Scientific Workflows: a Bottom-Up Perspective

Bruno Raffin,
DataMove
INRIA Grenoble Rhône-Alpes

ORAP, March 2019
Motivation

« Classical » workflows:
  – Code coupling
  – Data exchange through files
  – Grid support

Workflows from a bottom-up perspective:
  – Performance (No intermediate files)
  – Code development and maintenance
  – Tool coupling for added functionalities
    – Benefit from new programming paradigm shifts
Two Phase I/O

Goal: get I/Os outside of the simulation code for performance purpose

How:

– Overlap I/Os and computation with asynchronous I/Os

– Aggregate data per nodes

But also better code modularity

Examples: Adios, XSIO
Aggregate the results in situ on helper-core (1 per node) to the master node

Gromacs: standard molecular dynamics simulation code
Aggregate local results in situ on helper-core (1 per node) and write to disk
Two Phase I/O Performance

Gromacs no I/O

Two Phase I/O (one helper core)

Gromacs native I/O (in-simulation)

2048 cores (froggy@CIMENT)

Gromacs without I/O: 15 cores/node 3% slower than 16 cores/node
(-6% if scalability would have been perfect)

Developed with FlowVR
In Situ Processing

Goal: move from post mortem data processing to on-line/live analytics, i.e suppress intermediate files.

How:

– Overlap I/Os and analytics
– Aggregate data per nodes, process data on helper cores

Examples: Damaris, Decaf, Flexio, FlowVR, Tins
In S itu Processing

No analytics

In situ

Simulation iteration(s) → I/O → Simulation iteration(s)

Simulation iteration(s) → Data Copy → Simulation iteration(s)

Interferences

Simulation

It++

Data copy

Analytics

It++

In situ: simulation and analytics share the same nodes

Resource allocation strategies: time sharing or space sharing (dedicated helper core)

Node

Helper core

Analytics

Simulation

Simulation

Simulation

Simulation

Analytics
Most solutions are process based, but some like Tins [SCA 2017] show higher performance relying on tasks (based on intel TBB)
In Transit Processing

For practical or performance reasons, using nodes dedicated to analytics may be convenient to execute part of the analytics workflow.
In Situ + In Transit Processing

Analytics workflows can be complex calling for solutions that enable performance, modularity and flexibility.
Example: Parallel In Situ Isosurface Extraction [Dreher, CCGRID’14]

Compute a molecule surface based on atom density

Tested different distributions of processing steps to in situ and in transit nodes.
Performance [Dreher, CCGRID’14]

- In transit: 1 staging node every 64 compute nodes
- Density-intransit: costs 7% comp. to gromacs 15 cores
- Density-insitu costs 8% but use 1.5% less nodes than density-intransit
- Atoms-intransit costs 8.6% but enables other in-transit analytics (3x more data to move on staging nodes than Density-intransit)

Gromacs performance with Quicksurf

- gromacs-0-15cores
- gromacs-0-16cores
- quicksurf-density-intransit
- quicksurf-density-insitu
- quicksurf-atoms-intransit

Gromacs no I/O
Gromacs + Isosurface

2048 cores (froggy@CIMENT)
Data Staging

Ephemeral in transit store on staging nodes

PaDaWAn

Staging Area

Producer

Controller

Put

Delete

Publish

Get

Subscribe

Next

Close

Padawan

Simulation

put()/pub()

get()/sub()

d: Data object

m: Meta-data object

App1

App2

App3

Shared Space Model

DataSpace
BigData/Stream Processing

Take benefit of “novel” programming frameworks (Map/Reduce)

Molecular Dynamics App (MPI) → Flink Stream Processing → Distributed DB on top of HDFS

CoMD → CoMD → CoMD → Flink Worker → Flink Worker → HBase

[ISAV 2018]
High Performance Analytics

Compute Nodes

Staging Nodes

Sensor data

Simulation Processes

Analytics Processes

Simulation Processes

Analytics Processes

Simulation Processes

Analytics Processes

Simulation Processes

Analytics Processes

Staging Nodes

3D visualization + notebook

Ephemeral Object Store

In Transit Analytics

Batch Analytics

Emerging analytics: ML
In Situ Processing Frameworks

**MPI based**: Damaris, Decaf, FlexIO
  - One `mpirun`
  - One or several executables (SPMD mode)
  - Different communicators
    - Limited modularity and elasticity

**Broker/daemon/service based**: CCA, FlowVR
  - Communications handled by a third party
  - Enable coupling heterogeneous executables
The FlowVR Programming Model

1. Develop components:
   - While ( \texttt{wait(inputs)} )
     - \texttt{get()}
     - \texttt{compute()}
     - \texttt{put()}
   - Simple API (limit code intrusion)

2. Assemble components: Python script

3. Instantiate parameters and execute script

\url{http://flowvr.sf.net}
Parallel Simulation

N-to-1 Com

Analytics
N-to-1 Communication Pattern
(tree with arity k - unknown)

(N unknown – get value from incoming component)

Merge filter
(K inputs – unknown)
Parallel Simulation

N-to-1 Com

Analytics

N=4

K=2

FlowVR: example
N-to-1 Communication Pattern

(N=4)

Merge 2 inputs

Merge 2 inputs

Merge 2 inputs
Parallel Simulation

N-to-1 Com

Analytics

N=4

K=2
Let the Simulation Simulate

• Classical push paradigm: the simulation controls everything, in particular what/when data are pushed outside (for I/Os or analytics)

• Pull paradigm:
  – The simulation only exposes internal data structures
  – The consumer retrieves what is necessary when necessary
while (!computation_finished) {
    /* fill data buffer */
    It++;
    PDI_Expose ("newdata", "buffer", &buffer, size, "Iteration", it);
}

Simulation

Data Model

Contract:

Iteration: int
buffer: { type: array, subtype: double, size}
On_event: newdata trigger  foo("buffer","Iteration")

Pull code

foo()
{
    PDI_get("Iteration", &it)
    PDI_get ("buffer", &buffer, &size);
    if (need data at iteration it) do_something_with(&buffer, size);
}

Examples: Damaris, Adios, VTK/Ascent, PDI
High Performance Analytics

Compute Nodes

Simulation Processes

Analytics Processes

Sensor data

Simulation Processes

Analytics Processes

Simulation Processes

Analytics Processes

Simulation Processes

Analytics Processes

Staging Nodes

3D visualization + notebook

Ephemeral Object Store

Analytics Processes

GPU

NVM

Analytics Processes

GPU

NVM

Analytics Processes

In Transit Analytics

Batch Analytics

Simulation

In Situ Analytics

[Mommessin et al. Cluster'17]
Ensemble Runs

• From one to many simulation runs (with different parameters)
  – Statistics
  – Sensibility analysis, uncertainty quantification

• Example with molecular dynamics: Copernicus (SC’11)
  – Single simulation hard to scale (latency bounded)
  – Ensemble runs with adaptive sampling
  – « Killer use case » for exascale computing
Ensemble Runs: Sensitivity Analysis

P simulations with different parameter values

Output: one temperature \( T_1(x,y,z,t) \) per mesh cell and timestep

\[ T_p(x,y,z,t) \]

[Terraz et al. SC’17]

Collab. with EDF

I/O bottleneck at scale:
- slower simulations
- slower analytics

Basic solution: statistics at low resolution

Analytics: spatiotemporal statistics

E.g. \( T_{avg}(x,y,z,t) = \frac{1}{p} \sum_{i=1,p} T_i(x,y,z,t) \)

for a selection of probes \((x,y,z,t)\)
In Transit Sensitivity Analysis

Iterative statistics: one-pass algorithms

\[ \mu_i(x, y, z, t) = \mu_{i-1}(x, y, z, t) + \frac{1}{i} (u_i(x, y, z, t) - \mu_{i-1}(x, y, z, t)) \]

Supported stats: average, variances, skewness, kurtosis, min max, threshold exceedance, Sobol’ indices, quantiles
Fault Tolerance & Elasticity

**Control**
- Batch Scheduler
  - Run Jobs
  - Kill & restart failed jobs
  - Job status

**Launcher:**
- Detect failing server with timeout
- Detect zombies simulations with timeout
- Control #restarts per group

**Computations**
- Sim (i) fresh start
- Sim (j) restart
- Sim (k) zombie

**Parallel Melissa Server:**
- Register wallclock and timestep of last received data per group
- Discard duplicate messages
- Detect failing simulations with timeout
- Periodic checkpoint

**Checkpointing:** +0.5% exec time
Simulations: 8000 simulation runs
Massive run: up to 28K cores

8000 simulation runs (512 cores each)

**32 server nodes**: 740 CPU.h (2.1%)
Simulations: 34000 CPU.h

Simulation runs 13% faster on average than when writing to disk

**Massive run:**
- 80 000 simulation runs (24 cores each)
- 271 TB of data processed online (and thus not saved to disk)
Ensemble Runs

Other use cases:

**Statistical Data Assimilation**
Simulation are driven by data from observations (EnKF)

**Deep Reinforcement Learning**
Self-learn a score optimizing strategy from simulations (AlphaGoZero)
Melissa for DRL at Scale

**Control**
- Batch Scheduler
  - Pending Jobs
  - Run Jobs
  - New Jobs
  - Launcher: select simulations to run

**Computations**
- Atari game + DNN
  - Actor
  - Dynamics Connection to the server
  - Parallel Melissa Server
    - Parallel Learner (Deep NN)

- Periodically update NN weights

Join work between the DataMove And Sequel teams

Goal: to go beyond AlphaGoZero
Conclusion

Workflows driven by needs of:
   – Performance (the I/O bottleneck)
   – Data analytics

Emerging needs:
   – Data assimilation
   – Ensemble runs
   – Deep learning