Complex number representation

There are several ways to represent complex number arrays. If a code written for one of the representation and a new data representation are given, can we generate a code for the new data representation?

All of the following routines compute element-wise multiplication of two vectors of complex numbers.

The first code uses two separate arrays for real and imaginary parts. This form was used in vector processors, and is now reviving for GPUs.

```
void soa_cmul(int n, double *ar, double *ai, double *br, double *bi,
                    double *cr, double *ci) {
   int i;
   for (i=0; i< n; i++) {
      ar[i] = br[i] * cr[i] – bi[i] * ci[i];
      ai[i] = br[i] * ci[i] + bi[i] * cr[i];
   }
}
```

The second code uses a structure of two fields. This form is widely used in cache-based machines. However, some earlier versions of SSE instructions were incompatible with this form.

```
typedef struct {
   double r, i;
} complex;

void aos_cmul(int n, complex *a, complex *b, complex *c) {
   int i;
   for (i=0; i< n; i++) {
      a.r[i] = b.r[i] * c.r[i] – b.i[i] * c.i[i];
      a.i[i] = b.r[i] * c.i[i] + b.i[i] * c.r[i];
   }
}
```

Some old subroutines assume other forms of data structure. If such a subroutine is used in a program, we must either (1) (re)write application program with this form, (2) rewrite the subroutines in the application's data structure (but the source code is not always available), or (3) write marshaling and unmarshaling code. They are tedious and may bring an error.

Here, even element contains real part, and odd element contains imaginary part.

```
void pair_cml(int n, double *a, double *b, double *c) {
    int i;
    for (i=0; i< n; i++) {
        a[2*i+0] = b[2*i+0] * c[2*i+0] - b[2*i+1] * c[2*i+1];
        a[2*i+1] = b[2*i+0] * c[2*i+1] + b[2*i+1] * c[2*i+0];
    }
}
```

In the following case, the first half of the array contains the real part, and the last half of the array contains the imaginary part.

```
void half_cml(int n, double *a, double *b, double *c) {
    int i;
    for (i=0; i< n; i++) {
        a[i] = b[i] * c[i] - b[n+i] * c[n+i];
        a[n+i] = b[i] * c[n+i] + b[n+i] * c[i];
    }
}
```

Here, an array of complex numbers is stored as a two-dimensional array.

```
void twod_cml(int n, double a[n][2], double b[n][2], double c[n][2]) {
    int i;
    for (i=0; i< n; i++) {
        a[i][0] = b[i][0] * c[i][0] - b[i][1] * c[i][1];
        a[i][1] = b[i][0] * c[i][1] + b[i][1] * c[i][0];
    }
}
```

Similar to the previous one, but the order of the dimensions are inverted.

```
void twor_cml(int n, double a[2][n], double b[2][n], double c[2][n]) {
    int i;
    for (i=0; i< n; i++) {
        a[0][i] = b[0][i] * c[0][i] - b[1][i] * c[1][i];
        a[1][i] = b[0][i] * c[1][i] + b[1][i] * c[0][i];
    }
}
```