

03\_shared-memory/02\_thread-affinity

# Hands-on: Understanding thread affinity via STREAM

---

- Choose either C/C++ (`c`) or Fortran (`fortran`). Both of them are fine, as well.

## C/C++

How to execute

### 1. Edit a job script

- Before trying this hands-on, you need to do `00_stream` to compile STREAM.
- On `c/` or `ctrad/`, We have two working directories, `fj_zfill.close` and `fj_zfill.spread`. Under each of directories, you can find:
  - `run.sh` : a script to execute STREAM
  - `task.sh`: a job script to run STREAM with different kinds of settings
- edit `task.sh` (modify `--gname` option).
- Edit `BINDIR` variable in `run.sh` before the execution. You need to write your installed location of STREAM binary (e.g., `stream.exe`) there.

### 2. Run program

- You can run the program either:

```
## Here is an example of c/fj_zfill.close.
# To run as a batch job
$ cd c/fj_zfill.close
$ pjsub task.sh
# Or, to run in an interactive job
$ cd c/fj_zfill.close
$ bash task.sh
```

- Each of the cases in the Exercises will be completed within 3-4 minutes.
  - For safety, we set the job elapsed time in the job scripts is 6 minutes.

### Exercises A

- E1: Check `task.sh` in `c/fj_zfill.close` and `c/fj_zfill.spread`. Alternatively, you can check the files in `ctrad/`. You can find that the thread affinity is set by `TAFF` variable (See `run.sh`, as well).
- E2: Run the STREAM benchmark (`stream.exe`). Check the behaviors of `Triad`-based bandwidth with change of the number of threads. In particular, we suggest that you observe the cases of 12, 24, and 48 threads.

### Exercises B (advanced)

- E3: Consider a difference between `fj_zfill.close` and any examples in `01_bandwidth` from a core-binding point of view.
- E4: Measure the case of setting `TAFF` variable as `hbarrier` (i.e., use of in-core hardware barrier); It means that one uses the in-core hardware barrier in A64FX.
- E5: Try a more direct manipulation of core-binding settings with `GOMP_CPU_AFFINITY` in `run.sh`.

## Fortran

### How to execute

#### 1. Edit a job script

- Before trying this hands-on, you need to do `00_stream` to compile STREAM.
- We have one working directory, `fortran/fj_zfill.close`. Under the directory, you can find:
  - `run.sh`: a script to execute STREAM
  - `task.sh`: a job script to run STREAM with different kinds of settings
- edit `task.sh` (modify `--gname` option).
- Edit `BINDIR` variable in `run.sh` before the execution. You need to write your installed location of STREAM binary (e.g., `stream.exe`) there.

#### 2. Run program

- You can run the program either:

```
## Here is an example of fortran/fj_zfill.close.
# To run as a batch job
$ cd fortran/fj_zfill.close
$ pjsub task.sh
# Or, to run in an interactive job
$ cd fortran/fj_zfill.close
$ bash task.sh
```

- Each of the cases in the Exercises will be completed within 3-4 minutes.
  - For safety, we set the job elapsed time in the job scripts is 6 minutes.

### Exercises A

- E1: Check `task.sh` in `fortran/fj_zfill.close` and `fortran/fj_zfill.spread`. You can find that the thread affinity is set by `TAFF` variable (See `run.sh`, as well).
- E2: Run the STREAM benchmark (`stream.exe`). Check the behaviors of `Triad`-based bandwidth with change of the number of threads. In particular, we suggest that you observe the cases of 12, 24, and 48 threads.

### Exercises B (advanced)

- E3: Consider a difference between `fj_zfill.close` and any examples in `01_bandwidth` from a core-binding point of view.

- E4: Measure the case of setting `TAFF` variable as `spread` or `hbarrier` (i.e., use of in-core hardware barrier); It means that one uses the in-core hardware barrier in A64FX.
- E5: Try a more direct manipulation of core-binding settings with `GOMP_CPU_AFFINITY` in `run.sh`.