A simple ISPC program is given below. The following C code will call the following ISPC code:

```
------------------------------------------------------------------------
C program code: myprogram.cpp
------------------------------------------------------------------------
const int TOTAL_VALUES = 1024;
float a[TOTAL_VALUES];
float b[TOTAL_VALUES];
float c[TOTAL_VALUES]

// Initialize arrays a and b here.

sum(TOTAL_VALUES, a, b, c);

// Upon return from sumArrays, result of a + b is stored in c.
```

The corresponding ISPC code:

```
------------------------------------------------------------------------
ISPC code: myprogram.ispc
------------------------------------------------------------------------
export sum(uniform int N, uniform float* a, uniform float* b, uniform float* c)
{
  // Assumption programCount divides N evenly.
  for (int i=0; i<N; i+=programCount)
  {
    c[programIndex + i] = a[programIndex + i] + b[programIndex + i];
  }
}
```

The ISPC program code above interleaves the processing of array elements among instances.

However, rather than thinking about how to divide work among program instances (that is, how work is mapped to execution units), it is often more convenient, and more powerful, to instead focus only on the partitioning of a problem into independent parts.
ISPCs `foreach` construct provides a mechanism to express problem decomposition. Below, the `foreach` loop in the ISPC function `sum2` defines an iteration space where all iterations are independent and therefore can be carried out in any order. ISPC handles the assignment of loop iterations to concurrent program instances. The difference between `sum` and `sum2` below is subtle, but very important. `sum` is imperative: it describes how to map work to concurrent instances. The example below is declarative: it specifies only the set of work to be performed.

```
------------------------------------------------------------------------
ISPC code:
------------------------------------------------------------------------
export sum2(uniform int N, uniform float* a, uniform float* b, uniform float* c)
{
  foreach (i = 0 ... N)
  {
    c[i] = a[i] + b[i];
  }
}
```