Tapas:
Programming Framework for Hierarchical Particle Methods

Keisuke Fukuda[*], *Motohiko Matsuda*[+], *Naoya Maruyama*[+], *Rio Yokota*[*], *Kenjiro Taura*[¶], *Satoshi Matsuoka*[*]

[*] Tokyo Institute of Technology [+]RIKEN AICS [¶]University of Tokyo
fukuda@matsulab.is.titech.ac.jp

Highlights:

- ✓ Implicitly parallel C++ programming framework for tree-based particle simulations
- ✓ Algorithm-independent & automatic detection of irregular data access pattern and communication
- ✓ Competitive performance & scaling compared to manually-tuned implementation

Motivation

What are hierarchical particle methods?

- Direct methods for N-body algorithms is $O(N^2)$
- Hierarchical methods reduce the complexity to O(N logN) or O(N)
- Typical examples include Barnes-Hut algorithm and the Fast Multipole Method (FMM)

Why do they matter?

- Higher arithmetic intensity and communication locality
- Important applications : cosmology, molecular dynamics, H-matrix, BEM (Boundary element methods)

Why are they hard to implement?

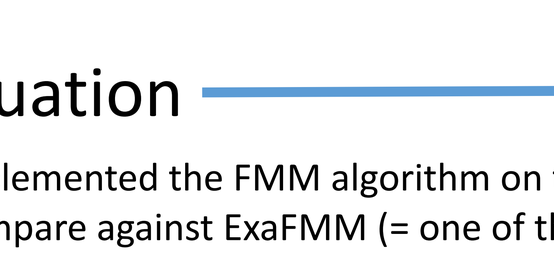
- **It is hard to implement correct / distributed / fast code**, due to their irregular data structure and access pattern.
- Most implementations are written by specialists and build from scratch (C++,Fortran & MPI) and little code & effort is shared.

We need a framework !!

- Separation of concern between computational scientists and computer scientists.
- “Tapas” is under development to be productive, high performance and practical.

What is the challenge?

- Irregular data access pattern across the tree, which is distributed over separate memory spaces
- Access pattern is algorithm-dependent and dynamic.
- How to encapsulate the complexity in the framework?



Framework Design

Core APIs:

- Data types:
  - `Body`, `Cell` : represents bodies and tree nodes
  - Programmers can plug-in application-specific data as body attributes and cell attributes
- Tree traversal operations:
  - Application-specific operations and computations are written as C++ template functor.
  - The functors are instantiated as the inspector and executor
- APIs to be used within functors:
  - `Map` : recursively apply a functor to children of a cell
  - `Reduce` : update values of the parent cell from a child.

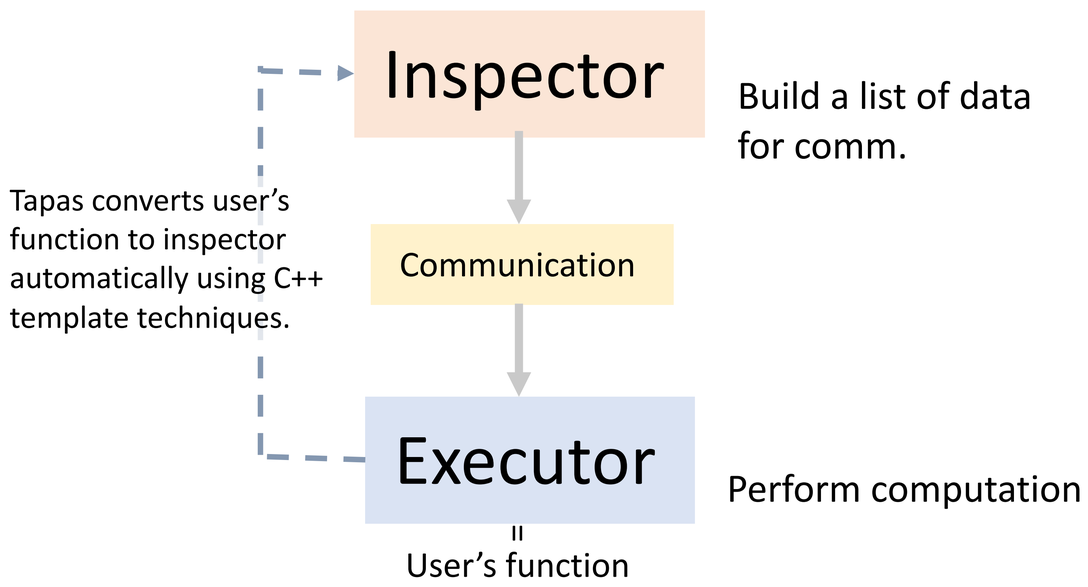
Code examples:

```
struct TraversePostOrder {  // Post-order(upward) tree traversal
functor
  template<typename Cell>
  void operator()(Cell &p, Cell &c) {
    ...; // do something using c;
    Map(TraversePostOrder(), c.SubCells());
    Reduce(p. p.attr().foo, val, sum);        }};

struct DualTreeTraversal {  // Dual tree traversal functor
  template<typename Cell>
  inline void operator()(Cell C1, Cell C2) {
    ...; // do something using C1 and C2
    Map(DualTreeTraversal(), Product(C1.SubCells(),
                                     C2.SubCells()); }};
```

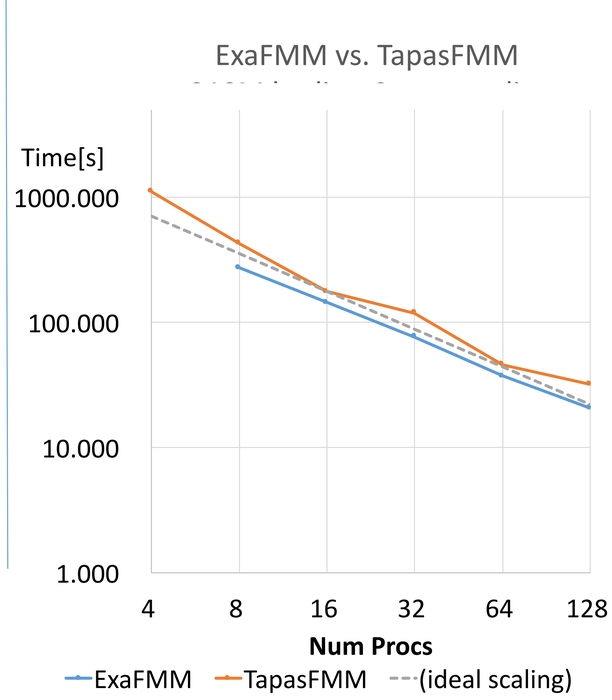
How to overcome the most challenging part: irregular data access across distributed memory space?

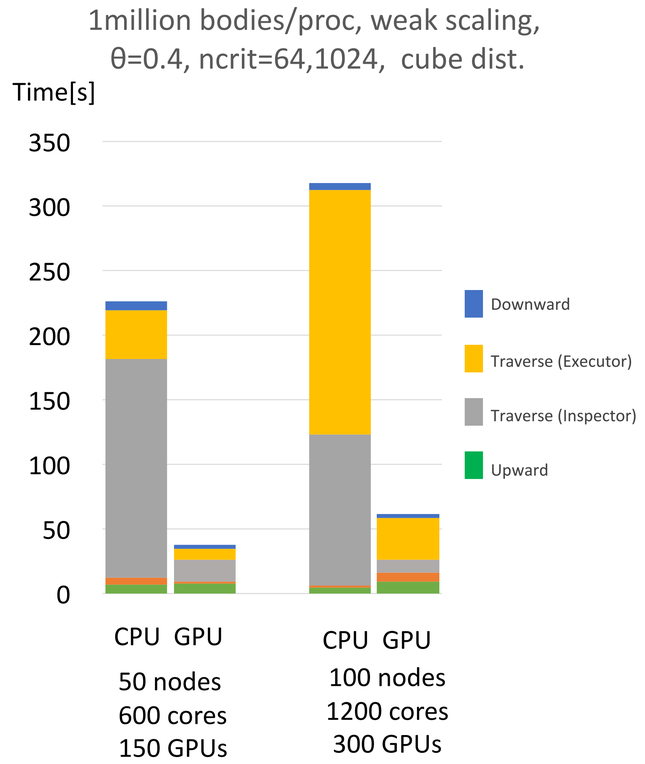
- Tapas employs inspector/executor model
- Inspector traverses the tree and build a list of necessary remote data for executor, and executor performs actual computation. Inspector is automatically generated.



Evaluation

- Implemented the FMM algorithm on top of Tapas (TapasFMM)
- Compare against ExaFMM (= one of the fastest manual implementation) : 64%-81% of ExaFMM





This project was partially supported by JST, CREST through itsresearch program: ``Highly Productive, High Performance ApplicationFrameworks for Post Petascale Computing'' and ``Advanced CoreTechnologies for Big Data Integration'', as well as JSPS KAKENHI GrantNumber 16H02827.