Pacific Northwest
NATIONAL LABORATORY

SHAD: Productive Programming for High-Performance Systems in Standard C++

Vito Giovanni Castellana

Senior Computer Scientist, PNNL

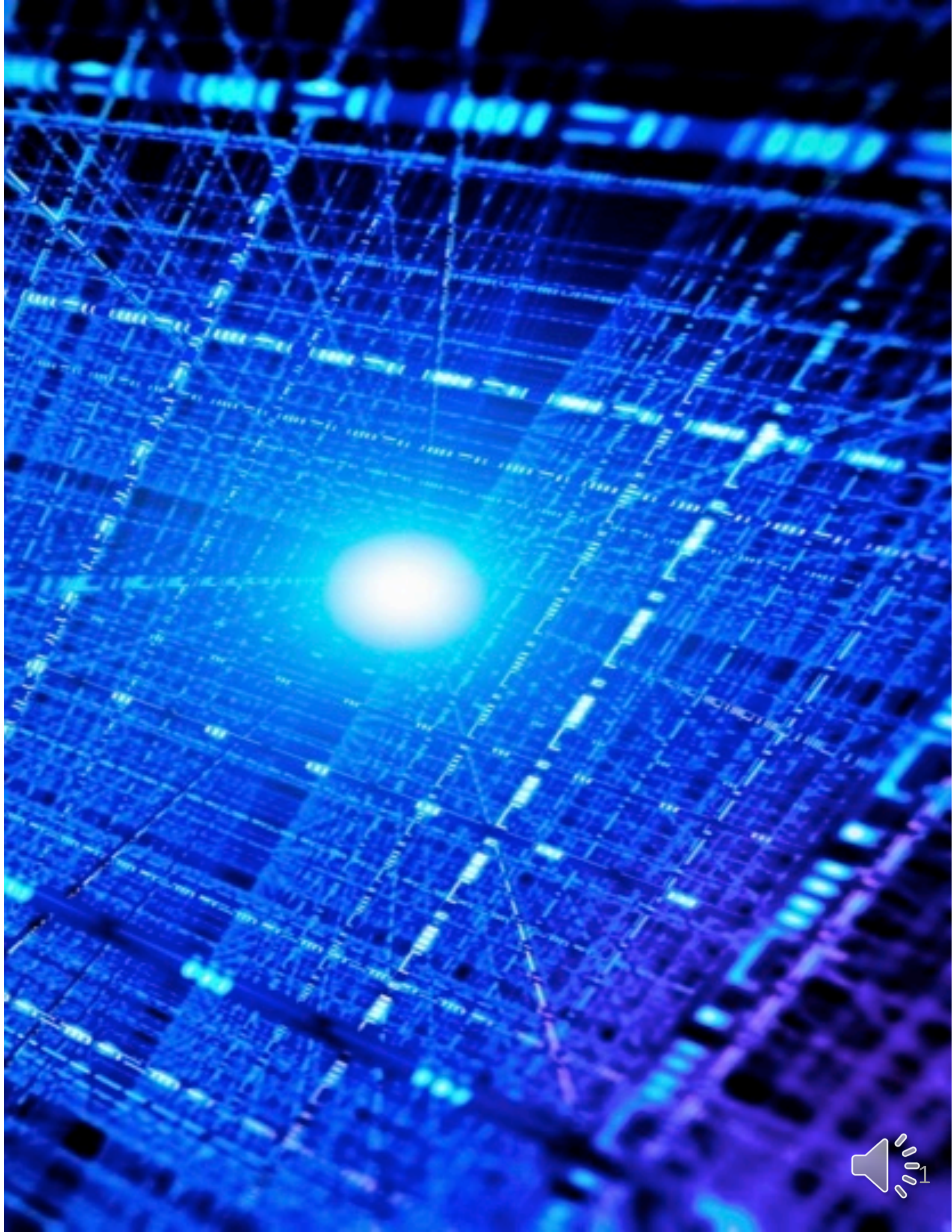
Maurizio Drocco, IBM Research

Marco Minutoli, PNNL

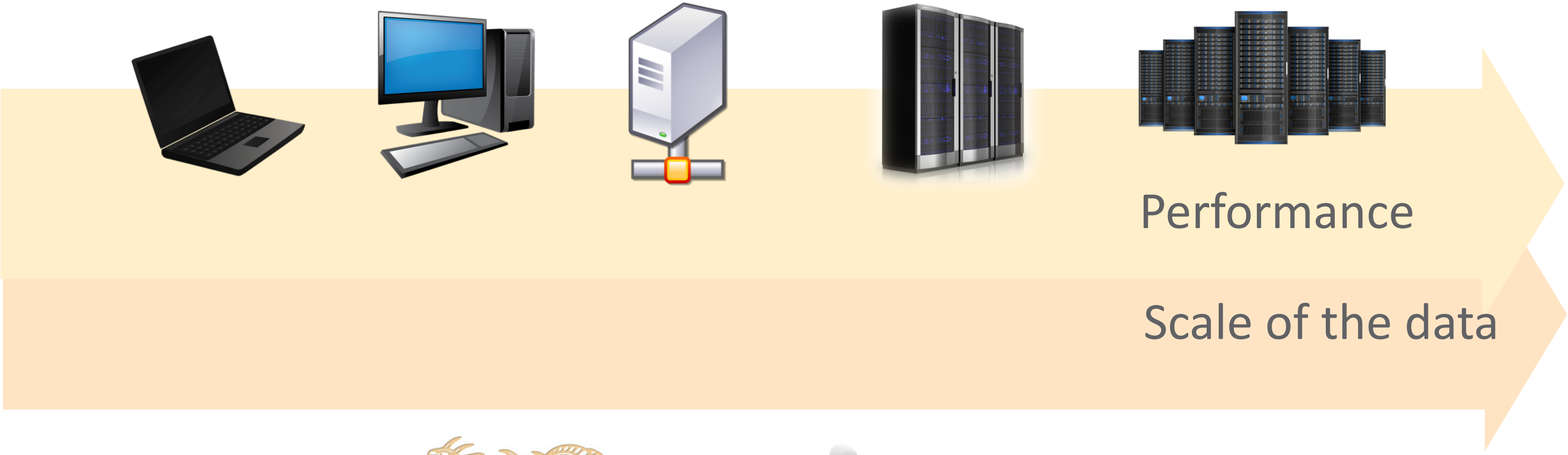
John Feo, PNNL

U.S. DEPARTMENT OF ENERGY BATTELLE

PNNL is operated by Battelle for the U.S. Department of Energy



Performance, Portability, and Productivity



Performance

Scale of the data

Teaser

Tackling the Stock Market with HPC – it’s all about money

- Problem: find the **highest price** in a set of **stock options**
  - Input: ~134 millions of stock-option descriptors
  - Output: the max-priced option
- Black-Scholes formula
  - Input: a stock-option descriptor
  - Output: its price
  - 127-line black-box C function, plenty of <`math.`h> stuff
  - [C. Bienia *et al.*, PARSEC Benchmark Suite, PACT,08]

“Old school” C++

```
price_t max_price(std::array<option_t, n> &a) {
  auto m = std::numeric_limits<price_t>::min();
  for (auto it = a.begin(); it != a.end(); ++it)
    m = std::max(res, blck_schls(*it));
  return m;
}
```

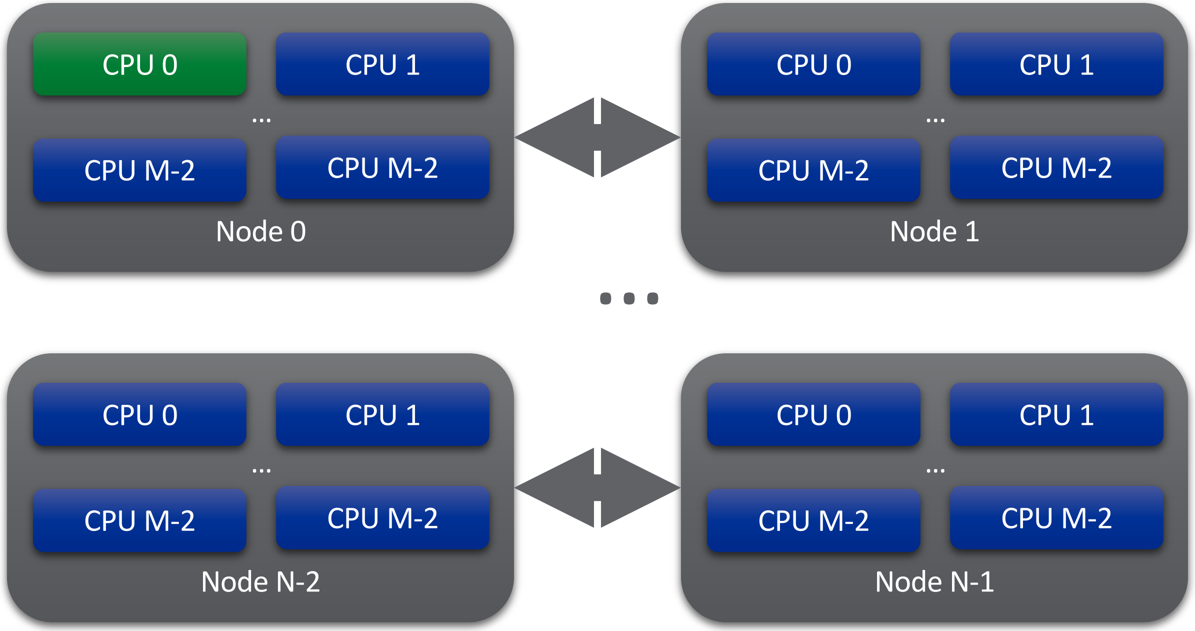
STL Containers + STL Iterators + for loop

“Old-school” C++ – HPC?

```
price_t max_price(…) {…}
```

STL Containers + STL Algorithms

- 1x CPU core
- Intel(R) Xeon(R) CPU @ 2.80GHz
- gcc 9.1



Node 0: CPU 0, CPU 1, …, CPU M-2, CPU M-2

Node 1: CPU 0, CPU 1, …, CPU M-2, CPU M-2

…

Node N-2: CPU 0, CPU 1, …, CPU M-2, CPU M-2

Node N-1: CPU 0, CPU 1, …, CPU M-2, CPU M-2

8.5 millions
options/sec

Modern(ish) C++

```
price_t max_price(std::array<option_t, n> &a) {
 std::array<price_t, n_options> p;
 std::transform(a.begin(), a.end(), p.begin(),
blck_schls);
 return *std::max_element(p.begin(),
```

STL Containers + STL Algorithms

Modern C++: execution policies may be our friends

```
price_t max_price(std::array<option_t, n> &a) {
 std::array<price_t, n_options> p;
 std::transform(std::execution::seq,
  a.begin(), a.end(), p.begin(), blck_schls);
 return *std::max_element(std::execution::seq,
  p.begin(), p.end());
}
```

STL Execution Policies

Modern C++: execution policies ARE our friends!

```
price_t max_price(std::array<option_t, n> &a) {
 std::array<price_t, n_options> p;
 std::transform(std::execution::par,
  a.begin(), a.end(), p.begin(), blck_schls);
 return *std::max_element(std::execution::par,
  p.begin(), p.end());
}
```

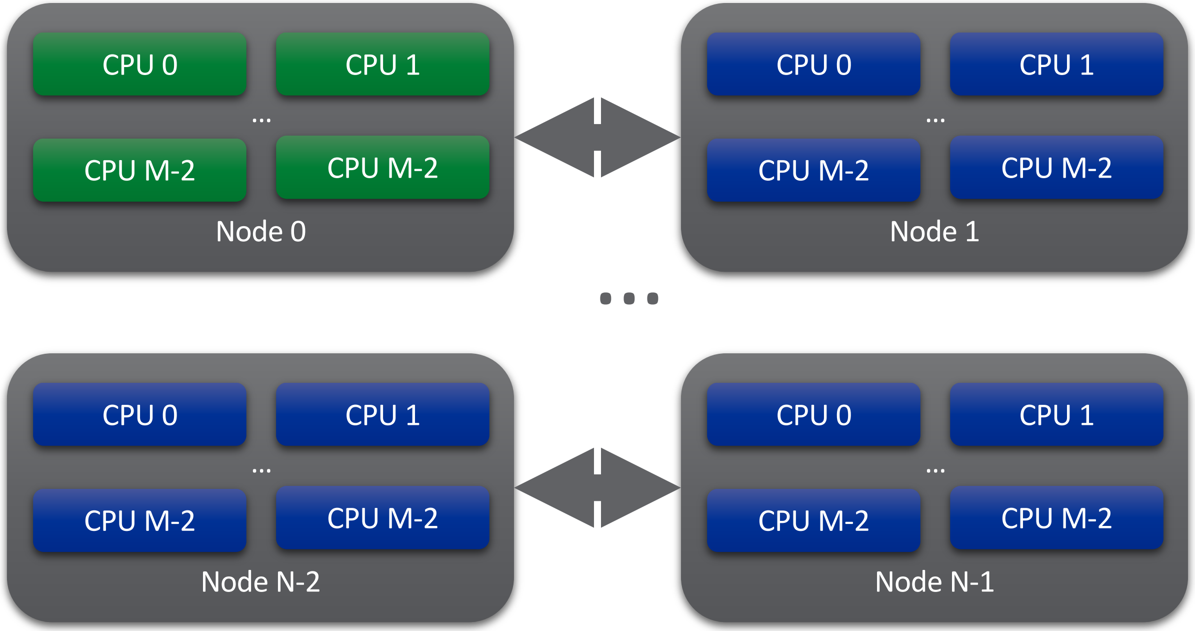
Parallel STL

Modern C++ – HPC?

```
price_t max_price(…) {…}
```

Parallel STL

- 10-core Socket
- Intel(R) Xeon(R) CPU @ 2.80GHz
- gcc 9.1



72.9 millions

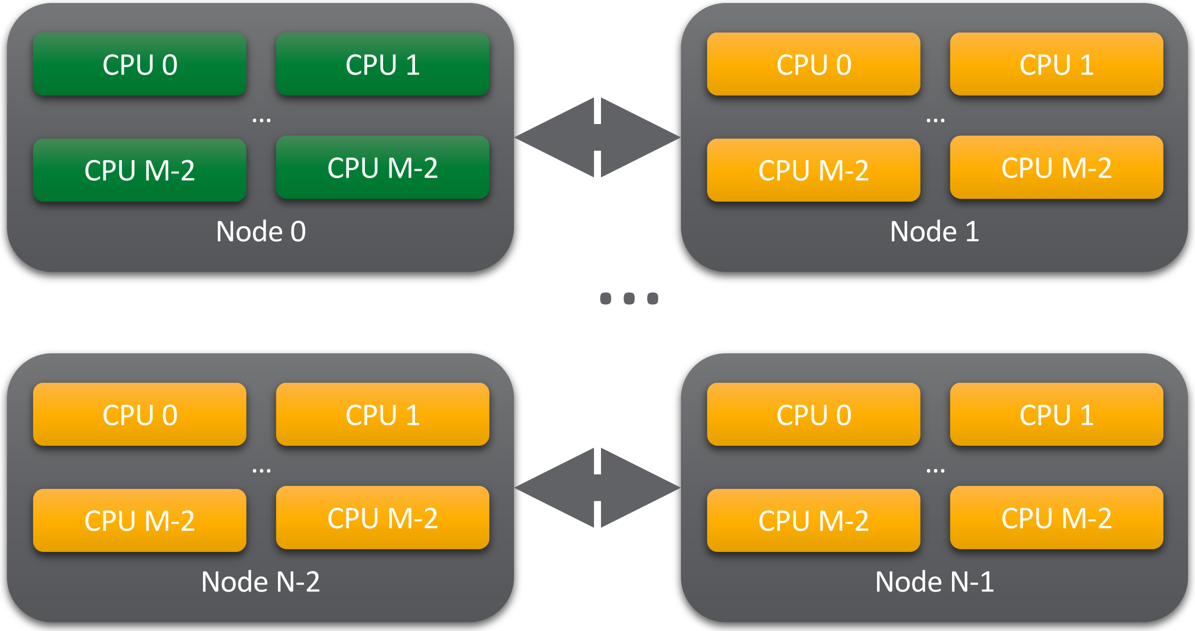
options/sec

~8.5x speedup

What about the other N-1 nodes?

```
price_t max_price(…) {…}
```

?



Pacific Northwest
NATIONAL LABORATORY

SCALABLE
HIGH-PERFORMANCE
ALGORITHMS &
DATA-STRUCTURES

https://github.com/pnnl/SHAD

Here comes the SHAD!

```
price_t max_price(shad::array<option_t, n> &a) {
 shad::array<price_t, n_options> p;
 shad::transform(shad::execution::par,
  a.begin(), a.end(), p.begin(), blck_schls);
 return *shad::max_element(shad::execution::par,
  p.begin(), p.end());
}
```

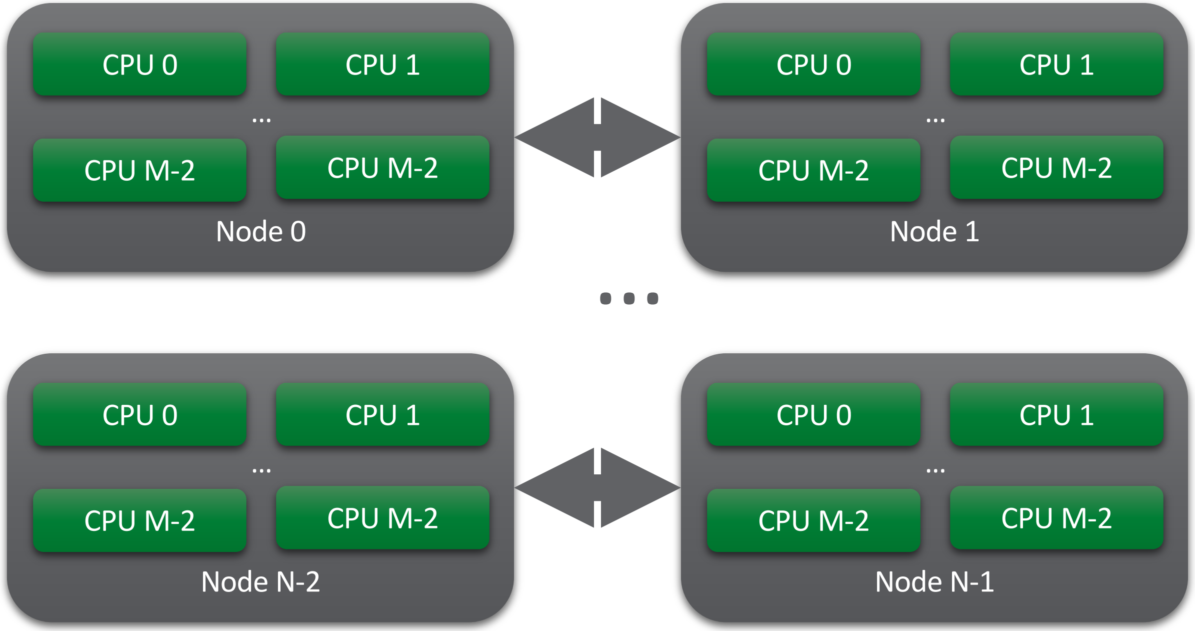
SHAD-powered Distributed STL

Here comes the SHAD! – HPC!

```
price_t max_price(…) {…}
```

SHAD-powered Distributed STL

- 16x 10-core sockets
- Intel(R) Xeon(R) CPU @ 2.80GHz
- gcc 9.1



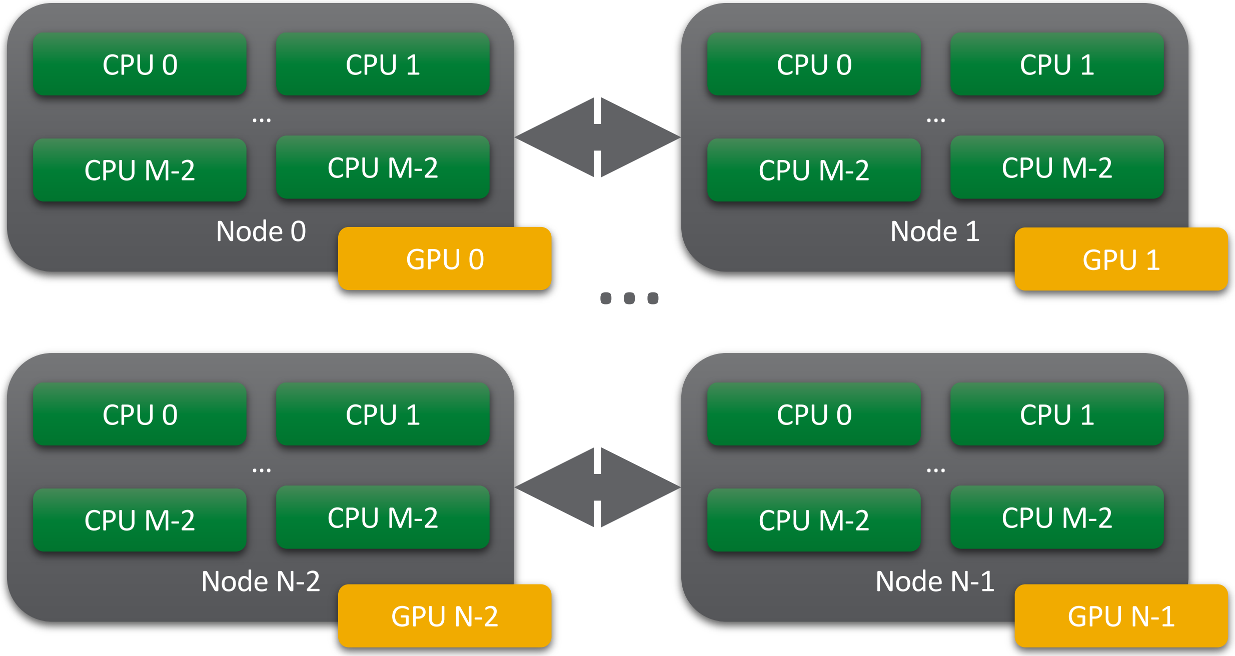
706.7 millions
options/sec

~82.5x speedup
vs plain STL

And what about… The GPUs?!

```
price_t max_price(…) {…}
```

?


CPU 0
CPU 1
…
CPU M-2
CPU M-2
Node 0
GPU 0

CPU 0
CPU 1
…
CPU M-2
CPU M-2
Node 1
GPU 1

…

CPU 0
CPU 1
…
CPU M-2
CPU M-2
Node N-2
GPU N-2

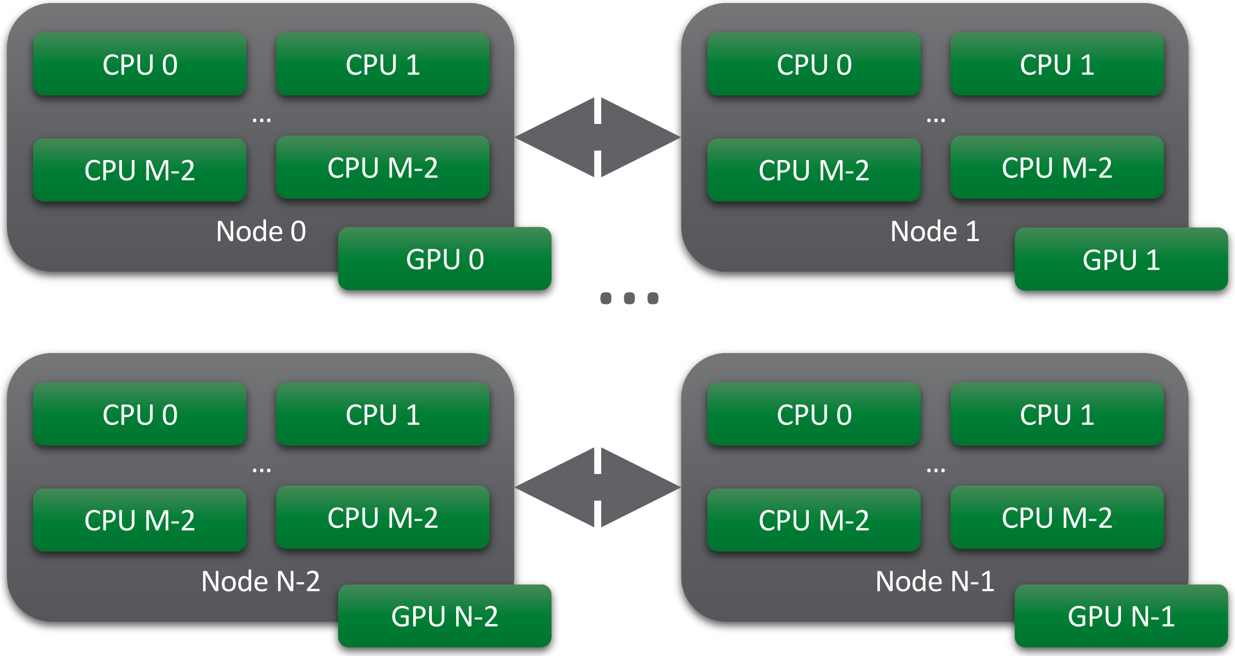
CPU 0
CPU 1
…
CPU M-2
CPU M-2
Node N-1
GPU N-1

Here comes the SHAD again! – HPC!!

```
price_t max_price(…) {…}
```

SHAD-powered Distributed STL

- 4x GPU equipped cluster nodes
- Intel(R) Xeon(R) CPU @ 2.80GHz
  + Nvidia Tesla GPU
- gcc 9.1 + nvcc (CUDA toolkit 9.2)



CPU 0 CPU 1 … CPU M-2 CPU M-2 Node 0 GPU 0

CPU 0 CPU 1 … CPU M-2 CPU M-2 Node 1 GPU 1

…

CPU 0 CPU 1 … CPU M-2 CPU M-2 Node N-2 GPU N-2

CPU 0 CPU 1 … CPU M-2 CPU M-2 Node N-1 GPU N-1

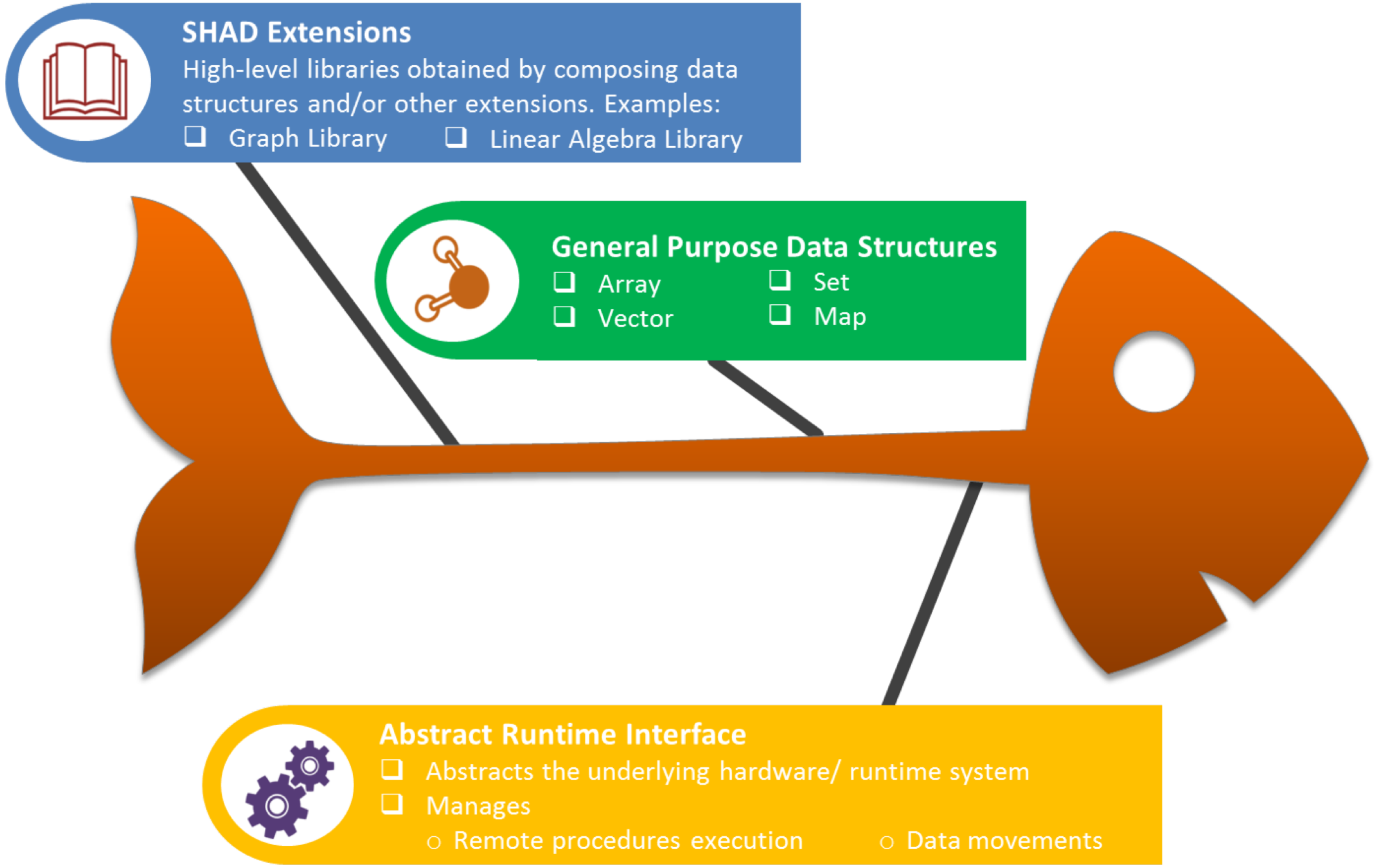
~5 Billions
options/sec

~585x speedup
vs plain STL

How did we get there?

Let’s take a closer look

SHAD Design Overview



SHAD Extensions

High-level libraries obtained by composing data structures and/or other extensions. Examples:

- □ Graph Library
- □ Linear Algebra Library

General Purpose Data Structures

- □ Array
- □ Vector
- □ Set
- □ Map

Abstract Runtime Interface

- □ Abstracts the underlying hardware/ runtime system
- □ Manages
  - ○ Remote procedures execution
  - ○ Data movements

Abstract Runtime Interface: main concepts

Machine Abstraction

- Locality
  - Entity in which memory is directly accessible
  - Examples: node in a cluster, core, NUMA domain
- Task
  - Basic unit of computation
  - Can be executed on any locality
  - Can be asynchronous
- “*Handles*”
  - Identifiers for spawning activities
  - Used to check for task completion
  - Multiple tasks may be associated to the same handle -> task groups

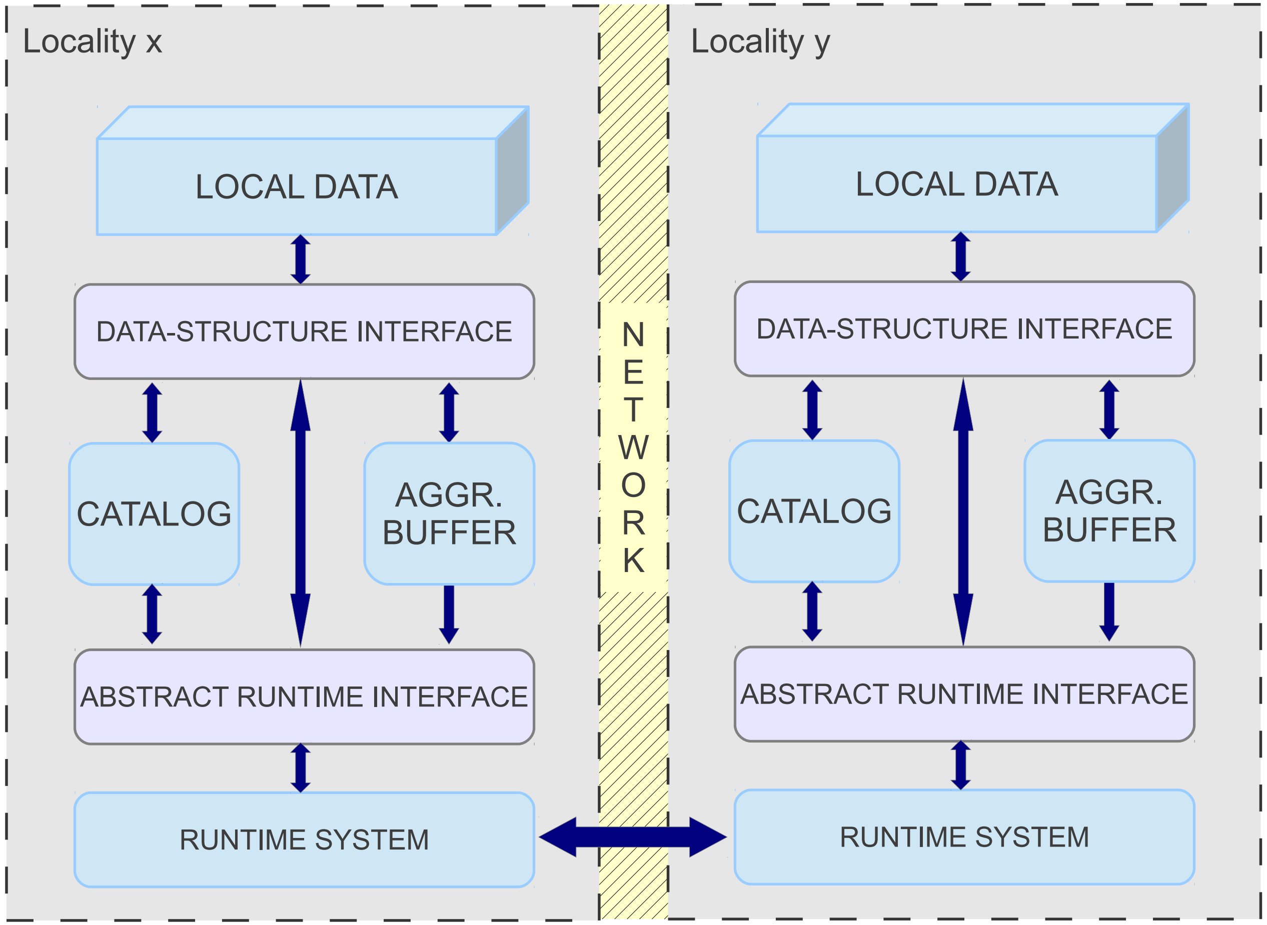
Abstract Runtime Interface Mappings

- Plain C++
  - Fast prototyping
- PNNL’s Global Memory and Threading (GMT) library
  - Targets commodity clusters
  - Available at **https://github.com/pnnl/gmt**
- Intel’ Threading Building Blocks (TBB)
  - Targets shared memory systems
    - … these may include your laptop ☺
- PNNL’s ARTS
  - Under development under the HIVE DARPA project
  - Available at **https://github.com/pnnl/ARTS**
  - SHAD-mapping not yet available on the repo
- Other mappings coming soon!

General Purpose Algorithms and Data-structures

- Include: array, vector, unordered set and map
- They "look like" STL, but they
  - Can be distributed on several localities
    - High capacity (TB+ scale data)
  - Are thread safe
  - Can be modified and accessed in parallel
    - High performance
  - Automatically manage synchronization and data-movements

Data-structures design template



SHAD extensions

- Higher-level or domain specific libraries
- Built on top of the General Purpose library
- Can be composed, to obtain application-specific libraries
  - High flexibility
  - *Evolving* framework

- Examples of SHAD extensions
  - Attributed Graph Lib (Prototype Available)
  - Linear Algebra Lib (Ongoing Work)
  - Machine Learning Lib (Future Work)

STL Interfaces

From STL *inspired* to STL compliant

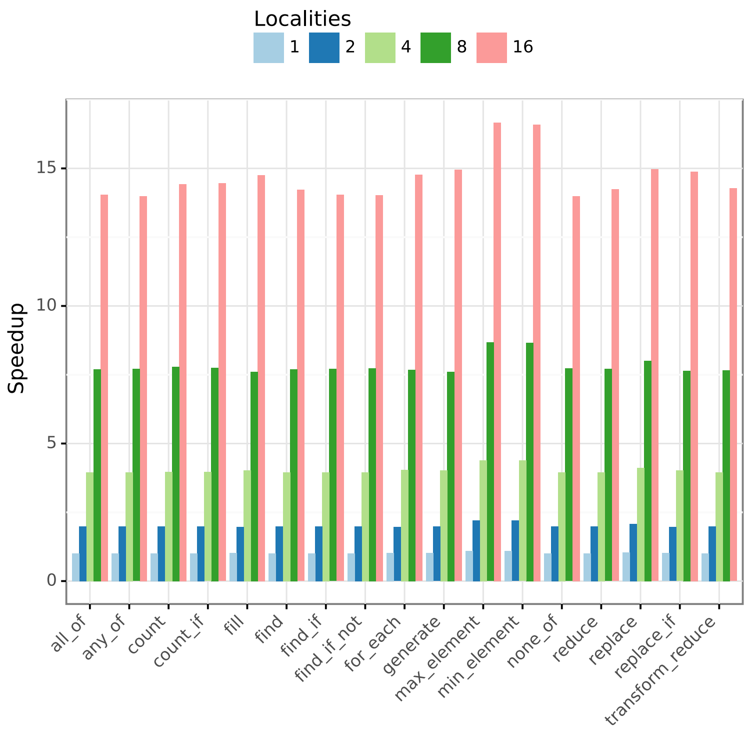
- Semantics, concepts and syntax analogous to STL’s APIs
  - Iterators, ranges, algorithms

- All STL’s algorithms can be executed on SHAD’s data structures
  - But you shouldn’t do that
    - Severe performance penalties due to sync remote memory operations

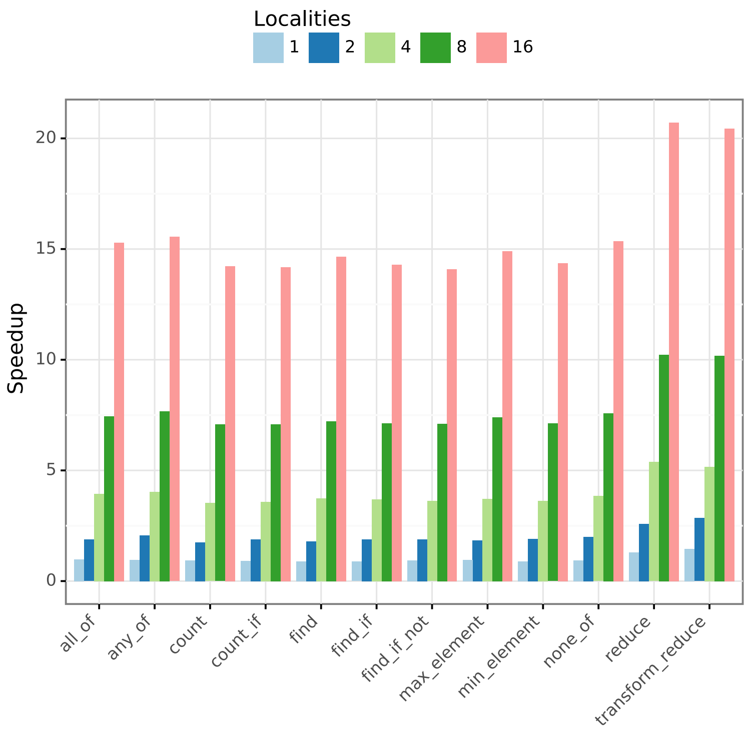
- Additional execution policies for performance
  - distributed_sequential
    - Algorithms with sequential semantics (e.g. left –folding)
  - distributed_parallel
    - Analougus to std::par

Preliminary results

array



unordered_set



- Commodity cluster equipped with Xeon E5-2680 v2 CPUs @ 2.8 GHz
- distributed_parallel policy
- 1 Billion elements of size_t type

And what if I don’t like C++?

SHADEs: The SHAD Exploration System

- Client-server Architecture, inspired by Arkouda
  - Client/server communications via ZMQ library
- Jupyter Notebook / Python frontend, SHAD backend
- Front-end commands are mapped to SHAD functions
- Multiple clients can connect to the same backend at the same time
- Clients can connect to multiple backends
- Debutted on Github!
  - https://github.com/pnnl/SHADEs

SHAD

https://github.com/pnnl/SHAD

Vito Giovanni Castellana
vitoGiovanni.castellana@pnnl.gov