Applied Parallel Computing LLC
http://parallel-computing.pro

USI

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Mini-stencil OpenACC C walkthrough

Dmitry Mikushin, William Sawyer, Radim Janalik

July 8, 2014

Code modifications for OpenACC compatibility

1. Move allocations in `cg_init` out of `ss_cg` and place it on top of main iterations loop:

```
printf(”INITIALIZING CG STATE\n”);
Ap    = (double*) malloc(N*sizeof(double));
r     = (double*) malloc(N*sizeof(double));
p     = (double*) malloc(N*sizeof(double));
Fx    = (double*) malloc(N*sizeof(double));
Fxold = (double*) malloc(N*sizeof(double));
v     = (double*) malloc(N*sizeof(double));
xold  = (double*) malloc(N*sizeof(double));
```

Code modifications for OpenACC compatibility

2. Explicitly specify `[start_index:length]` shape for every array in OpenACC pragmas:

```
#pragma acc data copy(x_old[0:nx*ny], x_new[0:nx*ny], deltax[0:nx*ny], Ap[0:nx*ny], p[0:nx*ny], r[0:nx*↩
    ny], b[0:nx*ny], v[0:nx*ny], Fx[0:nx*ny], Fxold[0:nx*ny], xold[0:nx*ny], bndN[0:nx], bndE[0:ny], ↩
    bndS[0:nx], bndW[0:ny], options)
```

```
#pragma acc parallel loop present(x[0:N], y[0:N], l[0:N], r[0:N])
for (i = 0; i < N; i++)
    y[i] = x[i] + alpha * (l[i] - r[i]);
```

... and all other places

Reason: in Fortran allocatable array references always implicitly contain dimensions configs; in C arrays are just raw pointers w/o any additional info.

Code modifications for OpenACC compatibility

2 Explicitly specify `[start_index:length]` shape for every array in OpenACC pragmas:

```
#pragma acc data copy(x_old‘[0:nx*ny]‘, x_new‘[0:nx*ny]‘, deltax‘[0:nx*ny]‘, Ap‘[0:nx*ny]‘, p‘[0:nx*ny↩
    ]‘, r‘[0:nx*ny]‘, b‘[0:nx*ny]‘, v‘[0:nx*ny]‘, Fx‘[0:nx*ny]‘, Fxold‘[0:nx*ny]‘, xold‘[0:nx*ny]‘, ↩
    bndN‘[0:nx]‘, bndE‘[0:ny]‘, bndS‘[0:nx]‘, bndW‘[0:ny]‘, options)
```

```
#pragma acc parallel loop present(x[0:N], y[0:N], l[0:N], r[0:N])
for (i = 0; i < N; i++)
    y[i] = x[i] + alpha * (l[i] - r[i]);
```

... and all other places

Reason: in Fortran allocatable array references always implicitly contain dimensions configs; in C arrays are just raw pointers w/o any additional info.

Code modifications for OpenACC compatibility

3. Initially set to zero two more arrays:

```
memset(x_old,  0, sizeof(double) * nx * ny);
memset(deltax, 0, sizeof(double) * N);
```

Code modifications for OpenACC performance

1. Assure OpenACC compiler input/output arrays do not intersect in memory:

```
void ss_copy(double* y, const double* x, const int N)
{
    int i;
    #pragma acc kernels loop present(x[0:N], y[0:N])
    for (i = 0; i < N; i++)
        y[i] = x[i];
}
```

W/o hints PGI compiler refuses to parallelize:

```
ss_copy:
    161, Generating present(x[:N])
         Generating present(y[:N])
    162, Complex loop carried dependence of ’x->’ prevents parallelization
         Loop carried dependence of ’y->’ prevents parallelization
         Loop carried backward dependence of ’y->’ prevents vectorization
         Accelerator scalar kernel generated
```

Code modifications for OpenACC performance

1. Assure OpenACC compiler input/output arrays do not intersect in memory:

Solution No.1: Add independent:

```
void ss_copy(double* y, const double* x, const int N)
{
    int i;
    #pragma acc kernels loop independent present(x[0:N], y[0:N])
    for (i = 0; i < N; i++)
        y[i] = x[i];
}
```

Now PGI compiler parallelizes:

```
ss_copy:
    162, Generating present(x[:N])
         Generating present(y[:N])
    163, Loop is parallelizable
         Accelerator kernel generated
    163, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
```

Code modifications for OpenACC performance

1. Assure OpenACC compiler input/output arrays do not intersect in memory:

Solution No.2: Add `__restrict__`:

```
void ss_copy(double* __restrict__ y, const double* const __restrict__ x, const int N)
{
int i;
    #pragma acc parallel loop present(x[0:N], y[0:N])
    for (i = 0; i < N; i++)
        y[i] = x[i];
}
```

Now PGI compiler parallelizes:

```
ss_copy:
    162, Generating present(x[:N])
         Generating present(y[:N])
    163, Loop is parallelizable
         Accelerator kernel generated
    163, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
```