TOP-C: Mature Parallel Computing

Gene Cooperman
gene@ccs.neu.edu

College of Computer Science
Northeastern University

Features of TOP-C model

Primary Goals:

- Ease-of-programming (small number of primitives, embedded in familiar language, ports to C, LISP, GAP, Java, etc.)
- Good latency tolerance (for commodity hardware)

Secondary Goals:

- Re-use of "legacy" sequential code
- Simple, task-oriented programmer's model
- Natural load balancing
- Robustness (in presence of very slow or dead processes)
- Dynamic attachment of new processors
- Checkpointing
- Meta-computing on Web
- Runs on top of message-passing or shared memory model (same application code, linked with different library)
- Small model (small library of code — easily maintained and modified)

FIRST-EVER COMPUTATIONS

Ly coset enum. (8,835,156 cosets)	J_4 perm. rep. (deg. 173,067,389) (over $GL(1333, 11)$)		Th condensation (from perm deg. 976,841,775 to matrix dim. 1,403) (over $GL(248, 2)$) J_4 condensation (from perm deg. 173,067,389 to matrix dim. 5,693) (over $GL(112, 2)$) Ly perm. rep. (deg. 9,606,125) (over $GL(111, 5)$)
TOP-C (shared mem.)	**TOP-C (dist. mem.)**	**Par/ GAP**	**LISP/MPI (GCL)**
POSIX threads	**MPI (Message Passing Interface)**		

Selected Papers About TOP-C and Applications

References

[1] G. Cooperman, "Parallel GAP: Mature Interactive Parallel", G. Cooperman, *Groups and Computation III*, DeGruyter Publishers, pp 123–138, to appear.

[2] G. Cooperman, "TOP-C: Task-Oriented Parallel C for Distributed and Shared Memory", *Workshop on Wide Area Networks and High Performance Computing*, Lecture Notes in Control and Information Sciences **249**, 1999, Springer Verlag, pp. 109–118.

[3] G. Cooperman, "Practical Task-Oriented Parallelism for Gaussian Elimination in Distributed Memory", *Linear Algebra and its Applications* **275-276**, 1998, pp. 107–120.

[4] G. Cooperman, "TOP-C: A Task-Oriented Parallel C Interface", 5^{th} *International Symposium on High Performance Distributed Computing* (HPDC-5), 1996, IEEE Press, pp. 141–150.

[5] G. Cooperman, "GAP/MPI: Facilitating Parallelism", *Proc. of Second DIMACS Workshop on Groups and Computation II* **28**, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, L. Finkelstein and W.M. Kantor (eds.), AMS, Providence, RI, 1997, pp. 69–84.

[6] G. Cooperman, "STAR/MPI: Binding a Parallel Library to Interactive Symbolic Algebra Systems", *Proc. of International Symposium on Symbolic and Algebraic Computation (ISSAC '95)*, ACM Press, 1995, pp. 126–132.

[7] L. Anchordoqui, G. Cooperman, V. Grinberg, T. McCauley, S. Reucroft, and J. Swain, *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics*, CHEP2000, Padova, Italy, 2000.

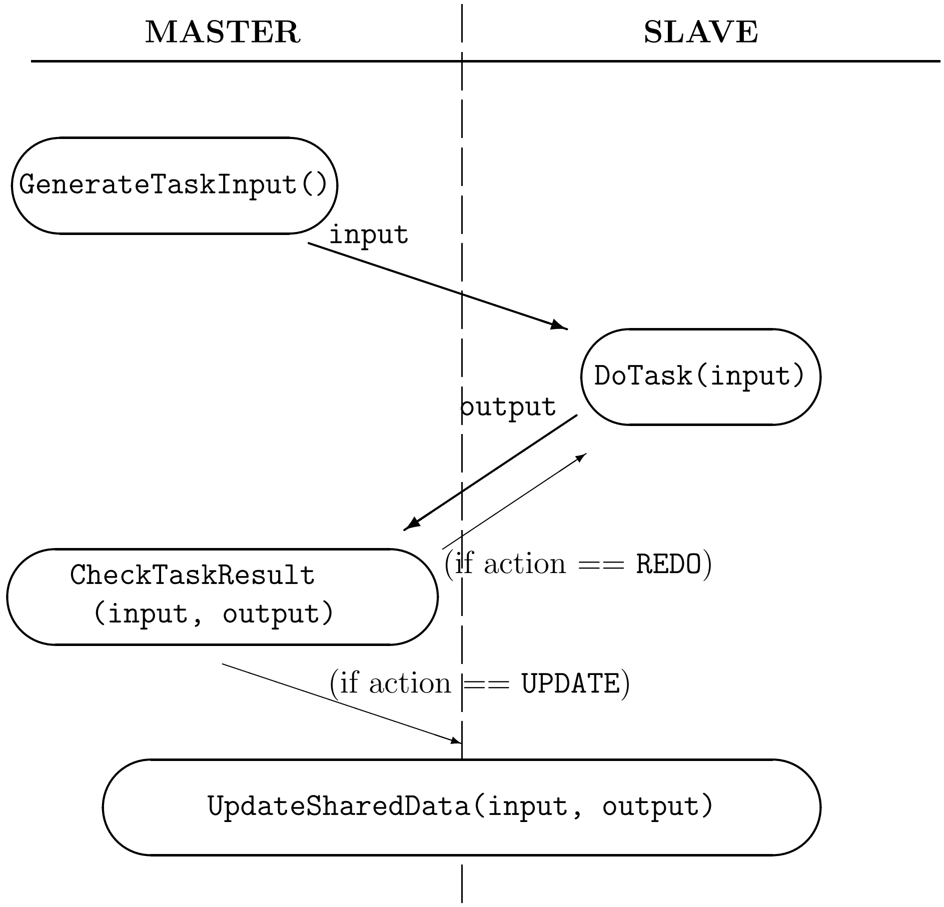
[8] G. Cooperman, L. Finkelstein, M. Tselman and B. York, "Constructing Permutation Representations for Matrix Groups", *J. Symbolic Computation* **24**, 1997, pp. 1–18.

[9] G. Cooperman and G. Havas, Practical parallel coset enumeration, *Proc. of Workshop on High Performance Computation and Gigabit Local Area Networks*, G. Cooperman, G. Michler and H. Vinck (eds.), Lecture notes in control and information sciences **226**, Springer Verlag, pp. 15–27.

[10] G. Cooperman, G. Hiss, K. Lux, and Jürgen Müller, The Brauer tree of the principal 19-block of the sporadic simple Thompson group, *J. of Experimental Mathematics* **6**(4), 1997, pp. 293–300.

[11] G. Cooperman and M. Tselman, New Sequential and Parallel Algorithms for Generating High Dimension Hecke Algebras using the Condensation Technique, *Proc. of International Symposium on Symbolic and Algebraic Computation (ISSAC '96)*, ACM Press, 155–160.

Life Cycle of A Task

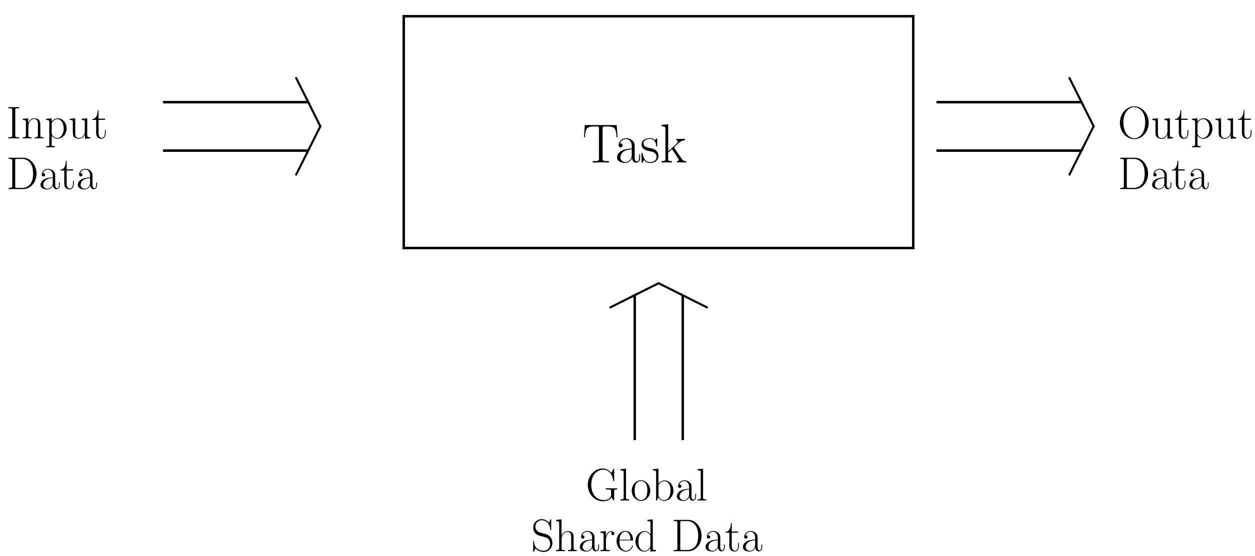


TOP-C Programmer's Model

```
MasterSlave(GenerateTaskInput, DoTask, CheckTaskResult, UpdateSharedData):

while ( NOTASK <> (input := GenerateTaskInput()) ) do [on master]
  redo:
  output := DoTask( input );                       [on slave]
  action := CheckTaskResult( input, output );      [on master]
  switch ( action )
    case NO_ACTION: /* do nothing */;
    case UPDATE:  UpdateSharedData(input, output);
                                  [on master and slave]
    case REDO:    goto redo;
    case CONTINUATION(param):  Like REDO, but pass "param"
```

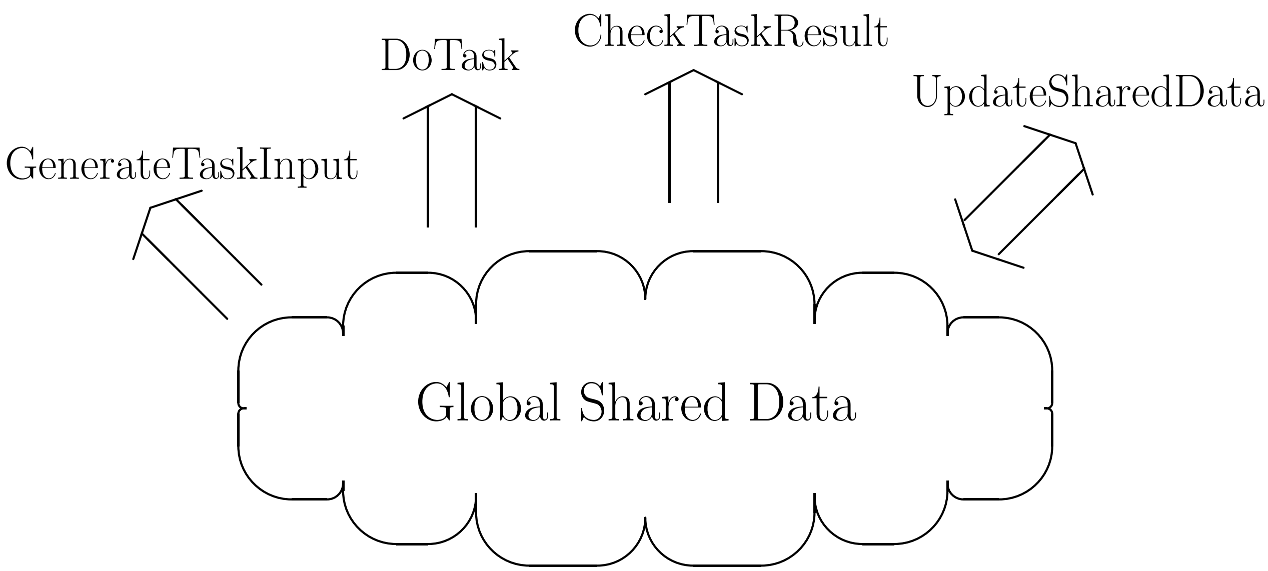
Concept 1: The Task

- Virtual star topology: single master process, many slave processes
- Task executed on a single slave process
- *Task input* generated by master as arbitrary buffer of bits and sent to slave process
- *Task output* produced by slave process as arbitrary buffer of bits and sent to slave process
- Task is a single routine executed by `DoTask()` and *never* interrupted or aborted.
- For a given *input* and a given state of *shared data*, task produces a unique *output*, independently of which slave process was invoked



Concept 2: The Shared Data

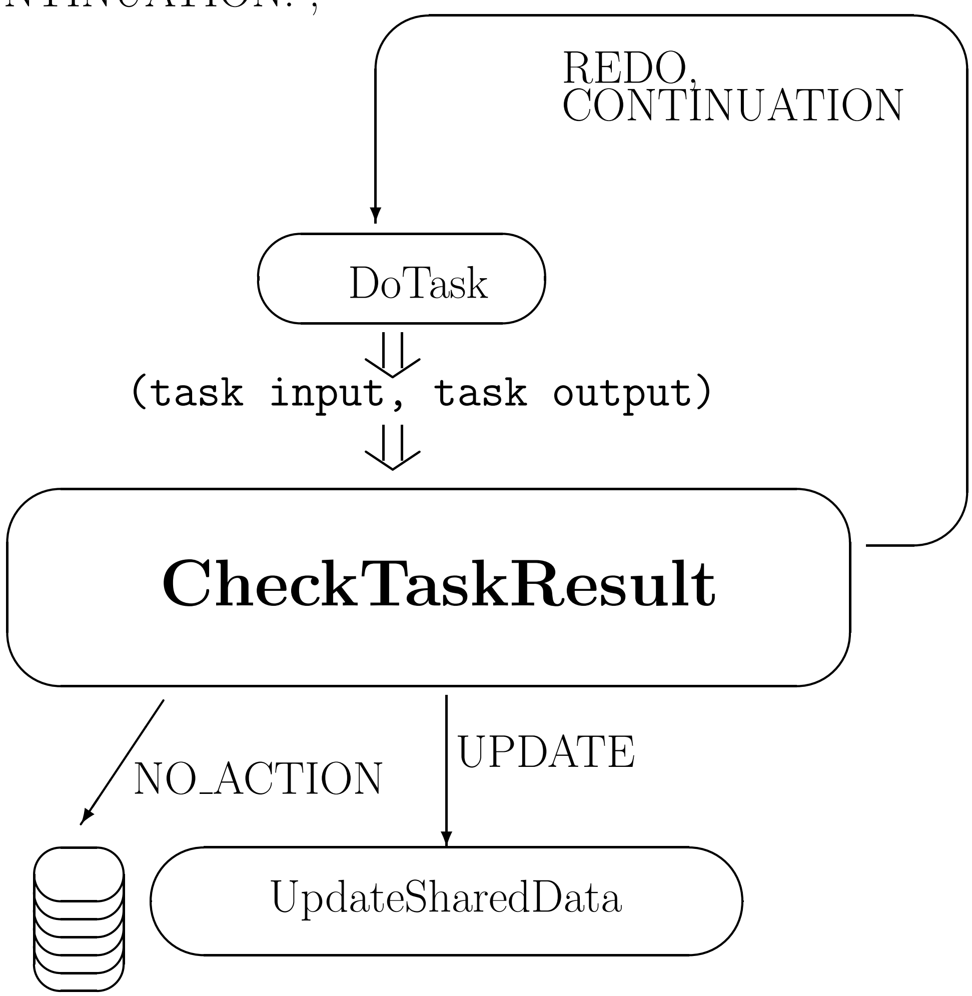
- **Definition:** Global, shared data, readable by all routines in all processes, but writable only by `UpdateSharedData()`
- Shared data modified only by application routine, `UpdateSharedData()`
- Lazy updates: `UpdateSharedData()` called by TOP-C on a given slave *after* current task is complete and *before* next task is processed



Concept 3: The Action

A task results in one of four actions:

- NO_ACTION: /* do nothing */;
- UPDATE: call UpdateSharedData(input, output);
  [on master and all slaves]
- REDO: Ask *same* slave to repeat calculation
  (executed only after *all* pending calls to UpdateSharedData)
  (Slave may have cached information from first computation)
- CONTINUATION: ;



CheckTaskResult(): The Heart of a Parallel Algorithm

```
TOPC_ACTION CheckTaskResult( void *input, void *output )
{ if (output == NULL) return NO_ACTION;
  if (TOPC_is_up_to_date()) return UPDATE;
  else return REDO; }
```

NOTE: Only two library functions for the parallel programmer to know about: `TOPC_master_slave()` and `TOPC_is_up_to_date()`

- Strategy 1: Suitable for few or no `UPDATE` actions.
  1. TRIVIAL PARALLELISM (always `NO_ACTION`: collect results in private variable of master; Report results at end of execution
  2. SEARCH: Most search branches fail, eliminate those cases (e.g.: SharedData is `best_solution_so_far` and prune branches with worse solutions)
- Strategy 2: Many `UPDATE`'s requiring `REDO` action; Slave sees:
  (a) `DoTask(new_task)`
  (b) `UpdateSharedData(...)`
  (c) `REDO:  DoTask(...)`
  1. Execute `REDO` more efficiently; (See next page)
- Strategy 3: Delay `UPDATE`, while merging data on master.
  1. Save next n task outputs from slaves in private global variable on master. Call `NO_ACTION` on first $n-1$ tasks, merge the n outputs, modify task output buffer of n-th task in place, and return `UPDATE`. (Slaves see modifed task output buffer.)

CheckTaskResult() (cont.): Strategy 2

Slave sees:

1. `DoTask(new_task)`
2. `UpdateSharedData(...)`
3. `REDO:  DoTask(...)`

Strategy 2:

- [STEP 1] Save temporary results of `DoTask(new_task)` in private global variable on slave.
- [STEP 2] Save temporary results of `UpdateSharedData(...)` in private global variable on slave.
- [STEP 3] Use temporary results of `DoTask(new_task)` and `UpdateSharedData(...)` to process `new_task` more quickly.

Invocation of TOP-C

- Write application source code including:
  1. initialization of global variables (including shared data);
  2. definition of four TOP-C application functions: `GenerateTaskInput`, `DoTask`, `CheckTaskResult`, `UpdateSharedData`;
  3. invocation of parallelism:
     `MasterSlave(GenerateTaskInput, DoTask, CheckTaskResult, UpdateSharedData);`
- Build application binary:
  1. Compile and link application source using `topcc` command
  - **NOTE:** same source code can be re-compiled for sequential (`topcc --seq`), message-passing (MPI: `topcc --mpi`), SMP (POSIX THREADS: `topcc --pthread`), DSM, and other architectures
- Write `procgroup` file on master specifying:
  1. number of slave processes;
  2. which computer to use (by hostname or Internet number) for each slave; and
  3. location of binary on slave computer
- Execute binary on master:
  This automatically invokes `MasterSlave()` in TOP-C library, which invokes MPI or other parallel library to start slave processes
- **NOTE:** Source code uses *SPMD style:* Master and all slave process execute identical code until they reach `MasterSlave()`

EXAMPLE: Trivial Parallelism

```
#include <topc.h>
/***********************************************************
 ** TO COMPILE: topcc [ --seq | --mpi | --pthread ] THIS_FILE
 ** ALGORITHM:    generic, with independent tasks
 ** shared data:  data written by initialize_data()
 ** task input:   input (case # from 1 to n)
 ** task output:  output (100 int's)
 ** update:  None
 ***********************************************************/
TOPC_MSG_BUF GenerateTaskInput( void );
TOPC_MSG_BUF DoTask( int *input );
TOPC_ACTION CheckTaskResult( int *input, int *output );
int n;
int results[MAX_RESULTS][100];

int main(int argc, char *argv[]) {
  if ( TOPC_is_master() ) {
    n = atoi(argv[1]);
    if ( n > MAX_RESULTS ) {
      printf("n too large\n"); exit(1); }
  }
  TOPC_init(&argc, &argv);
  initialize_data(); /* Called in all processes */
  TOPC_master_slave(GenerateTaskInput, DoTask,
                    CheckTaskResult, NULL); }
  TOPC_finalize();
}
```

EXAMPLE: Trivial Parallelism (cont.)

```
TOPC_MSG_BUF GenerateTaskInput() {
  static int case = 1;
  if ( case > n ) { return NOTASK; }
  return TOPC_MSG(&(case++), sizeof(case));
}
TOPC_MSG_BUF DoTask( int *input ) {
  int case = *input;
  int result[100];
  /* PROCESS case, WRITE TO result */
  return TOPC_MSG( result, 100*sizeof(result[0]) );
}
TOPC_ACTION CheckTaskResult( int *input, int output[] ) {
  int case = *input;
  memcpy( &(results[case]), output,
          100*sizeof(results[0][0] );
  return NO_ACTION;
}
void initialize_data() {
  /* INITIALIZE GLOBAL DATA; CALLED IN ALL PROC'S */
}
```

EXAMPLE: Factorization with naive sieve

```
#include <topc.h>
/***********************************************************
 ** TO COMPILE:  topcc [ --seq | --mpi | --pthread ] THIS_FILE
 ** ALGORITHM:  Euclidean sieve for factorization
 ** shared data:  num_to_factor (divided by each factor found)
 ** task input:   input (test factor for num_to_factor)
 ** task output:  1 or 0 (true or false)
 ** update:  Divide num_to_factor by num
 ***********************************************************/
void trace_input( long *input )
  { printf("%d", *input); }
void trace_result( long *input, long *output )
  { if (*output == 1) printf("TRUE: %d", *input);
    if (*output == 0) printf("FALSE: %d", *input); }
long factors[1000], max_factor=1, num_to_factor;

int main(int argc, char *argv[])
{ int i;
  TOPC_init(&argc, &argv, TOPC_OPTION("trace",1), // 1 is default
                          TOPC_OPTION("trace_input",trace_input),
                          TOPC_OPTION("trace_output",trace_output));

  num_to_factor = atol(argv[1]);
  factors[0]=0; /* number of factor found so far */
  TOPC_master_slave(GenerateTaskInput, DoTask, CheckTaskResult,
                    UpdateSharedData);
  if ( TOPC_is_master() ) {
    for (i = 1; i <= factors[ 0 ]; i++)
      printf("%d ", factors[i]);
    printf("\n");
  }
  TOPC_finalize();
}
```

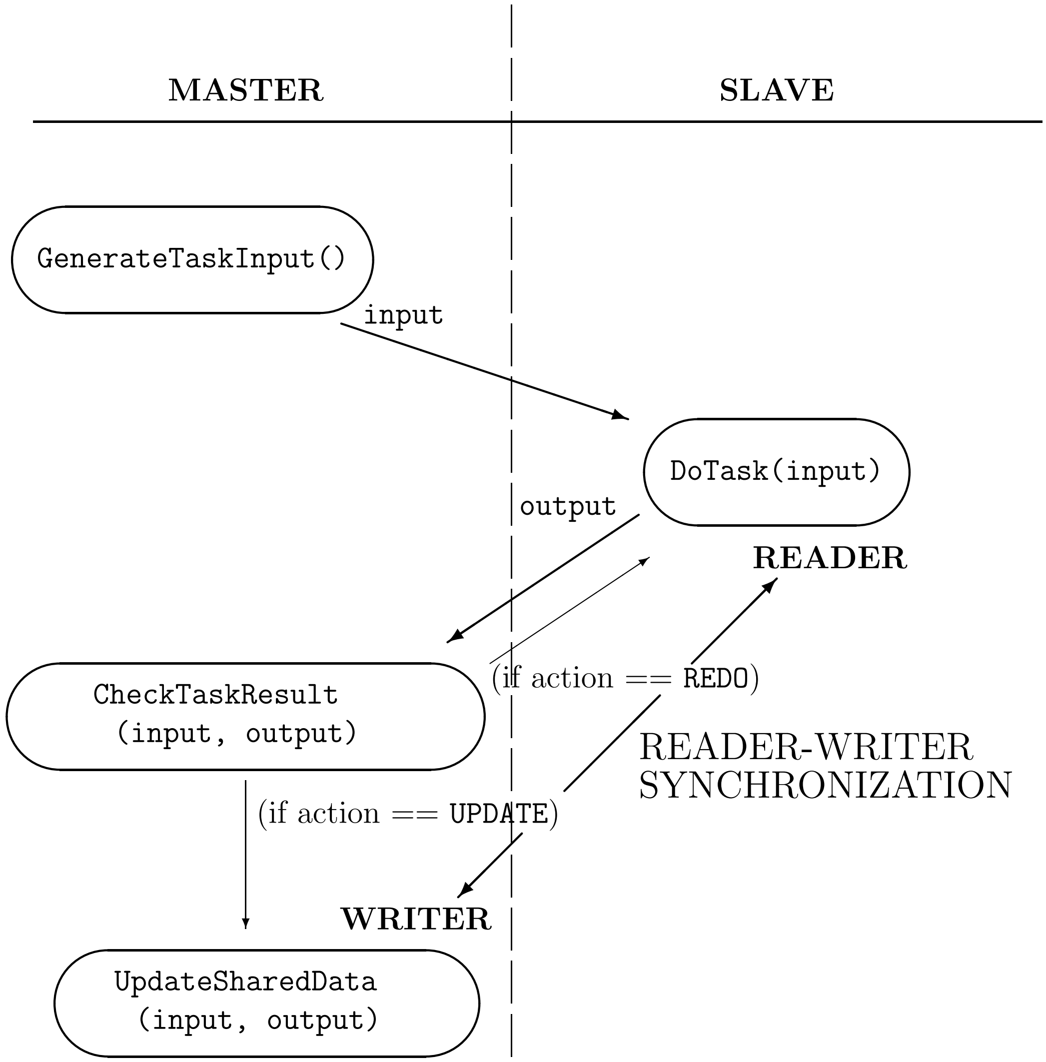
EXAMPLE: Factorization with naive sieve (cont.)

```
TOPC_MSG_BUF GenerateTaskInput() {
  static last_num = 2 - TASK_INTERVAL;
  last_num = last_num + TASK_INTERVAL;
  if ( last_num > num_to_factor ) { return NOTASK; }
  return TOPC_MSG(&last_num, sizeof(last_num)); }

TOPC_MSG_BUF DoTask( long *input )
{ long num = *input;
  long limit = limit = num + TASK_INTERVAL;
  static task_out;

  for ( ; num < limit; num++ )
    if (num_to_factor % num == 0) {
      task_out = 1;
      return TOPC_MSG(&task_out, sizeof(long)); }
  task_out = 0;
  return TOPC_MSG(&task_out, sizeof(task_out)); }
TOPC_ACTION CheckTaskResult( long *input, long *output )
{ if ( *output == 0 ) return NO_ACTION;
  if ( TOPC_is_up_to_date() ) return UPDATE;
  if ( *input < max_factor ) return UPDATE;
  else return REDO; }
void UpdateSharedData( long *input, long *output )
{ long limit, num = *input;
  int i;
  limit = num + TASK_INTERVAL;
  ... }
```

EXAMPLE: Factorization with naive sieve (cont.)

```
void UpdateSharedData( long *input, long *output )
{ long limit, num = *input;
  int i;

  limit = num + TASK_INTERVAL;
  // NOTE: Suppose we factor 12, 4 is reported as factor
  //       before 2 is reported
  for ( ; num < limit; num++ ) {
    if ( num_to_factor % num == 0 ) {
      if ( num < max_factor ) {
        max_factor = 1; // re-compute max_factor
        for( i = 1; i <= factors[0]; i++ ) {
          if (factors[i] > max_factor) max_factor = factors[i
          while (factors[i] % num == 0) {
            factors[i] = factors[i] / num;
            num_to_factor = num_to_factor * num;
          }
        }
      }
      while (num_to_factor % num == 0) {
        factors[ ++factors[0] ] = num;
        num_to_factor = num_to_factor / num;
        if (num_to_factor == 1) return;
      }
    }
  }
}
```

Life Cycle of A Task: Shared Memory



TOP-C Programmer's Model: Shared Memory

ISSUES: Scaling Up (current and future work)

- Overloading of single master for > 100 slaves
  + **Solution:** tree of masters with slaves at leaves
- Update of shared data creates too many sequential barriers
  + **Solution:** creation of multiple pages of shared data
- Very large shared data structure (TOP-C shared data) on top of shared memory hardware (SMP) or Distributed Shared Memory (DSM)
  + **Solution:** Atomic writes of each "record" in UpdateSharedData()
    (DSM resources: release consistency w/ acquire/release)
- Very large TOP-C shared data on top of distributed memory architecture
  + **Solution:** Pipelining data (DSM resources: release consistency w/ acquire/release)
- Data-Parallel applications (opposite extreme from task-oriented applications, needed for numerical applications or hybrid numeric/symbolic)
  + Example solution for Gaussian elimination: processing of row or "band of rows" becomes task

SCALING UP: Multiple Pages of Shared Data

Modified Syntax:

- `UPDATE` → `UPDATE("3,7,9-20")`
- `TOPC_is_up_to_date()` → `TOPC_is_up_to_date("5,21-24")`

Typical usage: `CheckTaskResult()`

```
TOPC_MSG_BUF CheckTaskResult ( void *input, void *output )
  if ( output == NULL ) return NO_ACTION;
  sharedMemPagesRead =
    SharedMemPagesUsedToCompute( output );
  if ( ! TOPC_is_up_to_date( sharedMemPagesRead ) )
    return REDO;
  sharedMemPagesToBeWritten =
    SharedMemPagesModifiedBy( output );
  return UPDATE( sharedMemPagesToBeWritten );
end;
```