
<td>1.8</td>
<td>51.2</td>
<td>40.9</td>
</tr>
<tr>
<td>LL 0.1/0.1</td>
<td>6483 k</td>
<td>2.6</td>
<td>99.9</td>
<td>64.9</td>
</tr>
<tr>
<td>N1024</td>
<td>8388 k</td>
<td>2.7</td>
<td>115.4</td>
<td>72.7</td>
</tr>
<tr>
<td>LL 0.05/0.05</td>
<td>25 927 k</td>
<td>6.1</td>
<td>252.2</td>
<td>102.8</td>
</tr>
</tbody>
</table>
Beware of the Floating Point

Users request a specific area of interest…

Lat <= 89.5 & Long  0.5

However **Floating Point** arithmetic…

• has **finite precision**
• is **non-associative**  \((a + b) + c != a + (b + c)\)
• some optimizations **relax truncation error assumptions**
Beware of the Floating Point (2)

Assume

\[ R = A \times B + C \times D \]

If \( A = C \) and \( B = -D \)

Then \( R = 0 \)

double c = sin(13./7.);
double d = tan(3./17.);
double a = c;
double b = -d;

double r = a * b + c * d;

Fused multiply-add
PGen Optimizations

- **Fast Linear Algebra**
  - Use Intel® MKL
  - Activate most FP optimizations

- **Correct and Deterministic**
  - Disable some optimizations
  - Fused multiply-add
  - Defined locations as **Factions of Integers**
    - A / B
Be mindful of cross-architecture reproducibility:

Be fast where you can,

But don’t sacrifice determinism
Programmable Pipeline Architecture

Construct an Action Plan

Decoding
Transform
Interpolation
Filter
Encoding

- GRIB
- NetCDF (basic)
- Raw (mem. buffer)
Programmable Pipeline Architecture

Construct an Action Plan

- Decode
- Transform
- Interpolation
- Filter
- Encode

SH to SH, VOD-2-UV
SH to Grid
Programmable Pipeline Architecture

Construct an Action Plan

- Decode
- Transform
- Interpolation
- Filter
- Encode

Compute Interpolation
Operator
Caching of operators
Linear algebra kernel
Programmable Pipeline Architecture

Construct an Action Plan

Decode → Transform → Interpolation → Filter → Encode

Crop Bitmap Frame

ECMWF
Programmable Pipeline Architecture

Construct an Action Plan

Decode → Transform → Interpolation → Filter → Encode

GRIB
Accuracy
Compression
Product Generation – PGen / MIR

- **Explicit Task Graph** analysis
  - *Users can update requests daily*
  - Factorise common tasks
  - Batch and Reorder execution
  - Compute time-series on-the-fly
PGen – Task Graph Analysis
PGen – Task Graph Analysis

Merge same input
Merge same interpolation target grids
PGen – Task Graph Analysis

*Merge same local area cropping*
Merge same local area frames
PGen – Task Graph Analysis

Caching of operators
Performance Analysis – Oper Stream 00Z run

e.g. step 24 ~ 18 min
Performance Analysis – Oper Stream 00Z run

e.g. step 24 ~ 4.3 min = 412% faster
Compose your complex workflow of manageable, deterministic building blocks
Thanks for your attention

Questions?