Compiler output for recur:
Source loop below has tag L4
L4 scheduled with steady-state cycle count = 8
L4 unrolled 4 times
L4 has 2 loads, 1 stores, 3 prefetches, 1 FPadds, 1 FPmuls, and 0 FPdivs per iteration
L4 has 0 int-loads, 0 int-stores, 5 alu-ops, 0 muls, 0 int-divs and 0 shifts per iteration
69. for (i = 1; i < n; i++)
70. x[i] = x[i-1]*a[i] + d[i];

Compiler output for recur2:
Source loop below has tag L4
L4 scheduled with steady-state cycle count = 9
L4 unrolled 4 times
L4 has 2 loads, 1 stores, 3 prefetches, 2 FPadds, 2 FPmuls, and 0 FPdivs per iteration
L4 has 0 int-loads, 0 int-stores, 5 alu-ops, 0 muls, 0 int-divs and 0 shifts per iteration
71. for (i = 2; i < n; i++)
72. {
73. cfa = a[i]*a[i-1];
74. cfd = a[i]*d[i-1]+d[i];
75. x[i] = x[i-2]*cfa + cfd;
76. }

Compiler output for recur3:
Source loop below has tag L4
L4 scheduled with steady-state cycle count = 6
L4 unrolled 4 times
L4 has 2 loads, 1 stores, 3 prefetches, 3 FPadds, 4 FPmuls, and 0 FPdivs per iteration
L4 has 0 int-loads, 0 int-stores, 5 alu-ops, 0 muls, 0 int-divs and 0 shifts per iteration
72. for (i = 3; i < n; i++)
73. {
74. cfa = a[i]*a[i-1]*a[i-2];
75. cfd = a[i]*a[i-1]*d[i-2]+a[i]*d[i-1]+d[i];
76. x[i] = x[i-3]*cfa + cfd;
77. }
Recurrences on Prism006 (Xeon) with SUNCC -fast -xrestrict -xvector=simd -xinline=no -xarch=native -g

Compiler does not seem to have applied any loop restructuring optimizations for recur, recur2, recur3
Recurrences on Prism006 (Xeon) with ICC -O3 -restrict -fno-inline -axW -vec-report3

Compiler output for recur (note that the use of 'restrict' did not seem to help eliminate assumed aliases):
recur.c(69) : (col. 3) remark: vector dependence: assumed FLOW dependence between x line 70 and d line 70.
recur.c(69) : (col. 3) remark: vector dependence: assumed FLOW dependence between x line 70 and a line 70.
recur.c(69) : (col. 3) remark: proven FLOW dependence between x line 70, and x line 70.
recur.c(69) : (col. 3) remark: loop was not vectorized: existence of vector dependence.

Compiler output for recur2:
recur2.c(71) : (col. 3) remark: LOOP WAS VECTORIZED.

Compiler output for recur3:
recur3.c(72) : (col. 3) remark: loop was not vectorized: vectorization possible but seems inefficient.
Code of recur bench test

```c
void
recurrence(double * restrict x, const double * restrict a, const double * restrict d, int n)
{
  int i;
  x[0] = d[0];
  for (i = 1; i < n; i++)
    x[i] = x[i-1]*a[i] + d[i];
}
```

Code of recur2 bench test

```c
x[0] = d[0];
x[1] = x[0]*a[1] + d[1];
for (i = 2; i < n; i++)
{
  cfa = a[i]*a[i-1];
  cfd = a[i]*d[i-1]+d[i];
  x[i] = x[i-2]*cfa + cfd;
}
```

Code of recur3 bench test

```c
x[0] = d[0];
x[1] = x[0]*a[1] + d[1];
for (i = 3; i < n; i++)
{
  cfa = a[i]*a[i-1]*a[i-2];
  cfd = a[i]*a[i-1]*d[i-2]+a[i]*d[i-1]+d[i];
  x[i] = x[i-3]*cfa + cfd;
}
```