ECE454, Fall 2015

Homework 5: Parallelization

Team Name: Team Herbert!!!

He Zhang (1000347546)
Joo Young Kang (995903330)

Following list summarizes the optimizations our team has done to achieve speedup.

Performance Summary
Original version: 110.86 seconds
Optimized version: 5.35 seconds

#1. Multithreading (number of threads=8) with pthreads API

If size of board is less or equal to 32, our program runs with sequential version of GoL
If input size N is greater than 10000, we print error statement and return inboard.
Otherwise, we initialize 8 threads to run parallel version of GoL and use pthread_join & pthread_barrier_wait as synchronization primitives. Each thread run ⅛ rows of the board.

#2. Loop Unrolling (4x)

Unrolling inner-most loop (j-loop) by 4 times in parallel_game_of_life

#3. Reuse of nearby cells.

During cell calculations, we reused right part of the cells (marked as big-o, O, in below diagrams), thereby number of calls for BOARD. We access them as local variable instead of accessing the matrix to get the number.

Timestamp1:	Timestamp2	….	Timestamp4
o O O	O O o		
o X O	O X o		
o O O	O O o		

Note that we unrolled loop by 4 (Timestamp1 - Timestamp4). In each timestamp calculation, we are saving 5 BOARD calls/calculations.

#4. Optimization of alivep function (util.h)

Changing comparison operator from AND to OR resulted in minor performance increase.

```
alivep (char count, char state)  {
  return (! state && (count == (char) 3)) ||
    (state && (count >= 2) && (count <= 3));  //Original
    (state && (count == 2 || count == 3));  //Optimized  }
```

#5. Optimization of MOD & BOARD helper function

Instead of using provided mod function, we inlined and optimized for bound checking (if an item is located on the edge boundary, return the opposite side.

Initialization of direction variables (no use of MOD function)

```
int up = i? i-1 : nrows-1;
int down = i == (nrows-1)? 0 : i+1;
int right = j ==( ncols-1 )? 0 : j+1;
int left = j? j-1: ncols -1;
```

<Optimized BOARDLICM function>

```
#define BOARD( __board, __i, __j )  (__board[(__i) + LDA*(__j)])  // Original
#define BOARDLICM( __board, __i, __jLDA )  (__board[(__i) + __jLDA])  //  Optimized
```

so instead of calculating LDA*_j each time, we calculated jLDS = j*LDS pass into the BOARD in order to do some loop invariant code motion.

#6. Code Motion and some common subexpression removal

-Loop Invariant Code Motion
-Use of registers for re-used cells mentioned in #2
-Re-use of registers within one iteration.
Does not increase much since compiler already done that for us

#7. Makefile Optimization Level (-O3)

This level give best optimization for speed.