

High Performance Computing

Parallelize dynprog.c using OpenMP

*A project made by Francesco Malferrari, Gianluca
Siligardi and Andrea Somenzi*

Profiling (1)

Perf command

A lot of branch-misses → A lot of cycles

```
somand@nano:~/hpc-assignments-group-7/dynprog$ perf stat -r 5 ./dynprog.exe
1.628108
-112589990684278448.00
1.697542
-112589990684278448.00
1.196933
-112589990684278448.00
1.191907
-112589990684278448.00
1.195159
-112589990684278448.00

Performance counter stats for './dynprog.exe' (5 runs):

      1220,764546      task-clock (msec)          #    0,874 CPUs utilized          ( +- 0,48% )
           25         context-switches          #    0,021 K/sec                  ( +- 46,23% )
           3          cpu-migrations            #    0,002 K/sec                  ( +- 43,17% )
          776         page-faults               #    0,636 K/sec                  ( +- 0,10% )
      1.805.160.393     cycles                    #    1,479 GHz                    ( +- 0,48% )
      2.057.058.241     instructions              #    1,14   insn per cycle        ( +- 0,07% )
      <not supported>   branches
      11.469.420        branch-misses                ( +- 0,58% )

      1,396696785 seconds time elapsed              ( +- 8,49% )
```

Profiling (2)

Gprof command

Almost all of the operations come from the function “kernel_dynprog”

Goal of parallelization

index	% time	self	children	called	name
<spontaneous>					
[1]	100.0	0.00	6.27		main [1]
		6.24	0.00	1/1	kernel_dynprog [2]
		0.03	0.00	3/3	polybench_alloc_data [3]
		0.00	0.00	1/1	polybench_timer_start [10]
		0.00	0.00	1/1	init_array [6]
		0.00	0.00	1/1	print_array [12]
		0.00	0.00	1/1	polybench_timer_print [9]
		0.00	0.00	1/1	polybench_timer_stop [11]

[2]	99.5	6.24	0.00	1/1	main [1]
		6.24	0.00	1	kernel_dynprog [2]

[3]	0.5	0.03	0.00	3/3	main [1]
		0.03	0.00	3	polybench_alloc_data [3]
		0.00	0.00	3/3	xmalloc [4]

[4]	0.0	0.00	0.00	3	polybench_alloc_data [3]
		0.00	0.00	3	xmalloc [4]

		0.00	0.00	1/2	polybench_timer_start [10]
		0.00	0.00	1/2	polybench_timer_stop [11]
[5]	0.0	0.00	0.00	2	rtclock [5]

[6]	0.0	0.00	0.00	1/1	main [1]
		0.00	0.00	1	init_array [6]

[7]	0.0	0.00	0.00	1/1	polybench_prepare_instruments [8]
		0.00	0.00	1	polybench_flush_cache [7]

[8]	0.0	0.00	0.00	1/1	polybench_timer_start [10]
		0.00	0.00	1	polybench_prepare_instruments [8]
		0.00	0.00	1/1	polybench_flush_cache [7]

[9]	0.0	0.00	0.00	1/1	main [1]
		0.00	0.00	1	polybench_timer_print [9]

[10]	0.0	0.00	0.00	1/1	main [1]
		0.00	0.00	1	polybench_timer_start [10]
		0.00	0.00	1/2	rtclock [5]
		0.00	0.00	1/1	polybench_prepare_instruments [8]

[11]	0.0	0.00	0.00	1/1	main [1]
		0.00	0.00	1	polybench_timer_stop [11]
		0.00	0.00	1/2	rtclock [5]

[12]	0.0	0.00	0.00	1/1	main [1]
		0.00	0.00	1	print_array [12]

Improving the algorithm

dynprog.c

```
/* Main computational kernel. The whole function will be timed,
   including the call and return. */
static void kernel_dynprog(int tsteps, int length,
                           DATA_TYPE POLYBENCH_2D(c, LENGTH, LENGTH, length, length),
                           DATA_TYPE POLYBENCH_2D(W, LENGTH, LENGTH, length, length),
                           DATA_TYPE POLYBENCH_3D(sum_c, LENGTH, LENGTH, LENGTH, length, length, length),
                           DATA_TYPE *out)
{
    int iter, i, j, k;

    DATA_TYPE out_l = 0;

    for (iter = 0; iter < _PB_TSTEPS; iter++)
    {
        for (i = 0; i <= _PB_LENGTH - 1; i++)
            for (j = 0; j <= _PB_LENGTH - 1; j++)
                c[i][j] = 0;
        for (i = 0; i <= _PB_LENGTH - 2; i++)
        {
            for (j = i + 1; j <= _PB_LENGTH - 1; j++)
            {
                sum_c[i][j][i] = 0;
                for (k = i + 1; k <= j - 1; k++)
                    sum_c[i][j][k] = sum_c[i][j][k - 1] + c[i][k] + c[k][j];
                c[i][j] = sum_c[i][j][j - 1] + W[i][j];
            }
        }
        out_l += c[0][_PB_LENGTH - 1];
    }
    *out = out_l;
}
```

rew_dynprog.c

```
/* Main computational kernel. The whole function will be timed,
   including the call and return. */
static void kernel_dynprog(int tsteps, int length,
                           DATA_TYPE POLYBENCH_1D(c, LENGTH, length),
                           DATA_TYPE POLYBENCH_1D(W, LENGTH, length),
                           DATA_TYPE sum_c,
                           DATA_TYPE *out)
{
    DATA_TYPE out_l = 0;
    sum_c = 0;

    for (int i = 1; i < _PB_LENGTH; i++)
    {
        for (int j = 1; j < i; j++)
            sum_c += c[j];
        c[i] = sum_c + W[i];
        sum_c = 0;
    }

    for (int k = 0; k < _PB_TSTEPS; k++)
        out_l += c[_PB_LENGTH - 1];

    *out = out_l;
}
```

Performance

STANDARD DATASET

dynprog.c = 1.190552 s

rew_dynprog.c = 0.000034 s

rew_dynprog.c also
requires less memory

```
EXT_CFLAGS="-DLENGTH=10000 -DTSTEPS=1"
```

```
0.026087
```

dynprog.c

```
[PolyBench] posix_memalign: cannot allocate memory.
```

OpenMP version

Parallel region for the inner loop



Reduction for sum_c

```
/* Main computational kernel. The whole function will be timed,
   including the call and return. */
static void kernel_dynprog(int tsteps, int length,
                           DATA_TYPE POLYBENCH_1D(c, LENGTH, length),
                           DATA_TYPE POLYBENCH_1D(W, LENGTH, length),
                           DATA_TYPE sum_c,
                           DATA_TYPE *out)
{
    DATA_TYPE out_l = 0;
    sum_c = 0;

    for (int i = 1; i < _PB_LENGTH; i++)
    {
        #pragma omp parallel for num_threads(NTHREADS) reduction(+:sum_c)
        for (int j = 1; j < i; j++)
            sum_c += c[j];
        c[i] = sum_c + W[i];
        sum_c = 0;
    }

    for (int k = 0; k < _PB_TSTEPS; k++)
        out_l += c[_PB_LENGTH - 1];

    *out = out_l;
}
```

Graphics accelerator version

- Enter and exit data from the GPU in the external loop
- Teams and distribution in the inner loop

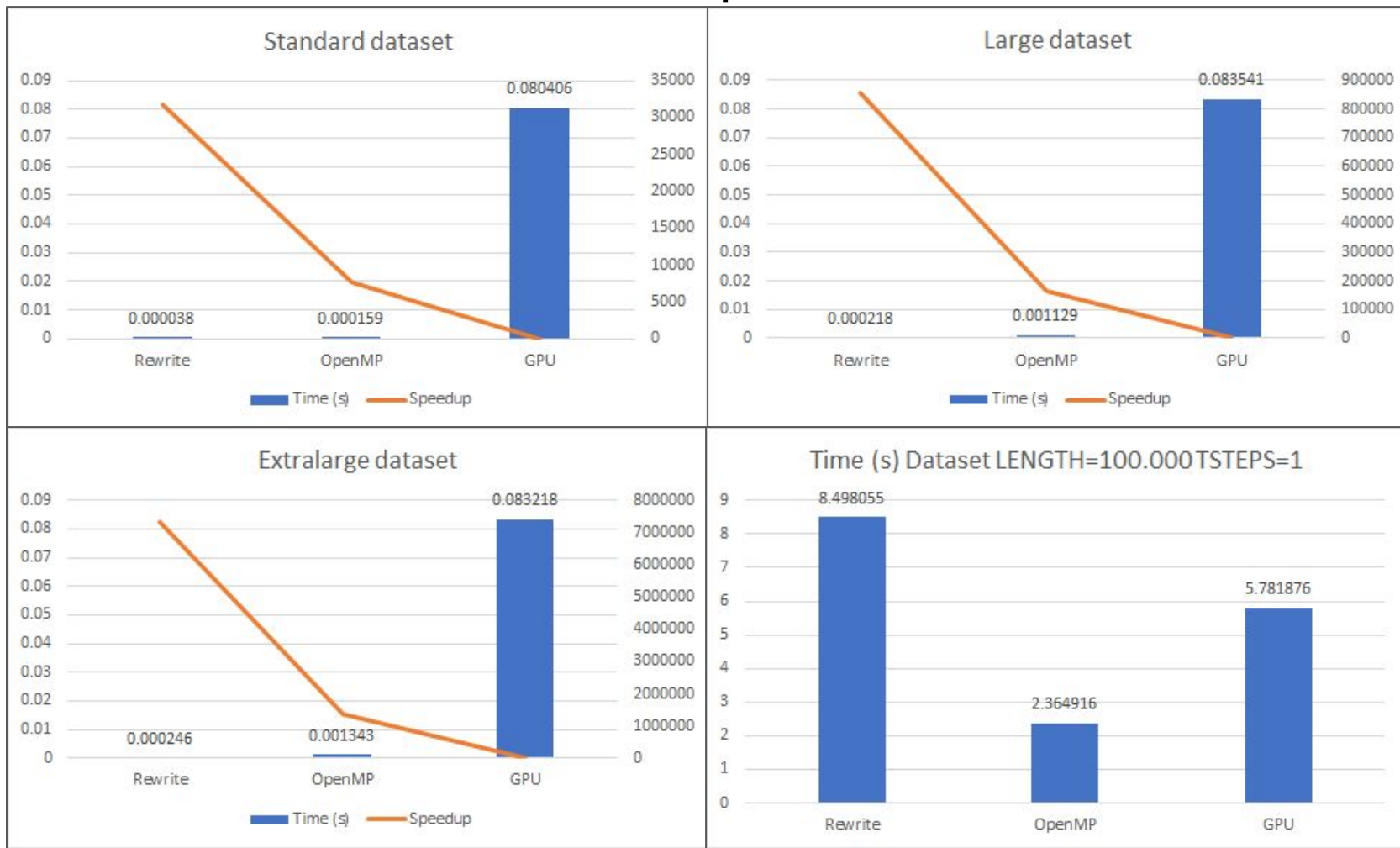
```
/* Main computational kernel. The whole function will be timed,
   including the call and return. */
static void kernel_dynprog(int tsteps, int length,
                           DATA_TYPE POLYBENCH_1D(c, LENGTH, length),
                           DATA_TYPE POLYBENCH_1D(W, LENGTH, length),
                           DATA_TYPE sum_c,
                           DATA_TYPE *out)
{
    DATA_TYPE out_l = 0;
    sum_c = 0;

    #pragma omp target enter data map(to: W[0:length-1], length, sum_c)
    for (int i = 1; i < _PB_LENGTH; i++)
    {
        #pragma omp teams num_teams(i/NTHREADS_GPU) thread_limit(NTHREADS_GPU)
        #pragma omp distribute parallel for reduction(+:sum_c) \
            num_threads(NTHREADS_GPU) dist_schedule(static, NTHREADS_GPU)
        for (int j = 1; j < i; j++)
            sum_c += c[j];
        c[i] = sum_c + W[i];
        sum_c = 0;
    }
    #pragma omp target exit data map(from: c[length-1])

    for (int k = 0; k < _PB_TSTEPS; k++)
        out_l += c[_PB_LENGTH - 1];

    *out = out_l;
}
```

Final comparison



**Thanks for
attention**