EXPOSING MASS PARALLELISM IN C++

Michael Garland and Jared Hoberock, November 2015
Programming Systems & Applications Research
ACCELERATED COMPUTING

Highly Parallel Program

CPU + GPU

High Performance Execution
RESEARCH QUESTION

How might programmers write highly parallel programs in a mainstream language like C++?
SUPPOSE PARALLELISM IS AVAILABLE

Example: Independent loop iterations represent latent parallelism

```c
void saxpy(int n, float a, float *x, float *y)
{
    // Sequential code with latent parallelism
    for(int i=0; i<n; ++i)
    {
        y[i] = a*x[i] + y[i];
    }
}
```
Individually launching threads leads to inefficiency

```cpp
void saxpy(int n, float a, float *x, float *y)
{
    for(int i=0; i<n; ++i)
    {
        async([=]{ y[i] = a*x[i] + y[i]; });
    }
}
```
void saxpy(int n, float a, float *x, float *y)
{
    #pragma omp parallel for
    for(int i=0; i<n; ++i)
    {
        y[i] = a*x[i] + y[i];
    }
}
void saxpy(int n, float a, float *x, float *y)
{
    auto I = interval(0, n);

    for_each(par, I.begin(), I.end(), [=](int i)
    {
        y[i] = a*x[i] + y[i];
    }
}
STANDARD TEMPLATE LIBRARY
Higher-level library built around algorithms

```
for_each(begin, end, function);
```

Operator: A named pattern of computation and communication.

Data: One or more collections to operate on.

Function: Caller-provided function object injected in pattern.
for_each(par, begin, end, function);

Execution Policy

Specify how operation may execute.
PARALLEL ALGORITHMS

Parallelizable algorithms in STL

for_each
transform
copy_if
sort
set_intersection
etc.

New additions for parallelism

reduce
exclusive_scan
inclusive_scan
transform_reduce
transform_inclusive_scan
transform_exclusive_scan
# EXECUTION POLICIES

Specify how algorithms may execute

<table>
<thead>
<tr>
<th>POLICY NAME</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq</td>
<td>Sequential execution alone is permitted.</td>
</tr>
<tr>
<td>par</td>
<td>Parallel execution is permitted.</td>
</tr>
<tr>
<td>par_vec</td>
<td>Vectorized parallel execution is permitted.</td>
</tr>
</tbody>
</table>

*parallel*: provided function objects can be executed in any order on one or more threads.

*vectorized*: provided function objects can be also be interleaved when on one thread.
EXECUTION POLICIES
Provide opportunities for customization

<table>
<thead>
<tr>
<th>POLICY NAME</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>seq</strong></td>
<td>Sequential execution alone is permitted.</td>
</tr>
<tr>
<td><strong>par</strong></td>
<td>Parallel execution is permitted.</td>
</tr>
<tr>
<td><strong>par_vec</strong></td>
<td>Vectorized parallel execution is permitted.</td>
</tr>
<tr>
<td><strong>cuda::par</strong></td>
<td>Always choose a CUDA-capable GPU.</td>
</tr>
<tr>
<td><strong>on_some_fpga</strong></td>
<td>Permit execution on FPGA where possible.</td>
</tr>
<tr>
<td><strong>vendors_policy</strong></td>
<td>Custom semantics not listed above.</td>
</tr>
</tbody>
</table>
Technical Specification for C++ Extensions for Parallelism


Draft available online


We’ve proposed adding this to C++17

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0024r0.html
THRUST LIBRARY
Programming with algorithms and policies today

Bundled with NVIDIA's CUDA Toolkit

Supports execution on GPUs and CPUs

Ongoing performance & feature improvements

Functionality beyond Parallel STL

Thrust Sort Speedup
CUDA 7.0 vs. 6.5 (32M samples)

From CUDA 7.0 Performance Report.
Run on K40m, ECC ON, input and output data on device
Performance may vary based on OS and software versions, and motherboard configuration
void saxpy(size_t n, float a, const float* x, const float* y, float* z)
{
    using namespace agency;

    bulk_invoke(par(n), [=](parallel_agent &self)
    {
        int i = self.index();
        z[i] = a * x[i] + y[i];
    });
}
void saxpy(size_t n, float a, const float* x, const float* y, float* z)
{
    using namespace agency;

    bulk_invoke(cuda::par(n), [=] __device__ (parallel_agent &self)
    {
        int i = self.index();
        z[i] = a * x[i] + y[i];
    });
}
void saxpy(size_t n, float a, const float* x, const float* y, float* z) {
    using namespace agency;

    auto f1 = bulk_async(par(n), [=](parallel_agent &self) {
        int i = self.index();
        z[i] = a * x[i] + y[i];
    });

    // Do other work here, asynchronously with kernel above
    f1.wait();
}
void other(size_t n, float a, const float* x, const float* y, float* z)
{
    using namespace agency;

    auto f1 = bulk_async(par(n), [=](parallel_agent &self)
    {
        int i = self.index();
        z[i] = a * x[i] + y[i];
    });

    auto f2 = bulk_then(f1, par(n), other_work);
    auto f3 = bulk_then(f1, par(n), more_work);
    when_all(f2, f3).wait();
}
EXECUTORS
Provide control over where execution happens

Placement is, by default, at discretion of the system.

```cpp
for_each(par, I.begin(), I.end(), [](int i) { y[i] += a*x[i]; });
```

In some cases, the programmer might want to control placement.

```cpp
auto place1 = choose_some_place();
auto place2 = choose_another_place();

for_each(par.on(place1), I.begin(), I.end(), ...);
for_each(par.on(place2), I.begin(), I.end(), ...);
```
EXECUTOR FRAMEWORK
Abstract platform details of execution

Create execution agents
Manage data they share
Advertise semantics
Mediate dependencies

```
class sample_executor
{
public:
    using execution_category = ...;

    using shape_type = tuple<size_t,size_t>;

    template<class T>
    using future = ...;

    template<class T>
    future<T> make_ready_future(T&& value);

    template<class Container, class Function, class Future>
    future<Container> then_execute(Function f,
                                    shape_type shape,
                                    Future& dependency);

    ...
};
```

See [http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0058r0.pdf](http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0058r0.pdf) for details.
Parallel Algorithms
Capture common patterns of parallel computation and communication.

Organized Collections of Parallel Agents
Direct implementation and control of parallel computation (e.g., kernels).

Executor Framework
Lowest-level abstraction of platform-specific details of work creation.
Expanded set of algorithms (cf. Thrust)

Consider range-based interfaces

Support for asynchrony in parallel algorithms

Explore desired parallel control structures

Complete definition of executor framework