Exercise 5, item 3: Parameters.

```
--- ../../step2_params/main.cpp
+++ main.cpp
@@ -9,7 +9,7 @@
          .description("Demo ALPSCore program")
          .define<std::string>("name","A name to greet")
          .define<int>("count", 1, "How many times to greet")
-         .define("loud", "Should we be \"loud\"?");
+         .define<int>("loud", 0, "Should we be \"loud\"?");

      if (p.help_requested(std::cerr))
          return 1;
@@ -17,7 +17,7 @@
      if (p.has_missing(std::cerr))
          return 2;

-     std::string ending=p["loud"]?"!!!":".";
+     std::string ending=(p["loud"]!=0)?"!!!":".";
      for (int i=0; i<p["count"]; ++i) {
          std::cout << "Hello, " << p["name"].as<std::string>() << ending <<
"\n";
      }
```

Exercise 6, item 5: Trivial MC.

```
--- ../../step3_trivial_mc/simulation.cpp
+++ simulation.cpp
@@ -23,6 +23,7 @@
 }

 double MySimulation::fraction_completed() const {
+    if (maxcount_==0) return 0;
     double frac=double(istep_)/maxcount_;
     return frac;
 }
```

Exercise 6, item 6: Trivial MC.

```
diff -u -b -r ../../step3_trivial_mc/simulation.cpp ./simulation.cpp
--- ../../step3_trivial_mc/simulation.cpp
+++ ./simulation.cpp
@@ -8,7 +8,7 @@
         verbose_=params["verbose"];
     }

-void MySimulation::update() {
+void MySimulation::renamed_update() {
     double r=random();
     if (verbose_) {
         std::cout << "Update at step " << istep_ << ", random=" << r <<
std::endl;


diff -u -b -r ../../step3_trivial_mc/simulation.hpp ./simulation.hpp
--- ../../step3_trivial_mc/simulation.hpp
+++ ./simulation.hpp
@@ -11,7 +11,7 @@
   public:
     MySimulation(const parameters_type& params, std::size_t
seed_offset=0);

-    void update();
+    void renamed_update();
     void measure();
     double fraction_completed() const;
```

Exercise 7, item 4: Computing Pi.

```
diff -u -b -r ../../step4_pi/simulation.hpp ./simulation.hpp
--- ../../step4_pi/simulation.hpp
+++ ./simulation.hpp
@@ -18,7 +18,7 @@
     static double objective_function(double x, double y);

     // Accumulator type to collect observables.
-    typedef alps::accumulators::FullBinningAccumulator<double>
my_accumulator_type;
+    typedef alps::accumulators::NoBinningAccumulator<double>
my_accumulator_type;

     MySimulation(const parameters_type& params, std::size_t
seed_offset=0);
```

```
diff -u -b -r ../../step4_pi/main.cpp ./main.cpp
--- ../../step4_pi/main.cpp
+++ ./main.cpp
@@ -53,9 +53,6 @@
                << " ... "
                << pi_res.mean<double>()+pi_res.error<double>()
                << std::endl;
-    std::cout << "Autocorrelation length: "
-              << pi_res.autocorrelation<double>()
-              << std::endl;

     return 0;
 }
```

Exercise 8, item 2: Checkpointng.

```
diff -u -b -r ../../step5_pi_checkpoint/main.cpp ./main.cpp
--- ../../step5_pi_checkpoint/main.cpp
+++ ./main.cpp
@@ -11,8 +11,7 @@
     alps::params p(argc, (const char**)argv);

     // Define the parameters
-    mysim_type::define_parameters(p)
-        .define<std::size_t>("timelimit", 5, "Time limit for the
computation");
+    mysim_type::define_parameters(p);

     if (p.help_requested(std::cerr) || p.has_missing(std::cerr))
         return 1;
```

Exercise 10, item 1: Parallelization of 2D Ising.

```
diff -u -b -r ../../step7_ising/main.cpp ./main.cpp
--- ../../step7_ising/main.cpp
+++ ./main.cpp
@@ -13,20 +13,29 @@
 #include <alps/mc/api.hpp>
 #include <alps/mc/mcbase.hpp>
 #include <alps/mc/stop_callback.hpp>
+#include <alps/mc/mpiadapter.hpp>

 int main(int argc, char* argv[])
 {
     // Define the type for the simulation
-    typedef ising_sim my_sim_type;
+    typedef alps::mcmpiadapter<ising_sim> my_sim_type;
+
+    // Initialize the MPI environment, and obtain the WORLD communicator
+    alps::mpi::environment env(argc, argv);
+    alps::mpi::communicator comm;
+    const int rank=comm.rank();
+    const bool is_master=(rank==0);
+

     try {

         // Creates the parameters for the simulation
         // If an hdf5 file is supplied, reads the parameters there
-        std::cout << "Initializing parameters..." << std::endl;
+        // This constructor broadcasts to all processes
+        if (is_master) std::cout << "Initializing parameters..." <<
std::endl;

-        alps::params parameters(argc, (const char**)argv);
+        alps::params parameters(argc, (const char**)argv, comm);
         my_sim_type::define_parameters(parameters);
```

```
         if (parameters.help_requested(std::cout) ||
@@ -34,31 +43,32 @@
             return 1;
         }

-        std::cout << "Creating simulation" << std::endl;
-        my_sim_type sim(parameters);
+        std::cout << "Creating simulation on rank " << rank << std::endl;
+        my_sim_type sim(parameters, comm);

         // If needed, restore the last checkpoint
         std::string checkpoint_file =
parameters["checkpoint"].as<std::string>();
+        if (!is_master)
checkpoint_file+="."+boost::lexical_cast<std::string>(rank);

         if (parameters.is_restored()) {
             std::cout << "Restoring checkpoint from " << checkpoint_file
-                      << std::endl;
+                      << " on rank " << rank << std::endl;
             sim.load(checkpoint_file);
         }

         // Run the simulation
-        std::cout << "Running simulation" << std::endl;
+        std::cout << "Running simulation on rank " << rank << std::endl;
         sim.run(alps::stop_callback(size_t(parameters["timelimit"])));

         // Checkpoint the simulation
         std::cout << "Checkpointing simulation to " << checkpoint_file
-                  << std::endl;
+                  << " on rank " << rank << std::endl;
         sim.save(checkpoint_file);

         alps::results_type<my_sim_type>::type results =
```

```
alps::collect_results(sim);

         // Print results
-        {
+        if (is_master) {
             std::cout << "All measured results:" << std::endl;
             std::cout << results << std::endl;

@@ -85,9 +95,11 @@
         return 0;
     } catch (const std::runtime_error& exc) {
         std::cout << "Exception caught: " << exc.what() << std::endl;
+        env.abort(2);
         return 2;
     } catch (...) {
         std::cout << "Unknown exception caught." << std::endl;
+        env.abort(2);
         return 2;
     }
 }
```