Experiences of Porting Structured and Unstructured Stencil Applications to FPGA using SYCL

Zadok Storkey, Steven Wright, Ian Gray

1. Context

1. **Context**
2. Approach
3. Results & Discussion
4. Future Work

Context

The potential benefits to reducing power consumption of HPC systems are clear:

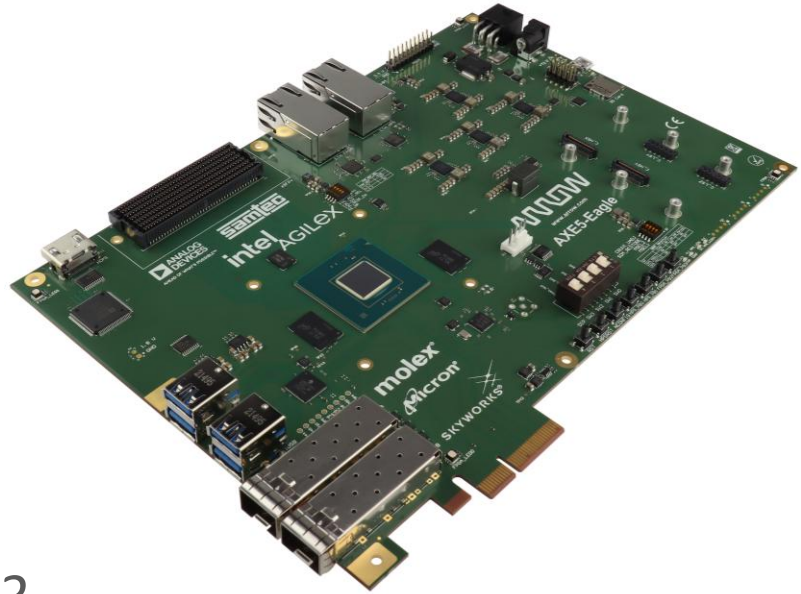
- Environmental - i.e. smaller carbon footprint
- Financial - i.e. lower power and cooling cost
- Infrastructural - i.e. less burden on energy infrastructure and water supply

Finding alternative low-energy accelerators is a good way of achieving this aim.

Context

FPGAs potentially fit this niche:

- Low energy
- Highly parallel



However, there are significant challenges to effectively utilise them:

- How can we use them *portably?*
- How can we use them *productively?*

2. Approach

1. Context
2. **Approach**
3. Results & Discussion
4. Future Work

Approach

Outline

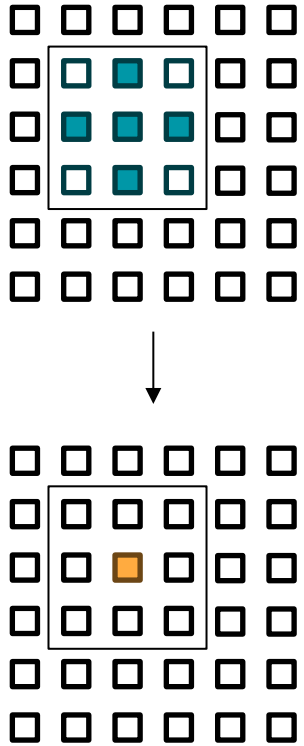
- Goal: Evaluate the process and results of adapting an existing CPU application portably for FPGA
- Application: Structured and Unstructured version of the *heat* mini application [1]
- Programming model: Intel OneAPI implementation of SYCL
- Targets: AMD EPYC 7713 CPU, Intel Stratix 10 MX FPGA
- Environment: EPCC Testbed

[1] T. Deakin, S. McIntosh-Smith, *Evaluating the Performance of HPC-Style SYCL Applications*, in: Proceedings of the International Workshop on OpenCL, IWOCL '20, 2020, pp. 1–11. 10.1145/3388333.3388643.

Approach

Application

- Based on the *heat* [1] mini-application
- Solves the heat equation over a 2D grid, using a finite difference method
- Uses an iterative approach with a 5-point stencil
- Two versions: structured grid and unstructured grid
- Unstructured version further divided by degree of memory shuffle



[1] T. Deakin, S. McIntosh-Smith, *Evaluating the Performance of HPC-Style SYCL Applications*, in: Proceedings of the International Workshop on OpenCL, IWOCL '20, 2020, pp. 1–11. 10.1145/3388333.3388643.

Approach

Implemented in three ways:

- CPU (Parallel For) – SYCL controls loop flow and assigns different iterations to different CPU cores
- FPGA (Parallel For) – SYCL controls loop flow and runs the kernel, which is synthesised and implemented onto an FPGA
- FPGA (Single Task) – The kernel maintains control of the loop flow, and is synthesised and implemented onto an FPGA

3. Results & Discussion

1. Context
2. Approach
3. **Results & Discussion**
4. Future Work

Results & Discussion

Discussion – How single source is the application actually?

```
cgh.parallel_for<class solve_kernel>(sycl::range<2>(nrows, ncols), [=](sycl::id
    <2> idx) [[sycl::reqd_work_group_size(16, 16)]] {
    const long row = idx[0];
    const long col = idx[1];
    const double temp_centre = get_previous_temps[row][col];
    double temp_adj_sum = 0;
    for (int face = 0; face < 4; face++) {
        const long adj_row = get_adj_row[row][col][face];
        const long adj_column = get_adj_col[row][col][face];
        if (adj_row != Neighbour::BOUNDARY) {
            adj_temperature_sum += get_previous_temps[adj_row][adj_col];
        }
    }
    set_next_temps[row][column] = r2 * temp_centre + r * temp_adj_sum;
});
```

Table 3: Code divergence of the kernel by target in lines (structured | unstructured)

	CPU	FPGA (parallel-for)	FPGA (single-task)
CPU	-	1 | 3	6 | 8
FPGA (parallel-for)	1 | 3	-	6 | 6
FPGA (single-task)	6 | 8	6 | 6	-

Results & Discussion

Discussion – How single source is the application actually?

```
cgh.parallel_for<class solve_kernel>(sycl::range<2>(nrows, ncols), [=](sycl::id
    <2> idx) [[sycl::reqd_work_group_size(16, 16)]] {
    const long row = idx[0];
    const long col = idx[1];
    const double temp_centre = get_previous_temps[row][col];
    double temp_adj_sum = 0;
    for (int face = 0; face < 4; face++) {
        const long adj_row = get_adj_row[row][col][face];
        const long adj_column = get_adj_col[row][col][face];
        if (adj_row != Neighbour::BOUNDARY) {
            adj_temperature_sum += get_previous_temps[adj_row][adj_col];
        }
    }
    set_next_temps[row][column] = r2 * temp_centre + r * temp_adj_sum;
});
```

```
cgh.parallel_for<class solve_kernel>(sycl::range<2>(nrows, ncols), [=](sycl::id
    <2> idx)
    [[intel::kernel_args_restrict, intel::num_simd_work_items(16), sycl::reqd
    _work_group_size(1, 16)]] {
    const long row = idx[0];
    const long column = idx[1];
    const double temp_centre = get_previous_temps[row][col];
    double temp_adj_sum = 0;
    #pragma unroll
    [[intel::ivdep]]
    for (int face = 0; face < 4; face++) {
        const long adj_row = get_adj_row[row][col][face];
        const long adj_column = get_adj_col[row][col][face];
        if (adj_row != Neighbour::BOUNDARY) {
            adj_temperature_sum += get_previous_temps[adj_row][adj_col];
        }
    }
    set_next_temps[row][column] = r2 * temp_centre + r * temp_adj_sum;
});
```

Table 3: Code divergence of the kernel by target in lines (structured | unstructured)

	CPU	FPGA (parallel-for)	FPGA (single-task)
CPU	-	1 | 3	6 | 8
FPGA (parallel-for)	1 | 3	-	6 | 6
FPGA (single-task)	6 | 8	6 | 6	-

Results & Discussion

Discussion – How single source is the application actually?

```
cgh.parallel_for<class solve_kernel>(sycl::range<2>(nrows, ncols), [=](sycl::id
    <2> idx) [[sycl::reqd_work_group_size(16, 16)]] {
    const long row = idx[0];
    const long col = idx[1];
    const double temp_centre = get_previous_temps[row][col];
    double temp_adj_sum = 0;
    for (int face = 0; face < 4; face++) {
        const long adj_row = get_adj_row[row][col][face];
        const long adj_column = get_adj_col[row][col][face];
        if (adj_row != Neighbour::BOUNDARY) {
            adj_temperature_sum += get_previous_temps[adj_row][adj_col];
        }
    }
    set_next_temps[row][column] = r2 * temp_centre + r * temp_adj_sum;
});
```

```
cgh.parallel_for<class solve_kernel>(sycl::range<2>(nrows, ncols), [=](sycl::id
    <2> idx)
    [[intel::kernel_args_restrict, intel::num_simd_work_items(16), sycl::reqd
    _work_group_size(1, 16)]] {
    const long row = idx[0];
    const long column = idx[1];
    const double temp_centre = get_previous_temps[row][col];
    double temp_adj_sum = 0;
    #pragma unroll
    [[intel::ivdep]]
    for (int face = 0; face < 4; face++) {
        const long adj_row = get_adj_row[row][col][face];
        const long adj_column = get_adj_col[row][col][face];
        if (adj_row != Neighbour::BOUNDARY) {
            adj_temperature_sum += get_previous_temps[adj_row][adj_col];
        }
    }
    set_next_temps[row][column] = r2 * temp_centre + r * temp_adj_sum;
});
```

Table 3: Code divergence of the kernel by target in lines (structured | unstructured)

	CPU	FPGA (parallel-for)	FPGA (single-task)
CPU	-	1 | 3	6 | 8
FPGA (parallel-for)	1 | 3	-	6 | 6
FPGA (single-task)	6 | 8	6 | 6	-

```
cgh.single_task<class solve_kernel>([=]() [[intel::kernel_args_restrict]] {
    [[intel::ivdep]]
    for (long row = 0; row < nrows; row++) {
        #pragma unroll 32
        [[intel::ivdep]]
        for (long column = 0; column < ncols; column++) {
            const double temp_centre = get_previous_temps[row][col];
            double temp_adj_sum = 0;
            #pragma unroll
            [[intel::ivdep]]
            for (int face = 0; face < 4; face++) {
                const long adj_row = get_adj_row[row][col][face];
                const long adj_column = get_adj_col[row][col][face];
                if (adj_row != Neighbour::BOUNDARY) {
                    adj_temperature_sum += get_previous_temps[adj_row][adj_col];
                }
            }
            set_next_temps[row][column] = r2 * temp_centre + r * temp_adj_sum;
        });
    }
});
```

Results & Discussion

Discussion – How easy was it to develop with FPGAs as a target

- The features of the toolchain helped with the productivity issues caused by the time taken to build FPGA images:
  - Fast-compile
  - Image re-use
  - Pre-synthesis reports
- Nevertheless, build times remain a challenge

Table 4: Build times (hh:mm:ss) by FPGA target variation and grid type

	Parallel-for	Single-task
Structured	03:19:28	04:38:37
Unstructured	02:47:57	06:36:37

Results & Discussion

Results – Solve times

Table 1: Solve times (s) by target and grid type

	CPU	FPGA (parallel-for)	FPGA (single-task)
Structured	15.27	252.90	208.42
Unstructured (No Shuffle)	88.60	8566.28	2074.41
Unstructured (Shuffle $(0.5 \cdot n_{cells})$)	91.93	9384.59	1790.39
Unstructured (Shuffle $(1.0 \cdot n_{cells})$)	93.91	9578.58	1795.50
Unstructured (Shuffle $(2.0 \cdot n_{cells})$)	96.27	9676.64	1795.82
Unstructured (Full Shuffle)	95.96	9882.52	2899.41

Results & Discussion

Results – Solve times

Table 1: Solve times (s) by target and grid type

	CPU	FPGA (parallel-for)	FPGA (single-task)
Structured	15.27	252.90	208.42
Unstructured (No Shuffle)	88.60	8566.28	2074.41
Unstructured (Shuffle $(0.5 \cdot n_{cells})$)	91.93	9384.59	1790.39
Unstructured (Shuffle $(1.0 \cdot n_{cells})$)	93.91	9578.58	1795.50
Unstructured (Shuffle $(2.0 \cdot n_{cells})$)	96.27	9676.64	1795.82
Unstructured (Full Shuffle)	95.96	9882.52	2899.41

Results & Discussion

Results – Solve times

Table 1: Solve times (s) by target and grid type

	CPU	FPGA (parallel-for)	FPGA (single-task)
Structured	15.27	252.90	208.42
Unstructured (No Shuffle)	88.60	8566.28	2074.41
Unstructured (Shuffle $(0.5 \cdot n_{cells})$)	91.93	9384.59	1790.39
Unstructured (Shuffle $(1.0 \cdot n_{cells})$)	93.91	9578.58	1795.50
Unstructured (Shuffle $(2.0 \cdot n_{cells})$)	96.27	9676.64	1795.82
Unstructured (Full Shuffle)	95.96	9882.52	2899.41

Results & Discussion

Results – Energy usage

Table 2: Energy Usage (kJ) by target and grid type

	CPU	FPGA (parallel-for)	FPGA (single-task)
Structured	2.92	6.02*	8.82*
Unstructured (No Shuffle)	17.33	212.70*	91.54*
Unstructured (Shuffle ($0.5 \cdot n_{cells}$))	18.23	233.02*	79.01*
Unstructured (Shuffle ($1.0 \cdot n_{cells}$))	18.64	237.84*	79.24*
Unstructured (Shuffle ($2.0 \cdot n_{cells}$))	18.78	240.27*	79.25*
Unstructured (Full Shuffle)	24.52	245.38*	127.95*

* estimates provided by Quartus

Results & Discussion

Results – Energy usage

Table 2: Energy Usage (kJ) by target and grid type

	CPU	FPGA (parallel-for)	FPGA (single-task)
Structured	2.92	6.02*	8.82*
Unstructured (No Shuffle)	17.33	212.70*	91.54*
Unstructured (Shuffle ($0.5 \cdot n_{cells}$))	18.23	233.02*	79.01*
Unstructured (Shuffle ($1.0 \cdot n_{cells}$))	18.64	237.84*	79.24*
Unstructured (Shuffle ($2.0 \cdot n_{cells}$))	18.78	240.27*	79.25*
Unstructured (Full Shuffle)	24.52	245.38*	127.95*

* estimates provided by Quartus

Results & Discussion

Results – Energy usage

Table 2: Energy Usage (kJ) by target and grid type

	CPU	FPGA (parallel-for)	FPGA (single-task)
Structured	2.92	6.02*	8.82*
Unstructured (No Shuffle)	17.33	212.70*	91.54*
Unstructured (Shuffle ($0.5 \cdot n_{cells}$))	18.23	233.02*	79.01*
Unstructured (Shuffle ($1.0 \cdot n_{cells}$))	18.64	237.84*	79.24*
Unstructured (Shuffle ($2.0 \cdot n_{cells}$))	18.78	240.27*	79.25*
Unstructured (Full Shuffle)	24.52	245.38*	127.95*

* estimates provided by Quartus

Results & Discussion

Discussion – Solve times and Energy Usage

- All FPGA builds perform rather poorly
- In terms of energy usage, FPGAs perform closer to CPUs, but still perform worse than expected
- Low computational intensity means that problem is likely limited more by bandwidth than compute
  - Supported by hardware resource utilisation amounts
- Age of hardware is also a consideration

4. Future Work

1. Context
2. Approach
3. Results & Discussion
4. **Future Work**

Future work

Future work

- Expand Hardware Targets
  - Compare GPU
  - Compare more recent Intel FPGAs
  - Compare AMD FPGA using SYCL for Vitis
- Better energy data collection

Future work

Future work

- Investigate the causes behind the poor performance
  - Investigate behaviour of memory transfers during kernel execution
  - Experiment with increasing the arithmetic intensity of the kernel
- Expand to larger, more complex unstructured applications

Experiences of Porting Structured and Unstructured Stencil Applications to FPGA using SYCL

Zadok Storkey, Steven Wright, Ian Gray