Asynchronous Programming in Modern C++

Futurize All The Things!

Hartmut Kaiser (hkaiser@cct.lsu.edu)
Today's Parallel Applications
Real-world Problems

• Insufficient parallelism imposed by the programming model
  • OpenMP: enforced barrier at end of parallel loop
  • MPI: global (communication) barrier after each time step

• Over-synchronization of more things than required by algorithm
  • MPI: Lock-step between nodes (ranks)

• Insufficient coordination between on-node and off-node parallelism
  • MPI+X: insufficient co-design of tools for off-node, on-node, and accelerators

• Distinct programming models for different types of parallelism
  • Off-node: MPI, On-node: OpenMP, Accelerators: CUDA, etc.
The Challenges

• Design a programming model and programming environment that:
  • Exposes an API that intrinsically
    • Enables overlap of computation and communication
    • Enables fine-grained parallelism
    • Requires minimal synchronization
    • Makes data dependencies explicit
    • Provides manageable paradigms for handling parallelism
  • Integrates well with existing C++ Standard
HPX
The C++ Standards Library for Concurrency and Parallelism

https://github.com/STEllAR-GROUP/hpx
HPX – The C++ Standards Library for Concurrency and Parallelism

• Exposes a coherent and uniform, standards-oriented API for ease of programming parallel, distributed, and heterogeneous applications.
  • Enables to write fully asynchronous code using hundreds of millions of threads.
  • Provides unified syntax and semantics for local and remote operations.

• Enables using the Asynchronous C++ Standard Programming Model
  • Emergent auto-parallelization, intrinsic hiding of latencies,
HPX – A C++ Standard Library

C++2z Concurrency/Parallelism APIs

Threading Subsystem

Active Global Address Space (AGAS)

Performance Counter Framework

Local Control Objects (LCOs)

Parcel Transport Layer (Networking)

Policy Engine/Policies
HPX – The API

- As close as possible to C++11/14/17/20 standard library, where appropriate, for instance:
  - std::thread, std::jthread
  - std::mutex
  - std::future
  - std::async
  - std::for_each(par, ...), etc.
  - std::experimental::task_block
  - std::latch, std::barrier, std::for_loop
  - std::bind
  - std::function
  - std::any
  - std::cout

  - hpx::thread (C++11), hpx::jthread (C++20)
  - hpx::mutex
  - hpx::future (including N4538, ‘Concurrency TS’)
  - hpx::async (including N3632)
  - hpx::parallel::for_each (N4507, C++17)
  - hpx::parallel::task_block (N4411)
  - hpx::latch, hpx::barrier, hpx::parallel::for_loop (TS V2)
  - hpx::bind
  - hpx::function
  - hpx::any (N3508)
  - hpx::cout
## Parallel Algorithms (C++17)

<table>
<thead>
<tr>
<th>adjacent_difference</th>
<th>adjacent_find</th>
<th>all_of</th>
<th>any_of</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>copy_if</td>
<td>copy_n</td>
<td>count</td>
</tr>
<tr>
<td>count_if</td>
<td>equal</td>
<td>exclusive_scan</td>
<td>fill</td>
</tr>
<tr>
<td>fill_n</td>
<td>find</td>
<td>find_end</td>
<td>find_first_of</td>
</tr>
<tr>
<td>find_if</td>
<td>find_if_not</td>
<td>for_each</td>
<td>for_each_n</td>
</tr>
<tr>
<td>generate</td>
<td>generate_n</td>
<td>includes</td>
<td>inclusive_scan</td>
</tr>
<tr>
<td>inner_product</td>
<td>inplace_merge</td>
<td>is_heap</td>
<td>is_heap_until</td>
</tr>
<tr>
<td>is_partitioned</td>
<td>is_sorted</td>
<td>is_sorted_until</td>
<td>lexicographical_compare</td>
</tr>
<tr>
<td>max_element</td>
<td>merge</td>
<td>min_element</td>
<td>minmax_element</td>
</tr>
<tr>
<td>mismatch</td>
<td>move</td>
<td>none_of</td>
<td>nth_element</td>
</tr>
<tr>
<td>partial_sort</td>
<td>partial_sort_copy</td>
<td>partition</td>
<td>partition_copy</td>
</tr>
<tr>
<td>reduce</td>
<td>remove</td>
<td>remove_copy</td>
<td>remove_copy_if</td>
</tr>
<tr>
<td>remove_if</td>
<td>replace</td>
<td>replace_copy</td>
<td>replace_copy_if</td>
</tr>
<tr>
<td>replace_if</td>
<td>reverse</td>
<td>reverse_copy</td>
<td>rotate</td>
</tr>
<tr>
<td>rotate_copy</td>
<td>search</td>
<td>search_n</td>
<td>set_difference</td>
</tr>
<tr>
<td>set_intersection</td>
<td>set_symmetric_difference</td>
<td>set_union</td>
<td>sort</td>
</tr>
<tr>
<td>stable_partition</td>
<td>stable_sort</td>
<td>swap_ranges</td>
<td>transform</td>
</tr>
<tr>
<td>uninitialized_copy</td>
<td>uninitialized_copy_n</td>
<td>uninitialized_fill</td>
<td>uninitialized_fill_n</td>
</tr>
<tr>
<td>unique</td>
<td>unique_copy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Parallel Algorithms (C++17)

- Add Execution Policy as first argument
- Execution policies have associated default executor and default executor parameters
  - `execution::parallel_policy`, generated with `par`
    - parallel executor, static chunk size
  - `execution::sequenced_policy`, generated with `seq`
    - sequential executor, no chunking

```cpp
// add execution policy
std::fill(
    std::execution::par, 
    begin(d), end(d), 0.0);
```
// uses default executor: par
std::vector<double> d = { ... };
fill(execution::par, begin(d), end(d), 0.0);

// rebind par to user-defined executor (where and how to execute)
my_executor my_exec = ...;
fill(execution::par.on(my_exec), begin(d), end(d), 0.0);

// rebind par to user-defined executor and user defined executor
// parameters (affinities, chunking, scheduling, etc.)
my_params my_par = ...
fill(execution::par.on(my_exec).with(my_par), begin(d), end(d), 0.0);
Execution Policies (Extensions)

• Extensions: asynchronous execution policies

  • `parallel_task_execution_policy` (asynchronous version of `parallel_execution_policy`), generated with `par(task)`
  • `sequenced_task_execution_policy` (asynchronous version of `sequenced_execution_policy`), generated with `seq(task)`

• In all cases the formerly synchronous functions return a future<>
• Instruct the parallel construct to be executed asynchronously
• Allows integration with asynchronous control flow
What is a (the) Future?

• Many ways to get hold of a (the) future, simplest way is to use (std) async:

```cpp
int universal_answer() { return 42; }

void deep_thought()
{
    future<int> promised_answer = async(&universal_answer);

    // do other things for 7.5 million years

    cout << promised_answer.get() << endl; // prints 42
}
```
What is a (the) future

- A future is an object representing a result which has not been calculated yet

- Enables transparent synchronization with producer
- Hides notion of dealing with threads
- Represents a data-dependency
- Makes asynchrony manageable
- Allows for composition of several asynchronous operations
- (Turns concurrency into parallelism)
Recursive Parallelism
Parallel Quicksort

template <typename RandomIter>
void quick_sort(RandomIter first, RandomIter last)
{
    ptrdiff_t size = last - first;
    if (size > 1) {
        RandomIter pivot = partition(first, last,
            [p = first[size / 2]](auto v) { return v < p; });

        quick_sort(first, pivot);
        quick_sort(pivot, last);
    }
}
Parallel Quicksort: Parallel

```cpp
template <typename RandomIter>
void quick_sort(RandomIter first, RandomIter last)
{
    ptrdiff_t size = last - first;
    if (size > threshold) {
        RandomIter pivot = partition(par, first, last,
            [p = first[size / 2]](auto v) { return v < p; });

        quick_sort(first, pivot);
        quick_sort(pivot, last);
    } else if (size > 1) {
        sort(seq, first, last);
    }
}
```
Parallel Quicksort: Futurized

template <typename RandomIter>
future<void> quick_sort(RandomIter first, RandomIter last)
{
    ptrdiff_t size = last - first;
    if (size > threshold) {
        future<RandomIter> pivot = partition(par(task), first, last,
            [p = first[size / 2]](auto v) { return v < p; });

        return pivot.then([=](auto pf) {
            auto pivot = pf.get();
            return when_all(quick_sort(first, pivot), quick_sort(pivot, last));
        });
    } else if (size > 1) {
        sort(seq, first, last);
    }
    return make_ready_future();
}
Parallel Quicksort: co_await

template <typename RandomIter>
future<void> quick_sort(RandomIter first, RandomIter last)
{
    ptrdiff_t size = last - first;
    if (size > threshold) {
        RandomIter pivot = co_await partition(par(task), first, last,
            [p = first[size / 2]](auto v) { return v < p; });

        co_await when_all(
            quick_sort(first, pivot), quick_sort(pivot, last));
    } else if (size > 1) {
        sort(seq, first, last);
    }
}
Asynchronous Communication
Asynchronous Channels

- High level abstraction of communication operations
  - Perfect for asynchronous boundary exchange
- Modelled after Go-channels
- Create on one thread, refer to it from another thread
  - Conceptually similar to bidirectional P2P (MPI) communicators
- Asynchronous in nature
  - `channel::get()` and `channel::set()` return futures
Phylanx
An Asynchronous Distributed Array Processing Toolkit

https://github.com/STEllAR-GROUP/phylanx
Phylanx: An Asynchronous Distributed Array Processing Toolkit

- High Performance Computing Challenges
  - Algorithms: need to be made work in distributed, requires data tiling
  - Programming Languages and Models: don’t directly support distributed execution
  - Heterogeneous hardware: difficult to deal with various programming models

- Domain experts, specially in the field of machine learning, have traditionally shied away from utilizing HPC resources due to such challenges

- HPC resources are (becoming) the only viable solution with the ever increasing size of datasets.

- Goal: Abstract away complexities of programming on High Performance Computing resources from domain experts.
Phylanx: An Asynchronous Distributed Array Processing Toolkit

- Uses a decorator, @Phylanx, to access the Python AST
  - Reinterpret the AST as C++ data structures
- Integrated job submission, performance measurement and visualization
- Consists of many parts
  - HPX
  - Blaze
  - APEX
  - Traveler
  - Agave/Tapis
  - Jupyter
Phylanx: An Asynchronous Distributed Array Processing Toolkit

- Combine performance of HPC systems with the ease of programming in a high level language
- Python frontend to abstract away complexities of lower level implementations
  - Integration with Jupyter notebooks
- Run NumPy code directly in Phylanx
- Distributed task graphs are generated from Python
- HPX acts as the execution engine to execute the task graphs
- Promising initial results with execution time comparable to NumPy on shared memory systems.
Phylanx Structure

Frontend
Expression: A + (-B)

Matrices A and B

Python:
@Phylanx
def work(A, B):
    return A + (-B)

Middleware
Internal representation (Abstract Syntax Tree)

PhySL:
define(work, A, B, A + (-B))

Backend
(Distributed) Execution Tree

HPX:
hpx::dataflow(…)
Phylanx: Frontend

```python
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import phylanx
>>> from phylanx.ast import Phylanx
>>> @Phylanx("PhySL")
... def print_somes():
...   print("hello world")
std::string const ::read_x_code = `R"(block(
    """
    // Read X-data from given CSV file
    """
    define(read_x, filepath, row_start, row_stop, col_start, col_stop,
          slice(file_read_csv(filepath),
              make_list(row_start, row_stop),
              make_list(col_start, col_stop))
        )
""
)"`;
```
Phylanx: Middleware

• Various transformations
  • Optimizations
  • Data Tiling and distribution

• Goal: Minimize computation and communication

• Specific for expression to be evaluated
Phylanx: Backend

- Adaptive, asynchronous execution using HPX
- Maximum speed, pure C++

Single System

Distributed System (tiled data)

Node 1

Node 2
Run Code Remotely (Jupyter/Agave)

- Start with Python
  - Call remote_run()
    - Send to remote resource
    - Convert to PhySL
    - Run through queue
    - Collect Performance Data
    - Collect Results

- Click link to visualize performance data

```python
In [4]:
def fib(n):
    if n < 2:
        return n
    else:
        return fib(n-1)+fib(n-2)

fibno = randint(10,15)
print('fib('+str(fibno)+')=...',sep='',flush=True)
job = remote_run(uv, fib, (fibno,), queue='fork', nodes=1, ppn=1)
job.wait()
print("result:",job.get_result())

fib(11)=...
```

```
fib(11)@ef66cf7a-c1bd-4044-8d15-749d32e85d49-007
```
Phylanx: Visualizing Performance

Traveler tools
Phylanx: Backend

- Distributed execution model, mostly SPMD
  - Duplicate execution trees
  - Nodes communicate as needed

Dot product

Asynchronous communication
Phylanx: Futurized Execution

// uses hpx::component for distributed operation
struct add : hpx::component<Node>
{
    // futurized implementation
    future<Data> eval(std::vector<Data> params) const override
    {
        // concurrently evaluate child nodes
        future<Data> lhs = children[0].eval(params);
        future<Data> rhs = children[1].eval(params);

        // simplify code with C++20
        co_return co_await lhs + co_await rhs;  // co_await for results
    }

    std::vector<Node> children;
};
Phylanx: CUDA Graph Execution

```cpp
// uses hpx::component for distributed operation
struct cuda_graph : hpx::component<Node>
{
    // futurized implementation
    future<Data> eval(std::vector<Data> params) const override
    {
        // evaluate children, execute CUDA graph when done
        auto args = co_await map(eval, children, params);
        co_return execute_cuda_graph(graph, args);
    }

    cudaGraph_t graph;
    std::vector<Node> children;
};
```
Asynchrony Everywhere
Futurization

- Technique allowing to automatically transform code
  - Delay direct execution in order to avoid synchronization
  - Turns ‘straight’ code into ‘futurized’ code
  - Code no longer calculates results, but generates an execution tree representing the original algorithm
  - If the tree is executed it produces the same result as the original code
  - The execution of the tree is performed with maximum speed, depending only on the data dependencies of the original code
- Execution exposes the emergent property of being auto-parallelized
Recent Results
Phylanx: Adaptive Inlining

Problem Sizes

- 100 x 100
- 400 x 400
- 200 x 200
- 700 x 700

Number of Threads

0 10 20 30 40 50 60

Improvement (%)
Phylanx: Scaling Results

![Graph showing scaling results](image)

**Non-distributed ALS Python on Dataset 4800 x 4800**

**Distributed ALS on different sizes of datasets**
Astrophysics: Merging White Dwarfs

Orbits: 4.13005
Primary Star Density
Donor Star Density

Primary Star Density
- 2e+3 Max
- 1e-3 Refine
- 1e-1
- 1e-5

Donor Star Density
- 2e+1 Max
- 1e-3 Refine
- 1e-5
- 1e-7
Adaptive Mesh Refinement

- DB: X.0.silo
- Cycle: 0
- Time: 1e-98

Mesh
Var: mesh
Adaptive Mesh Refinement

Strong-scaling efficiency: 68.1%

Weak-scaling efficiency: 78.4%
The Solution to the Application Problem

<table>
<thead>
<tr>
<th>Time</th>
<th>Thread 8</th>
<th>Thread 9</th>
<th>Thread 10</th>
<th>Thread 11</th>
<th>Thread 12</th>
<th>Thread 13</th>
<th>Thread 14</th>
<th>Thread 15</th>
<th>Thread 16</th>
<th>Thread 17</th>
<th>Thread 18</th>
<th>Thread 19</th>
<th>Thread 20</th>
<th>Thread 21</th>
<th>Thread 22</th>
<th>Thread 23</th>
<th>Thread 24</th>
<th>Thread 25</th>
</tr>
</thead>
<tbody>
<tr>
<td>0s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Solution to the Application Problems
FUTURIZE
ALL THE THINGS!