CSE 539S Project 3 Report

Amanda Hua, Funda Atik

In this project, we implemented the return protocols for a spawned child return (Closure_return) and a called child return (Cilk_set_return).

1.a. Our Cilk_set_return() method receives a worker as its parameter. We implement the return protocol by following the steps below:

- Lock the worker's ReadyDeque
- Extract the returning child from the bottom of the given worker's ReadyDeque
- Lock the child & the child's call_parent
- Place the parent's closure onto the bottom of the ReadyDeque
- Remove the link between child and parent using Closure_remove_callee()
- Perform an unconditional steal of the call parent's closure by calling setup_call_parent_resumption(), which updates the parent's status and worker variables
- Unlock the parent, child, and ReadyDeque, in order

1.b. Our Closure_return() method receives a worker and a returning child closure as its parameters. We perform the following steps to implement the return protocol.

- Lock the child & the child's spawn_parent
- Decrement the join counter of the parent
- If the returning child does not have a left sibling or the parent's fiber_child variable has not been set, this means that the child is not the leftmost child of the parent, so we will:
    - Deallocate the child's fiber to the fiber pool
- Otherwise, the returning child is the leftmost child of the parent, and we will:
    - Assign the parent's fiber_child to be the returning child's fiber
- If the parent's join counter is zero at this point and the parent's status is CLOSURE_SUSPENDED and the parent has no outstanding cilk callees, this means that the child must execute a provably good steal, where we will:
    - Call setup_for_sync(), which sets the parent's fiber to its leftmost child's fiber (which has been stored in fiber_child)
    - Set the worker's local state provably_good_steal status high
    - Set the parent's status to CLOSURE_READY
    - Set the parent to be the return closure
- Remove the link between the parent and child using Closure_remove_child (given code), which:
- Unlock the child and the parent, in order
- Returns a null closure (default) if a provably good steal was performed, returns the parent closure in the provably good steal scenario

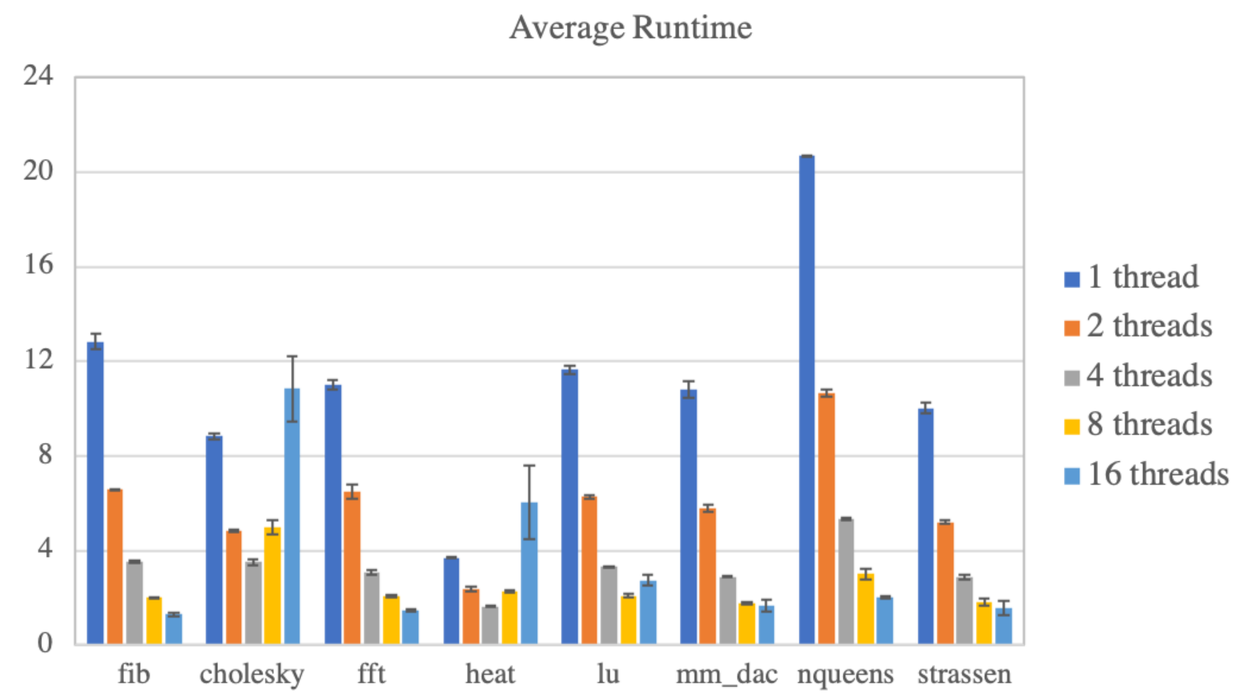
2. The output from running the commands in make check in the Makefile in bench.(in @linuxlab007)

```
./fib 42 --nproc 16
Cheetah: invoking user main with 16 workers.
Result: 267914296
Running time 1: 1.200025 s
Running time 2: 1.106036 s
Running time 3: 1.105412 s
Running time 4: 1.105165 s
Running time 5: 1.105623 s
Running time average: 1.124452 s
Std. dev: 0.042248 s (3.757%)
./cholesky --nproc 16 -n 4000 -z 40000 -c
Cheetah: invoking user main with 16 workers.
Running time 1: 0.864665 s
Running time 2: 0.865831 s
Running time 3: 0.844453 s
Running time 4: 0.842621 s
Running time 5: 0.855506 s
Running time average: 0.854615 s
Std. dev: 0.010894 s (1.275%)
Now check result ...

Cilk Example: cholesky
Error: 0.000000

Options: original size     = 4000
         original nonzeros = 40000
         input nonzeros    = 40096
         input blocks      = 35850
         output nonzeros   = 6350806
         output blocks     = 477040

./fft --nproc 16 -n 67108864
Cheetah: invoking user main with 16 workers.
Testing cos: -0.702713
Running time 1: 1.240781 s
Running time 2: 1.250175 s
Running time 3: 1.204489 s
Running time 4: 1.209309 s
Running time 5: 1.206744 s
Running time average: 1.222300 s
Std. dev: 0.021486 s (1.758%)

cilk example: fft
options:  number of elements   n = 67108864

./heat --nproc 16 -nx 2048 -ny 2048 -nt 500
Cheetah: invoking user main with 16 workers.
Testing exp: -0.000000
```

```
Running time 1: 1.335541 s
Running time 2: 1.209398 s
Running time 3: 1.205553 s
Running time 4: 1.206048 s
Running time 5: 1.208056 s
Running time average: 1.232919 s
Std. dev: 0.057388 s (4.655%)

Cilk Example: heat

   dx = 0.000767
   dy = 0.000767
   dt = 0.000000

 Stability Value for explicit method must be > 0:  0.499321

Options: granularity = 10
         nx          = 2048
         ny          = 2048
         nt          = 500
./lu --nproc 16 -n 4096 -c
Cheetah: invoking user main with 16 workers.
Running time 1: 0.938374 s
Running time 2: 0.933059 s
Running time 3: 0.936487 s
Running time 4: 0.932406 s
Running time 5: 0.931502 s
Running time average: 0.934365 s
Std. dev: 0.002929 s (0.313%)
Now check result ...

Cilk Example: lu
Options: (n x n matrix) n = 4096

./mm_dac --nproc 16 -n 2048 -c
Cheetah: invoking user main with 16 workers.
Running time 1: 0.867835 s
Running time 2: 0.869898 s
Running time 3: 0.867513 s
Running time 4: 0.865437 s
Running time 5: 0.868703 s
Running time average: 0.867877 s
Std. dev: 0.001647 s (0.190%)
```

```
Checking result.
MM_dac test passed.
./nqueens --nproc 16 14
Cheetah: invoking user main with 16 workers.
Running ./nqueens with n = 14.
Running time 1: 1.695522 s
Running time 2: 1.657656 s
Running time 3: 1.652501 s
Running time 4: 1.651849 s
Running time 5: 1.652925 s
Running time average: 1.662091 s
Std. dev: 0.018829 s (1.133%)
Total number of solutions : 365596
./strassen --nproc 16 -n 4096 -c
Cheetah: invoking user main with 16 workers.
Checking results ...
Checking results ...
Checking results ...
Checking results ...
Checking results ...
Running time 1: 1.003195 s
Running time 2: 0.995414 s
Running time 3: 0.977987 s
Running time 4: 1.100191 s
Running time 5: 1.007874 s
Running time average: 1.016932 s
Std. dev: 0.047913 s (4.712%)

Cilk Example: strassen
Options: n = 4096
```

3. The running times for all benchmarks in bench with CILK DEBUG turned off , on P=1,2,4,8,16 processors; each data point should be the average of 5 runs with standard deviations reported using the same inputs as in make check.



4. We met to talk about the project over Zoom meetings, starting with initial discussion about the exact steps we needed to take to implement the protocol. We wrote out our thoughts on the steps for each return method, and had at first decided to divide the work so that Amanda would implement the called child return and Funda would implement the spawned child return. However, it was much easier to talk about both methods as we implemented them on one computer to compile and test, so we ended up meeting several times to discuss the various bugs we were getting together. Both of us posted piazza questions about problems we were getting with our code, and we both attended office hours with Kyle to clarify our process.