ECE 514/CS 514

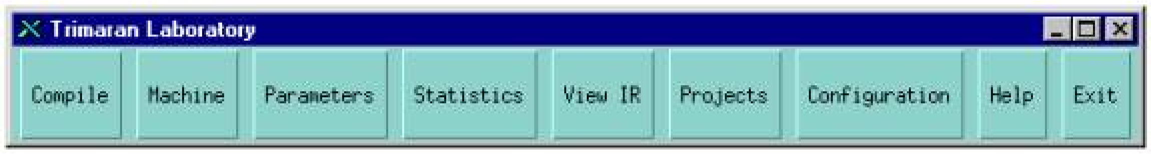
Lab 1 - Using Trimaran

All students must hand in an individual lab report by the due date. The reports are considered as portion the lab grade.

Goals:

1. Understand how to launch the Trimaran compiler and simulator via the X-window interface
2. Launching the compiler via shell script
3. Identify the phases of the compiler
4. Identify the files generated during the compilation process
5. Learn to read Rebel code well enough to match it with a simple C codeS

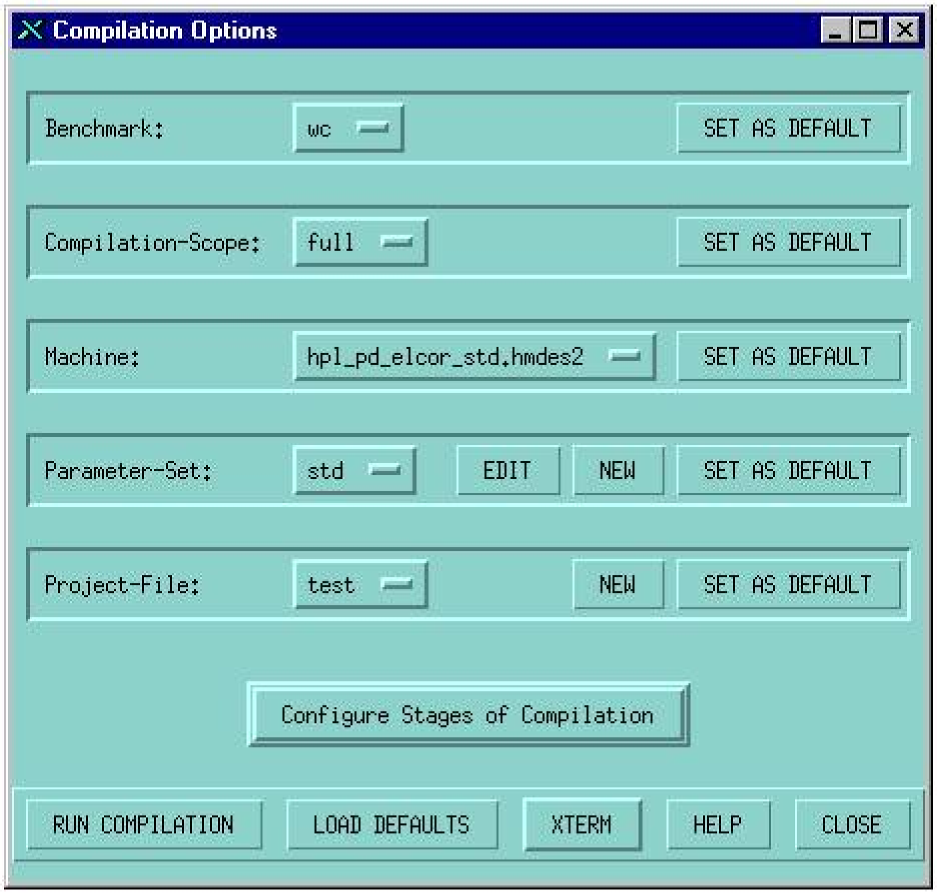
```
% trimaran/bin/trimaran-gui &
```



NOTE: You should also open a few shell terminal windows(you may want to open an editor like xemacs or kde's advanced editor). You will need these later in order to examine files that are created by Triamaran.

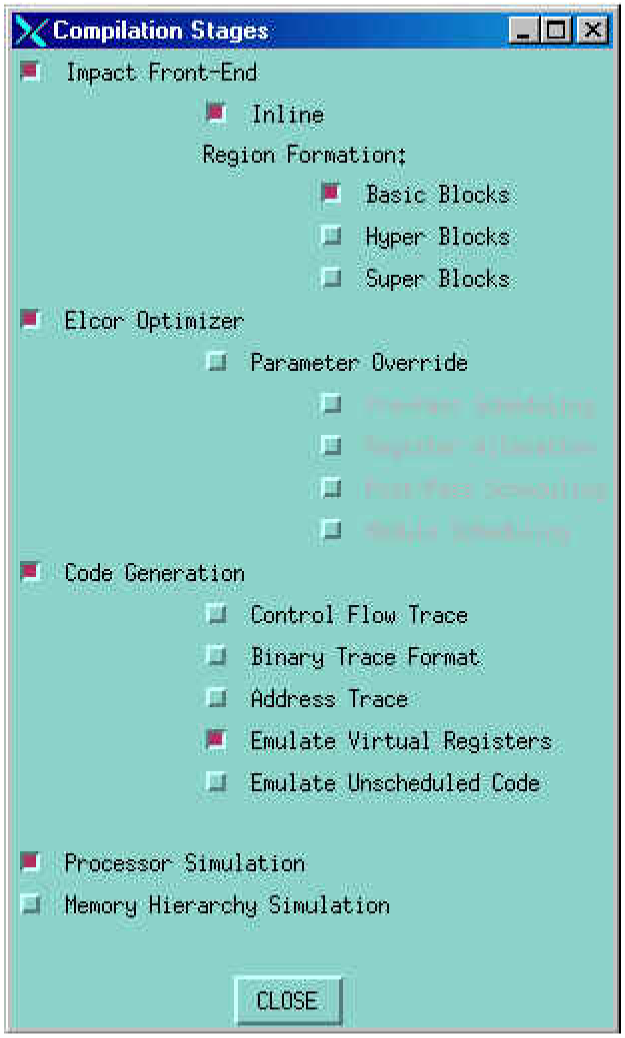
5.If this is the first time you are running Trimaran, it will attempt to create a workspace directory named **trimaran-workspace** in your home directory. You *must* have this workspace in order to work with Trimaran.

6.Choose the **Compile** button from the Trimaran menu. You will see the following window popping up:



Before your first compilation, you must give a name to your compilation project by going down to *Project-File* sub-window and selecting **New**.

7. Now let's compile the mm benchmark. Go to the *Benchmark* sub-window and select **mm** from the pull-down menu. mm is a matrix multiply benchmark. Now Click on the **Configure Stages of Compilation** button. The following screen or one very simular will show up.



Compilation Stages

- ☑ Impact Front-End
  - ☑ Inline
  - Region Formation:
    - ☑ Basic Blocks
    - ☐ Hyper Blocks
    - ☐ Super Blocks
- ☑ Elcor Optimizer
  - ☐ Parameter Override
    - ☐ Prepass Scheduling
    - ☐ Register Allocation
    - ☐ Post-Pass Scheduling
    - ☐ Modulo Scheduling
- ☑ Code Generation
  - ☐ Control Flow Trace
  - ☐ Binary Trace Format
  - ☐ Address Trace
  - ☑ Emulate Virtual Registers
  - ☐ Emulate Unscheduled Code
- ☑ Processor Simulation
- ☐ Memory Hierarchy Simulation

The 3 major sections of this window correspond to the 3 major parts of Trimaran. Impact is the front-end of the compilation system and performs most of the traditional front-end features of any compiler. Elcor is the back-end optimizer of the system, and performs the machine dependent back-end phases of the compiler. The simulator (SIMU + Dinero IV) is the 3rd component that allows this system to generate performace data on any compiled program.

Be sure that you are configured as above for *Inlining* and *Basic Block Formation* in the Front-end, standard settings in Elcor, and you want the simulator to be executed.

Click **Close** and then select the **Run Compilation** button. Ascreen will pop up and scroll by, don't worry about that now. We will show you where that is later.

8. After the compilation, identify the files created in the trimaran-workspace directory during the compilation process.

1. Change directories to ~/trimaran-workspace/projects/ and look at the contents of the directory. You should see a directory with the same name as your project name. Change to that directory.

2. Now you will see a directory name like: hpl_pd_elcor_std.std.mm.b.# where # is 1, 2, 3, etc. This name has a lot of meaning. The *hpl_pd_elcor_std* is the name of the MDES file, the *std* is the name of the parameter set used, the *mm* is the benchmark name, *b* is for basic block, and # is the instance of compilation where none of the previous parms have changed.

3. Change to a directory for one of your compilation instances. Examine this directory. You will see the following:

    1. compilation.log : a verbose recording of the compilation (i.e. The info in the scrolling screen during compilation)

    2. compile_script : a shell script that can be invoked e.g. ./compile_script to execute this version again.

    3. elcor_intermediate, simu_intermediate, and impact_intermediate: These directories will contain intermediate files created by Trimaran during the compilation. Each stage/phase will have IR files, data files, stats files, etc.

9. In the current directory for the mm run, locate the file **compile_script**. Execute it by typing the following:

```
% ./compile_script
```

What happens? Why would do you think this would be useful? What are the advantages over using the GUI again to compile?

10. **DELIVERABLE #1:** By reading the log file **compilation.log**, we can see every phase of the compilation process.

1. Determine which phase of Trimaran produce which file(s) in the intermediate directories? For example, which command from the compilation produces the .tr file.

2. Also, identify the work that is done in each of these phases. What is the functionality of each phase i.e. is this a scheduling phase, a parsing phase, or an scalar optimization phase just to name a few? Summarize all of this information

using an annotated **flow chart**. The flow chart should be annotated with input and output files, phase names, and the functionality of the phase. You will be adding other paths of compilation to the flow chart so make sure you leave room to branch out.

Completing this deliverable means you should have a clear picture of how a program is converted from a C program to a simulated executable within the context of the Trimaran compilation system. It is not important that you know all of the details of each phase; however, you should have a high-level understanding of the purpose of each phase.

11. Now you have a good picture of what Triamaran has to do to compile a program and simulate it. Now we want you to identify things that are different when you change the *Region Formation* types. Change the *Region Formation* option to *Super Block Fomation* and repeat the compilation.

APPEND TO DELIVERABLE #1: Identify the differences between *Basic Block Formation* and *Super Block Formation* in terms of the compilation phases and intermediate files. Update your flow chart with the alternative paths/steps/annotations.

12. Repeat (8) for *Hyper Block Formation*.S

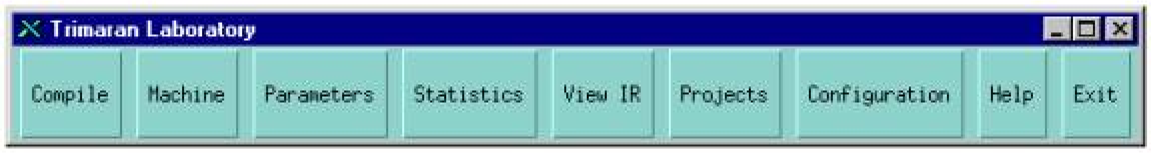
13. Now you need to update the simulator, by adding cache simulation. On the webpage, there is a dowload link called "Simu + Cache Addon". The file you download will be a .tgz file. Save the file to your trimaran/simu directory.

1. cd to your trimaran/simu directory (should be "~/trimaran/")
2. cd src/Emulib/static_lib
3. mv libequals.a libequals.a.old
4. cd ../../../
5. tar xzvf cache-sim-dinero.tgz

This will unpack to files (one is a symlink to the other). Compile and simulate an benchmark. Check the simu_intermediate directory for this execution. Verify that the file PD_STATS was generated. In addition see that SMACHS_GSTATS.sg file was generated; this file will contain cache statistics.

Reading Rebel files

Some of the files found in the intermediate directories are Rebel files. Rebel is the ascii representation of the Elcor IR. Over the next week or 2 in class you will be introduced to HPL-PD and Rebel. This acts as an introduction to how Rebel will play a role in Trimaran, and how the many ways that Triamaran will allow you to explore Rebel. If you closed you Trimaran process, launch **trimaran-gui**. You will see the following window popping up:



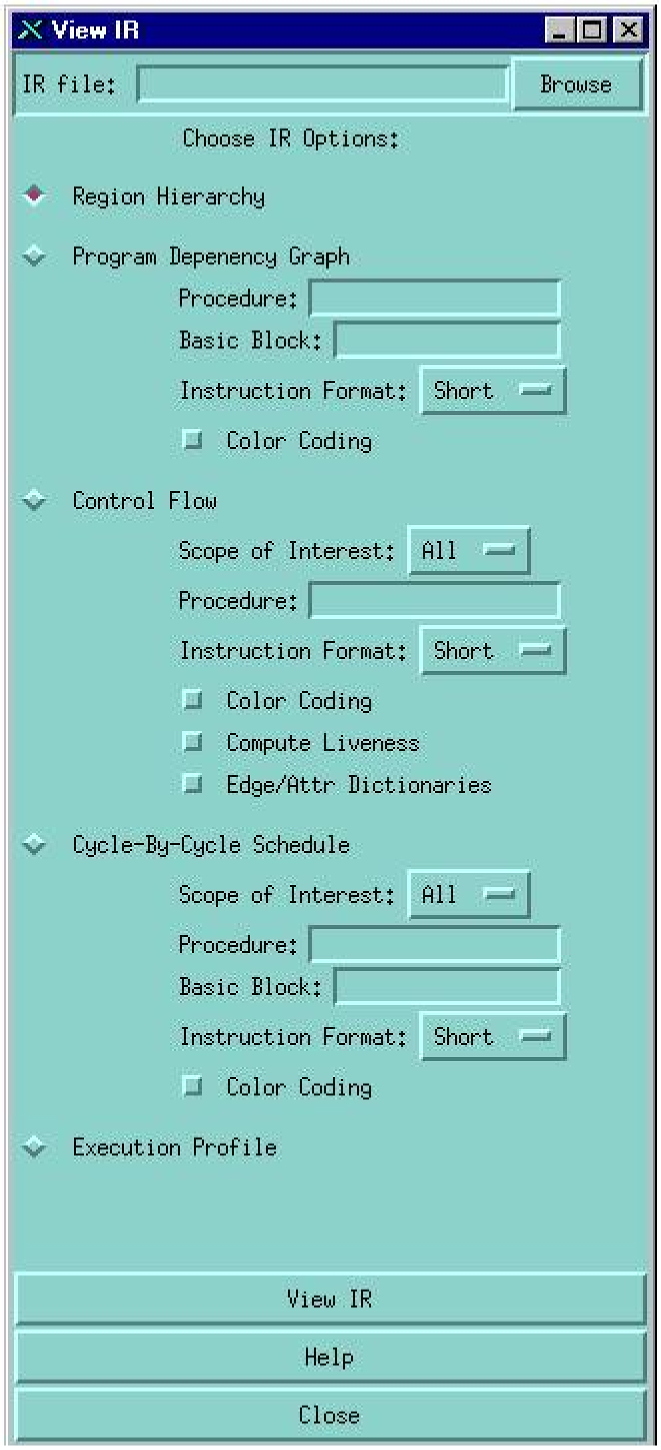
1. 1. Compile the **fact2** benchmark using any of the settings you used for mm before.

2. 2. Lets take a look at the source file. With your favorite editor, open the fact2.c file located in the **~/trimaran-workspace/benchmarks/fact2/src** directory. Examine the source code.

   - Compare the location of the printf statements and the output that you can find in the **compilation.log** file.

   - Leave this source file open so that you can compare with the IR views.

3. Choose the "View IR" from the above menu. You will see the following pop-up screen.



View IR

IR file: Browse

Choose IR Options:

- [✓ Region Hierarchy]
- Program Depenency Graph
  - Procedure:
  - Basic Block:
  - Instruction Format: Short
  - Color Coding
- Control Flow
  - Scope of Interest: All
  - Procedure:
  - Instruction Format: Short
  - Color Coding
  - Compute Liveness
  - Edge/Attr Dictionaries
- Cycle-By-Cycle Schedule
  - Scope of Interest: All
  - Procedure:
  - Basic Block:
  - Instruction Format: Short
  - Color Coding
- Execution Profile

View IR

Help

Close

This is the main window for where you can browse the IR from many different views. All of these options require you to load an Elcor .el file from the elcor_intermediate directory.

Using the "Browse" option, open the **fact2.O_el** file(follow the directory structure). **(checkpoint)** At what point in your flow chart is this file generated?

4. Try each of the following views of the IR, i.e. "Region Hierarchy", "Program Dependency Graph", "Control Flow", "Cycle-by-Cycle Schedule" and "Execution Profile".

- Start with the Execution Profile to get a feel for where the program spends a lot of its time.
- Use the Control Flow view to find the area where the work is being done.
- Map this are of the control flow back to the C program. For example, where is the loop, the system calls, and the if statement.
- Which basic block #'s correspond with which C code constructs i.e. using the viewer, identify the basic blocks in the source code.

5. **DELIVERABLE #2:** Identify an appropriate part of the program (i.e. not too small to be uninteresting nor too big to be messy). Using the information you looked at in step 14 describe the information being represented by the Region Hierarchy, Program Dependency Graph, Control Flow, Cycle-by-Cycle Schedule, and Execution Profile. In some cases, you may have to consider the whole program to have any reasonable results. This is left up to your discretion. When you have completed this deliverable, you should be comfortable with the different views of a program, and how these views relate to each other and the source code.

Tracing Rebel Code

Rebel operations have the same semantics as described in the HPL-PD ISA. At this time, you are being asked to trace some Rebel code using the semantics of the HPL-PD ISA. Generate a BB version of the ***eight*** benchmark. Trace the Rebel code, and determine what it is doing in the 1st 4 iterations of the main loop.

DELIVERABLE #3: You will submit the following:

- A table that shows a mapping of variable names found in the source code and the registers that are used to hold the values of those variables.

- A table that shows the execution of each op in the 1st and 4th iterations (Please do not submit all 4 iterations)
    - Each row of this table should contain the op #, opcode, src and dest registers and their corresponding values.
- A table that summarizes the state at the end of the 4th iteration; it contains the registers used and their contents.

--- *End of Lab* ---