Home Assignment Chapter 4&5 Programming

Yuxuan Jing

April 2020

Program 4.3

We use $f(x) = x^2$ to test the trapezoidal rule. We set $a = 0$, $b = 12$, $n = 40000000$. The number of threads is 4. Four cases are tested, using busy-waiting, mutexes, semaphores, and single thread. The output is as following:

```
yuxuan@yuxuan-XPS-13-9380:~/Dropbox/     k     /HPC/Assignments/Ex4_5_PA/4.3$
    ./run 4 0 12 40000000
With n = 40000000 terms,

Multi-threaded estimate of busy = 576.0
   Elapsed time = 3.204694e+00 seconds
Multi-threaded estimate of mutex = 576.0
   Elapsed time = 6.251407e-02 seconds
Multi-threaded estimate of semaphore = 576.0
   Elapsed time = 6.308293e-02 seconds
Single-threaded estimate = 576.0
   Elapsed time = 2.431848e-01 seconds
```

The answers for all three methods are correct and the run-time is:

Methods	busy-waiting	mutexes	semaphores	single-thread
time[s]	3.20	0.0625	0.0631	0.2433

Some conclusions can be draw:

- The speed of using mutexes and semaphores are almost the same and is much faster than the other two. The reason is that both methods are almost parallel computing thought the whole programs and only merge at the very end of the programs.
- The speed of the busy-waiting method is even slower than the single thread. The reason is that there are too many overheads for each busy-waiting while statement. And the overhead of each while loop is relatively large compared with the content of each while loop.

Program 5.6

We set three inputs of the program $< number\ of\ threads > < number\ of\ producer\ threads > < number\ of\ consumer\ threads >$.

Two kinds of cases are tested:

- Four files with numbers inside each files (input1 10-19, input2 20-29, input3 30-39, input4 40-49). We run the program with 4 threads, and the number pairs of producers and consumers are 2-2, 1-3, 3-1.
- Four files with some text inside (random paragraphs from the textbook). We run the program with 4 threads, 2 producers and 2 consumers.

The output of some tests are showed in Appendix A.

We can draw some conclusions:

- All the test cases are running paralleled.
- The computing time of Enqueue() is shorter than Tokenize()
- For the best performance we need to make a balance of producer threads and consumer threads, which means at a certain time duration the number of lines enqueued is equal to the number of lines tokenized.

$$number\ of\ enqueued\ lines = number\ of\ tokenized\ lines \tag{1}$$

$$number\ of\ enqueued\ lines = \frac{number\ of\ producers \times total\ time}{runtime\ per\ enqueued\ line} \tag{2}$$

$$number\ of\ tokenized\ lines = \frac{number\ of\ consumers \times total\ time}{runtime\ per\ tokenized\ line} \tag{3}$$

$$\frac{number\ of\ producers \times total\ time}{runtime\ per\ enqueued\ line} = \frac{number\ of\ consumers \times total\ time}{runtime\ per\ tokenized\ line} \tag{4}$$

$$\frac{number\ of\ producers}{number\ of\ consumers} = \frac{runtime\ per\ enqueued\ line}{runtime\ per\ tokenized\ line} \tag{5}$$

So only when the ratio between number of producers and consumers is equal to the ratio of run-time between equeueing a line and tokenizing a line, we can get the best performance.

- the run-time and ratio of equeueing a line and tokenizing a line are highly depend on the length of each line.

Appendix A

Case 1 (files with numbers inside), $< number\ of\ threads >= 4, < number\ of\ producer\ threads >= 2, < number\ of\ consumer\ threads >= 2$

```
yuxuan@yuxuan-XPS-13-9380:~/Dropbox/     k     /HPC/Assignments/Ex4_5_PA/5.6$
    ./run 4 2 2
Enter filenames
input1
input2
input3
input4

thread 0,   Enqueue 10 11 12 13 14
thread 0,   Enqueue 15 16 17 18 19
thread 2,   token 0 = 10
thread 2,   token 1 = 11
thread 3,   token 0 = 15
thread 1,   Enqueue 20 21 22 23 24
thread 0,   Enqueue 30 31 32 33 34
thread 1,   Enqueue 25 26 27 28 29
thread 0,   Enqueue 35 36 37 38 39
thread 3,   token 1 = 16
thread 1,   Enqueue 40 41 42 43 44
thread 2,   token 2 = 12
thread 3,   token 2 = 17
thread 1,   Enqueue 45 46 47 48 49
thread 3,   token 3 = 18
thread 3,   token 4 = 19
thread 2,   token 3 = 13
thread 2,   token 4 = 14
thread 2,   token 0 = 30
thread 2,   token 1 = 31
thread 2,   token 2 = 32
thread 2,   token 3 = 33
thread 2,   token 4 = 34
thread 2,   token 0 = 25
thread 2,   token 1 = 26
thread 2,   token 2 = 27
thread 2,   token 3 = 28
thread 2,   token 4 = 29
thread 2,   token 0 = 35
thread 2,   token 1 = 36
thread 2,   token 2 = 37
thread 2,   token 3 = 38
thread 2,   token 4 = 39
thread 2,   token 0 = 40
thread 2,   token 1 = 41
thread 2,   token 2 = 42
thread 2,   token 3 = 43
```

```
thread 2,   token 4 = 44
thread 2,   token 0 = 45
thread 2,   token 1 = 46
thread 2,   token 2 = 47
thread 2,   token 3 = 48
thread 2,   token 4 = 49
thread 3,   token 0 = 20
thread 3,   token 1 = 21
thread 3,   token 2 = 22
thread 3,   token 3 = 23
thread 3,   token 4 = 24
queue =
enqueued = 8, dequeued = 8
```

Case 1 (files with numbers inside), $< number\ of\ threads >= 4, < number\ of\ producer\ threads >= 2, < number\ of\ consumer\ threads >= 2$

```
yuxuan@yuxuan-XPS-13-9380:~/Dropbox/     k     /HPC/Assignments/Ex4_5_PA/5.6$
    ./run 4 2 2
Enter filenames
input1
input2
input3
input4

thread 0,   Enqueue 10 11 12 13 14
thread 1,   Enqueue 20 21 22 23 24
thread 0,   Enqueue 15 16 17 18 19
thread 2,   token 0 = 10
thread 2,   token 1 = 11
thread 2,   token 2 = 12
thread 3,   token 0 = 20
thread 3,   token 1 = 21
thread 3,   token 2 = 22
thread 3,   token 3 = 23
thread 3,   token 4 = 24
thread 1,   Enqueue 25 26 27 28 29
thread 3,   token 0 = 15
thread 3,   token 1 = 16
thread 3,   token 2 = 17
thread 3,   token 3 = 18
thread 3,   token 4 = 19
thread 2,   token 3 = 13
thread 2,   token 4 = 14
thread 0,   Enqueue 30 31 32 33 34
thread 1,   Enqueue 40 41 42 43 44
thread 0,   Enqueue 35 36 37 38 39
thread 3,   token 0 = 25
thread 3,   token 1 = 26
```

```
thread 3,   token 2 = 27
thread 3,   token 3 = 28
thread 3,   token 4 = 29
thread 2,   token 0 = 30
thread 2,   token 1 = 31
thread 1,   Enqueue 45 46 47 48 49
thread 3,   token 0 = 40
thread 3,   token 1 = 41
thread 3,   token 2 = 42
thread 3,   token 3 = 43
thread 3,   token 4 = 44
thread 3,   token 0 = 35
thread 3,   token 1 = 36
thread 3,   token 2 = 37
thread 3,   token 3 = 38
thread 3,   token 4 = 39
thread 3,   token 0 = 45
thread 3,   token 1 = 46
thread 3,   token 2 = 47
thread 3,   token 3 = 48
thread 3,   token 4 = 49
thread 2,   token 2 = 32
thread 2,   token 3 = 33
thread 2,   token 4 = 34
queue =
enqueued = 8, dequeued = 8
```

Case 1 (files with numbers inside), $< number\ of\ threads >= 4, < number\ of\ producer\ threads >= 3, < number\ of\ consumer\ threads >= 1$

```
yuxuan@yuxuan-XPS-13-9380:~/Dropbox/     k     /HPC/Assignments/Ex4_5_PA/5.6$
    ./run 4 3 1
Enter filenames
input1
input2
input3
input4

thread 0,   Enqueue 10 11 12 13 14
thread 1,   Enqueue 20 21 22 23 24
thread 0,   Enqueue 15 16 17 18 19
thread 2,   Enqueue 30 31 32 33 34
thread 1,   Enqueue 25 26 27 28 29
thread 0,   Enqueue 40 41 42 43 44
thread 0,   Enqueue 45 46 47 48 49
thread 2,   Enqueue 35 36 37 38 39
thread 3,   token 0 = 10
thread 3,   token 1 = 11
thread 3,   token 2 = 12
```

```
thread 3,   token 3 = 13
thread 3,   token 4 = 14
thread 3,   token 0 = 20
thread 3,   token 1 = 21
thread 3,   token 2 = 22
thread 3,   token 3 = 23
thread 3,   token 4 = 24
thread 3,   token 0 = 15
thread 3,   token 1 = 16
thread 3,   token 2 = 17
thread 3,   token 3 = 18
thread 3,   token 4 = 19
thread 3,   token 0 = 30
thread 3,   token 1 = 31
thread 3,   token 2 = 32
thread 3,   token 3 = 33
thread 3,   token 4 = 34
thread 3,   token 0 = 25
thread 3,   token 1 = 26
thread 3,   token 2 = 27
thread 3,   token 3 = 28
thread 3,   token 4 = 29
thread 3,   token 0 = 40
thread 3,   token 1 = 41
thread 3,   token 2 = 42
thread 3,   token 3 = 43
thread 3,   token 4 = 44
thread 3,   token 0 = 45
thread 3,   token 1 = 46
thread 3,   token 2 = 47
thread 3,   token 3 = 48
thread 3,   token 4 = 49
thread 3,   token 0 = 35
thread 3,   token 1 = 36
thread 3,   token 2 = 37
thread 3,   token 3 = 38
thread 3,   token 4 = 39
queue =
enqueued = 8, dequeued = 8
```

Case 1 (files with numbers inside), $<$ *number of threads* $>= 4, <$ *number of producer threads* $>= 1, <$ *number of consumer threads* $>= 3$

```
yuxuan@yuxuan-XPS-13-9380:~/Dropbox/     k     /HPC/Assignments/Ex4_5_PA/5.6$
    ./run 4 1 3
Enter filenames
input1
input2
input3
```

input4

```
thread 0,   Enqueue 10 11 12 13 14
thread 0,   Enqueue 15 16 17 18 19
thread 2,   token 0 = 10
thread 2,   token 1 = 11
thread 3,   token 0 = 15
thread 3,   token 1 = 16
thread 3,   token 2 = 17
thread 3,   token 3 = 18
thread 3,   token 4 = 19
thread 0,   Enqueue 20 21 22 23 24
thread 0,   Enqueue 25 26 27 28 29
thread 2,   token 2 = 12
thread 0,   Enqueue 30 31 32 33 34
thread 3,   token 0 = 25
thread 3,   token 1 = 26
thread 3,   token 2 = 27
thread 3,   token 3 = 28
thread 3,   token 4 = 29
thread 1,   token 0 = 20
thread 0,   Enqueue 35 36 37 38 39
thread 3,   token 0 = 30
thread 3,   token 1 = 31
thread 2,   token 3 = 13
thread 1,   token 1 = 21
thread 3,   token 2 = 32
thread 0,   Enqueue 40 41 42 43 44
thread 0,   Enqueue 45 46 47 48 49
thread 3,   token 3 = 33
thread 3,   token 4 = 34
thread 3,   token 0 = 35
thread 3,   token 1 = 36
thread 3,   token 2 = 37
thread 3,   token 3 = 38
thread 3,   token 4 = 39
thread 3,   token 0 = 40
thread 3,   token 1 = 41
thread 3,   token 2 = 42
thread 2,   token 4 = 14
thread 1,   token 2 = 22
thread 3,   token 3 = 43
thread 2,   token 0 = 45
thread 1,   token 3 = 23
thread 3,   token 4 = 44
thread 2,   token 1 = 46
thread 1,   token 4 = 24
thread 2,   token 2 = 47
thread 2,   token 3 = 48
thread 2,   token 4 = 49
```

```
queue =
enqueued = 8, dequeued = 8
```

Case 2 (files with texts inside), $< number\ of\ threads >= 4, < number\ of\ producer\ threads >= 2, < number\ of\ consumer\ threads >= 2$

```
yuxuan@yuxuan-XPS-13-9380:~/Dropbox/     k     /HPC/Assignments/Ex4_5_PA/5.6$
    ./run 4 2 2
Enter filenames
text1
text2
text3
text4

thread 0,   Enqueue requires approximately twice as much time as the
    time to execute f (i).
thread 1,   Enqueue However, the variables that are declared in the main
    function ( a , b , n,
thread 0,   Enqueue When we ran the program with n = 10,000 and one
    thread, the run-time was 3.67
thread 1,   Enqueue global result , and thread count ) are all
    accessible to all the threads in the team
thread 2,   token 0 = requires
thread 2,   token 1 = approximately
thread 2,   token 2 = twice
thread 2,   token 3 = as
thread 2,   token 4 = much
thread 2,   token 5 = time
thread 2,   token 6 = as
thread 2,   token 7 = the
thread 2,   token 8 = time
thread 2,   token 9 = to
thread 1,   Enqueue started by the parallel directive. Hence, the
    default scope for variables declared
thread 0,   Enqueue seconds. When we ran the program with two threads
    and the default assignment
thread 1,   Enqueue before a parallel block is shared. In fact, weve
    made implicit use of this: each
thread 0,   Enqueue iterations 05000 on thread 0 and iterations
    500110,000 on thread 1the run-time
thread 1,   Enqueue thread in the team gets the values of a , b , and n
    from the call to Trap . Since this call
thread 0,   Enqueue was 2.76 seconds. This is a speedup of only 1.33.
    However, when we ran the program
thread 0,   Enqueue with two threads and a cyclic assignment, the
    run-time was decreased to 1.84 seconds.
thread 1,   Enqueue takes place in the parallel block, its essential
    that each thread has access to a , b ,
thread 0,   Enqueue This is a speedup of 1.99 over the one-thread run
```

```
    and a speedup of 1.5 over the
thread 0,   Enqueue two-thread block partition!
thread 1,   Enqueue and n when their values are copied into the
    corresponding formal arguments.
thread 0,   Enqueue We can see that a good assignment of iterations to
    threads can have a very sig-
thread 1,   Enqueue Furthermore, in the Trap function, although global
    result p is a private vari-
thread 0,   Enqueue nificant effect on performance. In OpenMP, assigning
    iterations to threads is called
thread 2,   token 10 = execute
thread 2,   token 11 = f
thread 2,   token 12 = (i).
thread 2,   token 13 = !
thread 3,   token 0 = However,
thread 3,   token 1 = the
thread 3,   token 2 = variables
thread 3,   token 3 = that
thread 3,   token 4 = are
thread 3,   token 5 = declared
thread 3,   token 6 = in
thread 3,   token 7 = the
thread 3,   token 8 = main
thread 3,   token 9 = function
thread 3,   token 10 = (
thread 3,   token 11 = a
thread 3,   token 12 = ,
thread 3,   token 13 = b
thread 3,   token 14 = ,
thread 3,   token 15 = n,
thread 1,   Enqueue able, it refers to the variable global result which
    was declared in main before the
thread 0,   Enqueue scheduling, and the schedule clause can be used to
    assign iterations in either athread 2, token 0 = When
thread 2,   token 1 = we
thread 2,   token 2 = ran
thread 2,   token 3 = the
thread 2,   token 4 = program
thread 2,   token 5 = with
thread 2,   token 6 = n
thread 2,   token 7 = =
thread 2,   token 8 = 10,000
thread 2,   token 9 = and
thread 2,   token 10 = one
thread 2,   token 11 = thread,
thread 2,   token 12 = the
thread 2,   token 13 = run-time
thread 2,   token 14 = was
thread 2,   token 15 = 3.67
thread 1,   Enqueue parallel directive, and the value of global result
```

```
    is used to store the result thats
thread 0,   Enqueue If  youve  done a little C programming, theres
    nothing really new up to this point.
thread 0,   Enqueue When we start the program from the command line, the
    operating system starts a
thread 2,   token 0 = started
thread 2,   token 1 = by
thread 2,   token 2 = the
thread 2,   token 3 = parallel
thread 2,   token 4 = directive.
thread 2,   token 5 = Hence,
thread 2,   token 6 = the
thread 2,   token 7 = default
thread 1,   Enqueue printed out after the parallel block. So in the code
thread 0,   Enqueue single-threaded process and the process executes the
    code in the main function. How-
thread 1,   Enqueue  global  result p += my result;
thread 0,   Enqueue ever, things get interesting in Line 11. This is our
    first OpenMP directive, and were
thread 1,   Enqueue  its  essential that   global result p have shared
    scope. If it were private to each
thread 0,   Enqueue using it to specify that the program should start
    some threads. Each thread thats
thread 1,   Enqueue thread, there would be no need for the critical
    directive. Furthermore, if it were
thread 0,   Enqueue forked should execute the Hello function, and when
    the threads return from the call
thread 1,   Enqueue private, we would have a hard time determining the
    value of global result in main
thread 0,   Enqueue to Hello , they should be terminated, and the
    process should then terminate when it
thread 0,   Enqueue executes the return statement.
thread 0,   Enqueue  Thats  a lot of bang for the buck (or code). If you
    studied the Pthreads chapter,
thread 1,   Enqueue after completion of the parallel block.
thread 0,   Enqueue  youll  recall that we had to write a lot of code to
    fork and join multiple threads: we
thread 1,   Enqueue To summarize, then, variables that have been
    declared before a parallel direc-
thread 0,   Enqueue needed to allocate storage for a special struct for
    each thread, we used a for loop to
thread 1,   Enqueue tive have shared scope among the threads in the
    team, while variables declared in the
thread 0,   Enqueue start each thread, and we used another for loop to
    terminate the threads. Thus, its
thread 1,   Enqueue block (e.g., local variables in functions) have
    private scope. Furthermore, the value
thread 0,   Enqueue immediately evident that OpenMP is higher-level than
    Pthreads.
thread 1,   Enqueue of a shared variable at the beginning of the
```

```
    parallel block is the same as the
thread 0,   Enqueue  Weve  already seen that pragma s in C and C ++
    start withthread 1, Enqueue value before the block, and, after
    completion of the parallel block, the value of
thread 1,   Enqueue the variable is the value at the end of the block.
thread 1,   Enqueue  Well  shortly see that the default scope of a
    variable can change with other
thread 2,   token 8 = scope
thread 2,   token 9 = for
thread 2,   token 10 = variables
thread 2,   token 11 = declared
thread 1,   Enqueue directives, and that OpenMP provides clauses to
    modify the default scope.thread 2, token 0 = seconds.
thread 2,   token 1 = When
thread 1,   Enqueue Like Pthreads, OpenMP is an API for shared-memory
    parallel programming. The
thread 1,   Enqueue   MP   in OpenMP stands for multiprocessing,   a
    term that is synonymous with
thread 1,   Enqueue shared-memory parallel computing. Thus, OpenMP is
    designed for systems in which
thread 1,   Enqueue each thread or process can potentially have access
    to all available memory, and, when
thread 1,   Enqueue  were  programming with OpenMP, we view our system
    as a collection of cores or
thread 3,   token 0 = global
thread 3,   token 1 = result
thread 3,   token 2 = ,
thread 3,   token 3 = and
thread 3,   token 4 = thread
thread 3,   token 5 = count
thread 3,   token 6 = )
thread 3,   token 7 = are
thread 3,   token 8 = all
thread 3,   token 9 = accessible
thread 3,   token 10 = to
thread 3,   token 11 = all
thread 3,   token 12 = the
thread 3,   token 13 = threads
thread 3,   token 14 = in
thread 3,   token 15 = the
thread 3,   token 16 = team
thread 1,   Enqueue CPUs, all of which have access to main memory, as in
    Figure 5.1.
thread 3,   token 0 = before
thread 3,   token 1 = a
thread 3,   token 2 = parallel
thread 3,   token 3 = block
thread 3,   token 4 = is
thread 3,   token 5 = shared.
thread 3,   token 6 = In
```

```
thread 3,   token 7 = fact,
thread 3,   token 8 =  weve
thread 3,   token 9 = made
thread 3,   token 10 = implicit
thread 3,   token 11 = use
thread 3,   token 12 = of
thread 3,   token 13 = this:
thread 3,   token 14 = each
thread 1,   Enqueue Although OpenMP and Pthreads are both APIs for
    shared-memory programming,
thread 1,   Enqueue they have many fundamental differences. Pthreads
    requires that the programmer
thread 1,   Enqueue explicitly specify the behavior of each thread.
    OpenMP, on the other hand, some-
thread 1,   Enqueue times allows the programmer to simply state that a
    block of code should be executed
thread 1,   Enqueue in parallel, and the precise determination of the
    tasks and which thread should execute
thread 3,   token 0 = iterations
thread 3,   token 1 = 0 5000
thread 3,   token 2 = on
thread 3,   token 3 = thread
thread 3,   token 4 = 0
thread 3,   token 5 = and
thread 3,   token 6 = iterations
thread 3,   token 7 = 5001 10 ,000
thread 3,   token 8 = on
thread 3,   token 9 = thread
thread 3,   token 10 = 1 the
thread 3,   token 11 = run-time
thread 3,   token 12 = !
thread 1,   Enqueue them is left to the compiler and the run-time
    system. This suggests a further differ-
thread 1,   Enqueue ence between OpenMP and Pthreads, that is, that
    Pthreads (like MPI) is a library of
thread 1,   Enqueue functions that can be linked to a C program, so any
    Pthreads program can be used
thread 2,   token 2 = we
thread 2,   token 3 = ran
thread 3,   token 0 = thread
thread 3,   token 1 = in
thread 3,   token 2 = the
thread 3,   token 3 = team
thread 3,   token 4 = gets
thread 3,   token 5 = the
thread 3,   token 6 = values
thread 3,   token 7 = of
thread 3,   token 8 = a
thread 3,   token 9 = ,
thread 3,   token 10 = b
```

```
thread 3,   token 11 = ,
thread 3,   token 12 = and
thread 3,   token 13 = n
thread 3,   token 14 = from
thread 3,   token 15 = the
thread 3,   token 16 = call
thread 3,   token 17 = to
thread 3,   token 18 = Trap
thread 3,   token 19 = .
thread 3,   token 20 = Since
thread 3,   token 21 = this
thread 3,   token 22 = call
thread 1,   Enqueue with any C compiler, provided the system has a
    Pthreads library. OpenMP, on the
thread 1,   Enqueue other hand, requires compiler support for some
    operations, and hence its entirely
thread 1,   Enqueue possible that you may run across a C compiler that
    cant compile OpenMP programs
thread 1,   Enqueue into parallel programs.
thread 1,   Enqueue These differences also suggest why there are two
    standard APIs for shared-
thread 1,   Enqueue memory programming: Pthreads is lower level and
    provides us with the power to
thread 1,   Enqueue program virtually any conceivable thread behavior.
    This power, however, comes with
thread 3,   token 0 = was
thread 3,   token 1 = 2.76
thread 3,   token 2 = seconds.
thread 3,   token 3 = This
thread 3,   token 4 = is
thread 3,   token 5 = a
thread 3,   token 6 = speedup
thread 3,   token 7 = of
thread 3,   token 8 = only
thread 3,   token 9 = 1.33.
thread 3,   token 10 = However,
thread 3,   token 11 = when
thread 3,   token 12 = we
thread 3,   token 13 = ran
thread 3,   token 14 = the
thread 3,   token 15 = program
thread 2,   token 4 = the
thread 2,   token 5 = program
thread 2,   token 6 = with
thread 2,   token 7 = two
thread 2,   token 8 = threads
thread 2,   token 9 = and
thread 2,   token 10 = the
thread 2,   token 11 = default
thread 2,   token 12 =  assignment
```

```
thread 1,   Enqueue some associated  costits  up to us to specify every
    detail of the behavior of each
thread 2,   token 0 = with
thread 2,   token 1 = two
thread 2,   token 2 = threads
thread 2,   token 3 = and
thread 3,   token 0 = takes
thread 3,   token 1 = place
thread 3,   token 2 = in
thread 3,   token 3 = the
thread 3,   token 4 = parallel
thread 3,   token 5 = block,
thread 3,   token 6 =  its
thread 3,   token 7 = essential
thread 3,   token 8 = that
thread 3,   token 9 = each
thread 3,   token 10 = thread
thread 3,   token 11 = has
thread 3,   token 12 = access
thread 3,   token 13 = to
thread 3,   token 14 = a
thread 3,   token 15 = ,
thread 3,   token 16 = b
thread 3,   token 17 = ,
thread 1,   Enqueue thread. OpenMP, on the other hand, allows the
    compiler and run-time system to deter-
thread 3,   token 0 = This
thread 3,   token 1 = is
thread 3,   token 2 = a
thread 3,   token 3 = speedup
thread 3,   token 4 = of
thread 3,   token 5 = 1.99
thread 3,   token 6 = over
thread 3,   token 7 = the
thread 3,   token 8 = one-thread
thread 3,   token 9 = run
thread 3,   token 10 = and
thread 3,   token 11 = a
thread 3,   token 12 = speedup
thread 3,   token 13 = of
thread 3,   token 14 = 1.5
thread 3,   token 15 = over
thread 3,   token 16 = the
thread 1,   Enqueue mine some of the details of thread behavior, so it
    can be simpler to code some parallel
thread 1,   Enqueue behaviors using OpenMP. The cost is that some
    low-level thread interactions can be
thread 1,   Enqueue more difficult to program.
thread 1,   Enqueue OpenMP was developed by a group of programmers and
    computer scien-
```

```
thread 1,   Enqueue tists who believed that writing large-scale
    high-performance programs using APIs
thread 1,   Enqueue such as Pthreads was too difficult, and they defined
    the OpenMP specification
thread 1,   Enqueue so that shared-memory programs could be developed at
    a higher level. In fact,
thread 3,   token 0 = two-thread
thread 3,   token 1 = block
thread 3,   token 2 = partition!
thread 1,   Enqueue OpenMP was explicitly designed to allow programmers
    to incrementally parallelizethread 3, token 0 = and
thread 3,   token 1 = n
thread 3,   token 2 = when
thread 3,   token 3 = their
thread 3,   token 4 = values
thread 3,   token 5 = are
thread 3,   token 6 = copied
thread 3,   token 7 = into
thread 3,   token 8 = the
thread 3,   token 9 = corresponding
thread 3,   token 10 = formal
thread 3,   token 11 = arguments.
thread 3,   token 0 = We
thread 3,   token 1 = can
thread 3,   token 2 = see
thread 3,   token 3 = that
thread 3,   token 4 = a
thread 3,   token 5 = good
thread 3,   token 6 = assignment
thread 3,   token 7 = of
thread 3,   token 8 = iterations
thread 3,   token 9 = to
thread 3,   token 10 = threads
thread 3,   token 11 = can
thread 3,   token 12 = have
thread 3,   token 13 = a
thread 2,   token 4 = a
thread 2,   token 5 = cyclic
thread 2,   token 6 = assignment,
thread 2,   token 7 = the
thread 2,   token 8 = run-time
thread 2,   token 9 = was
thread 2,   token 10 = decreased
thread 2,   token 11 = to
thread 2,   token 12 = 1.84
thread 2,   token 13 = seconds.
thread 2,   token 0 = Furthermore,
thread 2,   token 1 = in
thread 2,   token 2 = the
thread 2,   token 3 = Trap
```

```
thread 2,   token 4 = function,
thread 2,   token 5 = although
thread 2,   token 6 = global
thread 2,   token 7 = result
thread 2,   token 8 = p
thread 2,   token 9 = is
thread 2,   token 10 = a
thread 2,   token 11 = private
thread 2,   token 12 = vari-
thread 2,   token 0 = nificant
thread 2,   token 1 = effect
thread 2,   token 2 = on
thread 2,   token 3 = performance.
thread 2,   token 4 = In
thread 2,   token 5 = OpenMP,
thread 2,   token 6 = assigning
thread 2,   token 7 = iterations
thread 2,   token 8 = to
thread 2,   token 9 = threads
thread 2,   token 10 = is
thread 2,   token 11 = called
thread 2,   token 0 = able,
thread 2,   token 1 = it
thread 2,   token 2 = refers
thread 2,   token 3 = to
thread 2,   token 4 = the
thread 2,   token 5 = variable
thread 2,   token 6 = global
thread 2,   token 7 = result
thread 2,   token 8 = which
thread 2,   token 9 = was
thread 2,   token 10 = declared
thread 2,   token 11 = in
thread 2,   token 12 = main
thread 2,   token 13 = before
thread 2,   token 14 = the
thread 2,   token 0 = scheduling,
thread 2,   token 1 = and
thread 2,   token 2 = the
thread 2,   token 3 = schedule
thread 2,   token 4 = clause
thread 2,   token 5 = can
thread 2,   token 6 = be
thread 2,   token 7 = used
thread 2,   token 8 = to
thread 2,   token 9 = assign
thread 2,   token 10 = iterations
thread 2,   token 11 = in
thread 2,   token 12 = either
thread 2,   token 13 = a
```

```
thread 2,   token 0 = parallel
thread 2,   token 1 = directive,
thread 2,   token 2 = and
thread 2,   token 3 = the
thread 2,   token 4 = value
thread 2,   token 5 = of
thread 2,   token 6 = global
thread 2,   token 7 = result
thread 2,   token 8 = is
thread 2,   token 9 = used
thread 2,   token 10 = to
thread 2,   token 11 = store
thread 2,   token 12 = the
thread 2,   token 13 = result
thread 2,   token 14 =  thats
thread 2,   token 15 = %
thread 2,   token 0 = If
thread 2,   token 1 =  youve
thread 2,   token 2 = done
thread 2,   token 3 = a
thread 2,   token 4 = little
thread 2,   token 5 = C
thread 2,   token 6 = programming,
thread 2,   token 7 =  theres
thread 2,   token 8 = nothing
thread 2,   token 9 = really
thread 2,   token 10 = new
thread 2,   token 11 = up
thread 2,   token 12 = to
thread 2,   token 13 = this
thread 2,   token 14 = point.
thread 2,   token 15 = !
thread 2,   token 0 = When
thread 2,   token 1 = we
thread 2,   token 2 = start
thread 2,   token 3 = the
thread 2,   token 4 = program
thread 2,   token 5 = from
thread 2,   token 6 = the
thread 2,   token 7 = command
thread 2,   token 8 = line,
thread 2,   token 9 = the
thread 2,   token 10 = operating
thread 2,   token 11 = system
thread 2,   token 12 = starts
thread 2,   token 13 = a
thread 2,   token 0 = printed
thread 2,   token 1 = out
thread 2,   token 2 = after
thread 2,   token 3 = the
```

```
thread 2,   token 4 = parallel
thread 2,   token 5 = block.
thread 2,   token 6 = So
thread 2,   token 7 = in
thread 2,   token 8 = the
thread 2,   token 9 = code
thread 2,   token 0 = single-threaded
thread 2,   token 1 = process
thread 2,   token 2 = and
thread 2,   token 3 = the
thread 2,   token 4 = process
thread 2,   token 5 = executes
thread 2,   token 6 = the
thread 2,   token 7 = code
thread 2,   token 8 = in
thread 2,   token 9 = the
thread 2,   token 10 = main
thread 2,   token 11 = function.
thread 2,   token 12 = How-
thread 2,   token 0 =  global
thread 2,   token 1 = result
thread 2,   token 2 = p
thread 2,   token 3 = +=
thread 2,   token 4 = my
thread 2,   token 5 = result;
thread 2,   token 0 = ever,
thread 2,   token 1 = things
thread 2,   token 2 = get
thread 2,   token 3 = interesting
thread 2,   token 4 = in
thread 2,   token 5 = Line
thread 2,   token 6 = 11.
thread 2,   token 7 = This
thread 2,   token 8 = is
thread 2,   token 9 = our
thread 2,   token 10 = first
thread 2,   token 11 = OpenMP
thread 2,   token 12 = directive,
thread 2,   token 13 = and
thread 2,   token 14 =  were
thread 2,   token 0 =  its
thread 2,   token 1 = essential
thread 2,   token 2 = that
thread 2,   token 3 =
thread 2,   token 4 = global
thread 2,   token 5 = result
thread 2,   token 6 = p
thread 2,   token 7 = have
thread 2,   token 8 = shared
thread 2,   token 9 = scope.
```

```
thread 2,   token 10 = If
thread 2,   token 11 = it
thread 2,   token 12 = were
thread 2,   token 13 = private
thread 2,   token 14 = to
thread 2,   token 15 = each
thread 2,   token 16 = %
thread 2,   token 0 = using
thread 2,   token 1 = it
thread 2,   token 2 = to
thread 2,   token 3 = specify
thread 2,   token 4 = that
thread 2,   token 5 = the
thread 3,   token 14 = very
thread 3,   token 15 = sig-
thread 3,   token 0 = thread,
thread 3,   token 1 = there
thread 3,   token 2 = would
thread 3,   token 3 = be
thread 3,   token 4 = no
thread 3,   token 5 = need
thread 3,   token 6 = for
thread 3,   token 7 = the
thread 3,   token 8 = critical
thread 3,   token 9 = directive.
thread 3,   token 10 = Furthermore,
thread 3,   token 11 = if
thread 3,   token 12 = it
thread 3,   token 13 = were
thread 3,   token 0 = forked
thread 3,   token 1 = should
thread 3,   token 2 = execute
thread 3,   token 3 = the
thread 3,   token 4 = Hello
thread 3,   token 5 = function,
thread 3,   token 6 = and
thread 3,   token 7 = when
thread 3,   token 8 = the
thread 3,   token 9 = threads
thread 3,   token 10 = return
thread 3,   token 11 = from
thread 3,   token 12 = the
thread 3,   token 13 = call
thread 3,   token 0 = private,
thread 3,   token 1 = we
thread 3,   token 2 = would
thread 3,   token 3 = have
thread 3,   token 4 = a
thread 3,   token 5 = hard
thread 3,   token 6 = time
```

```
thread 3,   token 7 = determining
thread 3,   token 8 = the
thread 3,   token 9 = value
thread 3,   token 10 = of
thread 3,   token 11 = global
thread 3,   token 12 = result
thread 3,   token 13 = in
thread 3,   token 14 = main
thread 3,   token 0 = to
thread 3,   token 1 = Hello
thread 3,   token 2 = ,
thread 3,   token 3 = they
thread 3,   token 4 = should
thread 3,   token 5 = be
thread 3,   token 6 = terminated,
thread 3,   token 7 = and
thread 3,   token 8 = the
thread 3,   token 9 = process
thread 3,   token 10 = should
thread 3,   token 11 = then
thread 3,   token 12 = terminate
thread 3,   token 13 = when
thread 3,   token 14 = it
thread 3,   token 0 = executes
thread 3,   token 1 = the
thread 3,   token 2 = return
thread 3,   token 3 = statement.
thread 3,   token 0 =  Thats
thread 3,   token 1 = a
thread 3,   token 2 = lot
thread 3,   token 3 = of
thread 3,   token 4 = bang
thread 3,   token 5 = for
thread 3,   token 6 = the
thread 3,   token 7 = buck
thread 3,   token 8 = (or
thread 3,   token 9 = code).
thread 3,   token 10 = If
thread 3,   token 11 = you
thread 3,   token 12 = studied
thread 3,   token 13 = the
thread 3,   token 14 = Pthreads
thread 3,   token 15 = chapter,
thread 3,   token 0 = after
thread 3,   token 1 = completion
thread 3,   token 2 = of
thread 3,   token 3 = the
thread 3,   token 4 = parallel
thread 3,   token 5 = block.
thread 3,   token 6 = %
```

```
thread 3,   token 0 =  youll
thread 3,   token 1 = recall
thread 3,   token 2 = that
thread 3,   token 3 = we
thread 3,   token 4 = had
thread 3,   token 5 = to
thread 3,   token 6 = write
thread 3,   token 7 = a
thread 3,   token 8 = lot
thread 3,   token 9 = of
thread 3,   token 10 = code
thread 3,   token 11 = to
thread 3,   token 12 = fork
thread 3,   token 13 = and
thread 3,   token 14 = join
thread 3,   token 15 = multiple
thread 3,   token 16 = threads:
thread 3,   token 17 = we
thread 3,   token 0 = To
thread 3,   token 1 = summarize,
thread 3,   token 2 = then,
thread 3,   token 3 = variables
thread 3,   token 4 = that
thread 3,   token 5 = have
thread 3,   token 6 = been
thread 3,   token 7 = declared
thread 3,   token 8 = before
thread 3,   token 9 = a
thread 3,   token 10 = parallel
thread 3,   token 11 = direc-
thread 3,   token 0 = needed
thread 3,   token 1 = to
thread 3,   token 2 = allocate
thread 3,   token 3 = storage
thread 3,   token 4 = for
thread 3,   token 5 = a
thread 3,   token 6 = special
thread 3,   token 7 = struct
thread 3,   token 8 = for
thread 3,   token 9 = each
thread 3,   token 10 = thread,
thread 3,   token 11 = we
thread 3,   token 12 = used
thread 3,   token 13 = a
thread 3,   token 14 = for
thread 3,   token 15 = loop
thread 3,   token 16 = to
thread 3,   token 0 = tive
thread 3,   token 1 = have
thread 3,   token 2 = shared
```

```
thread 3,   token 3 = scope
thread 3,   token 4 = among
thread 3,   token 5 = the
thread 3,   token 6 = threads
thread 3,   token 7 = in
thread 3,   token 8 = the
thread 3,   token 9 = team,
thread 3,   token 10 = while
thread 3,   token 11 = variables
thread 3,   token 12 = declared
thread 3,   token 13 = in
thread 3,   token 14 = the
thread 3,   token 0 = start
thread 3,   token 1 = each
thread 3,   token 2 = thread,
thread 3,   token 3 = and
thread 3,   token 4 = we
thread 3,   token 5 = used
thread 3,   token 6 = another
thread 3,   token 7 = for
thread 3,   token 8 = loop
thread 3,   token 9 = to
thread 3,   token 10 = terminate
thread 3,   token 11 = the
thread 3,   token 12 = threads.
thread 3,   token 13 = Thus,
thread 3,   token 14 =  its
thread 3,   token 0 = block
thread 3,   token 1 = (e.g.,
thread 3,   token 2 = local
thread 3,   token 3 = variables
thread 3,   token 4 = in
thread 3,   token 5 = functions)
thread 3,   token 6 = have
thread 3,   token 7 = private
thread 3,   token 8 = scope.
thread 3,   token 9 = Furthermore,
thread 3,   token 10 = the
thread 3,   token 11 = value
thread 3,   token 0 = immediately
thread 3,   token 1 = evident
thread 3,   token 2 = that
thread 3,   token 3 = OpenMP
thread 3,   token 4 = is
thread 3,   token 5 = higher-level
thread 3,   token 6 = than
thread 3,   token 7 = Pthreads.
thread 3,   token 0 = of
thread 3,   token 1 = a
thread 3,   token 2 = shared
```

```
thread 3,   token 3 = variable
thread 3,   token 4 = at
thread 3,   token 5 = the
thread 3,   token 6 = beginning
thread 3,   token 7 = of
thread 3,   token 8 = the
thread 3,   token 9 = parallel
thread 3,   token 10 = block
thread 3,   token 11 = is
thread 3,   token 12 = the
thread 3,   token 13 = same
thread 3,   token 14 = as
thread 3,   token 15 = the
thread 3,   token 0 =  Weve
thread 3,   token 1 = already
thread 3,   token 2 = seen
thread 3,   token 3 = that
thread 3,   token 4 = pragma
thread 3,   token 5 = s
thread 3,   token 6 = in
thread 3,   token 7 = C
thread 3,   token 8 = and
thread 3,   token 9 = C
thread 3,   token 10 = ++
thread 3,   token 11 = start
thread 3,   token 12 = with
thread 3,   token 0 = value
thread 3,   token 1 = before
thread 3,   token 2 = the
thread 3,   token 3 = block,
thread 3,   token 4 = and,
thread 3,   token 5 = after
thread 3,   token 6 = completion
thread 3,   token 7 = of
thread 3,   token 8 = the
thread 3,   token 9 = parallel
thread 3,   token 10 = block,
thread 3,   token 11 = the
thread 3,   token 12 = value
thread 3,   token 13 = of
thread 3,   token 0 = the
thread 3,   token 1 = variable
thread 3,   token 2 = is
thread 3,   token 3 = the
thread 3,   token 4 = value
thread 3,   token 5 = at
thread 3,   token 6 = the
thread 3,   token 7 = end
thread 3,   token 8 = of
thread 3,   token 9 = the
```

```
thread 3,   token 10 = block.
thread 3,   token 0 =  Well
thread 3,   token 1 = shortly
thread 3,   token 2 = see
thread 3,   token 3 = that
thread 3,   token 4 = the
thread 3,   token 5 = default
thread 3,   token 6 = scope
thread 3,   token 7 = of
thread 3,   token 8 = a
thread 3,   token 9 = variable
thread 3,   token 10 = can
thread 3,   token 11 = change
thread 3,   token 12 = with
thread 3,   token 13 = other
thread 3,   token 0 = directives,
thread 3,   token 1 = and
thread 3,   token 2 = that
thread 3,   token 3 = OpenMP
thread 3,   token 4 = provides
thread 3,   token 5 = clauses
thread 3,   token 6 = to
thread 3,   token 7 = modify
thread 3,   token 8 = the
thread 3,   token 9 = default
thread 3,   token 10 = scope.
thread 3,   token 0 = Like
thread 3,   token 1 = Pthreads,
thread 3,   token 2 = OpenMP
thread 3,   token 3 = is
thread 3,   token 4 = an
thread 3,   token 5 = API
thread 3,   token 6 = for
thread 3,   token 7 = shared-memory
thread 3,   token 8 = parallel
thread 3,   token 9 = programming.
thread 3,   token 10 = The
thread 3,   token 0 =   MP
thread 3,   token 1 = in
thread 3,   token 2 = OpenMP
thread 3,   token 3 = stands
thread 3,   token 4 = for
thread 3,   token 5 = multiprocessing,
thread 3,   token 6 = a
thread 3,   token 7 = term
thread 3,   token 8 = that
thread 3,   token 9 = is
thread 3,   token 10 = synonymous
thread 3,   token 11 = with
thread 3,   token 0 = shared-memory
```

```
thread 3,   token 1 = parallel
thread 3,   token 2 = computing.
thread 3,   token 3 = Thus,
thread 3,   token 4 = OpenMP
thread 3,   token 5 = is
thread 3,   token 6 = designed
thread 3,   token 7 = for
thread 3,   token 8 = systems
thread 3,   token 9 = in
thread 3,   token 10 = which
thread 3,   token 0 = each
thread 3,   token 1 = thread
thread 3,   token 2 = or
thread 3,   token 3 = process
thread 3,   token 4 = can
thread 3,   token 5 = potentially
thread 3,   token 6 = have
thread 3,   token 7 = access
thread 3,   token 8 = to
thread 3,   token 9 = all
thread 3,   token 10 = available
thread 3,   token 11 = memory,
thread 3,   token 12 = and,
thread 3,   token 13 = when
thread 3,   token 0 =  were
thread 3,   token 1 = programming
thread 3,   token 2 = with
thread 3,   token 3 = OpenMP,
thread 3,   token 4 = we
thread 3,   token 5 = view
thread 3,   token 6 = our
thread 3,   token 7 = system
thread 3,   token 8 = as
thread 3,   token 9 = a
thread 3,   token 10 = collection
thread 3,   token 11 = of
thread 3,   token 12 = cores
thread 3,   token 13 = or
thread 3,   token 0 = CPUs,
thread 3,   token 1 = all
thread 3,   token 2 = of
thread 3,   token 3 = which
thread 3,   token 4 = have
thread 3,   token 5 = access
thread 3,   token 6 = to
thread 3,   token 7 = main
thread 3,   token 8 = memory,
thread 3,   token 9 = as
thread 3,   token 10 = in
thread 3,   token 11 = Figure
```

```
thread 3,   token 12 = 5.1.
thread 3,   token 0 = Although
thread 3,   token 1 = OpenMP
thread 3,   token 2 = and
thread 3,   token 3 = Pthreads
thread 3,   token 4 = are
thread 3,   token 5 = both
thread 3,   token 6 = APIs
thread 3,   token 7 = for
thread 3,   token 8 = shared-memory
thread 3,   token 9 = programming,
thread 3,   token 0 = they
thread 3,   token 1 = have
thread 3,   token 2 = many
thread 3,   token 3 = fundamental
thread 3,   token 4 = differences.
thread 3,   token 5 = Pthreads
thread 3,   token 6 = requires
thread 3,   token 7 = that
thread 3,   token 8 = the
thread 3,   token 9 = programmer
thread 3,   token 0 = explicitly
thread 3,   token 1 = specify
thread 3,   token 2 = the
thread 3,   token 3 = behavior
thread 3,   token 4 = of
thread 3,   token 5 = each
thread 3,   token 6 = thread.
thread 3,   token 7 = OpenMP,
thread 3,   token 8 = on
thread 3,   token 9 = the
thread 3,   token 10 = other
thread 3,   token 11 = hand,
thread 3,   token 12 = some-
thread 3,   token 0 = times
thread 3,   token 1 = allows
thread 3,   token 2 = the
thread 3,   token 3 = programmer
thread 3,   token 4 = to
thread 3,   token 5 = simply
thread 3,   token 6 = state
thread 3,   token 7 = that
thread 3,   token 8 = a
thread 3,   token 9 = block
thread 3,   token 10 = of
thread 3,   token 11 = code
thread 3,   token 12 = should
thread 3,   token 13 = be
thread 3,   token 14 = executed
thread 3,   token 0 = in
```

```
thread 3,   token 1 = parallel,
thread 3,   token 2 = and
thread 3,   token 3 = the
thread 3,   token 4 = precise
thread 3,   token 5 = determination
thread 3,   token 6 = of
thread 3,   token 7 = the
thread 3,   token 8 = tasks
thread 3,   token 9 = and
thread 3,   token 10 = which
thread 3,   token 11 = thread
thread 3,   token 12 = should
thread 3,   token 13 = execute
thread 3,   token 14 = %
thread 3,   token 0 = them
thread 3,   token 1 = is
thread 3,   token 2 = left
thread 3,   token 3 = to
thread 3,   token 4 = the
thread 3,   token 5 = compiler
thread 3,   token 6 = and
thread 3,   token 7 = the
thread 3,   token 8 = run-time
thread 3,   token 9 = system.
thread 3,   token 10 = This
thread 3,   token 11 = suggests
thread 3,   token 12 = a
thread 3,   token 13 = further
thread 3,   token 14 = differ-
thread 3,   token 0 = ence
thread 3,   token 1 = between
thread 3,   token 2 = OpenMP
thread 3,   token 3 = and
thread 3,   token 4 = Pthreads,
thread 3,   token 5 = that
thread 3,   token 6 = is,
thread 3,   token 7 = that
thread 3,   token 8 = Pthreads
thread 3,   token 9 = (like
thread 3,   token 10 = MPI)
thread 3,   token 11 = is
thread 3,   token 12 = a
thread 3,   token 13 = library
thread 3,   token 14 = of
thread 3,   token 0 = functions
thread 3,   token 1 = that
thread 3,   token 2 = can
thread 3,   token 3 = be
thread 3,   token 4 = linked
thread 3,   token 5 = to
```

```
thread 3,   token 6 = a
thread 3,   token 7 = C
thread 3,   token 8 = program,
thread 3,   token 9 = so
thread 3,   token 10 = any
thread 3,   token 11 = Pthreads
thread 3,   token 12 = program
thread 3,   token 13 = can
thread 3,   token 14 = be
thread 3,   token 15 = used
thread 3,   token 0 = with
thread 3,   token 1 = any
thread 3,   token 2 = C
thread 3,   token 3 = compiler,
thread 3,   token 4 = provided
thread 3,   token 5 = the
thread 3,   token 6 = system
thread 3,   token 7 = has
thread 3,   token 8 = a
thread 3,   token 9 = Pthreads
thread 3,   token 10 = library.
thread 3,   token 11 = OpenMP,
thread 3,   token 12 = on
thread 3,   token 13 = the
thread 3,   token 0 = other
thread 3,   token 1 = hand,
thread 3,   token 2 = requires
thread 3,   token 3 = compiler
thread 3,   token 4 = support
thread 3,   token 5 = for
thread 3,   token 6 = some
thread 2,   token 6 = program
thread 2,   token 7 = should
thread 2,   token 8 = start
thread 2,   token 9 = some
thread 2,   token 10 = threads.
thread 2,   token 11 = Each
thread 2,   token 12 = thread
thread 2,   token 13 =  thats
thread 3,   token 7 = operations,
thread 3,   token 8 = and
thread 3,   token 9 = hence
thread 3,   token 10 =  its
thread 3,   token 11 = entirely
thread 3,   token 0 = into
thread 3,   token 1 = parallel
thread 3,   token 2 = programs.
thread 3,   token 3 = %
thread 3,   token 0 = These
thread 3,   token 1 = differences
```

```
thread 3,   token 2 = also
thread 3,   token 3 = suggest
thread 3,   token 4 = why
thread 3,   token 5 = there
thread 3,   token 6 = are
thread 3,   token 7 = two
thread 3,   token 8 = standard
thread 3,   token 9 = APIs
thread 3,   token 10 = for
thread 3,   token 11 = shared-
thread 3,   token 0 = memory
thread 3,   token 1 = programming:
thread 3,   token 2 = Pthreads
thread 3,   token 3 = is
thread 3,   token 4 = lower
thread 3,   token 5 = level
thread 3,   token 6 = and
thread 3,   token 7 = provides
thread 3,   token 8 = us
thread 3,   token 9 = with
thread 3,   token 10 = the
thread 3,   token 11 = power
thread 3,   token 12 = to
thread 3,   token 0 = program
thread 3,   token 1 = virtually
thread 3,   token 2 = any
thread 3,   token 3 = conceivable
thread 3,   token 4 = thread
thread 3,   token 5 = behavior.
thread 3,   token 6 = This
thread 3,   token 7 = power,
thread 3,   token 8 = however,
thread 3,   token 9 = comes
thread 3,   token 10 = with
thread 3,   token 0 = some
thread 3,   token 1 = associated
thread 3,   token 2 =   costits
thread 3,   token 3 = up
thread 3,   token 4 = to
thread 3,   token 5 = us
thread 3,   token 6 = to
thread 3,   token 7 = specify
thread 3,   token 8 = every
thread 3,   token 9 = detail
thread 3,   token 10 = of
thread 3,   token 11 = the
thread 3,   token 12 = behavior
thread 3,   token 13 = of
thread 3,   token 14 = each
thread 3,   token 0 = thread.
```

```
thread 3,   token 1 = OpenMP,
thread 3,   token 2 = on
thread 3,   token 3 = the
thread 3,   token 4 = other
thread 3,   token 5 = hand,
thread 3,   token 6 = allows
thread 3,   token 7 = the
thread 3,   token 8 = compiler
thread 3,   token 9 = and
thread 3,   token 10 = run-time
thread 3,   token 11 = system
thread 3,   token 12 = to
thread 3,   token 13 = deter-
thread 3,   token 0 = mine
thread 3,   token 1 = some
thread 3,   token 2 = of
thread 3,   token 3 = the
thread 3,   token 4 = details
thread 3,   token 5 = of
thread 3,   token 6 = thread
thread 3,   token 7 = behavior,
thread 3,   token 8 = so
thread 3,   token 9 = it
thread 3,   token 10 = can
thread 3,   token 11 = be
thread 3,   token 12 = simpler
thread 3,   token 13 = to
thread 3,   token 14 = code
thread 3,   token 15 = some
thread 3,   token 16 = parallel
thread 3,   token 17 = %
thread 3,   token 0 = behaviors
thread 3,   token 1 = using
thread 3,   token 2 = OpenMP.
thread 3,   token 3 = The
thread 3,   token 4 = cost
thread 3,   token 5 = is
thread 3,   token 6 = that
thread 3,   token 7 = some
thread 3,   token 8 = low-level
thread 3,   token 9 = thread
thread 3,   token 10 = interactions
thread 3,   token 11 = can
thread 3,   token 12 = be
thread 3,   token 0 = more
thread 3,   token 1 = difficult
thread 3,   token 2 = to
thread 3,   token 3 = program.
thread 3,   token 0 = OpenMP
thread 3,   token 1 = was
```

```
thread 3,   token 2 = developed
thread 3,   token 3 = by
thread 3,   token 4 = a
thread 3,   token 5 = group
thread 3,   token 6 = of
thread 3,   token 7 = programmers
thread 3,   token 8 = and
thread 3,   token 9 = computer
thread 3,   token 10 = scien-
thread 3,   token 0 = tists
thread 3,   token 1 = who
thread 3,   token 2 = believed
thread 3,   token 3 = that
thread 3,   token 4 = writing
thread 3,   token 5 = large-scale
thread 3,   token 6 = high-performance
thread 3,   token 7 = programs
thread 3,   token 8 = using
thread 3,   token 9 = APIs
thread 2,   token 0 = possible
thread 2,   token 1 = that
thread 2,   token 2 = you
thread 2,   token 3 = may
thread 2,   token 4 = run
thread 2,   token 5 = across
thread 3,   token 0 = such
thread 3,   token 1 = as
thread 3,   token 2 = Pthreads
thread 3,   token 3 = was
thread 3,   token 4 = too
thread 3,   token 5 = difficult,
thread 3,   token 6 = and
thread 3,   token 7 = they
thread 3,   token 8 = defined
thread 3,   token 9 = the
thread 3,   token 10 = OpenMP
thread 3,   token 11 = specification
thread 3,   token 0 = so
thread 3,   token 1 = that
thread 3,   token 2 = shared-memory
thread 3,   token 3 = programs
thread 3,   token 4 = could
thread 3,   token 5 = be
thread 3,   token 6 = developed
thread 3,   token 7 = at
thread 3,   token 8 = a
thread 3,   token 9 = higher
thread 3,   token 10 = level.
thread 3,   token 11 = In
thread 3,   token 12 = fact,
```

```
thread 3,   token 0 = OpenMP
thread 3,   token 1 = was
thread 3,   token 2 = explicitly
thread 3,   token 3 = designed
thread 3,   token 4 = to
thread 3,   token 5 = allow
thread 3,   token 6 = programmers
thread 3,   token 7 = to
thread 3,   token 8 = incrementally
thread 3,   token 9 = parallelize
thread 2,   token 6 = a
thread 2,   token 7 = C
thread 2,   token 8 = compiler
thread 2,   token 9 = that
thread 2,   token 10 =  cant
thread 2,   token 11 = compile
thread 2,   token 12 = OpenMP
thread 2,   token 13 = programs
queue =
enqueued = 80, dequeued = 80
```