# XCOMPUTE:

## Algorithms and Instruction Sequences for CFD/FEA Multiphysics

G.J. Orr, R.J. Kwan, M. Doudar

Xplicit Computing, Inc

11 July 2022

An interactive paper is available 11-15 July 2022 at:
https://xplicitcomputing.com/docs/xc-arch/paper_overview.html

# Problem Statement

- Hardware is improving, but still, HPC is inaccessible to most.
- CAE software is decades behind and increasingly-specialized.
- Teams cannot collaborate effectively across disciplines and tools.

**A platform standard is required for performant collaboration,**

**providing an open schema as a Rosetta Stone between computing efforts.**

# Technology Roadmap

- Modest single-thread performance increase
- Shift toward parallelism and local clustering
- Introduction of heterogeneous computing

**Static Host Programs**
- compiled application binaries

2011:
C++11 +
OpenCL1.2

**CPU**
- C++/OMP
- 0.1-1 TFLOPS/socket

**Dynamic Device Programs**
- generated and compiled at runtime

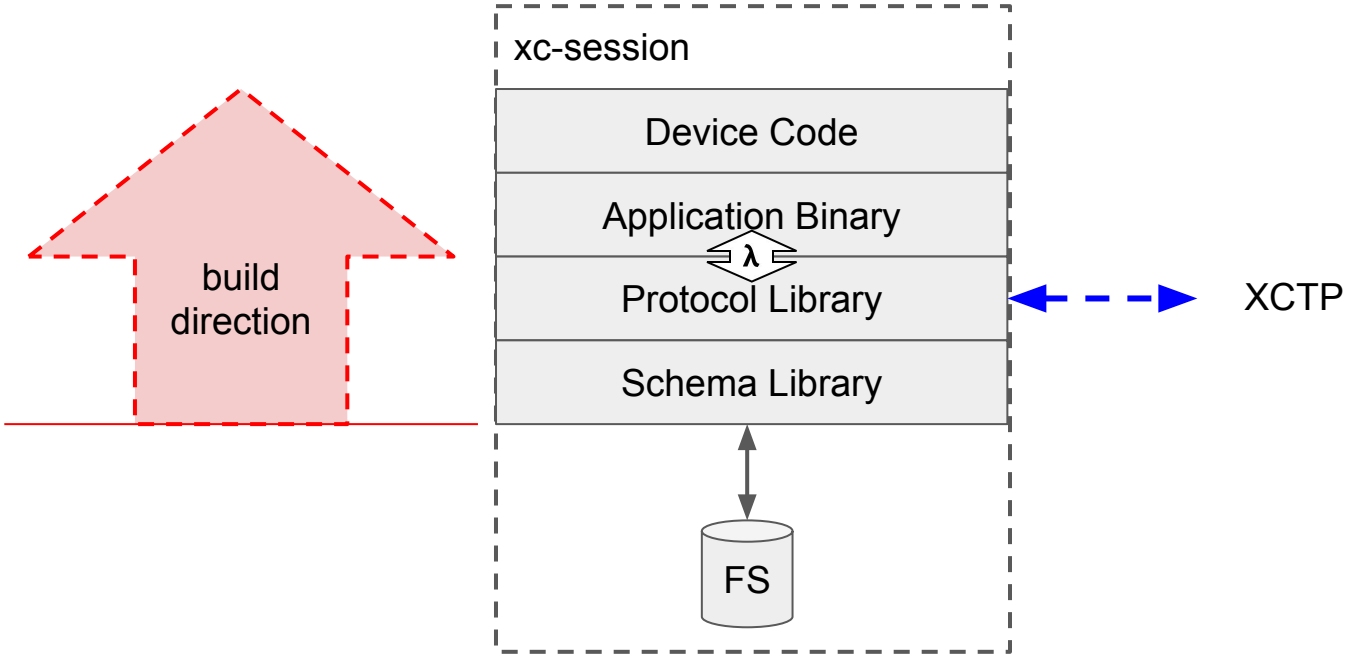
2022

**GPU**
- C/OpenCL/OpenGL
- 1-100 TFLOPS/card
- exascale aggregate

**ASIC**
- C/OpenCL
- Q/OpenQL?
- multiprecision?

G.J. Orr, R.J. Kwan, M. Doudar. Xplicit Computing, Inc. USA info@xplicitcomputing.com
ICCFD11-2022-0802

# XC Software Stack



# Messages Library - libxcmessages

*Messages* provides open flexible encoding/decoding similar to XML and JSON but faster and denser. A machine-generated library contains utilities to flatten and reconstruct object-oriented and vectorized data structures.

Xplicit Computing created the *Messages*™ file & wire schema based on Google's Protocol Buffers, a platform-neutral serialization mechanism for structured data. The *Messages* schema is defined by .proto files that are passed into the *protoc* compiler to generate high-performance binary-encoded accessors.

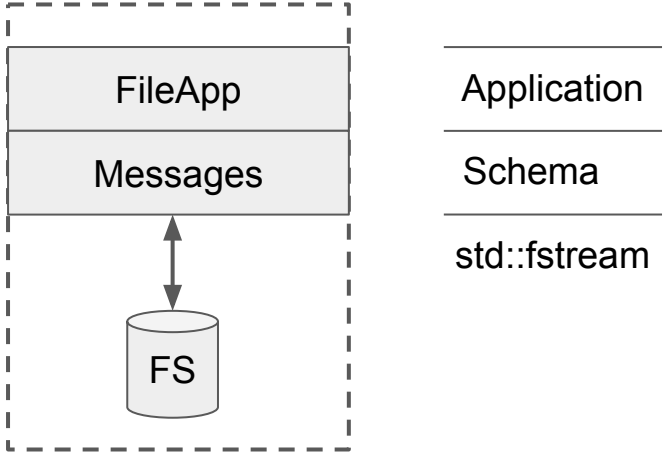



Illustration of serialization & I/O in a typical filesystem application.

# Common Library - libxccommon

The common library contains proprietary platform machinery that is shared across xcompute sessions, including low-level math utilities and transport protocol.

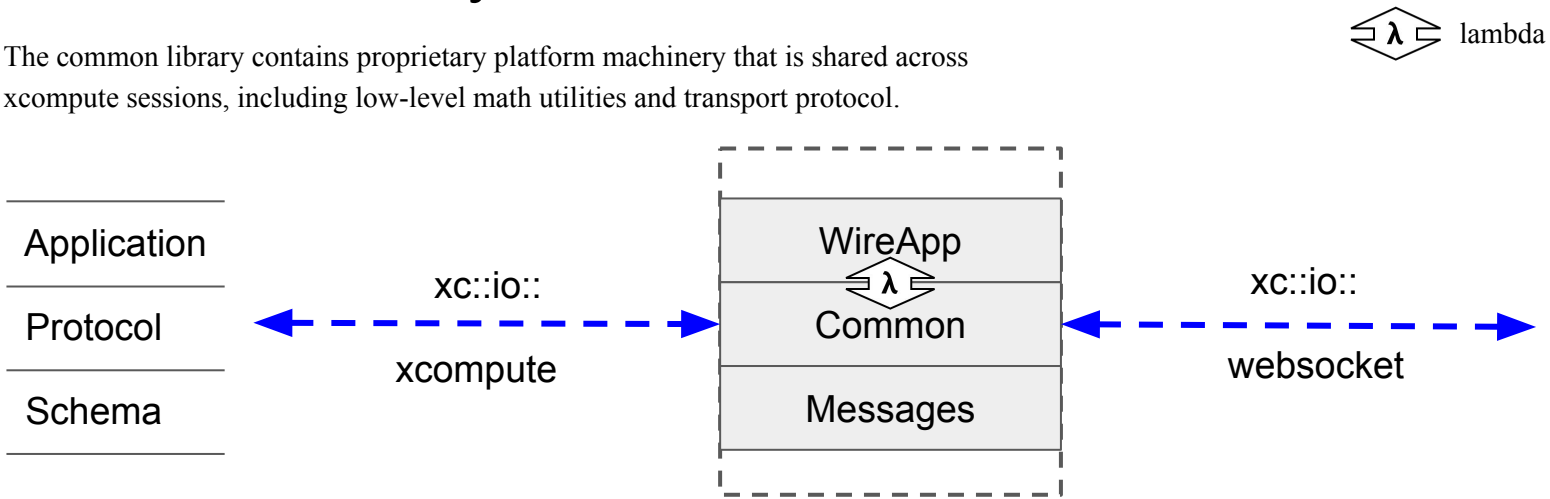
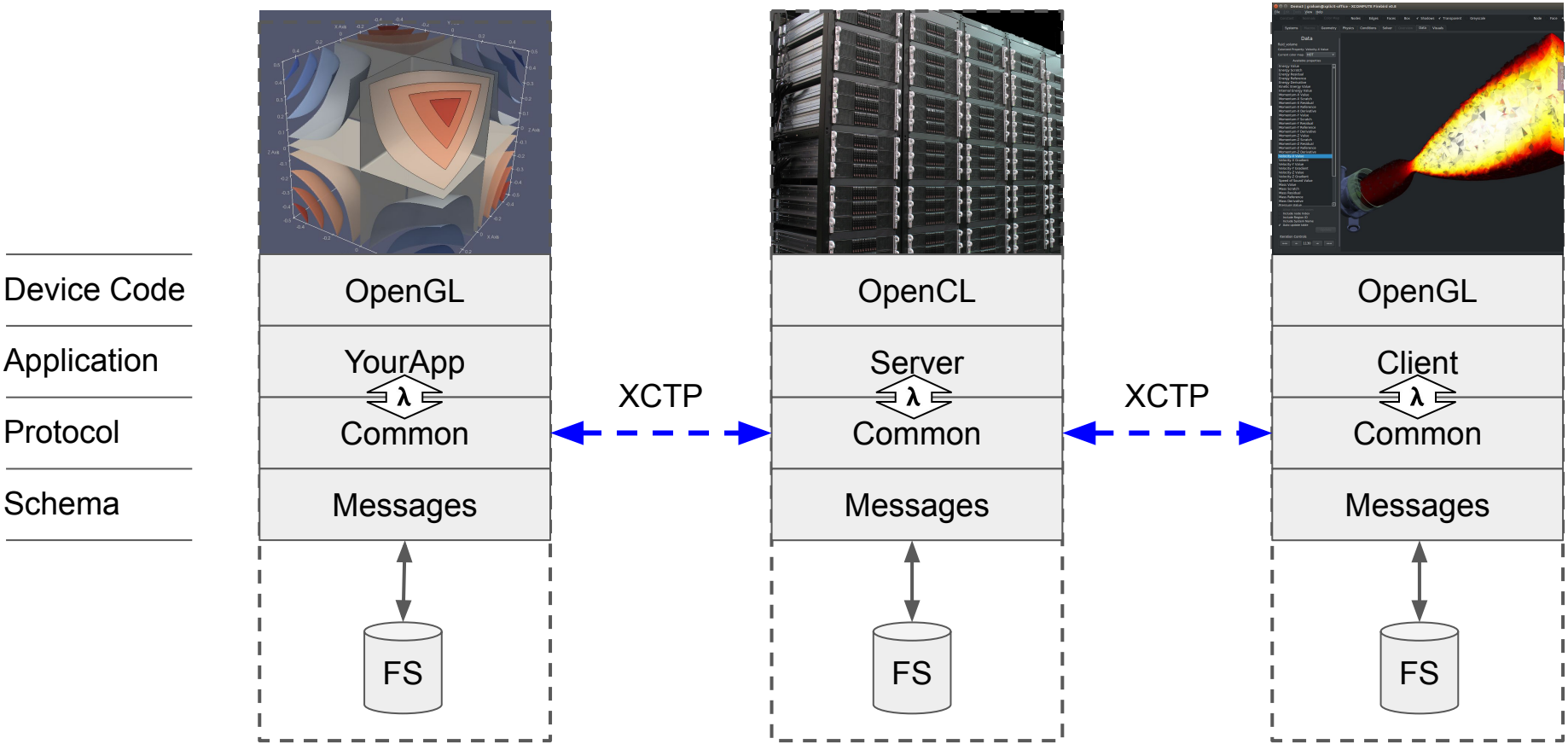



Illustration of serialization & I/O in a typical wire application.

The *xcompute* protocol (XCTP) comprises a 32-bit hash-encoded function call followed by variadic arguments, packaging payloads into a transmission defining the expected types and sizes. The intended function is decoded, parameters deserialized, and the function call is dispatched to an appropriate lambda overload, using a Y-combinator to facilitate the recursion on function arguments.

# Platform Overview



# Server Responsibilities

1. **Passive XCTP Protocol**

- Serve push/pull/do with *uid*
- Push *Messages::Meta* on change

2. **Compute Resources**

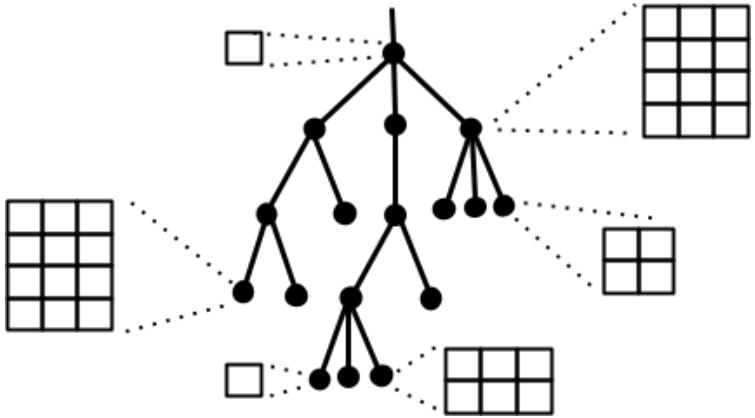
- Platforms & Devices
- Solvers & Programs
- Algorithms & Instructions
- Collections of Models

3. **Systems Tree**

- Data
- Geometry
- Physics
- Conditions
- Solvers

# System (noun)
## Messages::System

System is an extrinsic object (noun) - a scoped numerical domain with abstract or physical form and behavior, representing components, assemblies, or fields. Its boundaries may be artificially-driven with conditions, or coupled with regional links for global multi-fidelity, each using different types/kinds of algorithms. A system can supervene and manage any number of subsystems in a parent-child (one-way) relational tree hierarchy, permitting spatial and temporal subspaces each with assignable host and device resources, allowing a complex problem to be parallelized on SIMD hardware at more reasonable local scales.



Contains: Data, Geometry*, Physics*, Conditions, Solvers, Subsystems…

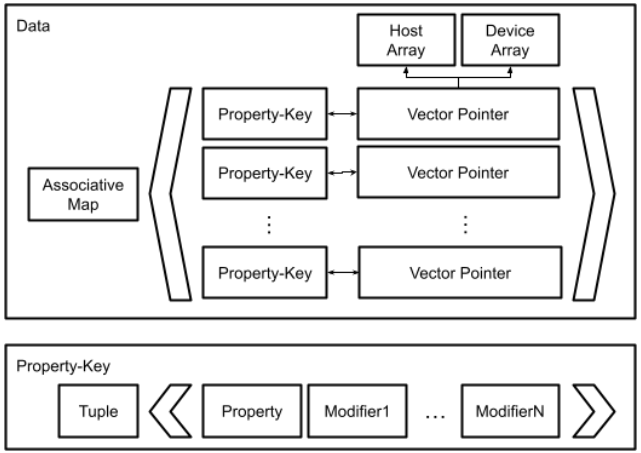
G.J. Orr, R.J. Kwan, M. Doudar. Xplicit Computing, Inc. USA info@xplicitcomputing.com ICCFD11-2022-0802

# Data

## Repeated Messages::Vector64

Data are scalars and vectors for singular and nodal values stored in a convenient format accessed as *system.data(PK)*. A given system's data also defines its name, owner, and permissions; such attributes are intrinsic to data but can be utilized by parent objects.

The dimensions of data entries are determined by the number of nodes in the system. The number of rows is equal to the number of nodes (or elements), while the number of columns is dictated by the number of components associated with the given PK and spatial dimensionality.




**auto& vector = data(pk);**

G.J. Orr, R.J. Kwan, M. Doudar. Xplicit Computing, Inc. USA info@xplicitcomputing.com 

# PropertyKeys

## {Property} < {Property|Modifier} …

A Property-Key (PK) is the composite of a *property* tag, followed by optional *modifier* tags, enabling:

- Human and machine readable data tagging and retrieval.
- Ideal lexicographical sorting & searching (in-session)
- Compatibility comparison by string name (out-of-session)
- Dynamic I/O for algorithms that use modular substitution (in-code)

Using Property-Keys, applications can dynamically allocate memory for data scalars, spatial vectors, or tensors. Memory allocation for varying objects is automatically handled by the bound instruction and/or sequence as part of the larger program.

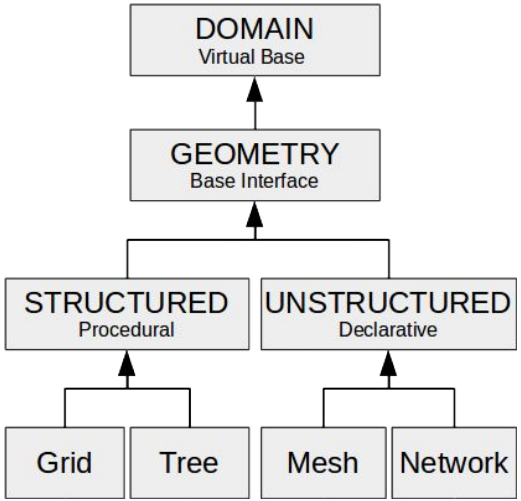
Concrete PK Examples

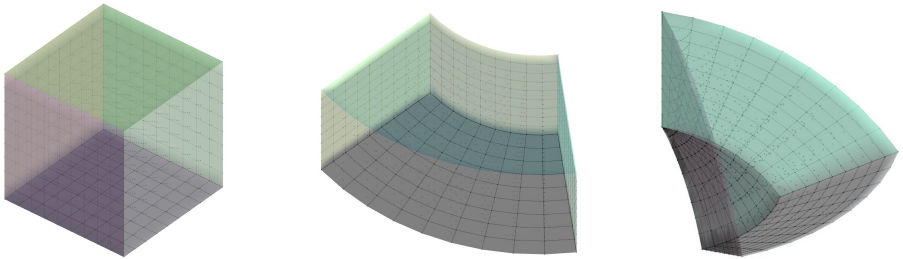
{Position}
{Temperature}
{Pressure}
{Temperature|Gradient}
{Pressure|Gradient}
{Mass|Density}
{Momentum|Density}
{Energy|Density}
{Charge|Density}
{Mass|Density|Residual}
{Mass|Density|Min}
{Sutherland|Coefficient}
{AnyProperty|Density}
…

# Geometry

## Messages::Topology

Geometry is an optional pointer to a local geometry, which is transformed to a global space instance using a global model matrix. This permits a geometry to be shared between one or more systems and greatly reduces memory consumption with large numbers of duplicates or patterns. The global model matrix is computed as the recursive product of parent local matrices, providing linear complexity scaling for refresh starting at the root system and performing similar breadth-first recursive refresh to all systems. Geometries are shared across systems and managed by the server application.



G.J. Orr, R.J. Kwan, M. Doudar. Xplicit Computing, Inc. USA info@xplicitcomputing.com

# XCOMPUTE: Algorithms and Instruction Sequences for CFD/FEA Multiphysics

## Implicit Mesh Generation

a) Solve SDF background grid

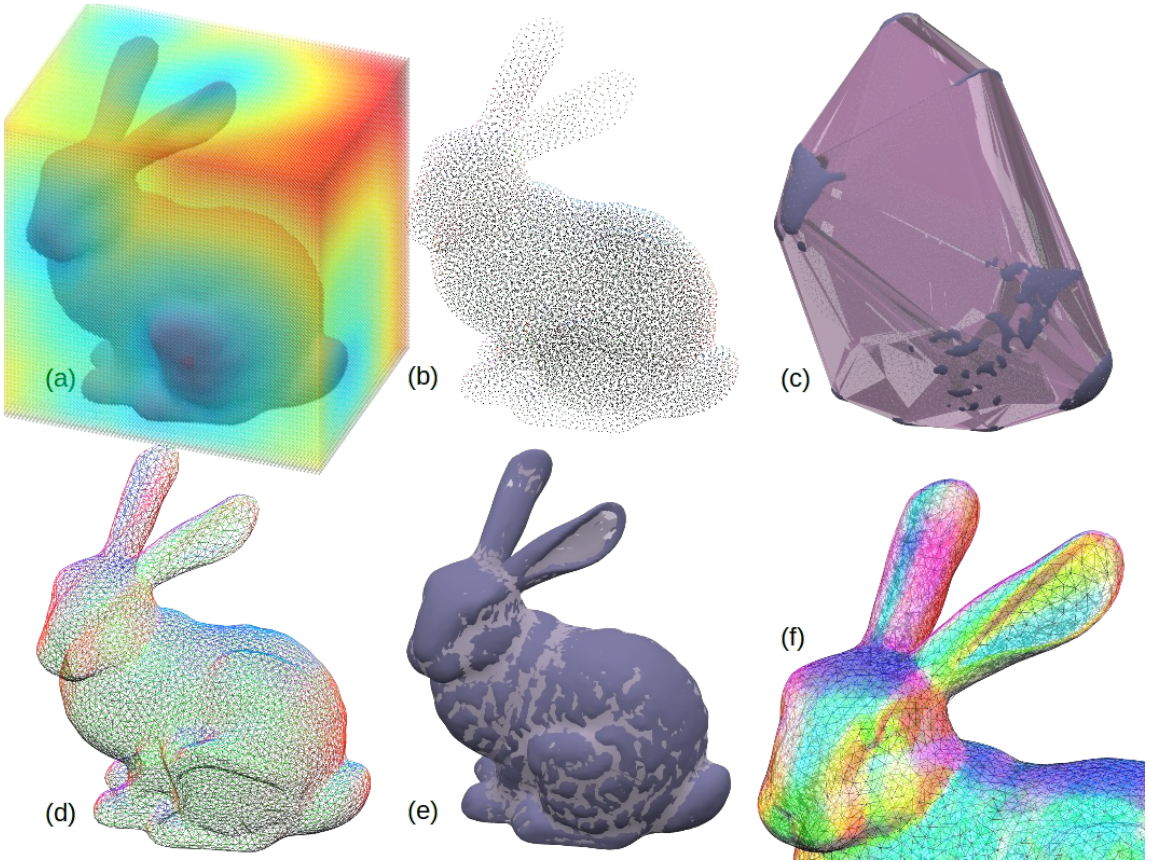
b) Inject nodes and anneal to valid SDF

c) Triangulate volumetric mesh convex hull

d) Unwrap volumetric mesh simplexes

e) Original surface and resulting volume

f) Close-up of unrefined mesh

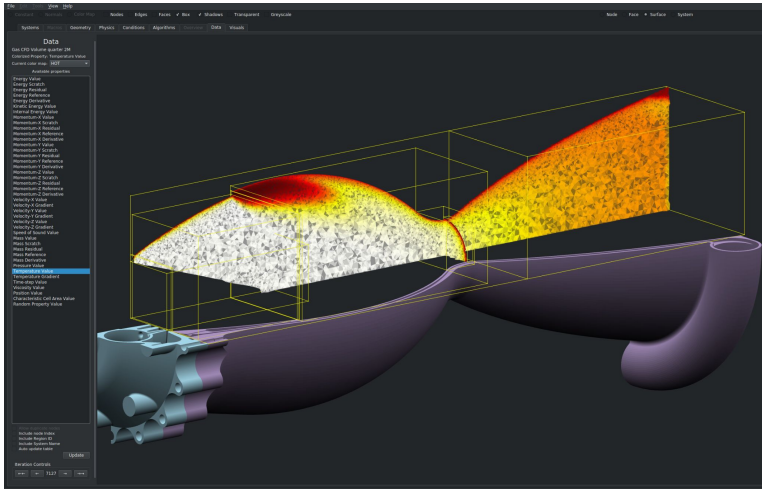



G.J. Orr, R.J. Kwan, M. Doudar. Xplicit Computing, Inc. USA info@xplicitcomputing.com
ICCFD11-2022-0802

# Physics

## Repeated Messages::Model

Physics is an optional pointer (address reference) to a physical model, which contributes algorithms to the system's solver in preparation for execution. As a model is defined to be a collection of algorithms to achieve an approximate result, physics is defined as the superposition of models, itself a type of model. Most physical models utilize algorithms to define a system's spatial transport and state; similar method families may be combined.



Operator T applied to the state defined by K degree-of-freedom ={1, 2, ..., K} quantities to yield some discrete change $\Delta\Phi$ or state $\Phi$ in geometric domain $\Omega$, and boundary conditions $\Phi o(\delta\Omega)$ and any additional arguments:

$$(\Delta)\Phi(\Omega) = T(\Phi(\Omega),\Phi o(\delta\Omega),...).$$

The (change of) state in the domain is equal to the transformation that occurs upon the state within the domain given some set of prescribed conditions (or mediating links to other systems) on the boundaries. The far-left ($\Delta$) expression is to indicate generality for explicit and implicit schemes.

G.J. Orr, R.J. Kwan, M. Doudar. Xplicit Computing, Inc. USA info@xplicitcomputing.com ICCFD11-2022-0802

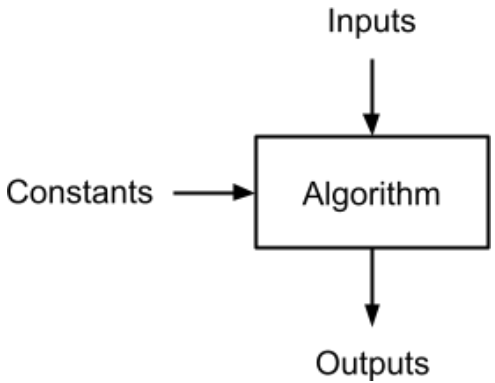
# Algorithm (verb)

## Messages::Function

An algorithm is an intrinsic operator (verb) - a reusable global callable function-class defining a procedure to achieve a desired numeric operation.

An algorithm is defined by its procedure (as code) and the argument types required for processing. Objects are not directly bound to an algorithm, for algorithms have a singleton-like pattern; they are referenced and invoked as needed within other algorithms, models, and solvers bound to instruction and argument objects - itself rarely made new or mutated. Property-key inputs and outputs can be specified upon algorithm construction, later interpreted by sequencing machinery to connect relevant data inputs/outputs in a chain. An algorithm's callable operator dispatches static host bytecode, and provides an analogous method to define dynamic OpenCL code fragments to be assembled and compiled into a device program at runtime.

***Requirements*** - A set of object types expected as bound arguments to execute the defined set of routines.

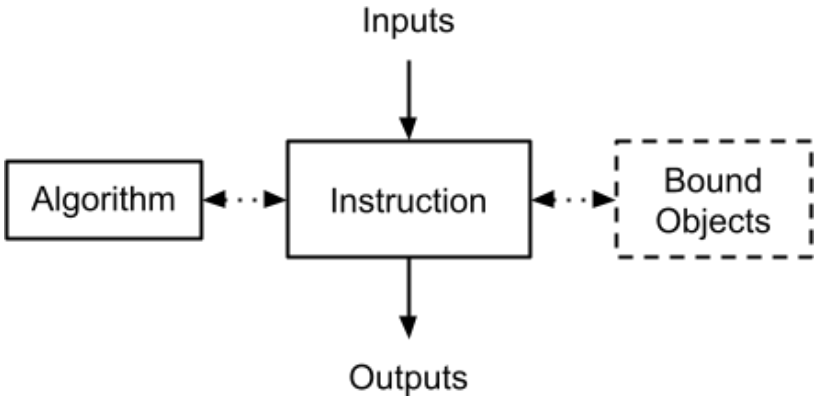


# Instruction

## Messages::Command

An Instruction is an executable instance comprising an algorithm bound to arguments.

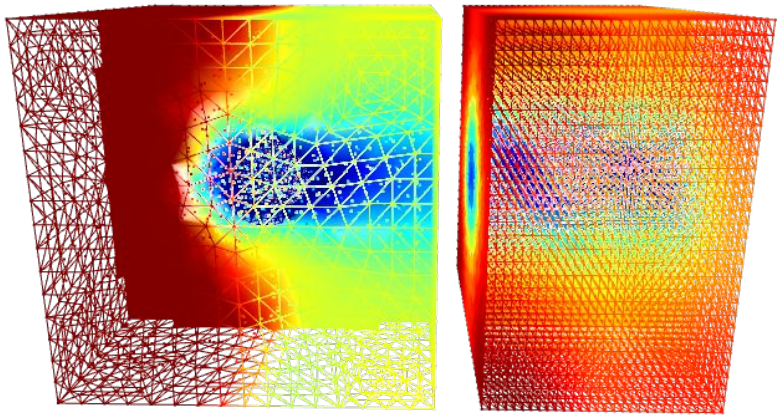
An instruction performs the operation of binding objects together in a temporary callable instance that can be invoked in a managing sequence. Inputs and outputs can be functionally queried, returning underlying PK inputs and outputs with relevant *AnyProperty* and *AnyModifier* substitutions (necessitating a Property-Key as an algorithm requirement). An instruction may be invoked individually with its callable operator, to be called from a sequence (and/or solver). This calling structure permits instructions to call either host functions or device kernel functions as part of a larger compute program. Any algorithmic preparation or post-processing can be included in optional before and after sequences.



# Condition

## Messages::Command

A condition is an instruction applied to temporal or spatial boundaries to provide numerical closure. Initial conditions are typically applied to systems as part of a preprocessor, while boundary conditions are applied to regions repeatedly within the main solver sequence. Conditions are bound to instructions and inserted into the sequence; upon building a solver the algorithms are consolidated into the proper execution sequence. Explicit methods often assert conditions on *system.data* entries, while implicit methods typically manipulate the *system.adjacency* matrix and/or state.



Coupling (contacts & links) are similar to conditions, but utilize a duplexed interpolation algorithm between regions. In this example, two systems are coupled permitting two-way information flow in the elliptic numerical domain.

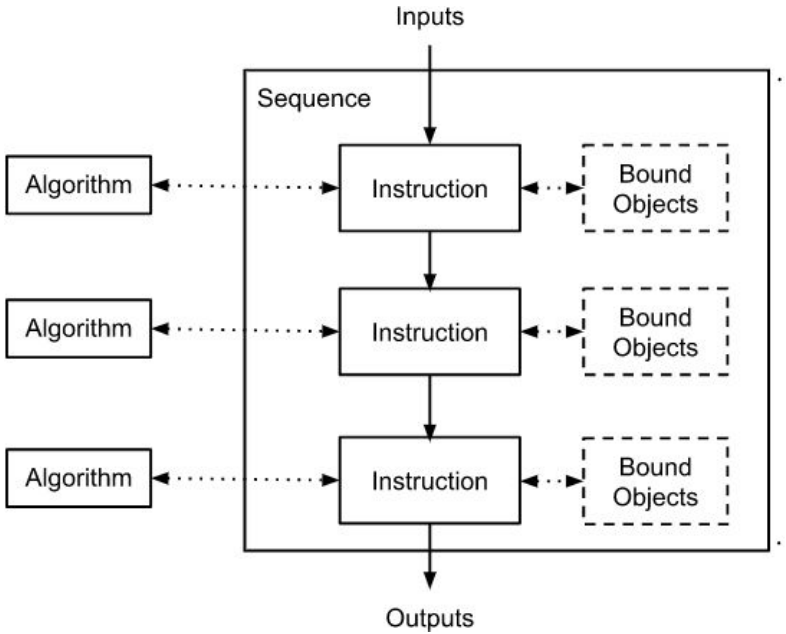
G.J. Orr, R.J. Kwan, M. Doudar. Xplicit Computing, Inc. USA info@xplicitcomputing.com
ICCFD11-2022-0802

# Sequence

## Repeated Messages::Command

A sequence is an executable container of instructions that assists in object binding and solver preparation.

Algorithms are hardcoded and immutable in static bytecode, but with the help of runtime containers, unique high-level behavior can be achieved via custom sequences to comprise physical models and numerical methods (aka. “solvers”). Together, the physical model and solver transform the PDE into a discrete ODE, typically integrating in spatial and temporal dimensions, respectively.



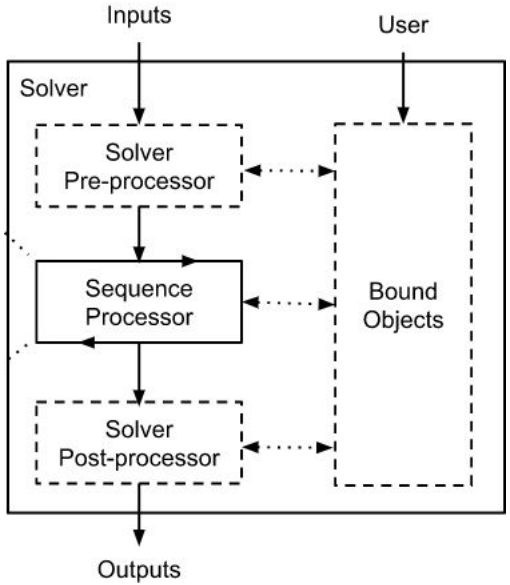

G.J. Orr, R.J. Kwan, M. Doudar. Xplicit Computing, Inc. USA info@xplicitcomputing.com ICCFD11-2022-0802

# Solver

## Messages::Script

A solver is a list of instructions to be executed as part of a workflow.

Typically, there is a single system per solver instance; solvers are bound and executed against their owning system. A solver usually defines a self-contained numerical process such as mesh generation, finite volume, or finite element methods using a main iterative sequence plus optional recursive pre and post processor stages. A solver contains faculties to compile a master sequence, assemble OpenCL code fragments, and generate a device compute program. One or more solvers can be listed to define a system's workflow.




| | |
|---|---|
| Explicit solvers | Solve-Tree, Runge-Kutta (FDM, FVM), Stream-Collide (LBM) |
| Implicit solvers | Linear-Steady (FEM), Linear-Euler (FEM), Linear-Newmark (FEM) |

# XCOMPUTE: Algorithms and Instruction Sequences for CFD/FEA Multiphysics

## Example - perhaps the most complex geometric algorithm? Implicit mesh generation! 7-level DAG:

```
L1.  resize_background(MeshControl, GridGeometry) : host
L1.  sdf_solve() : host  -> outputs={SignedDistance}
L2.      initialize_sdf(GridGeometry, DataReference, SurfaceSet) : host  -> outputs={SignedDistance}
L2.      sdf_sign_correction(GridGeometry, DataReference, SurfaceSet) : host  -> inputs={SignedDistance} -> outputs={SignedDistance}
L2.      sdf_fast_marching_method(GridGeometry, DataReference) : host  -> inputs={SignedDistance} -> outputs={SignedDistance}
L1.  grid_gradient(PropertyKey, GridGeometry, DataReference) : host  -> inputs={SignedDistance} -> outputs={SignedDistance|Gradient}
L1.  size_function_from_sdf(MeshControl, GridGeometry, DataReference) : host  -> inputs={SignedDistance} -> outputs={Element|Size}
L1.  sdf_point_injection(MeshControl, GridGeometry, MeshGeometry, DataReference, SurfaceSet) : host  -> inputs={SignedDistance, Element|Size}
L1.  constrain_with_domain(GridGeometry, MeshGeometry) : host
L1.  constrain_with_sdf(MeshControl, GridGeometry, MeshGeometry, DataReference) : host  -> inputs={SignedDistance, SignedDistance|Gradient}
L1.  remove_duplicate_nodes(MeshGeometry) : host
L0.  create_mesh(System, MeshControl) : host
L2.      gflip3d(TriangulationData, MeshGeometry) : host
L3.          initialize_triangulation(TriangulationData) : host
L3.          add_infinity_point(TriangulationData) : host
L3.          initialize_predicates(TriangulationData) : device
L3.          make_first_tetrahedra(TriangulationData) : device
L3.          initialize_points_fast(TriangulationData) : device
L3.          initialize_points_exact(TriangulationData) : device
L3.          cleanup_initial_construction(TriangulationData) : host
L4.              split_tetrahedra(TriangulationData) : host
L5.                  vote_for_point(TriangulationData) : device
L5.                  pick_winner_point(TriangulationData) : device
L5.                  negate_inserted_verts(TriangulationData) : device
L5.                  mark_tet_empty(TriangulationData) : device
L5.                  split_points_fast(TriangulationData) : device
L5.                  split_points_exact(TriangulationData) : device
L5.                  split_tetra(TriangulationData) : device
L3.          split_and_flip(TriangulationData) : host
L6.                  update_flip_trace(TriangulationData) : device
L5.                  relocate_all(TriangulationData) : host
L6.                      relocate_points_fast(TriangulationData) : device
L6.                      relocate_points_exact(TriangulationData) : device
L4.              do_flipping_loop_fast(TriangulationData) : host
L6.                  check_triangulation_fast_fast(TriangulationData) : device
L5.                  dispatch_check_triangulation(TriangulationData) : host
L6.                      check_triangulation_fast_sos(TriangulationData) : device
L6.                      check_triangulation_exact(TriangulationData) : device
L5.                  mark_rejected_flips(TriangulationData) : device
L5.                  allocate_flip_23_slot(TriangulationData) : device
L5.                  flip_tetras(TriangulationData
L5.                  update_opp(TriangulationDa
L4.              mark_special_tets(TriangulationData) : devi
L6.                  update_flip_tr
L5.              relocate_all(TriangulationDat
L6.                  relocate_point
L6.                  relocate_point
L4.              do_flipping_loop_exact(TriangulationData) :
L6.                  check_triangu
L5.                  dispatch_check_triangulatior
L6.                  check_triangu
L6.                  check_triangu
L5.                  mark_rejected_flips(Triangul
L5.                  allocate_flip_23_slot(Triangu
L5.                  flip_tetras(TriangulationData
L5.                  update_opp(TriangulationDa
L5.                  update_flip_trace(Triangulati
L4.              relocate_all(TriangulationData) : host
L5.                  relocate_points_fast(Triangul
L5.                  relocate_points_exact(Triang
L3.          output_triangulation(TriangulationData, MeshGeometry) :
L4.              compact_tetras(TriangulationData) : host
L5.                  collect_free_slots(Triangulati
L5.                  make_compact_map(Triangu
L5.                  compact_tets(TriangulationD
L4.              gather_failed_verts(TriangulationData) : dev
L3.          star_splaying(TriangulationData) : host
L3.          finish_triangulation(TriangulationData, MeshGeometry) : h
L2.      mark_cells_by_sdf(MeshControl, GridGeometry, MeshGeometry, DataRef
L2.      mark_cells_by_quality(MeshGeometry) : host
L2.      build_interior_edges(MeshGeometry) : host
L1.  equalization_loop(MeshControl, MeshGeometry) : host
L2.      force_equilibrium(DeviceVector<double>, GridGeometry, MeshGeometry
L2.      constrain_with_domain(GridGeometry, MeshGeometry) : host
L2.      constrain_with_sdf(MeshControl, GridGeometry, MeshGeometry, DataRe
L1.  clean_cells_by_sdf(MeshControl, GridGeometry, MeshGeometry, DataReference) : host
L1.  clean_cells_by_quality(MeshGeometry) : host
L1.  reset_cell_ids(MeshGeometry) : host
L1.  build_surfaces_from_faces(MeshGeometry) : host
L1.  check_mesh(System, MeshControl, GridGeometry, DataReference) : host  -> inputs={Sign
```

G.J. Orr, R.J. Kwan, M. Doudar. Xplicit Computing, Inc. USA info@xplicitcomputing.com

ICCFD11-2022-0802

# Configuration

## Messages::Variables

To set default runtime parameters, each *xcompute* session can load a simple human-editable *.cfg file at launch and/or throughout runtime. In all cases, the application expects at least one config file in the local execution or install path.

Example *compute.cfg* ingested by xcompute-server at runtime:

```
### XCOMPUTE CONFIG ### Sat Aug  1 23:14:50 2020
OptimizeJIT=true
threads=12
Iterations=1
HostOnly=false
debug=true
DefaultDevice=1
Resolution=1000000
ElementCount=100000
```

# Execution

## ./xcompute-server

There are three known ways to define and execute an *xcompute* process:

1) shell command line (in beta)
2) protobuf-generated messages I/O
3) interactive server-client protocol

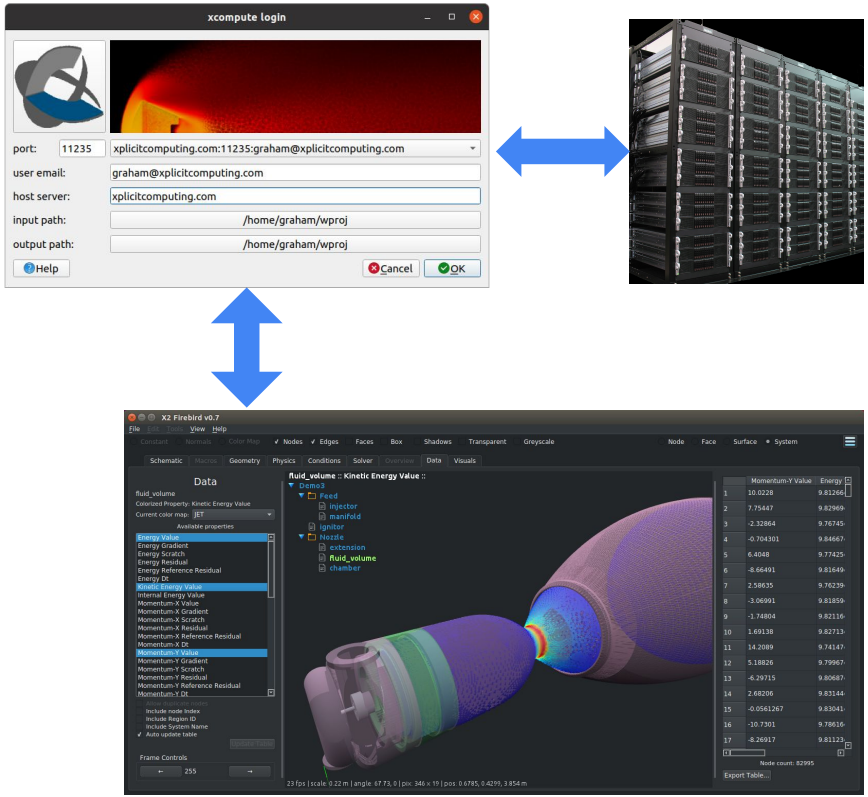
G.J. Orr, R.J. Kwan, M. Doudar. Xplicit Computing, Inc. USA info@xplicitcomputing.com 

# Interactive Session

To facilitate network transmission, the *xcompute* transport protocol (XCTP) over TCP/IP is used, with the service name ‘xcompute’ on IANA-reserved port 11235.

As the central nervous system of a distributed computing network, the *xcompute* protocol comprises a 32-bit hash-encoded function call followed by variadic arguments that are encoded Protobuf messages and/or standard primitives. The *xc::io* transport layer packages payloads into a transmission defining the expected types and sizes.

The intended function is decoded, parameters deserialized, and the function call is dispatched to an appropriate lambda overload, using a Y-combinator to facilitate the recursion on function arguments, with much of the heavy-lifting leveraging code-generation.




G.J. Orr, R.J. Kwan, M. Doudar. Xplicit Computing, Inc. USA info@xplicitcomputing.com

ICCFD11-2022-0802

# Messages I/O

| | |
|---|---|
| vector.proto | XCO numeric data object arrays using packed arena allocation |
| spatial.proto | XCG geometry/topology of elements and regions discretization |
| concept.proto | XCS system case setup, models, parameters, associations |
| meta.proto | XCM meta-data and user-graphics media for a specific system |

## C/C++ Example

```
#include "vector.pb.h"

Messages::Vector64 msg; // empty message

msg.set_name("Position|Value"); // set the pk

msg.set_components(3);

msg.add_values(pos.x);  // push back x value
msg.add_values(pos.y);  // push back y value
msg.add_values(pos.z);  // push back z value
```

## Python Example

```
import vector_pb2 as vector

msg = vector.Vector64()

msg.name = "Position|Value"

msg.components = 3

msg.values.append(pos.x)
msg.values.append(pos.y)
msg.values.append(pos.z)
```

Note: {AnyProperty} and {AnyProperty|Value} are equivalent, but we mention the “Value” modifier here for instructive purposes.

In Conclusion,

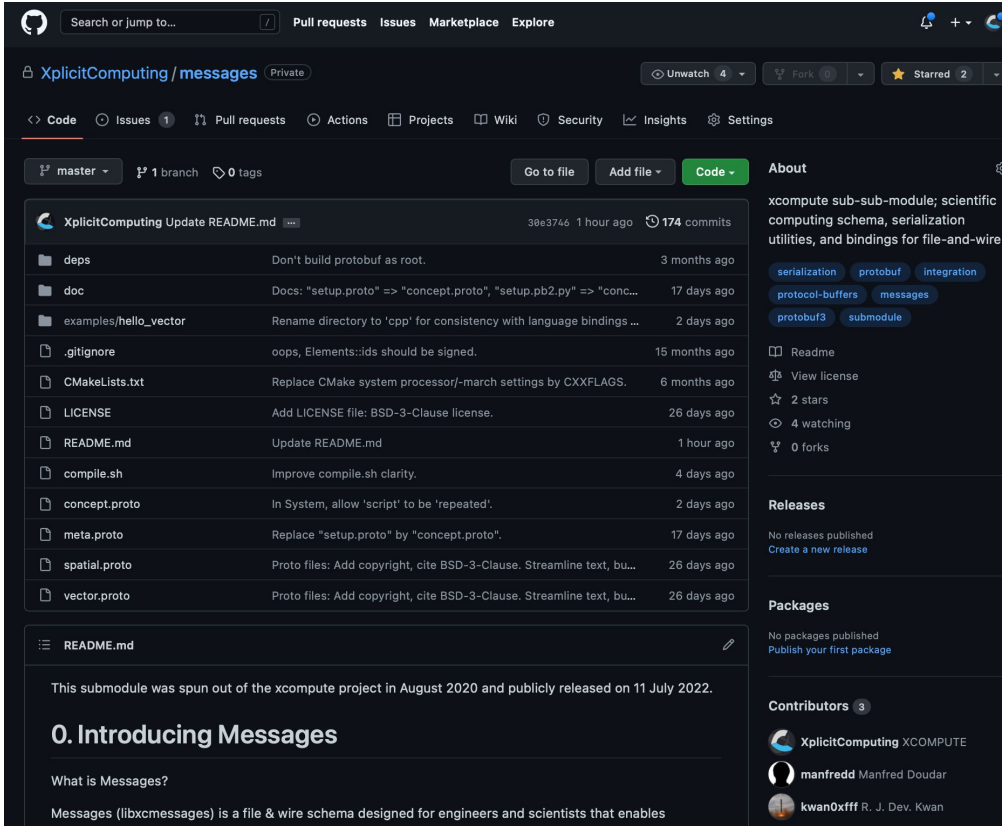
# Messages™

## Released Today!

https://xplicitcomputing.com

Interoperability between CAE projects and teams!

Free & Open BSD-3 license




G.J. Orr, R.J. Kwan, M. Doudar. Xplicit Computing, Inc. USA info@xplicitcomputing.com

ICCFD11-2022-0802