Performance, Portability, Productivity: Which two do you want?

Bryan Lawrence ’
NCAS, University of Reading
…and a cast of thousands.
Outline

Motivation

1. Performance

2. Evolution of hardware and software

3. The Chasm - What Chasm?
   - What’s in the Gap
   - Building Blocks for crossing the chasm
   - Next Steps

Summary
The long-term trend in computing requirement (over decades); compute and storage:

(Data courtesy of Mick Carter, UKMO)

UKMO Storage and Compute

TB Stored
Peak (Tflops)

Predicted ExaFLOPS
Predicted ExaBYTES

ExaFLOPS Long-Term

Year


Date

UKMO Storage
UKMO Compute
ncar Storage
ncar Compute
dkrz Storage
dkrz Compute

Terabytes Stored or Peak Teraflops Available

10^8
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0


Date

UKMO Storage
UKMO Compute
ncar Storage
ncar Compute
dkrz Storage
dkrz Compute

Performance, Portability, Productivity: Which two do you want?
Bryan Lawrence - Gung Ho Network Meeting, 18th July 2017
The long-term trend in computing requirement (over decades); compute and storage:

...but how do we get to exploit this new capability/capacity if it transpires?
CPMIP: measurements of real computational performance of Earth system models in CMIP6

Balaji, Maisonnave, Zadeh, Lawrence, Biercamp, Fladrich, Aloisio, Benson, Caubel, Durachta, Foujols, Lister, Mocavero, Underwood, Wright
1. How long will the experiment take (including data transfer and post-processing)?

2. How many nodes can be efficiently used in different phases of the experiment?

3. Can/should the experiment be split up into parallel chunks (e.g., how many ensemble members should be run in parallel)? What is the best use of my (limited) allocation?

4. *How much short-term/medium-term/long-term storage (disk, tape, etc.) is needed?*

5. Are there bottlenecks in the experiment workflow, either from software or from system policies, such as queue structure and resource allocation?
Comparing Performance

CPMIP Metrics

Criteria:

1. They are universally available from current ESMs, and applicable to any underlying numerics, as well as any underlying hardware architecture;

2. They are representative of the actual performance of the ESMs running as they would in a science setting, not under ideal conditions, or collected from representative subsets of code;

3. They measure performance across the entire lifecycle of modeling, and cover both data and computational load; and

4. They are extremely easy to collect, requiring no specialised instrumentation or software, but can be acquired in the course of routine production computing.
Comparing Performance

Metric Scope

1. Speed, of which more later ...
2. Computational cost
   2.1 based on the number of degrees of freedom in the model, factored into the influence of resolution and complexity;
   2.2 cost of load balancing and coupling;
3. Memory Boundedness;
4. Input/Output;
5. System policies (influence of queuing time etc)

Many of these are dependent on the Platform!
Comparing Performance

Speed or Throughput

Model performance can have two optimal points of interest:

1. **Speed**: Minimising time to solution, maximising simulated years per day or SYPD — termed S-Mode;

2. **Throughput**: Best use of a resource allocation (Minimizing compute hours per simulated year, or CHSY — termed T-Mode).

A single ESM experiment may contain both phases, e.g. spinup in S-mode, science in T-Mode.
### The Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>Number of gridpoints, summed over components.</td>
</tr>
<tr>
<td>Complexity</td>
<td>Number of prognostic variables, summed over components (NOT, the number of lines of code).</td>
</tr>
<tr>
<td>SYPD</td>
<td>Simulated Years Per Day (in both Speed and Throughput mode).</td>
</tr>
<tr>
<td>ASYPD</td>
<td>Actual SYPD measured from the start of enqueuing until the last data has reached its destination.</td>
</tr>
<tr>
<td>CHSY</td>
<td>Core-Hours per simulated year.</td>
</tr>
<tr>
<td>Parallelisation</td>
<td>Total number of cores allocated for the job.</td>
</tr>
<tr>
<td>JPSY</td>
<td>Energy cost of the simulation, in Joules per simulated year.</td>
</tr>
<tr>
<td>Coupling Cost</td>
<td>Normalised difference between actual run time, and the sum of the individual component times, with suitable parallelisation weightings.</td>
</tr>
<tr>
<td>Memory Bloat</td>
<td>Ratio of actual memory size (less memory for executable code) to ideal memory size (the size of memory you’d use if you had only one copy of your prognostic variables).</td>
</tr>
<tr>
<td>Data Output Cost</td>
<td>The cost of doing I/O (compared to an I/O free run).</td>
</tr>
<tr>
<td>Data Intensity</td>
<td>The data produced divided by the compute used: GB/CH.</td>
</tr>
</tbody>
</table>

All but the first two are dependent on the platform, so there are additional metrics aimed at understanding the nature of the platform.
Comparison for T-mode!

- Different hardware, and different complexity: complexity shown as the size of the square.
- In general the low-complexity models are AOGCMS and the high complexity models are ESMs.
- On similar hardware, we might expect to see models of similar complexity to cluster along similar resolution SYPD slopes, but we don’t have similar complexity or hardware!
- Aiming to do a much more comprehensive analysis for CMIP6!
Comparing Performance

Is the new hardware fast?

Consider these two (NOAA) machines:

- **c1+c2 (2014):** Cray XE6 (120 320 AMD Interlagos cores rated at 3.6 GHz on a Cray Gemini fabric).
- **c3 (2016):** Cray XC40 (48128 Intel Haswell cores rated at 2.3 GHz but with higher clock-cycle concurrency and Cray Aries interconnector).
Is the new hardware fast?

Consider these two (NOAA) machines:

- c1+c2 (2014): Cray XE6 (120 320 AMD Interlagos cores rated at 3.6 GHz on a Cray Gemini fabric).
- c3 (2016): Cray XC40 (48128 Intel Haswell cores rated at 2.3 GHz but with higher clock-cycle concurrency and Cray Aries interconnect).

And these results

<table>
<thead>
<tr>
<th>Model</th>
<th>Machine</th>
<th>Resol</th>
<th>SYPD</th>
<th>CHSY</th>
<th>JPSY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM4 S</td>
<td>gaea/c2</td>
<td>$1.2 \times 10^8$</td>
<td>4.5</td>
<td>16000</td>
<td>$8.92 \times 10^8$</td>
</tr>
<tr>
<td>CM4 S</td>
<td>gaea/c3</td>
<td>$1.2 \times 10^8$</td>
<td>10</td>
<td>7000</td>
<td>$3.40 \times 10^8$</td>
</tr>
<tr>
<td>CM4 T</td>
<td>gaea/c2</td>
<td>$1.2 \times 10^8$</td>
<td>3.5</td>
<td>15000</td>
<td>$8.36 \times 10^8$</td>
</tr>
<tr>
<td>CM4 T</td>
<td>gaea/c3</td>
<td>$1.2 \times 10^8$</td>
<td>7.5</td>
<td>7000</td>
<td>$3.40 \times 10^8$</td>
</tr>
</tbody>
</table>

Core for core, the new machine shows a speedup of 2.2X, which one could not have inferred from the clock ratings (influence of clock-cycle concurrency).

However, the total number of cores has dropped by 2.5X. Thus, in aggregate, c3 provides about 87% (2.2 / 2.5) of the capacity of the older c1 and c2 partitions combined, for the GFDL workload.

…however, that the PF rating of c3 is considerably higher than c1 and c2 combined (1.77 PF vs. 1.12 PF).

Cray Cray Aries is showing a manifest increase in performance, with the same CHSY in both configurations (i.e., with different numbers of PEs) unlike Gemini.

There is a concrete and substantial fall in the total energy cost of simulation science!
Comparing Performance

Is the new hardware fast?

Consider these two (NOAA) machines:

- **c1+c2 (2014):** Cray XE6 (120 320 AMD Interlagos cores rated at 3.6 GHz on a Cray Gemini fabric).
- **c3 (2016):** Cray XC40 (48128 Intel Haswell cores rated at 2.3 GHz but with higher clock-cycle concurrency and Cray Aries interconnect).

And these results:

<table>
<thead>
<tr>
<th>Model</th>
<th>Machine</th>
<th>Resol</th>
<th>SYPD</th>
<th>CHSY</th>
<th>JPSY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM4 S</td>
<td>gaea/c2</td>
<td>$1.2 \times 10^8$</td>
<td>4.5</td>
<td>16000</td>
<td>$8.92 \times 10^8$</td>
</tr>
<tr>
<td>CM4 S</td>
<td>gaea/c3</td>
<td>$1.2 \times 10^8$</td>
<td>10</td>
<td>7000</td>
<td>$3.40 \times 10^8$</td>
</tr>
<tr>
<td>CM4 T</td>
<td>gaea/c2</td>
<td>$1.2 \times 10^8$</td>
<td>3.5</td>
<td>15000</td>
<td>$8.36 \times 10^8$</td>
</tr>
<tr>
<td>CM4 T</td>
<td>gaea/c3</td>
<td>$1.2 \times 10^8$</td>
<td>7.5</td>
<td>7000</td>
<td>$3.40 \times 10^8$</td>
</tr>
</tbody>
</table>

1. Core for core, the new machine shows a speedup of 2.2X, which one could not have inferred from the clock ratings (influence of clock-cycle concurrency).
2. However, the total number of cores has dropped by 2.5X. Thus, in aggregate, c3 provides about 87% (2.2 / 2.5) of the capacity of the older c1 and c2 partitions combined, for the GFDL workload.
   - …however, that the PF rating of c3 is considerably higher than c1 and c2 combined (1.77 PF vs. 1.12 PF).
3. Cray Cray Aries is showing a manifest increase in performance, with the same CHSY in both configurations (i.e., with different numbers of PEs) unlike Gemini.
4. There is a concrete and substantial fall in the total energy cost of simulation science!
We need to design codes for here!

We design codes for here

Microprocessor performance
~55% per annum

Memory capacity
~49% per annum
(and slowing down?)

Memory bandwidth
~30% per annum
(and slowing down?)

Memory latency
<<30% per annum
Evolving Hardware

**Broadwell**

**KNL**

- **Performance, Portability, Productivity: Which two do you want?**
- **Bryan Lawrence - Gung Ho Network Meeting, 18th July 2017**
Evolving Software

Requirement: Ensure model quality (the models available are meeting the advancing scientific requirements)!

Available really means models with:

1. **Performance** — satisfactory SYPD for affordable CHSY given real data intensity.

2. **Portability** — models need to run on all the relevant platforms currently in use, or likely to be in use in the foreseeable future — without excessive lead times associated with porting, and

3. **Productivity** — the ability to work with the codes from a scientific perspective – changing algorithms, adding processes, etc, in a reasonable period of time, and without inadvertently compromising on reliability/accuracy of code.
To be productive, we need to be able to exploit mathematics!

The best mathematical methods to use at any given time are going to be slaves to the nature of the hardware – which will be problematic if the hardware is changing quicker than the mathematics and its implementation!

What is the lead time from maths to code?

How hard is it to make that code performant?

And will that code be portable?

Our best guess, today, is that it won’t be long before you can have only any two of those! The free X86 based free lunch is over!
Crossing the Chasm: How to develop weather and climate models for next generation computers?

Lawrence, Rezny, Budich, Bauer, Behrens, Carter, Deconinck, Ford, Maynard, Mullerworth, Osuna, Porter, Serradell, Valcke, Wedi, and Wilson

Targeting GMD, submission within weeks!
IS-ENES2 Deliverable 3.2
Software changing slowly & slowing!

Hardware changing rapidly & accelerating!

How far is it between our scientific aspiration and our ability to develop and/or rapidly adapt our codes to the available hardware?
Science Code

How do we bridge the gap?

Compilers, OpenMP, MPI etc

Hardware & Operating System

Motivation

Performance

Evolution

Chasm

In the Gap

Building Blocks

Next Steps

Summary

Statement

Hardware & Operating System

Compilers, OpenMP, MPI etc

Science Code

How do we bridge the gap?

Compilers, OpenMP, MPI etc

Hardware & Operating System

Motivation

Performance

Evolution

Chasm

In the Gap

Building Blocks

Next Steps

Summary

Statement

Hardware & Operating System

Compilers, OpenMP, MPI etc

Science Code

How do we bridge the gap?

Compilers, OpenMP, MPI etc

Hardware & Operating System

Motivation

Performance

Evolution

Chasm

In the Gap

Building Blocks

Next Steps

Summary

Statement

Hardware & Operating System

Compilers, OpenMP, MPI etc

Science Code

How do we bridge the gap?

Compilers, OpenMP, MPI etc

Hardware & Operating System

Motivation

Performance

Evolution

Chasm

In the Gap

Building Blocks

Next Steps

Summary

Statement

Hardware & Operating System

Compilers, OpenMP, MPI etc

Science Code

How do we bridge the gap?

Compilers, OpenMP, MPI etc

Hardware & Operating System

Motivation

Performance

Evolution

Chasm

In the Gap

Building Blocks

Next Steps

Summary

Statement

Hardware & Operating System

Compilers, OpenMP, MPI etc

Science Code

How do we bridge the gap?

Compilers, OpenMP, MPI etc

Hardware & Operating System
Too many levels of parallelism!

1. Vectorisation within a CPU or GPU core or accelerator (via the compiler, or compiler directive languages such as OpenACC),
2. Shared parallelism across CPU and accelerators/GPUs.
3. Threading providing shared memory concurrency within nodes/sockets (using a tool such as OpenMP),
4. Distributed memory concurrency across nodes, either by
   ▶ utilising MPI (traditional “domain decomposition”); directly, or with a library, or
   ▶ exploiting a PGAS implementation (e.g. CoArray Fortran)
5. Internal component concurrency (using, e.g. ESMF) or manually provisioned using OpenMP,
6. Concurrent coupled model components, either
   ▶ executed independently using a coupler such as OASIS, or executed together using a framework, or
   ▶ concurrent models running as part of a single executable ensemble
7. I/O parallelism (using an I/O server such as XIOS)
Parallelism

Best practice for parallelism?

Current best practice for addressing these modes of concurrency is to

1. Code for hierarchies of parallelism (loops, blocks, do-mains),
2. Use standard directives (OpenMP/OpenACC),
3. Optimise separately for many-core/GPU, and
4. Try to minimise code differences associated with architectural optimisations.

However, this is no longer seen as a successful strategy – at least on its own!

With the advent of exascale systems, entirely new programming models are likely to be necessary, with entirely new constructs such as thread pools and task-based parallelism possible.
Clearly lots of activity addressing these problems, including, e.g:

1. Early experiences re-coding for new processors which did effective rewrites which became orphaned because they used the left the science code owners behind (wrong language etc). (Performant, but not portable or Productive).

2. Aggressive code development and maintenance with many lines of directives with great scope for error. Opinion divided as to how easy it will be to maintain the resulting codes (productivity?)

3. Code translation: works to an extent, but require invasive changes (productivity).

4. Experiments with new programming models suggest that the developer really needs to understand the fundamental algorithms and vice versa (issues for productivity).

All these have addressed “fine-grained” parallelisation and have delivered modest results.
Route 1: The Massive Edifice

- No group has enough effort to do all the work needed.
- No group has all the relevant expertise.

Route 2: Incremental Advances

- The peril of the local minimum
- Any given span/leap may not be sufficient to cross the next gap!
Motivation
Performance
Evolution
Chasm
In the Gap
Building Blocks
Next Steps
Summary

Architecture for crossing?

Science Code

- PSyclone
- GridTools
- ESMF
- OASIS
- YAC
- ESCAPE
- XIOS
- NetCDF4
- HDF5

Compilers, OpenMP, MPI etc

Hardware & Operating System

Performance, Portability, Productivity: Which two do you want?
Bryan Lawrence - Gung Ho Network Meeting, 18th July 2017
Why and What is a DSL?

Why?

- Humans currently produce the best optimised code!

- Humans can inspect an algorithm, and exploit domain-specific knowledge to reason how to improve performance – but a compiler or generic parallelisation tool doesn’t have that knowledge.

- Result: Humans better than generic tools every time, but it’s big slow task and mostly not portable!
Why and What is a DSL?

Why?

▶ Humans currently produce the best optimised code!
▶ Humans can inspect an algorithm, and exploit domain-specific knowledge to reason how to improve performance – but a compiler or generic parallelisation tool doesn’t have that knowledge.
▶ Result: Humans better than generic tools every time, but it’s big slow task and mostly not portable!

What?

▶ A domain specific compiler, with a set of rules!
▶ Inclusive knowledge includes things like
  ▶ Operations are performed over a mesh,
  ▶ The same operations are typically performed independently at each mesh point/volume/element,
  ▶ the meshes themselves typically have consistent properties.
  ▶ …
▶ Let the tools exploit that knowledge, and leave a much smaller task for the humans!
DSLs in the Wild

Two major projects:

- GridTools (formerly Stella)
- PSyclone (evolved from Gung Ho)

Both are DS ELs ... domain specific **embedded** languages.

- Embedded in C++
- Embedded in Fortran
- Originally targeted finite difference lat-lon LAM.
- Originally targeted finite element irregular mesh.
- Backends (via human experts) mapped to the science description via C++ templates.
- A recipe of optimisations (via human experts) is used by PSyclone to produce targeted code.

In both cases the DSL approach allows mathematical experts to do their thing, while HPC experts do their thing, and the DSL provides a separation of concerns.
Whither the DSL?

- DSLs are becoming more common across disciplines.
- The Domains are more or less specific …
  - the more specific, the cleaner a domain specific separation of concerns, but the larger the technical debt (maintaining the code and the teams of experts for the backends)
  - the more generic, the less the DSL can do for you, and the less the separation of concerns.
DSLs are becoming more common across disciplines.

The Domains are more or less specific …

- the more specific, the cleaner a domain specific separation of concerns, but the larger the technical debt (maintaining the code and the teams of experts for the backends)
- the more generic, the less the DSL can do for you, and the less the separation of concerns.

The holy grail is to add further separation of concerns inside the DSL …e.g. can we imagine a GridTools and a PSyclone front end to a vendor managed intermediate DSL compiler?

- compare with MPI: successful because vendors manage their own specific backends with a defined API that we all work with to develop our own libraries (e.g. GCOM, YAXT etc)
Weather and Climate Dwarfs

1. Flat computational profiles: select exemplar codes, ”mini-apps”, typical of key functionality, so-called dwarfs (after the Berkely Dwarfs).

2. Aim for computational performance challenges which are fundamentally different between dwarfs just like their functional separation within an Earth system model.

3. ESCAPE (Energy-efficient Scalable Algorithms 60 for Weather Prediction at Exascale; www.hpc-escape.eu) project is investigating this approach for weather and climate.

4. For each dwarf ESCAPE targets performance on existing and emerging processor technologies (specifically Intel Xeon, Xeon Phi and NVIDIA GPGPU and a novel technique employing optical interferometry particularly suitable for Fourier transforms), but it is also targeting programming methodologies.
## ESCAPE: Dwarfs

<table>
<thead>
<tr>
<th>D</th>
<th>Spectral Transform</th>
<th>Spherical harmonics based transform to facilitate semi-implicit solvers on the sphere.</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Spectral Transform</td>
<td>A 2D Fourier spectral transform for regional model applications.</td>
</tr>
<tr>
<td>D</td>
<td>Advection</td>
<td>A flux-form advection algorithm for sign-preserving and conservative transport of prognostic variables and species.</td>
</tr>
<tr>
<td>I</td>
<td>3D Interp.</td>
<td>Interpolation algorithm representing a wide class of interpolation and remapping uses in NWP &amp; Climate.</td>
</tr>
<tr>
<td>D</td>
<td>Elliptic Solver</td>
<td>An iterative solver for the 3D elliptic problem arising in semi-implicit time-stepping algorithms.</td>
</tr>
<tr>
<td>D</td>
<td>Advection, SLT</td>
<td>An implementation of the semi-Lagrangian advection algorithm.</td>
</tr>
<tr>
<td>P</td>
<td>Cloud MicroPx</td>
<td>Cloud microphysics scheme from IFS, an exemplar of a range of physical parameterisations.</td>
</tr>
<tr>
<td>P</td>
<td>Radiation</td>
<td>An exemplar radiation scheme used in regional NWP modelling.</td>
</tr>
</tbody>
</table>
1. No established convention or standard for layout in memory grids and meshes or how the various cell entities are associated with each cell: scope for data models and associated efficiency.

2. Exploit knowledge about data types and data movement: Improved MPI libraries

3. How best to optimise, find bottlenecks? Better use of tools for optimisation
If we work together …

...can we reduce the problem to a set of small leaps?
Science Code

- Defined Interfaces and Contracts
  - High Level Libraries and Tools
    - Defined Interfaces and Contracts
  - Libraries and Tools
    - Defined Interfaces and Contracts
  - Low-Level Libraries and Tools
    - Defined Interfaces and Contracts
  - Compilers, OpenMP, MPI etc

Hardware & Operating System

Turning the chasm into a bunch of creeks
What do we need?

▶ Progressing at the community level will require methods to allow the community to discuss, specify, design, develop, maintain, and document the necessary libraries and tools.

▶ …that is, a commonly deployed structured approach to sharing, one that maximises delivery of requirements, while minimising risk of future technical burden – the sort of approach that has delivered the MPI libraries upon which nearly all of HPC depends.

▶ While a fully fledged standards track is probably beyond the will of the community at this point, it is certainly possible for the community to take more steps towards joint working!
What steps can the community make now?

Begin by recognising that:

- business as usual, consisting of modest incremental steps, is unlikely to deliver the requisite next generation models,
- none of us have enough internal resource to take the leap to the next generation alone, and most importantly,
- there are library or tool projects which can be exploited, some of which may be from outside our traditional communities of collaborators.
What institutional characteristics are necessary?

They will most probably:

- Have **understood** the issue fully at the management level, the science level, and in the infrastructure teams,
- Be able to **reward individuals** for innovation in, and/or contributions to, **external** projects,
- Recognise the **benefit of external scrutiny** and contributions into their own projects,
- Have the **courage to stop** existing activities and **pickup and use/integrate** third party libraries and tools, and
- Have the ability to **recognise the cost-benefit trade-off** between “doing it themselves” and contributing intellectually and financially to third party solutions, and
- Be ready to **apply more sophisticated and complex** software engineering techniques, and encourage more computational science research.
What project characteristics are necessary?

They will:

- be open source and have an open development process,
- have clear goals, scope, and where appropriate, deliver stable software interfaces,
- have a mechanism to understand and respond to the timescales of collaborators (that is, some sort of governance mechanism which assimilates and responds to requirements),
- potentially be able to accumulate and spend funds to provide user-support, training, and documentation,
- be not initially disruptive of existing solutions, and ideally
- engage both the scientific community and vendors (compare with MPI where vendor implementations are often key to enhanced MPI performance).
What do we need?

To work out what components exist and which can be shared. To do that we need a taxonomy. Such a taxonomy will cover at least:

- Tools for exposing mathematical algorithms for implementation on a sphere (domain specific languages),
- Tools for describing and using data models for the variables in those algorithms (including stencils for computation),
- Mathematical Libraries such as Fast Fourier and Spherical Transforms,
- Solvers which can exploit specific data models and algorithms,
- Interpolation and Regridding Libraries,
- Embedded and standalone visualisation tools,
- Exchange libraries (for problems ranging from domain halo exchanges to 3D field exchange between high level components),
- Fully fledged couplers (e.g. OASIS) and frameworks (e.g. ESMF),
- I/O servers (such as XIOS),
- Data Assimilation tools such as minimisers and adjoint compilers,
- Clock, calendar, time-stepping and event handling libraries (events such as "do at first time step, do every three hours, etc),
- Testing Frameworks,
- Performance and Debugging tools,
- Domain specific tools for automatic code documentation.
Lots of pieces in play:

- Performance is tough.
- Getting all of Performance, Portability and Productivity is going to become much harder!

To get timely and manageable progress:

- We are going to have to change the way we work, and work more together.
- Working together isn’t going to be easy either!

Cambrian explosion in hardware means: