Extending C++ with Co-Array semantics

Antoine Tran Tan $^1$
Hartmut Kaiser $^2$

Louisiana State University
Center for Computation and Technology
The STEllAR Group
$^1$ atrantan@lsu.edu  $^2$ hkaiser@cct.lsu.edu

C++ Now - May 12, 2016
Context

Issues coming from the hardware

- *Data access* more costly than *data processing*
- More and more disjoint memories to increase the bandwidth
- More and more complex parallel architectures to increase the peak performance
Issues coming from the hardware

- Data access more costly than data processing
- More and more disjoint memories to increase the bandwidth
- More and more complex parallel architectures to increase the peak performance

Software solutions to adapt to these changes

- Data locality with Single Programming Multiple Data
- Remote Memory Access with a Partitioned Global Address Space
- Load balance flexibility with Asynchronous programming
Plan

What are Co-Arrays and why are they important

HPX - High Performance Parallel

Implementation of Co-arrays in C++

Performance evaluation
Co-arrays in few words

- Fortran extension introduced by Numrich and Reid \(^1\)
- Co-array is a strict implementation of the PGAS Model
- Part of the actual Fortran Standard\(^2\)

---

\(^1\) Co-array Fortran for Parallel Programming, - R.W. Numrich et al. - ACM SIGPLAN Fortran forum, 1998
\(^2\) Co-arrays in the next Fortran Standard - R.W. Numrich et al. - ACM SIGPLAN Fortran forum, 2005
Illustration

Locality 0  |  Locality 1  |  Locality $i$  |  Locality $N - 1$
-----------|-------------|----------------|----------------
Memory     | Memory      | Memory         | Memory         
CPUs       | CPUs        | CPUs           | CPUs           

Global Address Space
real :: a(3)
Illustration

real :: a(3)
Illustration

```
real :: a(3)[*]
```

```
Locality 0

```
```
Locality 1

```
```
Locality i

```
```
Locality N – 1

```

<table>
<thead>
<tr>
<th>Locality 0</th>
<th>Locality 1</th>
<th>Locality i</th>
<th>Locality N – 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUs</td>
<td>CPUs</td>
<td>CPUs</td>
<td>CPUs</td>
</tr>
<tr>
<td>[1]</td>
<td>[2]</td>
<td>[i + 1]</td>
<td>[N]</td>
</tr>
<tr>
<td>(1) (2) (3)</td>
<td>(1) (2) (3)</td>
<td>(1) (2) (3)</td>
<td>(1) (2) (3)</td>
</tr>
</tbody>
</table>
Illustration

Where is the element \( a(2)[N] \) ?

Locality 0  |  Locality 1  |  Locality \( i \)  |  Locality \( N - 1 \)
--- | --- | --- | ---
[1] | [2] | \([i + 1]\) | \([N]\)
(1) (2) (3) | (1) (2) (3) | (1) (2) (3) | (1) (2) (3)
CPUs | CPUs | CPUs | CPUs
Illustration

Where is the element \( a(2)[N] \)?

Locality 0  Locality 1  Locality \( i \)  Locality \( N - 1 \)

<table>
<thead>
<tr>
<th>CPUs</th>
<th>CPUs</th>
<th>CPUs</th>
<th>CPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>(2)</td>
<td>(3)</td>
<td>(1)</td>
</tr>
<tr>
<td>(2)</td>
<td>(3)</td>
<td>(1)</td>
<td>(2)</td>
</tr>
<tr>
<td>[1]</td>
<td>[2]</td>
<td>([i + 1])</td>
<td>([N])</td>
</tr>
</tbody>
</table>
From Co-array Fortran to Co-array C++

Why co-arrays?

- Improve data locality in distributed applications
- Access to remote references are done via array-based subscripts
- Widely accepted by the Fortran community

---

3 Extending C++ with co-array semantics - A. Tran Tan et al - ACM SIGPLAN ARRAY, 2016 (soon)
From Co-array Fortran to Co-array C++

Why co-arrays?

- Improve data locality in distributed applications
- Access to remote references are done via array-based subscripts
- Widely accepted by the Fortran community

Our Approach

- Enable co-array semantics with a C++ library approach
- Use of a C++ runtime system to manage parallel/distributed tasks
- New features of the C++ Standard 11/14 ➞ Easy API design

---

3 Extending C++ with co-array semantics - A. Tran Tan et al - ACM SIGPLAN ARRAY, 2016 (soon)
Plan

What are Co-Arrays and why are they important

HPX - High Performance Parallex

Implementation of Co-arrays in C++

Performance evaluation
HPX : High Performance Parallex

A C++ runtime system for applications of any scale \(^4,5\)

\[^4\text{Parallex an advanced parallel execution model for scaling-impaired applications - H. Kaiser et al - ICPPW, 2009}\]

\[^5\text{A Task Based Programming Model in a Global Address Space - H. Kaiser et al - PGAS, 2014}\]
HPX : High Performance Parallex

A C++ runtime system for applications of any scale \(^4,^5\)

---

\(^4\) Parallex an advanced parallel execution model for scaling-impaired applications - H. Kaiser et al - ICPPW, 2009

\(^5\) A Task Based Programming Model in a Global Address Space - H. Kaiser et al - PGAS, 2014
HPX : High Performance Parallex

A C++ runtime system for applications of any scale \(^4,5\)

---

Plan

What are Co-Arrays and why are they important

HPX - High Performance Parallel

Implementation of Co-arrays in C++

Performance evaluation
Instantiation of a co-array object: Fortran vs C++

// Fortran Code
real :: z[10,*]

// C++ Code
spmd_block block;
coarray<double,2> z( block, "z", {10,_}, partition<double>(1));
spmd_block block;
coarray<
double,1> z( block, "z", { }, partition<double>(1));

if ( block
    .this_image() == 0 )
{
    std::cin >> z.data(_);

    int num_images = block
        .get_num_images();
    for ( int image = 1; image < num_images; image++ )
    {
        z(image) = z.data(_);
    }
}

block
    .barrier_sync("b"); // sync_all() in Fortran
Co-Array C++ sample code

```cpp
spmd_block block;
coarray<double,1> z( block, "z", { _ }, partition<double>(1));

if ( block.this_image() == 0 )
{
    std::cin >> z.data(_);

    int num_images = block.get_num_images();
    for( int image = 1; image < num_images; image++ )
    {
        z(image) = z.data(_);
    }
}

future<void> fb = block.barrier("b");
```
Traversal of co-indexed elements with iterators

```cpp
spmd_block block;
coarray<double,3> a ( block, "a", {{4,4,_}}, partition<double>(5));

int idx = 0;
if ( block.this_image() == 0 )
{
    for (auto i = a.begin(); i != a.end(); i++)
        *i = std::vector<double>(5, idx++);
}
block.barrier_sync("b");

auto alocal = local_view(a);

for (auto ii = alocal.begin(); ii != alocal.end(); ii++)
{
    std::vector<double> & ref = *ii;
    ...
}
```
... with range-based for loops

```cpp
spmd_block block;
coarray<double,3> a ( block, "a", {4,4,_}, partition<double>(5));

int idx = 0;
if ( block.this_image() == 0 )
{
    for (auto && proxy : a)
        proxy = std::vector<double>(5, idx++);
}
block.barrier_sync("b");

auto alocal = local_view( a );

for (std::vector<double> & ref : alocal)
{
    ...
}
```
Creation of a distributed vector in HPX

A coarray is a *multi-dimensionnmal view* tied to a distributed vector.
Creation of a distributed vector in HPX

A coarray is a multi-dimensional view tied to a distributed vector

```cpp
int N, n;

std::vector<hpx::id_type> locs = hpx::find_all_localities();

auto layout = hpx::container_layout(n, locs);

// Creation of the distributed vector
hpx::partitioned_vector<double> v(N, 0.0, layout);
```
Creation of a SPMD region

A SPMD region is the mean to invoke images in multiple localities
Creation of a SPMD region

A SPMD region is the mean to invoke *images* in multiple localities

```cpp
void example_image(spmd_block block)
{
    ...
}
HPX_DEFINE_PLAIN_ACTION(example_image, my_action);

int main()
{
    std::vector<hpx::id_type> locs = hpx::find_all_localities();

    // Invocation of the spmd region
    define_spmd_block(locs, my_action);

    return 0;
}
```
Plan

What are Co-Arrays and why are they important

HPX - High Performance Parallel

Implementation of Co-arrays in C++

Performance evaluation
Benchmark 1: Matrix Transpose

```c
void transpose_coarray(spmd_block & block,
            coarray<double,2> & out,
            coarray<double,2> & in,
            int height, int width,
            int local_height,
            int local_width,
            int local_leading_dimension)
{
    // Outer Transpose operation
    for(int j = 0; j<width; j++)
        for(int i = 0; i<height; i++)
        {
            // Put operation
            out(j,i) = in(i,j);
        }
    block.barrier_sync("outer_transpose");

    /* */
```
Benchmark 1: Matrix Transpose

```cpp
// Inner Transpose operation
for (std::vector<double> & elt : out_local)
{
    for (int jj = 0; jj < local_width - 1; jj++)
        for (int ii = jj + 1; ii < local_height; ii++)
        {
            std::swap( elt[jj + ii*local_leading_dimension], elt[ii + jj*local_leading_dimension]);
        }
}
block.barrier_sync("inner_transpose");
```
Benchmark 1: Matrix Transpose
performed in a $2 \times 8$ core machine
Benchmark 2: **Sparse Matrix Vector Multiplication**

```cpp
struct spmatrix
{
    // Constructor definition ...
    int m_, n_, nnz_;  
    std::vector<int> rows_, indices_;  
    std::vector<double> values_;  
    std::vector<int> begins_, sizes_;  
};

void spmv_coarray(  
    spmd_block & block  
    , spmatrix const & a,  
    std::vector<double> & x  
    , coarray<double,1> & y)
{
    int image_id = block.this_image();
    int begin = a.begins_[image_id];
    int chunksize = a.sizes_[image_id];

    /* */
```
Benchmark 2: Sparse Matrix Vector Multiplication

```c
/* */

double * out = y.data(_).data();
const int * row = a.rows_.data() + begin;
const int * idx = a.indices_.data() + *row - 1;
const double * val = a.values_.data() + *row - 1;

for(int i = 0; i < chunksize; i++, row++, out++)
{
    double tmp = 0.;
    int end = *(row + 1);

    for(int o = *row; o < end; o++, val++, idx++)
        tmp += *val * x[*idx - 1];

    *out = tmp;
}

block.barrier_sync("spmv");
```
Benchmark 2: Sparse Matrix Vector Multiplication
performed in a $2 \times 8$ core machine
Thanks for your attention