UPC++ Hello World

```
#include <stdio.h>
#include <upcxx/upcxx.hpp>

int main(int argc, char* argv[])
{
  upcxx::init();
  rankId     = upcxx::rank_me();
  rankCount = upcxx::rank_n();

  printf("Hello, I am rank %d of %d\n", rankId, rankCount);

  upcxx::finalize();
  return 0;
}
```

```
>upcxx-run -n 8 ./example1
Hello, I am rank 5 of 8
Hello, I am rank 3 of 8
Hello, I am rank 7 of 8
Hello, I am rank 1 of 8
Hello, I am rank 0 of 8
Hello, I am rank 4 of 8
Hello, I am rank 6 of 8
Hello, I am rank 2 of 8
```

UPC++ Barriers

```
int main(int argc, char* argv[])
{
   upcxx::init();
   int rankId     = upcxx::rank_me();
   int rankCount = upcxx::rank_n();

   printf("Rank %d: Let's do this first.\n", rankId);

   upcxx::barrier();

   printf("Rank %d: Let's do this second.\n", rankId);

  upcxx::finalize();
   return 0;
}
```

```
Rank 2: Let's do this first.
Rank 3: Let's do this first.
Rank 5: Let's do this first.
Rank 1: Let's do this first.
Rank 0: Let's do this first.
Rank 6: Let's do this first.
Rank 7: Let's do this first.
Rank 4: Let's do this first.
Rank 0: Let's do this second.
Rank 1: Let's do this second.
Rank 4: Let's do this second.
Rank 2: Let's do this second.
Rank 6: Let's do this second.
Rank 5: Let's do this second.
Rank 3: Let's do this second.
Rank 7: Let's do this second.
```

UPC++ (Blocking) Broadcast

```
#include <stdio.h>
#include <upcxx/upcxx.hpp>

int main(int argc, char* argv[])
{
   upcxx::init();
   int rankId     = upcxx::rank_me();
   int rankCount = upcxx::rank_n();

   int number = 0;
   if (rankId == 0) number = 500;

   upcxx::broadcast(&number, 1 /*Count*/, 0 /*Root*/).wait();

   printf("Rank %d: Number is %d.\n", rankId, number);

  upcxx::finalize();
   return 0;
}
```

```
>upcxx-run -n 8 ./example3
Rank 4: Number is 500.
Rank 0: Number is 500.
Rank 1: Number is 500.
Rank 2: Number is 500.
Rank 3: Number is 500.
Rank 5: Number is 500.
Rank 6: Number is 500.
Rank 7: Number is 500.
```

UPC++ (Non-Blocking) Broadcast

```
auto future = upcxx::broadcast(&number, 1 /*Count*/, 0 /*Root*/);

printf("Rank %d: (Before) Number is %d.\n", rankId, number);

// Do other stuff

future.wait();

printf("Rank %d: (After) Number is %d.\n", rankId, number);
```

```
>upcxx-run -n 8 ./example4
Rank 2: (Before) Number is 0.
Rank 1: (Before) Number is 0.
Rank 3: (Before) Number is 0.
Rank 6: (Before) Number is 0.
Rank 5: (Before) Number is 0.
Rank 7: (Before) Number is 0.
Rank 0: (Before) Number is 500.
Rank 4: (Before) Number is 0.
Rank 0: (After) Number is 500.
Rank 1: (After) Number is 500.
Rank 4: (After) Number is 500.
Rank 5: (After) Number is 500.
Rank 6: (After) Number is 500.
Rank 7: (After) Number is 500.
Rank 2: (After) Number is 500.
Rank 3: (After) Number is 500.
```

Accessing the Global Address Space (I)

Broadcasting the global pointer to a single partition.

```
upcxx::global_ptr<int> gptr;
if (rankId == 0) gptr = upcxx::new_array<int>(rankCount);

upcxx::broadcast(&gptr, 1, 0).wait();

auto future = upcxx::rput(&rankId, gptr + rankId, 1);

printf("Rank %d - Updating the global allocation.\n", rankId);

future.wait();

upcxx::barrier();

if (rankId == 0)
{
 int* lptr = gptr.local();
 printf("{");
 for (int i = 0; i < rankCount; i++)
  printf("%d,", lptr[i]);
 printf("}\n");
}
```

```
>upcxx-run -n 8 ./example5
Rank 0 - Updating the global allocation.
Rank 1 - Updating the global allocation.
Rank 2 - Updating the global allocation.
Rank 4 - Updating the global allocation.
Rank 3 - Updating the global allocation.
Rank 5 - Updating the global allocation.
Rank 6 - Updating the global allocation.
Rank 7 - Updating the global allocation.
{0,1,2,3,4,5,6,7,}
```

Accessing the Global Address Space (II)

Accessing multiple partitions via a distributed object.

```
auto myPartition = upcxx::new_<double>(rankId);
upcxx::dist_object<upcxx::global_ptr<double>> partitions(myPartition);

*myPartition.local() = sqrt((double)rankId);

upcxx::barrier();

if (rankId == 0) for (int i = 0; i < rankCount; i++)
{
  auto partition = partitions.fetch(i).wait();
  double val = upcxx::rget(partition).wait();
  printf("SquareRoot(%f) = %f\n", (double)i, val);
}
```

```
>upcxx-run -n 8 ./example11
SquareRoot(0.000000) = 0.000000
SquareRoot(1.000000) = 1.000000
SquareRoot(2.000000) = 1.414214
SquareRoot(3.000000) = 1.732051
SquareRoot(4.000000) = 2.000000
SquareRoot(5.000000) = 2.236068
SquareRoot(6.000000) = 2.449490
SquareRoot(7.000000) = 2.645751
```

Asynchronous Execution with RPCs

Remote Procedure Call - Using Lambda Expressions.

```
upcxx::init();
rankId    = upcxx::rank_me();
int rankCount = upcxx::rank_n();

finished = false;
upcxx::barrier();

if (rankId == 0)
  for (int i = 1; i < rankCount; i++)
    upcxx::rpc(i, [](int par){
     printf("Rank %d: Received RPC with Parameter: %d\n", rankId, par);
     finished = true;
   }, i*rankCount).wait();

if (rankId > 0) while (!finished) upcxx::progress();
```

```
>upcxx-run -n 8 ./example6
Rank 1: Received RPC with Parameter: 8
Rank 2: Received RPC with Parameter: 16
Rank 3: Received RPC with Parameter: 24
Rank 4: Received RPC with Parameter: 32
Rank 5: Received RPC with Parameter: 40
Rank 6: Received RPC with Parameter: 48
Rank 7: Received RPC with Parameter: 56
```

Conjoining Futures

So that we can continue doing stuff while they finish

```
upcxx::future<> futs = upcxx::make_future();

if (rankId == 0)
 for (int i = 1; i < rankCount; i++)
 {
  auto fut = upcxx::rpc(i, [](int par){
    printf("Rank %d: Received RPC with Parameter: %d\n", rankId, par);
    finished = true;
   }, i*rankCount);
  futs = upcxx::when_all(futs, fut);
 }

if (rankId > 0) while (!finished) upcxx::progress();

if (rankId == 0) printf("Not all finished yet.\n");
 if (rankId == 0) futs.wait();
 if (rankId == 0) printf("All finished now.\n");
```

```
>upcxx-run -n 8 ./example9
Rank 1: Received RPC with Parameter: 8
Rank 2: Received RPC with Parameter: 16
Rank 3: Received RPC with Parameter: 24
Rank 4: Received RPC with Parameter: 32
Not all finished yet.
Rank 5: Received RPC with Parameter: 40
Rank 6: Received RPC with Parameter: 48
Rank 7: Received RPC with Parameter: 56
All finished now.
```

Composing Futures

Useful for defining callbacks to previous RPCs

```
if (rankId == 0)
{
 upcxx::future<> futs = upcxx::make_future();
 for (int i = 1; i < rankCount; i++)
 {
  auto f1 = upcxx::rpc(i, [](int par){
   printf("Rank %d: Received RPC with Parameter: %d\n", rankId, par);
   finished = true;
  }, i*rankCount);
  f1.then([i](){printf("Rank 0: Rank %d Came back.\n", i);});
  futs = upcxx::when_all(futs,f1);
 }
 futs.wait();
}
```

```
upcxx-run -n 8 ./example12
Rank 1: Received RPC with Parameter: 8
Rank 3: Received RPC with Parameter: 24
Rank 6: Received RPC with Parameter: 48
Rank 4: Received RPC with Parameter: 32
Rank 7: Received RPC with Parameter: 56
Rank 5: Received RPC with Parameter: 40
Rank 2: Received RPC with Parameter: 16
Rank 0: Rank 1 Came back.
Rank 0: Rank 3 Came back.
Rank 0: Rank 6 Came back.
Rank 0: Rank 4 Came back.
Rank 0: Rank 7 Came back.
Rank 0: Rank 5 Came back.
Rank 0: Rank 2 Came back.
```

RPC with Return Values

```
int calculateSquare(int x) { return x*x; }

int main(int argc, char* argv[])
{
  upcxx::init();
  int rankId     = upcxx::rank_me();
  int rankCount = upcxx::rank_n();

  if (rankId == 0)
   for (int i = 1; i < rankCount; i++)
    printf("Value: %d - Square %d\n", i, upcxx::rpc(i, calculateSquare, i).wait());
```

```
>upcxx-run -n 8 ./example10
Value: 1 - Square 1
Value: 2 - Square 4
Value: 3 - Square 9
Value: 4 - Square 16
Value: 5 - Square 25
Value: 6 - Square 36
Value: 7 - Square 49
```

Quiescense (Fire and Forget)

When we don't need a response

```
int main(int argc, char* argv[])
{
   upcxx::init();
   rankId     = upcxx::rank_me();
   int rankCount = upcxx::rank_n();

   finished = false;
   upcxx::barrier();

   if (rankId == 0)
     for (int i = 1; i < rankCount; i++)
        upcxx::rpc_ff(i, [](int par){
         printf("Rank %d: Received RPC with Parameter: %d\n", rankId, par);
         finished = true;
      }, i*rankCount);

   if (rankId > 0) while (!finished) upcxx::progress();
```

```
>upcxx-run -n 8 ./example7
Rank 2: Received RPC with Parameter: 16
Rank 1: Received RPC with Parameter: 8
Rank 3: Received RPC with Parameter: 24
Rank 4: Received RPC with Parameter: 32
Rank 5: Received RPC with Parameter: 40
Rank 6: Received RPC with Parameter: 48
Rank 7: Received RPC with Parameter: 56
```

Nested RPCs

When the reciever also needs the sender to do something

```
upcxx::barrier();

if (rankId == 0)
 for (int i = 1; i < rankCount; i++)
  upcxx::rpc_ff(i, [](int par){
   printf("Rank %d: Received RPC with Parameter: %d\n", rankId, par);
   finished = true;
   upcxx::rpc_ff(0, []()
   {
    responses++;
    printf("Rank 0: Received responses: %d\n", responses);
    if (responses == rankCount-1) finished = true;
   });
  }, i*rankCount);
```

```
>upcxx-run -n 8 ./example8
Rank 2: Received RPC with Parameter: 16
Rank 1: Received RPC with Parameter: 8
Rank 3: Received RPC with Parameter: 24
Rank 5: Received RPC with Parameter: 40
Rank 4: Received RPC with Parameter: 32
Rank 6: Received RPC with Parameter: 48
Rank 7: Received RPC with Parameter: 56
Rank 0: Received responses: 1
Rank 0: Received responses: 2
Rank 0: Received responses: 3
Rank 0: Received responses: 4
Rank 0: Received responses: 5
Rank 0: Received responses: 6
Rank 0: Received responses: 7
```