Empirical Experiment Bare-metal Fork-Join Scheduling Algorithm

This folder provides the sources and scripts to reproduce the results and data from the empirical experiment in **Section VIII**.

Folder Contents

- `README.md` - the Markdown source of this file that explains how to use.
- `bin` folder includes several core scripts that help with the empirical evaluation. The scripts are as follows:
  - `amortize.py` - amortizes the base and incremental cost of all of the MRTC objects.
  - `bicosts.py` - calculates the base and incremental costs for the MRTC benchamrks.
  - `cycles.py` - calculates the number of instruction cycles per core.
  - `mqemu.py` - runs a riscv32-img within qemu and calculates a series of data such as cycle average, cache misses and a longest schedule.
  - `poc-graph.py` - generates a graph based off data generated from mqemu.py
  - `qemu.sh` - Runs a RISC-V simulated machine given a kernel image.
- `fjlib` folder contains an early alpha libray of the fork-join scheduling library used by fork-join tasks.
- `fjlibgroup` fork-join scheduling library but headers set to different parallel sections for different fork-join group tasks.
- `fjtasks` folder contains folders of each task that was performed and evaluated in the RISC-V empircal experiment. The tasks were:
  - FJ-791
  - FJ-956
  - FJ-484
- `makefile` - running **make** will compile and evaluate all tasks in the fjtasks folder.
- `export.sh` - exports generated graphs to a designated folder.

Step 0 - Setup and Prerequisites

Download the fjoin.ova with the required software and environment configuration.

Step 0 - Setup and Prerequisites (without OVA).

Environment

The following instructions have been tested on the Linux distribution **Ubuntu 22.04 LTS**.

The default user interface on **Ubuntu 22.04 LTS** is DASH however, several scripts depend on the shell interface to be BASH. This can be changed with the following commands:

```
sudo dpkg-reconfigure dash
( Select <No> when requested to use dash as the default system shell )
```

The PATH variable for the user will need to be updated to accomodate future software dependencies. The command below will need to be updated if the installation paths for **QEMU** and the **RISC-V toolchain** are changed in the next step.

```
export PATH=$PATH:/opt/riscv/bin:/home/user/bin/qemu-8.0.0/bin
(user should be changed to the current user of the system)
```

Software dependencies

The bare-metal Fork-Join scheduling algorithm was implemented for a 32bit 8-core RISC-V processor thus requires several pieces of software to run succesfully. The list of software and instructions required to run the experiment and reproduce the graphs can be found below:

RISCV Tool Chain

Due to the empirical experiment being a bare-metal the RISCV Tool Chain needs to be installed from source and configured to accomodate this. First the required dependecies for the tool chain must be installed.

```
 $ sudo apt-get install autoconf automake autotools-dev curl python3  python3-pip
libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf
libtool patchutils bc zlib1g-dev libexpat-dev ninja-build git cmake libglib2.0-dev
libpixman-1-dev
```

After installing the dependencies the RISCV source can be obtained by:

```
 $ git clone https://github.com/riscv/riscv-gnu-toolchain
```

The source directory needs to be prepped and can be configured by:

```
cd riscv-gnu-toolchain
git checkout tags/2023.01.31
mkdir build
cd build
../configure --prefix=/opt/riscv --with-arch=rv32gc --with-abi=ilp32d
sudo make -j NUMBER-OF-CORES-TO-USE
```

QEMU

The experiment was only evaluated with a stable version of QEMU 8.0.0.
The source can be obtained with:

```
git clone https://gitlab.com/qemu-project/qemu.git qemu
```

Change into new directory and checkout the appropriate branch:

```
cd qemu
git checkout c1eb2ddf
```

Create build directory and configure source:

```
mkdir build
cd build
```

```
../configure --prefix=${HOME}/bin/qemu-8.0.0 --target-list="riscv32-softmmu riscv64-softmmu riscv32-linux-user riscv64-linux-user" --enable-plugins --enable-kvm --enable-multiprocess --enable-qcow1 --enable-qed --enable-sdl --enable-tcg --enable-tools --disable-sdl
```

Make QEMU from the source and install:

```
make -j NUMBER-OF-CORES-TO-USE
make install
```

The QEMU source does not make its plugins with its scripts and needs to be done and copied manually:

```
cd contrib/plugins
make
cd ../../
cp -r contrib /home/user/bin/qemu-8.0.0/contrib
```

FJCOLO graph dependencies

The data and graph generated from the ran experiments requires several graph libraries and fonts.
FJCOLO graph dependencies can be installed with:

```
sudo apt-get install python3 python3-numpy rename texlive-latex-extra -y cm-super dvipng
```

Additional required python modules are:

```
python3 -m pip install -U alive-progress matplotlib pandas
```

Step 1 - Compile the Fork-join Library

Compiling the Fork-join library is a simple process of running the makefile in the `fjlib` directory:

```
cd fjlib
make
```

The default configuration of the Fork-join library takes an assumption of one parallel section and a max number of 256 threads per core schedule. This assumption can be

changed within the fjlib folder before running the make file and further details can be found in the readme in fjlib.

Step 2 - Compile the Fork-join tasks and graphs

For convenience a `makefile` has been provided in the fjcolo/simul folder. The `makefile` will compile, run, and generate all graphs for all fjtasks. Each respective graph can be found in each task folder e.g fjtasks/task-484.
To run all build all fjtasks enter the command:

```
make
```

Step 3 - Verify the generated Graphs

After running Step 2 newly generated graphs should have been generated in the respective task `graphs` folder. **Navigate to the directory and verify the graphs for each task that was compiled:**

```
cd fjtasks/task-XXX/graphs
```