Georgia Institute of Technology

CREST

Trimaran: An Infrastructure for Research in Instruction-Level Parallelism

Lakshmi-Narasimhan Chakrapani

Rodric M. Rabbah

Krishna V. Palem

Center for Research on Embedded Systems and Technology
Georgia Institute of Technology, Atlanta, GA

Introduction

What Is Trimaran ?

- A parametric compilation and performance monitoring system
  - A full-blown C compiler for the HPL-PD instruction set architecture (ISA)
  - A cycle-by-cycle parametric machine simulator + cache simulator
  - A suite of optimization and analysis tools
- Uses HPL-PD a parameterized very long instruction word (VLIW) ISA
  - Supports predication, control and data speculation and compiler controlled management of the memory hierarchy
- Compiles for target architectures specified by a machine description language
  - Can compile optimized code for a variety of VLIW and Superscalar architectures

Trimaran Goals

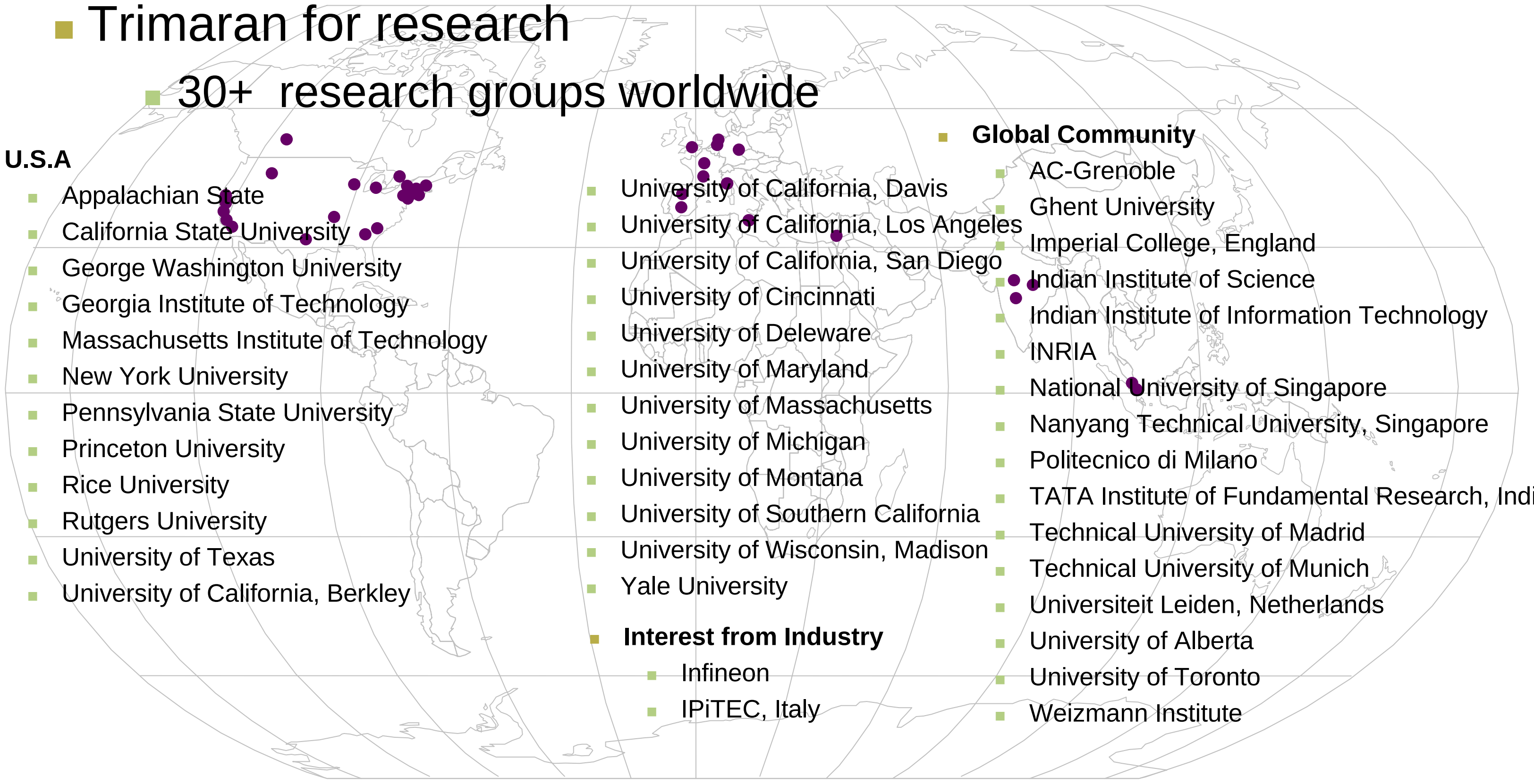
- To provide a vehicle for implementation and experimentation for state of the art research in compiler techniques
  - Consists of a full suite of analysis and optimization modules
  - Optimizations and analysis modules can be easily added, deleted or bypassed, thus facilitating compiler optimization research
    - R. M. Rabbah and K. V. Palem. "Data remapping for design space optimization of embedded memory systems.", *In ACM Transactions on Embedded Computing Systems (TECS)*, 2(2), 2003
- Study and evaluation of novel architectural features
  - Currently, the infrastructure is oriented towards Explicitly Parallel Instruction Computing (EPIC) architectures
    - But can also support compiler research for Superscalar and other novel architectures
    - S. Talla. "Adaptive Explicitly Parallel Instruction Computing.", PhD thesis, New York University, Department of Computer Science, 2000.
- Study and evaluation of language features and architecture modeling languages
    - S. P. Seng, K. V. Palem, R. M. Rabbah, W.-F.Wong, W. Luk, and P. Cheung. "PD-XML: Extensible markup language for processor description." *In Proceedings of the IEEE International Conference on Field-Programmable Technology (ICFPT)*, Dec. 2002.

Trimaran Availability and Support

- Developed through a collaborative effort
  - Compiler and Architecture Research Group at Hewlett Packard Laboratories
  - IMPACT Group at the University of Illinois
  - Center for Research on Embedded Systems and Technology (CREST) at the Georgia Institute of Technology
    - CREST was the ReaCT-ILP Laboratory at New York University
- Result of many man-years of research and development
  - Distributed without charge for non-commercial use
    - http://www.trimaran.org
- Website has instruction and installation manuals, reading lists of compiler optimizations, an expanded version of this tutorial
  - Support includes forum for discussion, newsletters
    - Email questions, concerns, and comments support@trimaran.org

Trimaran Today

- Trimaran for research
  - 30+ research groups worldwide



- **U.S.A**
  - Appalachian State
  - California State University
  - George Washington University
  - Georgia Institute of Technology
  - Massachusetts Institute of Technology
  - New York University
  - Pennsylvania State University
  - Princeton University
  - Rice University
  - Rutgers University
  - University of Texas
  - University of California, Berkley
  - University of California, Davis
  - University of California, Los Angeles
  - University of California, San Diego
  - University of Cincinnati
  - University of Deleware
  - University of Maryland
  - University of Massachusetts
  - University of Michigan
  - University of Montana
  - University of Southern California
  - University of Wisconsin, Madison
  - Yale University
- **Interest from Industry**
  - Infineon
  - IPiTEC, Italy
- **Global Community**
  - AC-Grenoble
  - Ghent University
  - Imperial College, England
  - Indian Institute of Science
  - Indian Institute of Information Technology
  - INRIA
  - National University of Singapore
  - Nanyang Technical University, Singapore
  - Politecnico di Milano
  - TATA Institute of Fundamental Research, India
  - Technical University of Madrid
  - Technical University of Munich
  - Universiteit Leiden, Netherlands
  - University of Alberta
  - University of Toronto
  - Weizmann Institute

- Trimaran In the classroom
  - Georgia Institute of Technology, New York University, University of Alberta, Politecnico di Milano, Appalachian State University, Indian Institute of Information Technology

Quick Overview

Infrastructure Components

- A machine description language, HMDES, for describing ILP architectures.
- A parameterized ILP Architecture called HPL-PD
  - Current instantiation in the infrastructure is as a EPIC architecture
- A compiler front-end for C, performing parsing, type checking, and a large suite of high-level (i.e. machine independent) optimizations.
  - This is the IMPACT module (IMPACT group, University of Illinois)
- A compiler back-end, parameterized by a machine description, performing instruction scheduling, register allocation, and machine-dependent optimizations
  - Each stage of the back-end may easily be replaced or modified by a compiler researcher
  - Primarily implemented as part of the ELCOR effort by the CAR Group at HP Labs
  - Augmented with a scalar register allocator from the CREST

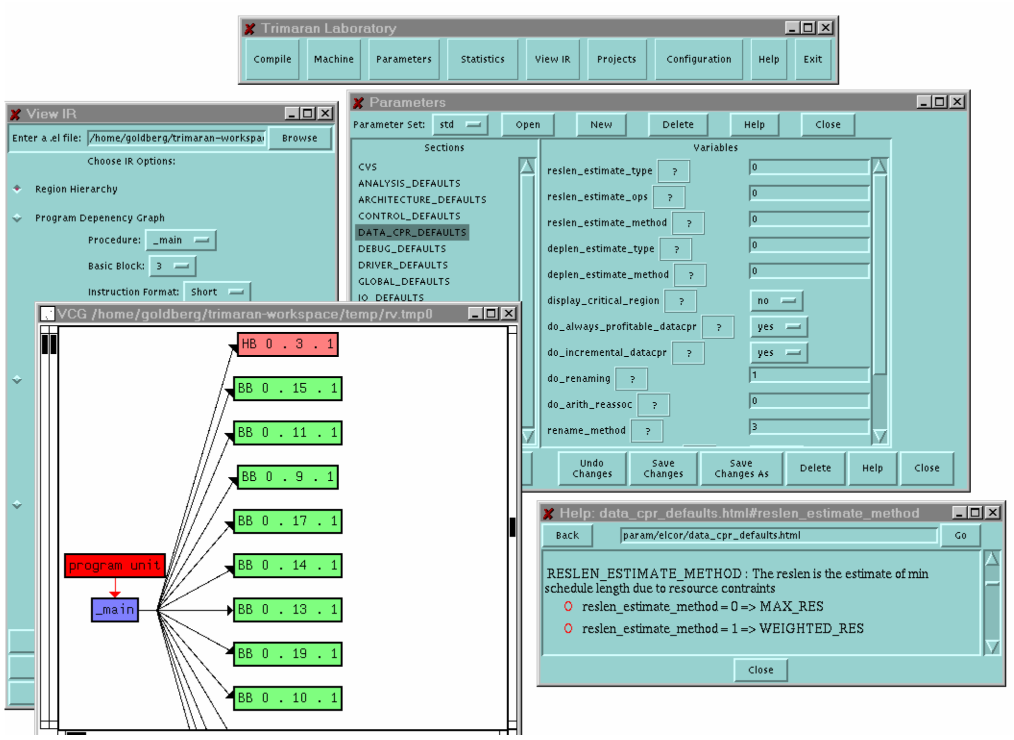
Infrastructure Components

- An extensible IR (intermediate program representation)
  - Has both an internal and textual representation, with conversion routines between the two. The textual language is called REBEL
  - Supports modern compiler techniques by representing control flow, data and control dependence, and many other attributes
  - Easy to use in its internal representation (clear C++ object hierarchy) and textual representation (human-readable)

- A cycle-level simulator of the HPL-PD, configurable by a MDES and provides run-time information on execution time, branch frequencies, and resource utilization
  - This information can be used for profile-driven optimizations, as well as to provide validation of new optimizations
  - The HPL-PD simulator was implemented by the ReaCT_ILP group at NYU

Infrastructure Components

- An Integrated graphical user interface (GUI) for configuring and running the Trimaran system



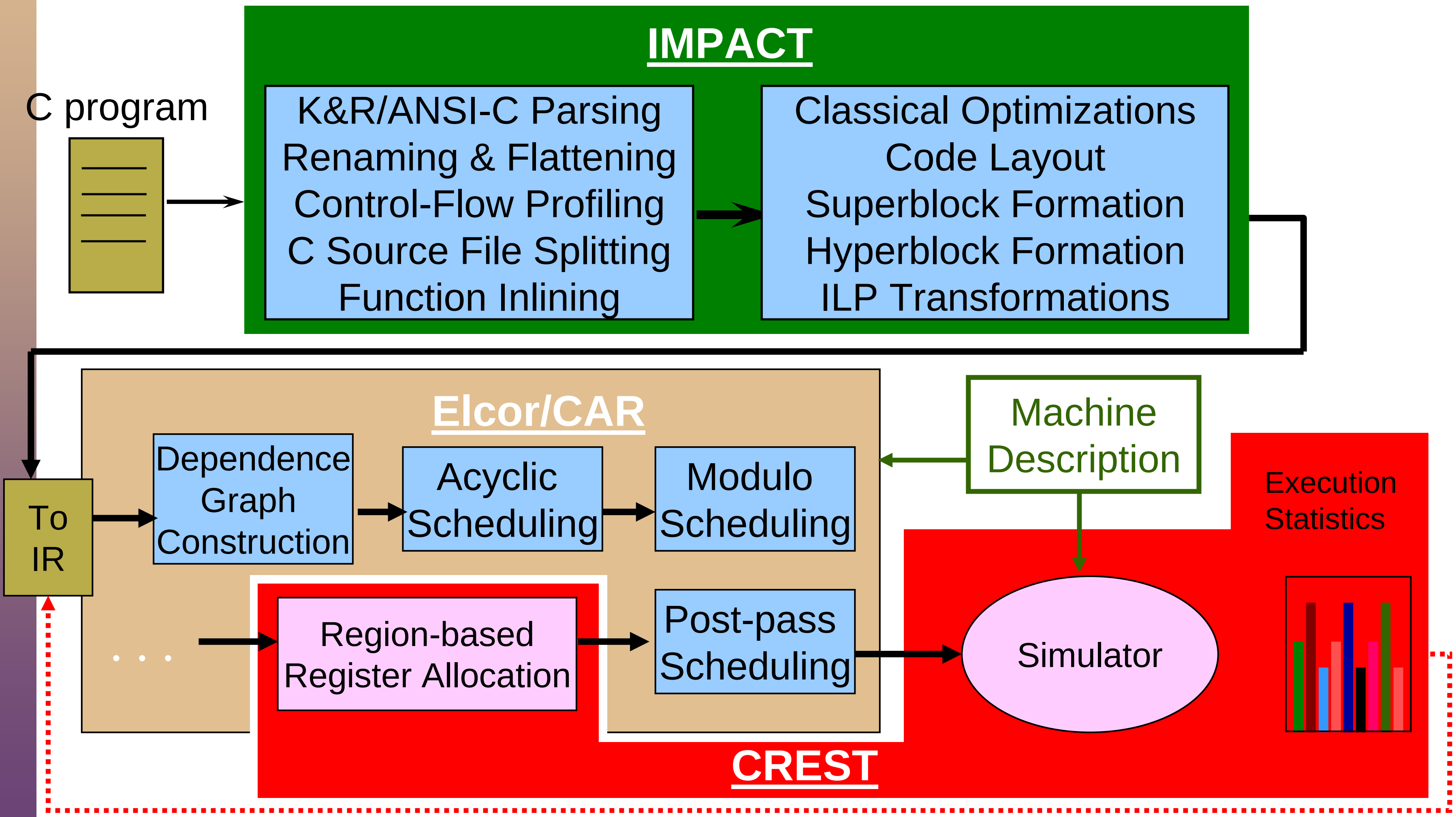
reslen_estimate_type	0
reslen_estimate_ops	0
reslen_estimate_method	0
deplen_estimate_type	0
deplen_estimate_method	0
display_critical_region	no
do_always_profitable_datacpr	yes
do_incremental_datacpr	yes
do_renaming	1
do_arith_reassoc	0
rename_method	3

RESLEN_ESTIMATE_METHOD : The reslen is the estimate of min schedule length due to resource contraints

- reslen_estimate_method = 0 => MAX_RES
- reslen_estimate_method = 1 => WEIGHTED_RES

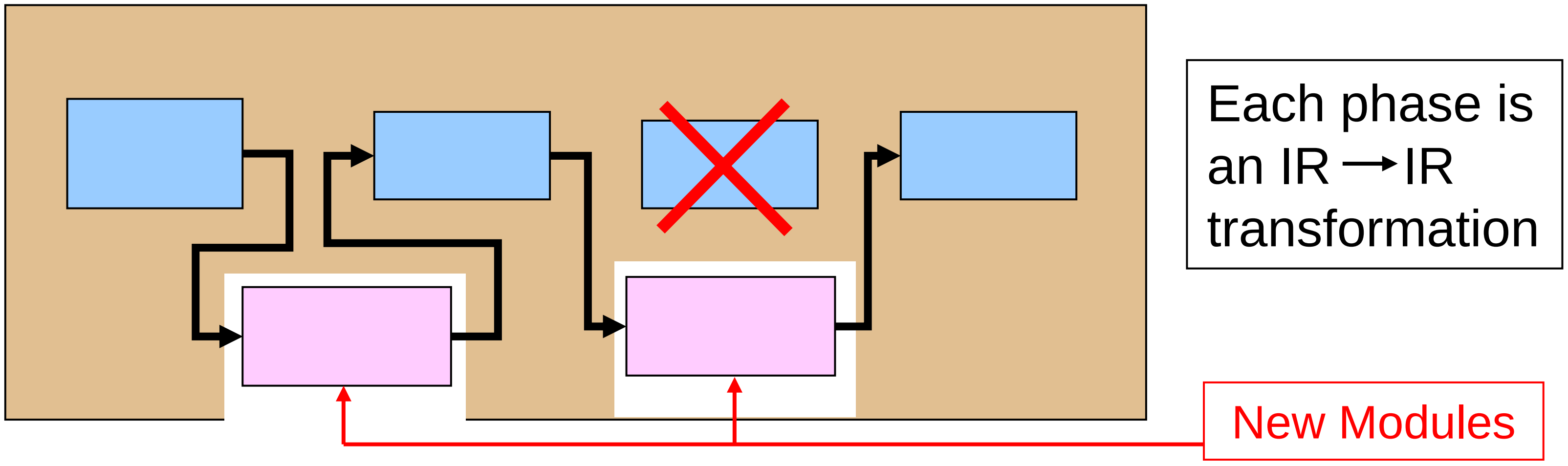
System Organization

- A compiler researcher's view of the infrastructure:



The Research Process

- The infrastructure is used for designing, implementing, and testing new compilation modules to be incorporated into the back end
  - These phases may augment or replace existing ILP optimization modules
- New modules may be the result of research in scheduling, register allocation, program analysis, profile-driven compilation, etc.
  - For example, NYU has added a region-based register allocator



HPL-PD
A Parameterized Research Architecture

Overview of HPL-PD

- HPL-PD is a parameterized ILP architecture
  - serves as a vehicle for processor architecture and compiler optimization research.
  - Admits both EPIC and superscalar implementations

- The HPL-PD parameter space includes:
  - Number and types of functional units
  - Number and types of registers (in register files)
  - Width of the instruction word (for EPIC)
  - Instruction latencies

Features of HPL-PD

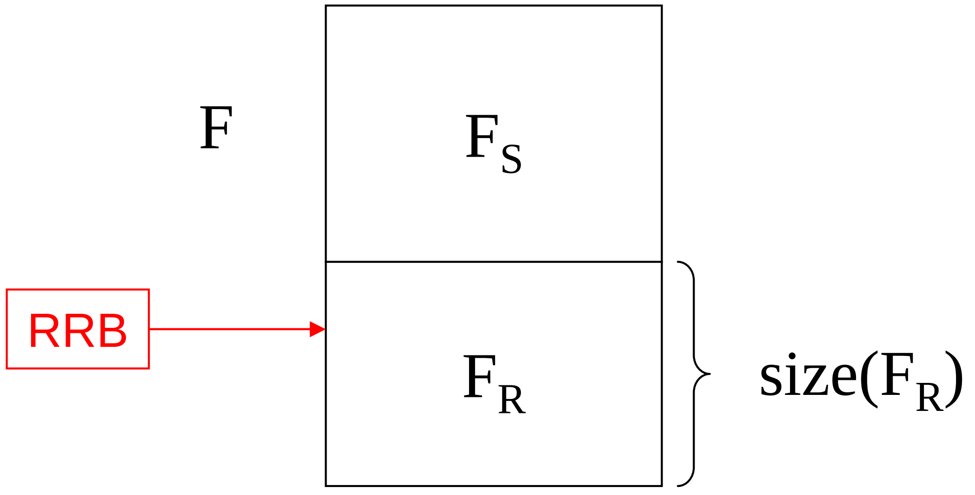
- Support for speculative execution
  - Data speculation (run-time address disambiguation)
  - Control speculation (eager execution)
- Predicated (guarded) execution
  - Conditionally enable/disable instructions
- Memory system
  - Compiler-visible cache hierarchy
  - Serial behavior of parallel reads/writes

Features of HPL-PD

- Branch architecture
  - Architecturally visible separation of fetch and execute of branch target
- Unusual simultaneous write semantics
  - Hardware allows multiple simultaneous writes to registers
- Software loop pipelining support
  - Rotating registers for efficient software pipelining of tight inner loops
  - Branch instructions with loop support (shifting the rotating register window, etc)

Register Files in HPL-PD

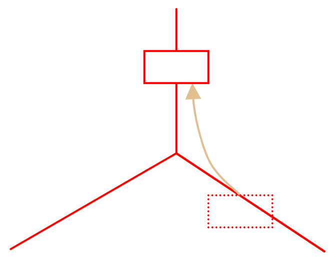
- Register Files are of various types
  - General purpose (GPR), Floating point (FPR), Predicate (PR), Branch target (BTR)
- Each register file may have a static and a rotating portion
  - The ith static register in file F is named Fi
  - The ith rotating register in file F is named F[i].
  - Indexed off the RRB, the rotating register base register.



F [i] =FR [(RRB + i) % size(FR)]

Control Speculation Support

- Control speculation is the execution of instructions that may not have been executed in un-optimized code
  - Generally occurs due to code motion across conditional branches
  - e.g. An instruction in one branch is moved above the conditional jump



- This transformation is generally safe
  - If the effect of the speculative instruction can be ignored or undone if the other branch is taken
  - eg. if a speculative instruction causes an exception, the exception should not be raised if the other branch is taken
  - HPL-PD provides hardware support for this

Speculative Operations

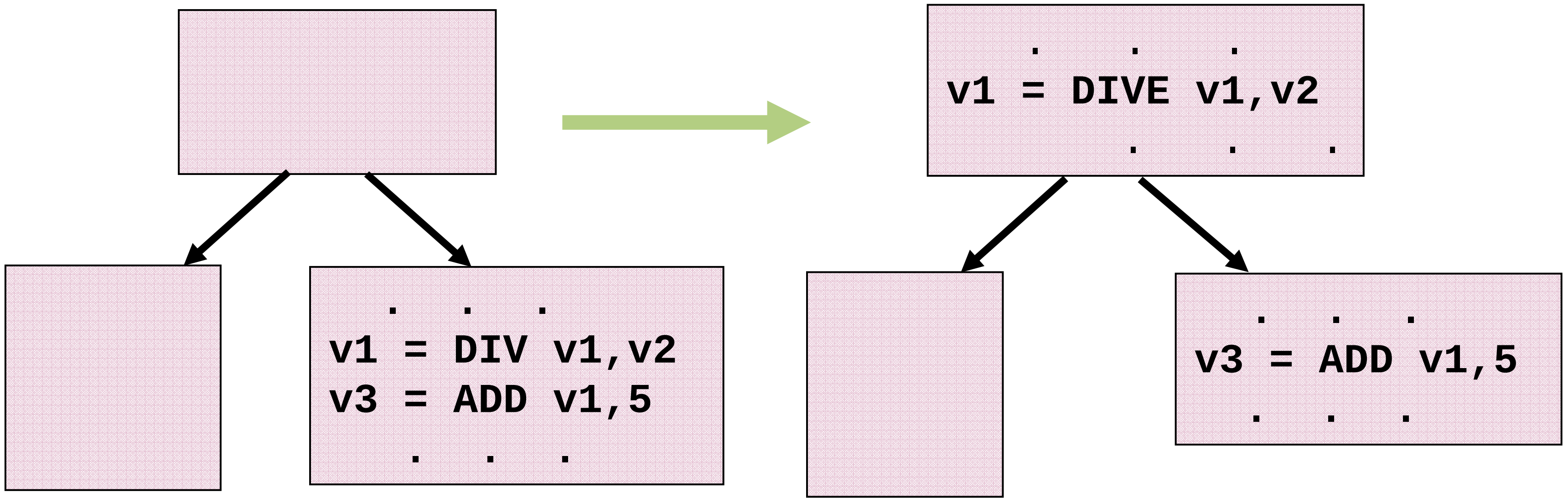
- Speculative operations are written identically to their non-speculative counterparts, but with an “E” appended to the operation name.
  - e.g. DIVE ADDE PBRRE

- If an exceptional condition occurs during a speculative operation, the exception is not raised
  - A bit is set in the result register to indicate that such a condition occurred
  - More information (e.g. type of condition, IP of instruction) is stored
  - Not currently specified how or where

- If a non-speculative operation has an operand with its speculative bit set, an exception is raised

Speculative Operations

- An example



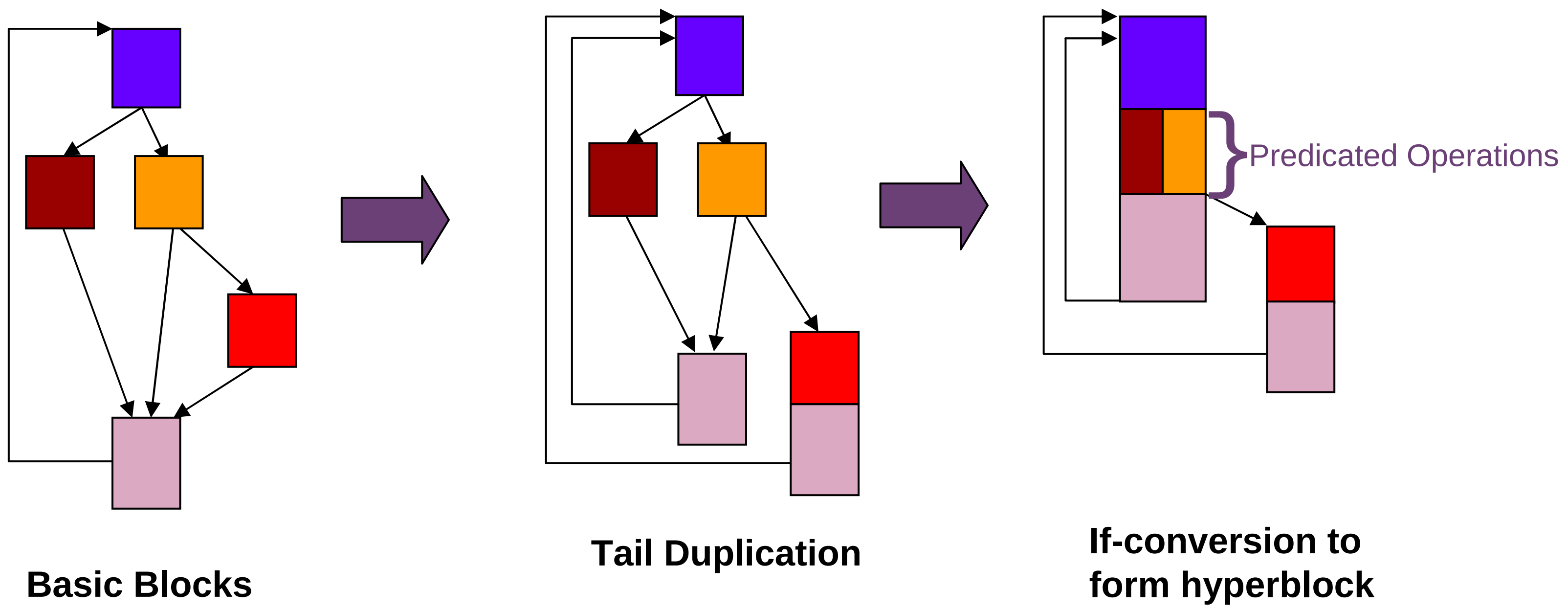
- The effect of the DIV latency is reduced
- If a divide-by-zero occurs, an exception will be raised by ADD

Predication in HPL-PD

- In HPL-PD, most operations can be predicated
  - they can have an extra operand that is a one-bit predicate register.
  - r2 = ADD.W r1, r3 **if p2**
  - If the predicate register contains 0, the operation is not performed
- The values of predicate registers are typically set by "compare-to-predicate" operations
  - p1 = (CMPP.< r4, r5)
- HPL-PD provides two-output CMPP instructions
  - p1,p2 = CMPP.W.<.UN.UC r1,r2
    - U means unconditional, N means normal, C means complement
    - There are other possibilities (conditional, or, and)
- Predication, in its simplest form, has several uses
  - If-conversion
  - To aid code motion by instruction scheduler.
    - e.g. hyperblocks
  - Height reduction of control dependences

Use of Predication: An Example

- In hyperblock formation, if-conversion is used to form larger blocks of operations than the usual basic blocks
  - Tail duplication used to remove some incoming edges in middle of block
  - if-conversion applied after tail duplication
  - larger blocks provide a greater opportunity for code motion to increase ILP



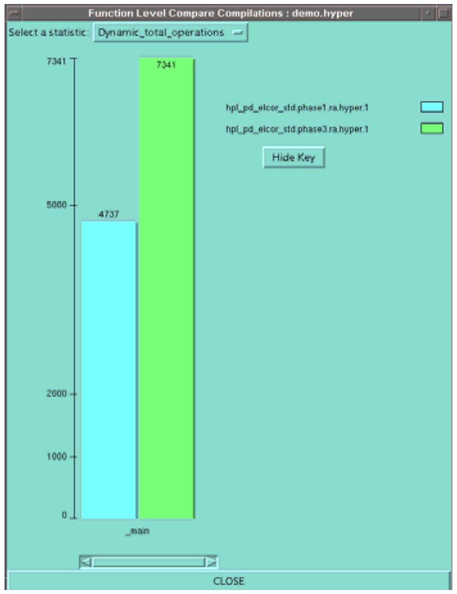
Hyperblock Performance Comparison

- Although the total number of operations executed increases, so does the parallelism

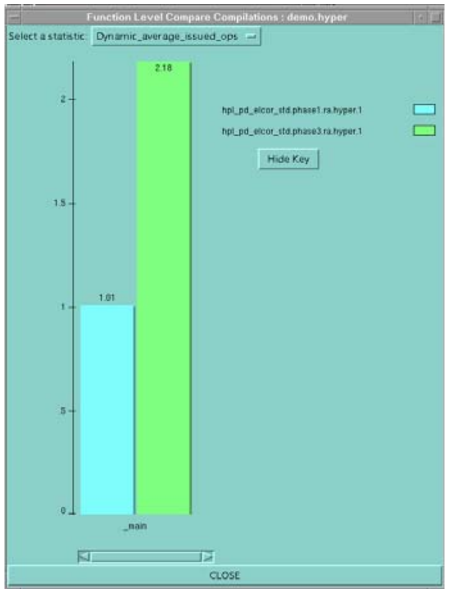
🟦 without hyperblock formation

🟩 with hyperblock formation

Total number of operations executed

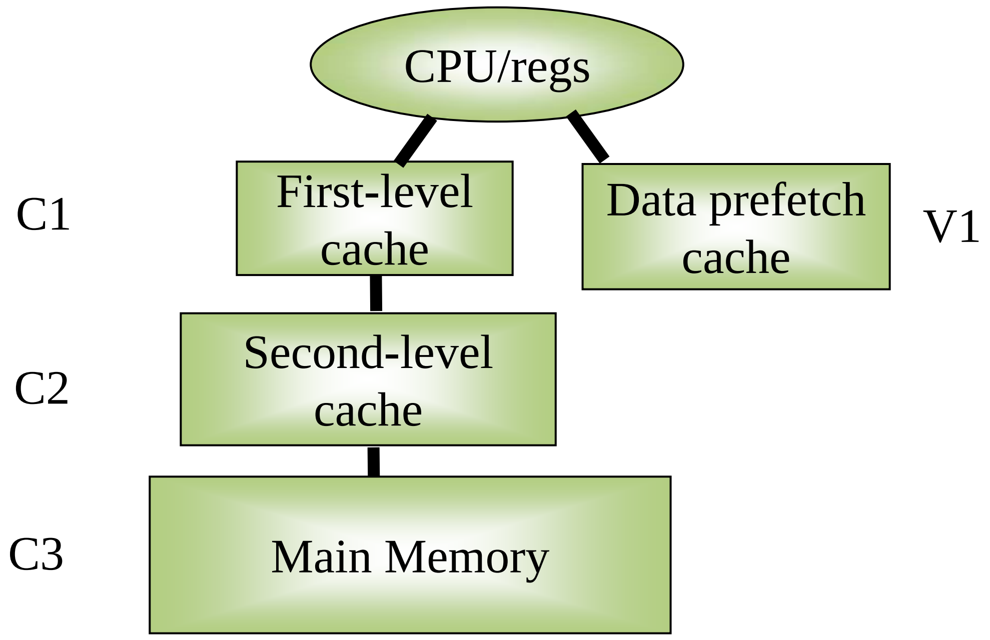


Average number of operations executed per cycle



The HPL-PD Memory Hierarchy

- HPL-PD's memory hierarchy is unusual in that it is visible to the compiler
  - In store instructions, compiler can specify in which cache the data should be placed
  - In load instructions, the compiler can specify in which cache the data is expected to be found and in which cache the data should be left
- This supports static scheduling of load/store operations with reasonable expectations that the assumed latencies will be correct

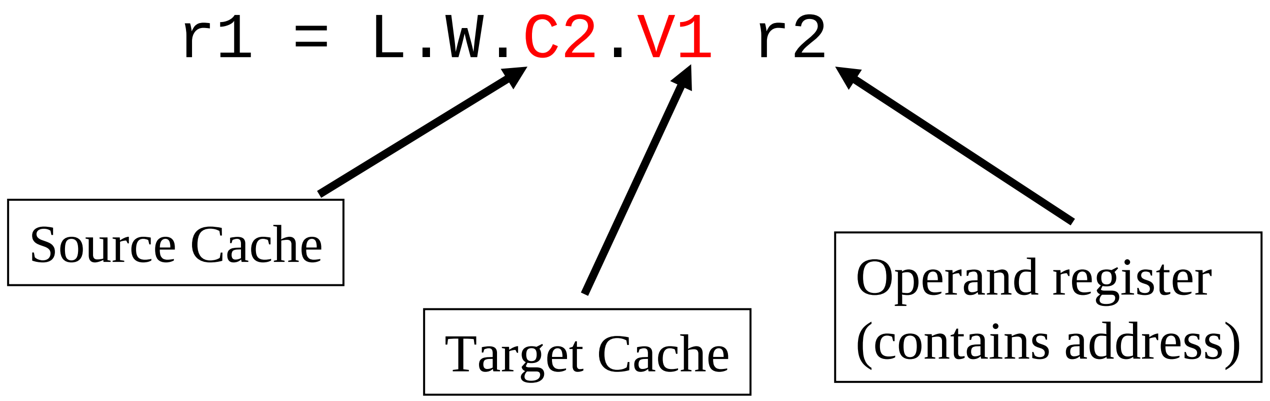


<u>data-prefetch cache</u>

- Independent of the first-level cache
- Used to store large amounts of cache-polluting data
- Doesn't require sophisticated cache-replacement mechanism

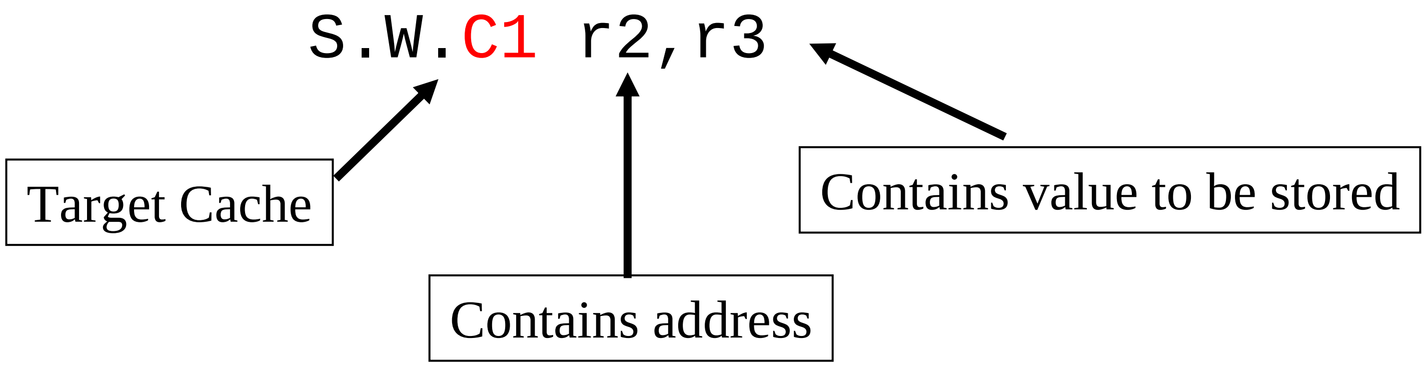
Load/Store Instructions

Sample Load Instruction



```
r1 = L.W.C2.V1 r2
```

Sample Store Instruction



```
S.W.C1 r2,r3
```

Run-time Memory Disambiguation

- Here’s a desirable optimization (due to long load latencies):

```
         . . .
              S r3, 4
r1  =    L r2
r1  =    ADD r1,7
```

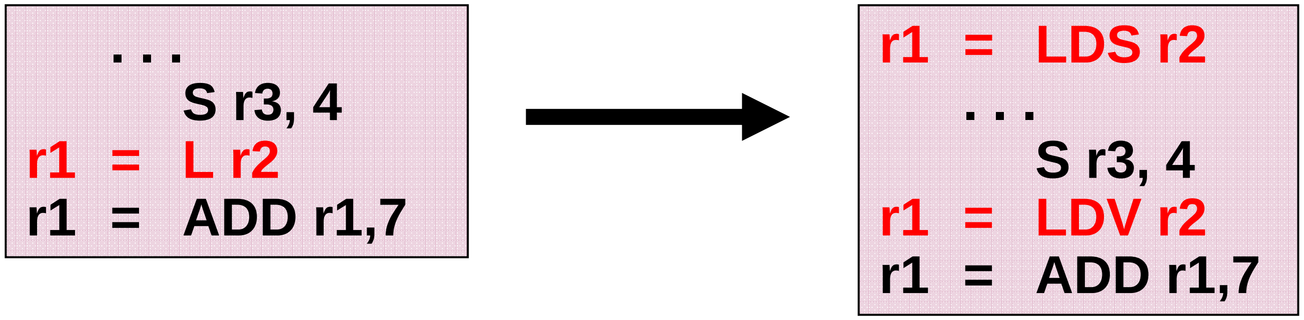
$\longrightarrow$

```
r1  =   L r2
       . . .
           S r3, 4
 r1 =   ADD r1,7
```

- However, this optimization is not valid if the load and store reference the same location
  - i.e. if r2 and r3 contain the same address.
  - This cannot be determined at compile time
- HPL-PD solves this by providing run-time memory disambiguation

Run-time Memory Disambiguation

- HPL-PD provides two special instructions that can replace a single load instruction:
- r1 = LDS r2 ; speculative load
  - initiates a load like a normal load instruction
    - A log entry is made in a table to store the memory location
- r1 = LDV r2 ; load verify
  - checks to see if a store to the memory location has occurred since the LDS
  - if so, the new load is issued and the pipeline stalls
    - Otherwise, it's a no-op



```
      . . .
         S r3, 4
r1  =    L r2
r1  =    ADD r1,7
```

```
r1  =    LDS r2
      . . .
         S r3, 4
r1  =    LDV r2
r1  =    ADD r1,7
```

The HPL-PD Branch Architecture

- HPL-PD replaces conventional branch operations with two operations

- Prepare-to-Branch operations (PBRR, etc)
  - Loads target address into a branch target register
  - Initiates prefetch of the branch target instruction to minimize branch delay
  - Contains field specifying whether the branch is likely to be taken
  - Must precede any branch instruction

- Branch operations (BRU, etc)
  - Branches to address contained in a branch target register
  - There are branch instructions for function calls, loops, software pipelining, and conditional branch related to memory disambiguation

Software Pipelining Support

- Software Pipelining is a technique for exploiting parallelism across iterations of a loop
  - Iterations are overlaid

- HPL-PD's rotating registers support a form of software pipelining called Modulo Scheduling
  - Rotating registers provide automatic register renaming across iterations
  - The rotating base register, RRB, is decremented by the BRLC instruction.
  - Thus, r[i] in one iteration is referenced as r[i+1] in the next iteration

Modulo Scheduling Example

- Initial C code

```
for(i = 0; i < N; i++)
  a[i] += M;
```

- Non-pipelined code (r and s are GPR registers)

```
   LC =  MOV N-1
   s  =  MOV a
   b1 =  PBRR Loop,1
Loop:
   r  =  L s
   r  =  ADD r,M
         S s,r
   s  =  Add s,4
         BRLC b1
```

Modulo Scheduling Example

- We can overlay the iterations…..

```
r = L s
----          L r,s
r = Add r,M   ----           L r,s
    S s,r   r = Add r,M      ----          L r,s
s = Add s,4     S s,r      r = Add r,M     ----         L r,s
            s = Add s,4         S s,r    r = Add r,M    ----
                           s = Add s,4       S s,r    r4 = Add r,M
                                         s = Add s,4      S s,r
                                                      s = Add s,4
```

- ….and take a slice to be executed as a single EPIC instruction:

```
s[0] =    Add s[1],4        ; increment i
S s[4],r[3]                 ; store a[i-3]
r[2] =    Add r[2],M        ; a[i-2]= a[i-2]+M
-----
r[0] =    L s[1]            ; load a[i]
```

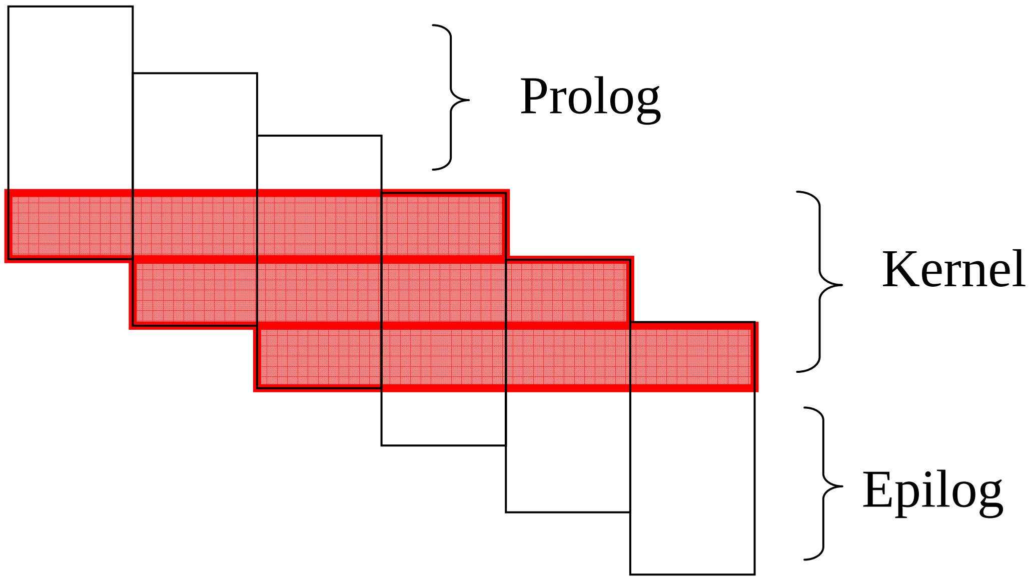
Modulo Scheduling

- With rotating registers, we can overlay iterations of the loop.
  - e.g. r[j] in one iteration was r[j-1] in the previous iteration, r[j-2] in the iteration before that, and so on
  - thus a single EPIC instruction could conceivably contain an operation from each of the n previous iterations
    - where n is the size of the rotating portion of a register file

- Loop Prolog and Epilog

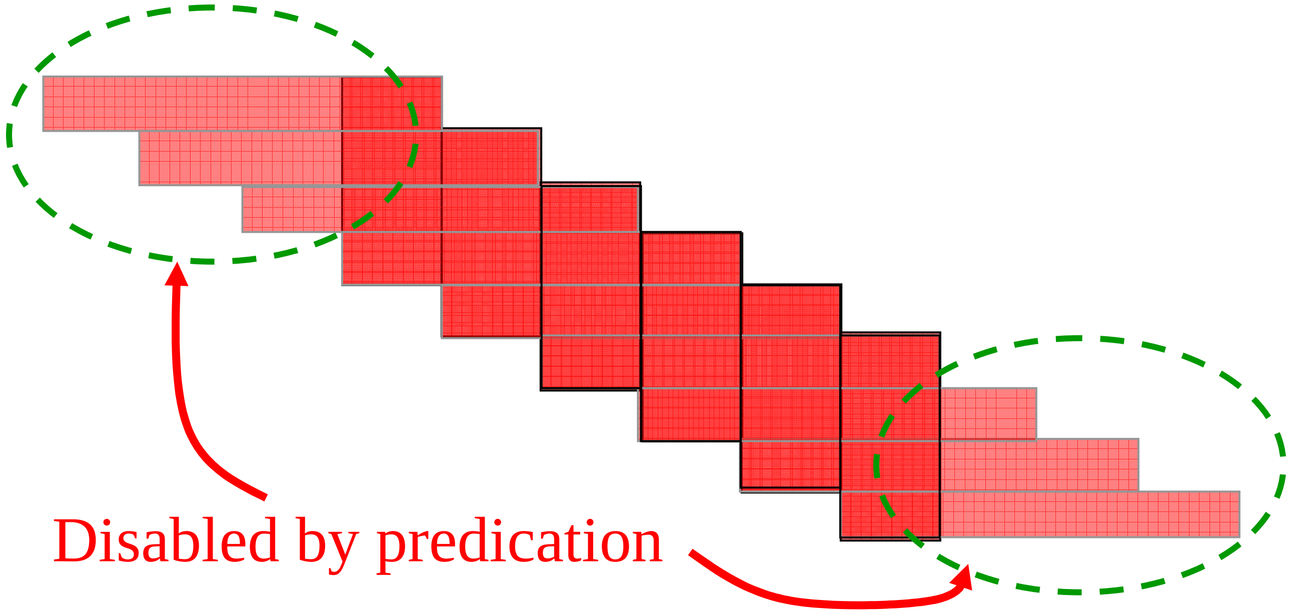
Only the shaded part, the loop kernel, involves executing the full width of the EPIC instruction.

- The loop prolog and epilog contain only a subset of the instructions
- “ramp up” and “ramp down” of the parallelism



Modulo Scheduling With Predication

- We can also view the overlay of iterations as



- Where the loop kernel is executed in every iteration, but with the undesired instructions disabled by predication
  - Supported by rotating predicate registers

Modulo Scheduling Performance (Matmult)

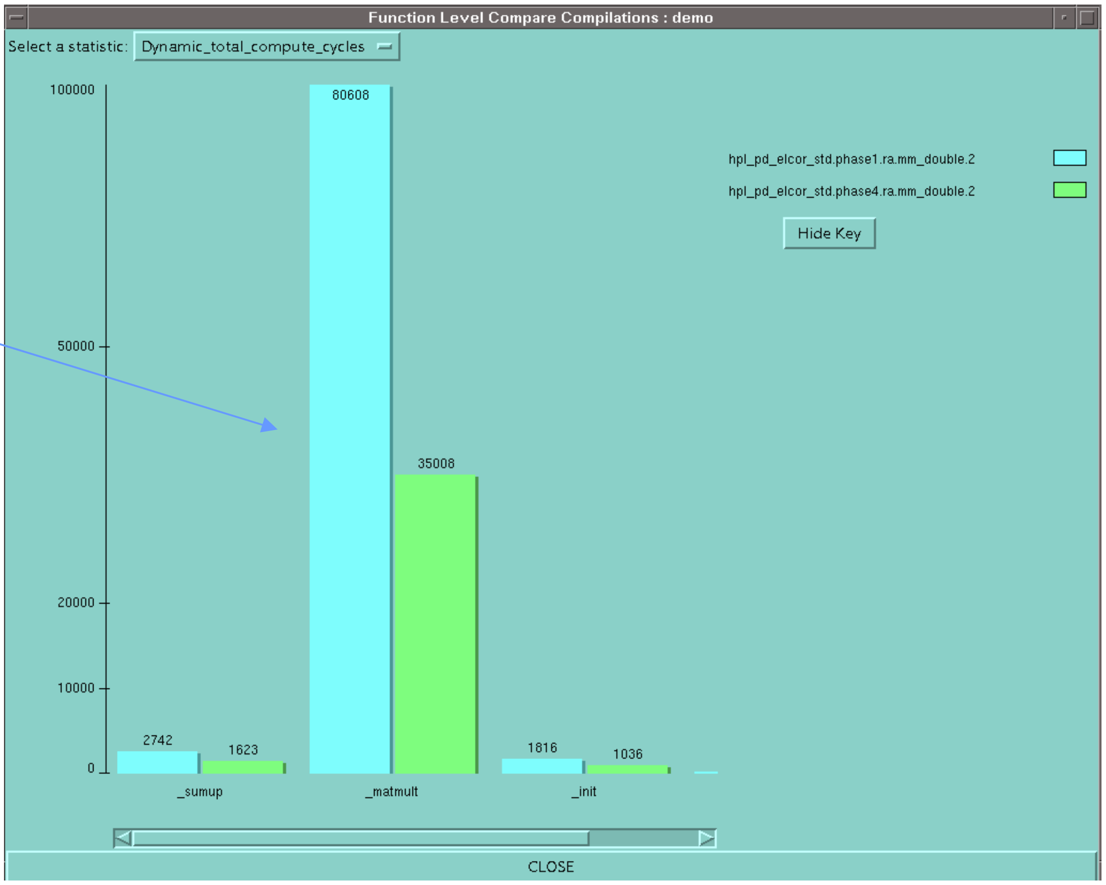
- Speedup of 2.2 due to Modulo scheduling

Total number of cycles

Matmult

🟦 Without modulo scheduling 80608 cycles

🟩 With modulo scheduling 35008 cycles



Summary

- HPL-PD is a flexible ILP architecture
    - Encompassing both superscalar and EPIC machine classes
- HPL-PD is a very interesting target for compiler optimizations
    - Many useful, novel features
    - Increased opportunities for instruction scheduling
        - Predication, speculative instructions
    - Other optimizations

The Elcor Intermediate Representation

Factors Motivating the Design

- Global scheduling is key to exploiting ILP
  - We are moving towards bigger and complex regions
- Frequency-based regions have more complex structure than traditional structure-based regions
  - Even a trace is multiple-entry multiple-exit region
- Many of the ILP enhancing techniques, e.g., height reduction, rely on estimates of height and resource usage
  - Such estimates may be helpful even in earlier phases
- Analysis like memory disambiguation are expensive
  - Need to represent and maintain their results accurately

Factors motivating the design

- Flexibility in phase ordering
  - Because we don't fully understand the right phase order
- Flexibility and ability to grow
  - In many cases, we don't fully understand the requirements
  - IR highly optimized for a specific purpose may not be the right one
  - Put general mechanism to support various policies
  - Well defined interfaces to modules and encapsulation
- Uniformity
  - Easy to build software, modify and grow

IR Features

- Multi-state IR
- Provides mechanism for representing
  - Traditional control flow graph
  - Control dependences
  - Data dependences for both registers and memory in various forms
  - Various forms of register usage – single assignment, multiple assignments
  - Expanded virtual registers (EVRs)
  - Predicated execution
- Data section
  - Global symbols, arrays, etc.
- Registers carry values, edges represent dependences
- A uniform, edge-based representation of control flow and data dependences
- Supports threading of data dependences
  - dependence flow graphs
- Hierarchical non-overlapping region structure (a tree)

Internal vs. Textual Representation

- Each component of the graph data structure is a C++ object
  - All modules of the Elcor use this IR
  - Optimization are simply IR-to-IR transformations
- There is an ASCII intermediate representation, called Rebel
  - Phases of Elcor may communicate using Rebel
  - A reader procedure is provided that reads Rebel and constructs the corresponding internal program representation
  - A writer procedure is provided for generating Rebel from the internal representation

Program Representation

- A program unit is represented by a graph of operations connected by edges
    - Control flow is represented explicitly and at the operation level
- A region structure over the operation graph (a tree)
    - The root of the tree is the program unit, e.g. a procedure
    - The leaf nodes of the tree are operations
- Operation graph elements
    - Op(eration) class
    - Operand class
    - Edge class

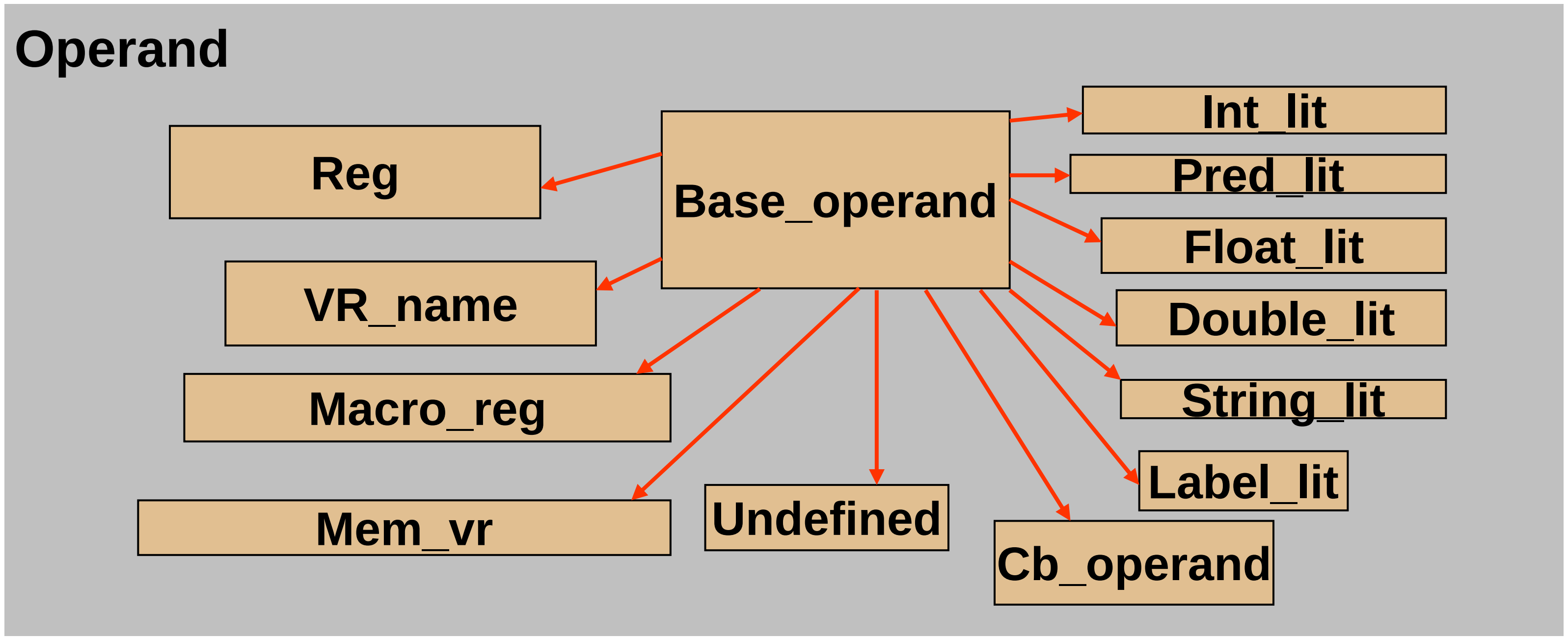
Op class

- Represents an operation
  - Machine operation
  - Compiler operations (e.g.,CONTROL_MERGE, PRED_CLEAR)
- Has source and destination operands including guarding predicate (their number is determined by MDES)
  - dest1, ..., destm = opcode(src1, ..., srcn) if p
- May have implicit sources and destinations
  - e.g., parameter passing registers for BRL
- Memory dependence "sources" and "destinations"
  - Memory dependences are encoded as "def" and "use" of special variables

```
<$a> r3 = load (r4)
 store(r1, r2) <$a, $b, ...>
```

  - Simplifies dependence graph construction
- Set of input edges and set of output edges
- Schedule time, latency queries for sources/destinations

Operand Class Hierarchy



Operand class is a wrapper for all operand types.

- Provides Boolean methods for class type testing
- Provides access methods to class specific fields
- Provides comparison operators
- Manages symbol table

Extended Virtual Registers

- EVRs allow multiple values from a sequence of assignments to be live at the same time
- An EVR is a linearly ordered set of VRs
  - Elements are referenced using the notation t[0], t[1], etc.
  - A special remap operation to "shift" reference coordinates

```
t = 0;     // t means t[0]
remap(t); // Previous
  value
           // of t is now
  t[1]
t = 1;
remap (t);
t = t[1] + t[2] // t = 0
  + 1
```

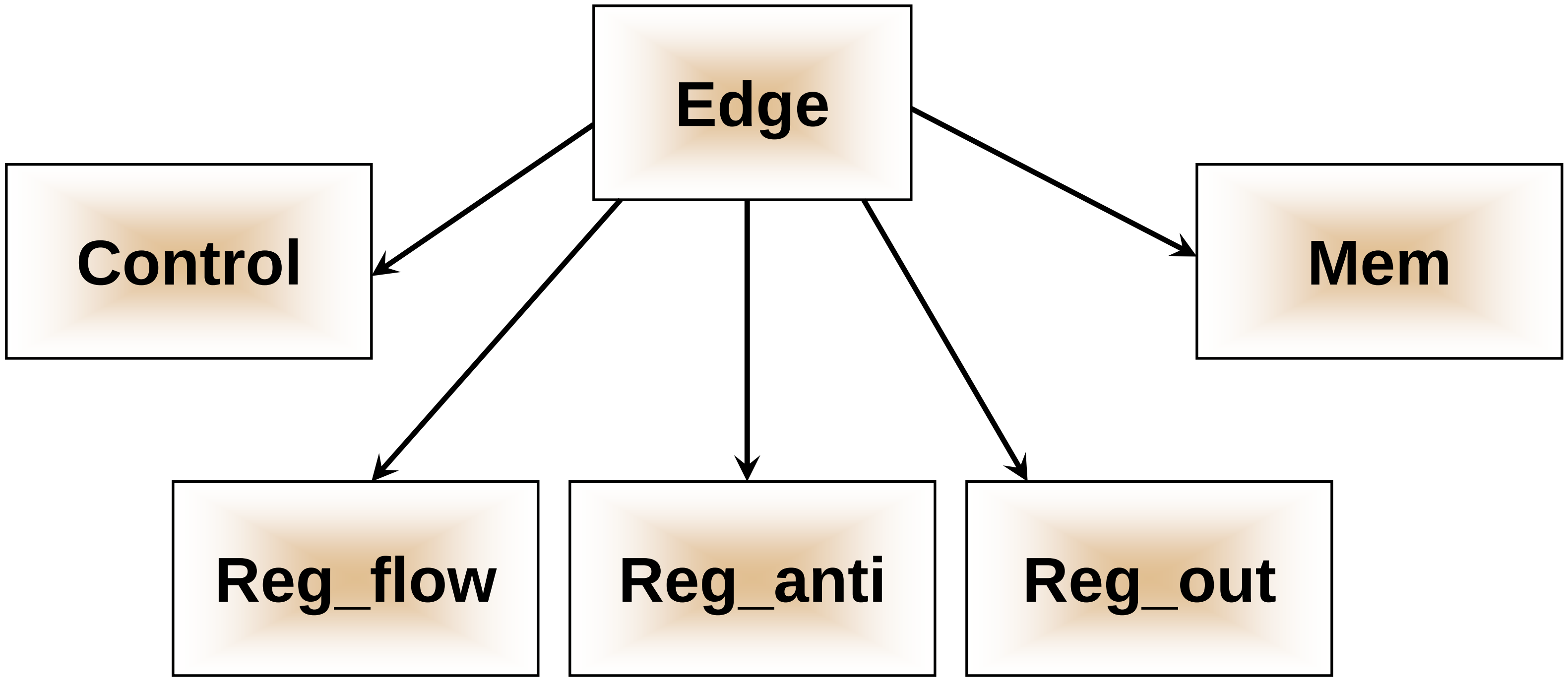
- EVRs allow
  - Accurate representation of value flow across zero or more iterations of a loop
  - Representation of results of analysis and transformation without unrolling or unnecessary copies
    E.g., The use of the value loaded in previous previous iteration as t[2]
  - Representation in dynamic single assignment form to eliminate inter-iteration anti- and output dependences
- Use of EVRs in IR doesn't imply use of rotating registers in hardware
  - Code can be unrolled at a later stage if rotating registers are not supported

Edge Class

- Edges Represents dependence constraints between operations
- Edges do not represent value-flow like data flow graphs
- Edge types:
    - Control (sequential control flow, control dependence)
    - Flow, anti and output dependences on registers
    - Flow, anti and output memory dependences classified as "certain" or "maybe"
- An edge has pointers to source and destination ops
- An edge also contains more detailed reason for dependence
    - Represented in terms of "Port" for source and destination operands
        - e.g., register flow edge from DEST1 of op1 to SRC2 of op2
- Latency setting and querying functions

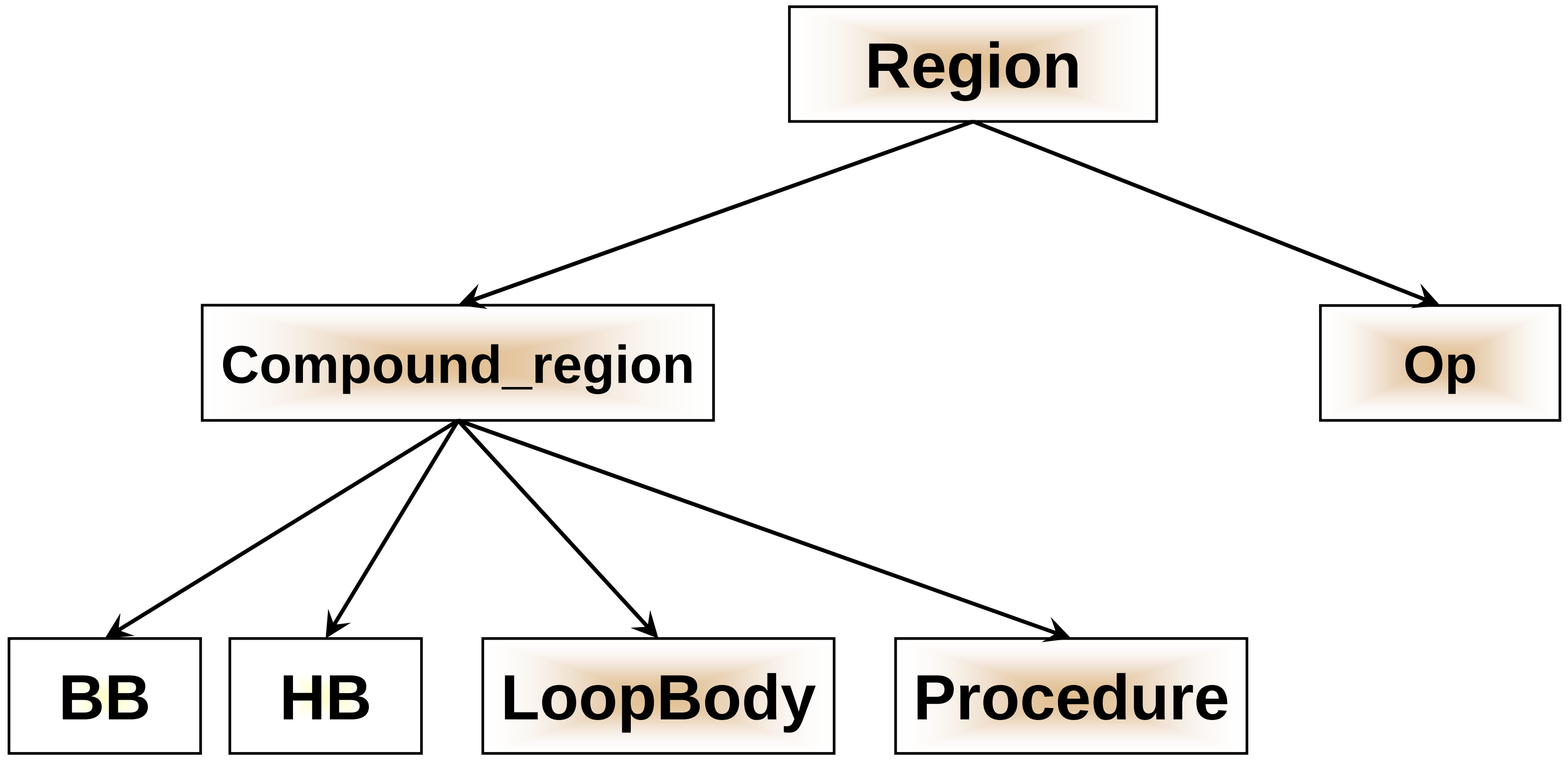
Edge Class Hierarchy

- The hierarchy is based on how latency for an edge is computed



Region Class Hierarchy

- Region class is an abstract base class
- Compound regions can contain other regions in the region tree



IR Attributes

- The intermediate representation allows annotations on Regions and Edges
  - Used for module specific purposes
  - Used when the information is sparse

- There are two kinds of attributes
  - Heavy weight
    - Type safe
    - Can be represented in ASCII form of IR (can be printed and parsed in)
    - If the object it is attached to is deleted the attributes are deleted
  - Light weight
    - Stored and retrieved using string keys
    - Not type safe

Using the IR iterators

Elcor provides a collection of iterators to walk data structures

```
void check_region_hierarchy(Region* r)
{
   // Iterator over subregions
   Region_subregions subreg_iter;

   if (r->is_op()) return;
   Compound_region* cr = (Compound_region*) r;
   for(subreg_iter(cr) ; subreg_iter!=0 ; subreg_iter++) {
        Region* current_subregion = (*subreg_iter);
        assert(current_subregion->parent() == r);
        check_region_hierarchy(current_subregion);
   }
}
```

Initialize iterator

We aren’t done, are we?

Move to next

Current item, please

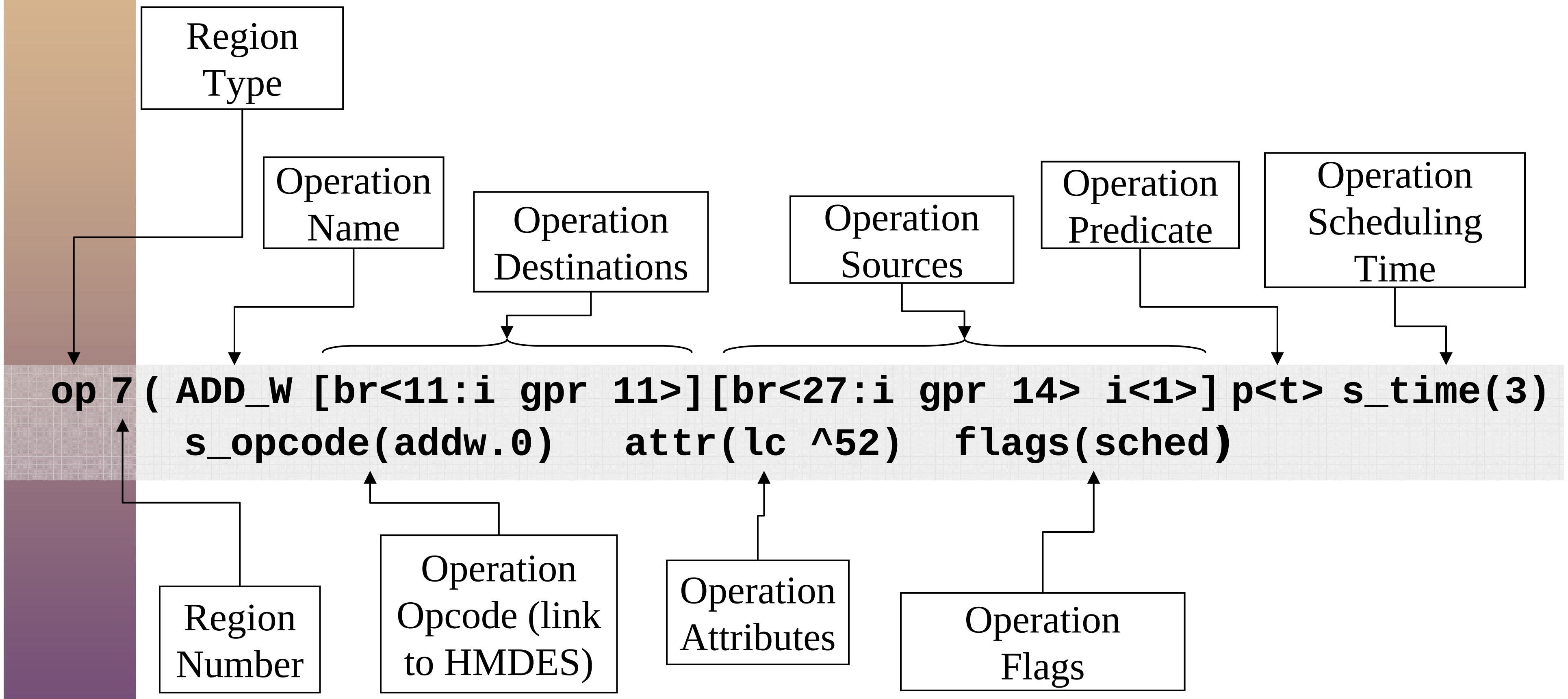
Textual Representation of the IR

- Rebel is the ASCII representation of the IR
- It is human-readable
  - Can be parsed by a recursive descent parser

- It has the same structure and elements as the data structures of IR
  - Region based
  - Sufficiently powerful to express program properties at various stages of compilation
    - Before/after scheduling
    - Before/after register allocation

Operation in Rebel

Here is how an operation region looks in Rebel



```
op 7 ( ADD_W [br<11:i gpr 11>][br<27:i gpr 14> i<1>] p<t> s_time(3)
    s_opcode(addw.0)   attr(lc ^52)   flags(sched)
```

Summary

- Elcor Intermediate Representation is
  - Graph based with explicit representation of dependence and control flow
  - Region based

- There are two forms of the intermediate representation that a researcher can use.
  - Internal representation
    - C++ object based
    - Used by all Elcor modules
  - Textual representation (Rebel)
    - Complete program representation
    - Easily parsed, readable

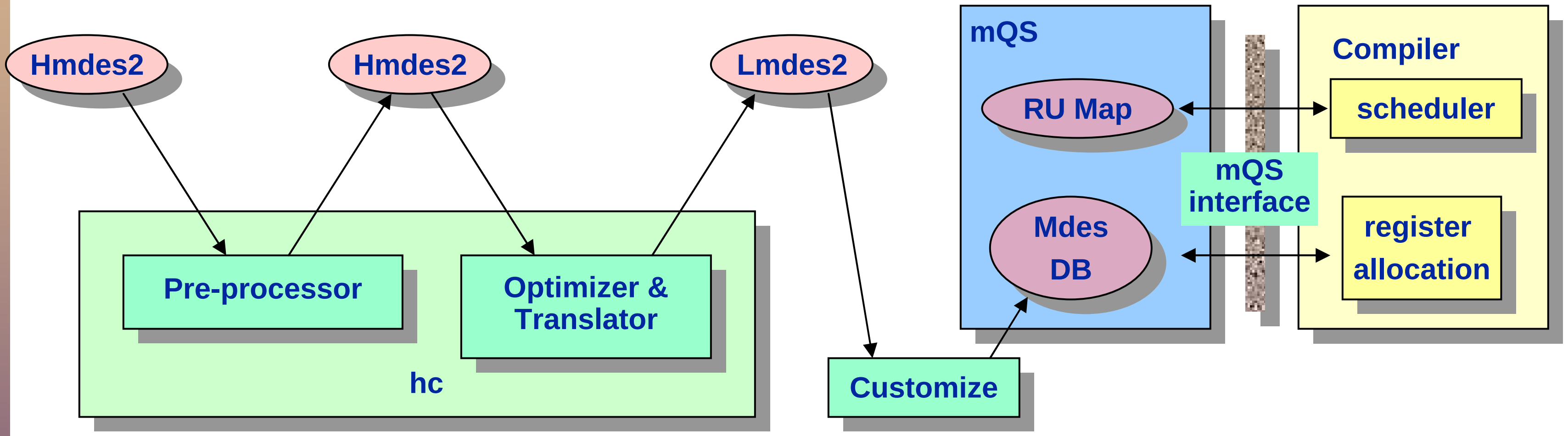
Trimaran
Machine Description System

Target Machine Description

- Trimaran includes an advanced Machine Description facility, called Mdes, for describing a wide range of ILP architectures
- A high-level language, Hmdes2, for specifying machine features precisely
  - functional units, register files, instruction set, instruction latencies, etc.
- Human writable and readable
- A translator converting Hmdes2 to Lmdes2, an optimized low-level machine representation
- A query system, mQS, used to configure the compiler and simulator based on the specified machine features

Mdes Overview

- The goal: to minimize the number of assumptions built into the compiler back-end regarding the target machine



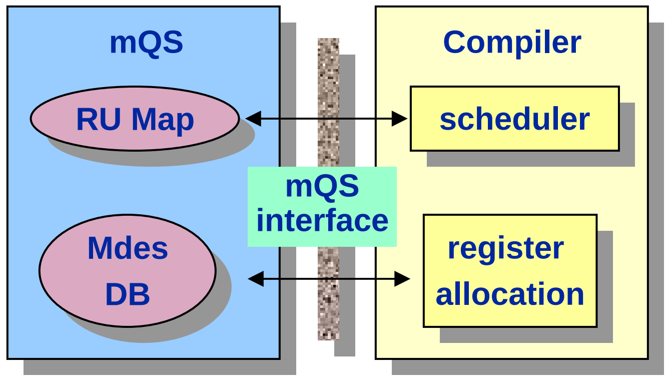
- The processor types have been served by mdes thus far
  - Cydrome: Cydra 5 (VLIW, complex, non-parametric)
  - HPL-PD (VLIW, simplified, parametric)
  - IMPACT: products (superscalar, complex, non-parametric)

Information Required By The Compiler

- For ILP code selection
  - **I/O descriptor**: source / destination register file constraints for operations
  - **Register file**: the set of compatible register types
- For edge drawing
  - **Register**: overlapping registers, i.e., that have at least one bit in common
- For edge delays
  - **Latency descriptor**: source sampling / result update times for operations
- To determine legality of scheduling at a given time with respect to resource conflicts
  - **Reservation table**: resource usage over time for each operation
- For lifetime calculation
  - **Latency descriptor**: register allocation and de-allocation times
- To determine register allocation options
  - **Register file**: the set of legal registers for allocation

The Compiler/Machine Description Interface

- The interface between the compiler and the machine description is the mdes Query System, mQS
  - New modules implemented by researchers will need to use the mQS
- The compiler queries mQS via a set of C++ procedures
  - Each class of machine feature corresponds to a separate C++ procedure



Summary

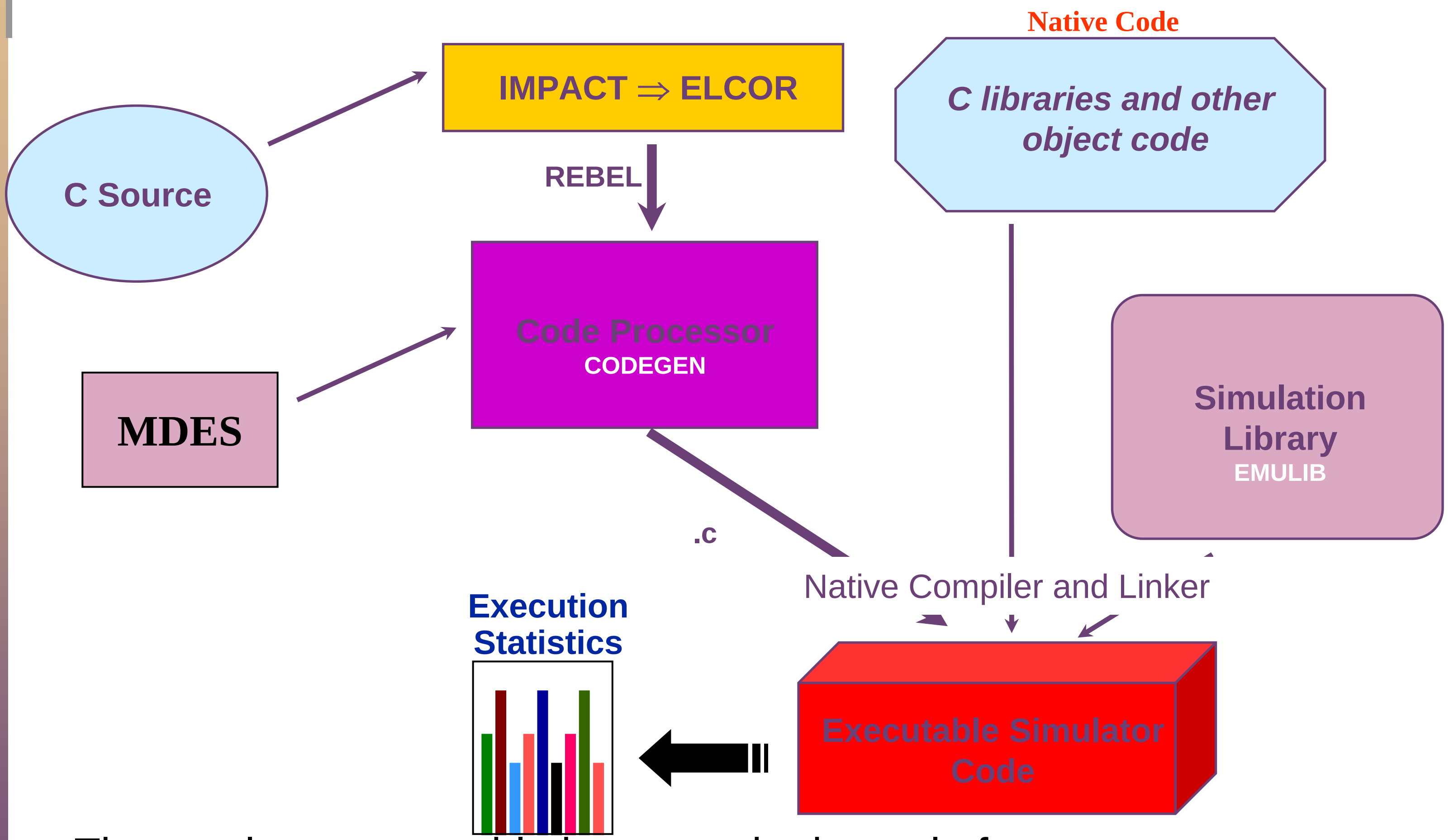
- Reconfiguring the target machine is quite easy
  - GUI speeds up the process substantially for modest changes.
  - Extensive changes can be made using Hmdes2
    - there are plenty of sample Hmdes2 files to look at
- Adding new machine-dependent compilation modules is also quite easy
  - mQS provides a clean interface between the compiler and the machine description

Simulator Support in Trimaran

The Framework

- The goals of the HPL-PD simulation framework are
  - Emulate the execution of the generated REBEL code on a virtual HPL-PD processor
  - Have the ability to adapt to changes in the machine description
  - Generate *accurate* run-time information
    - Execution clock cycles
    - Dynamic control flow and call trace
    - Address trace
    - Average number of operations executed per cycle

Simulator: Overview



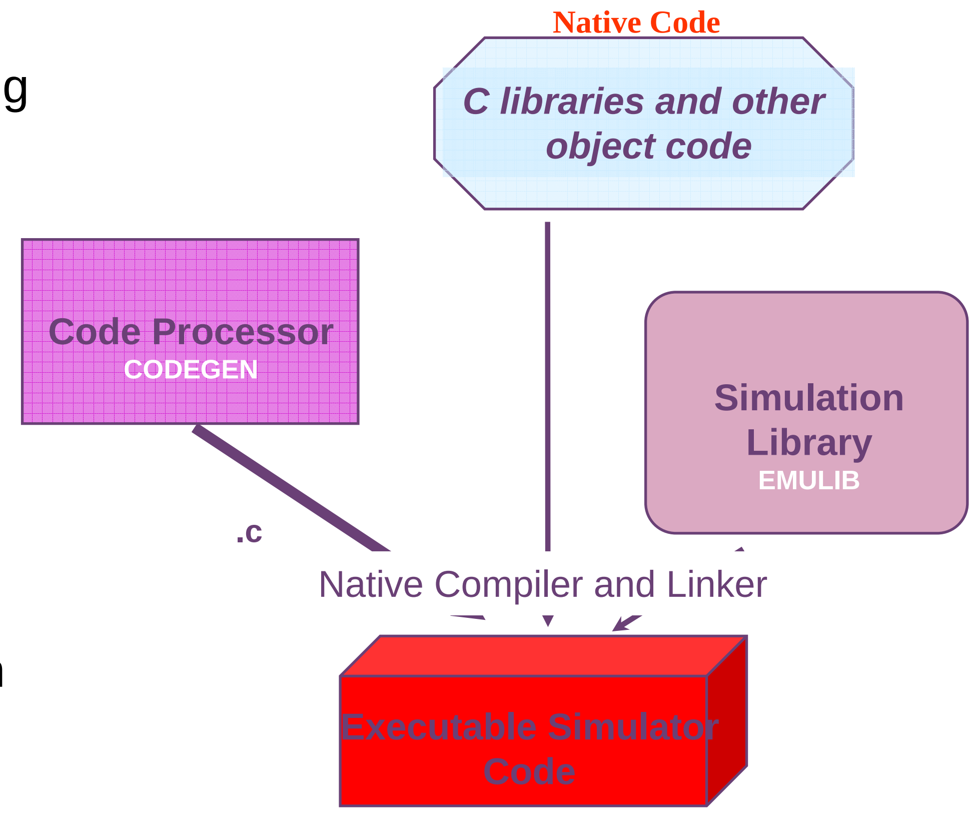
- The resultant executable is run on the host platform to generate statistics and dynamic profile information

Simulator: Codegen

- Codegen
  - Input: REBEL code that is generated by Elcor
  - Output (one for each function of the benchmark):
    - Benchmark.simu.c file wrapper functions for emulating the assembly code
    - Benchmark.simu.c.tbls file for declaring the tables of assembly operations
    - Benchmark.simu.c.inc file contains global declarations and the statistics tracking data structures
- The .tbls and .inc files are #included into the .c file.

Simulator: EMULIB

- Collection of files corresponding to operation types:
  - PD_load_store_ops.c
  - PD_move_ops.c
  - PD_int_arith_ops.c
  - Etc.
- These files contain a function for each variation of the HPL-PD operations
- These functions emulate the operation during the simulation
- Other files support the register files, function call mechanics, statistics collection, etc.


Native Code
C libraries and other object code
Code Processor
CODEGEN
Simulation Library
EMULIB
.c
Native Compiler and Linker
Executable Simulator Code

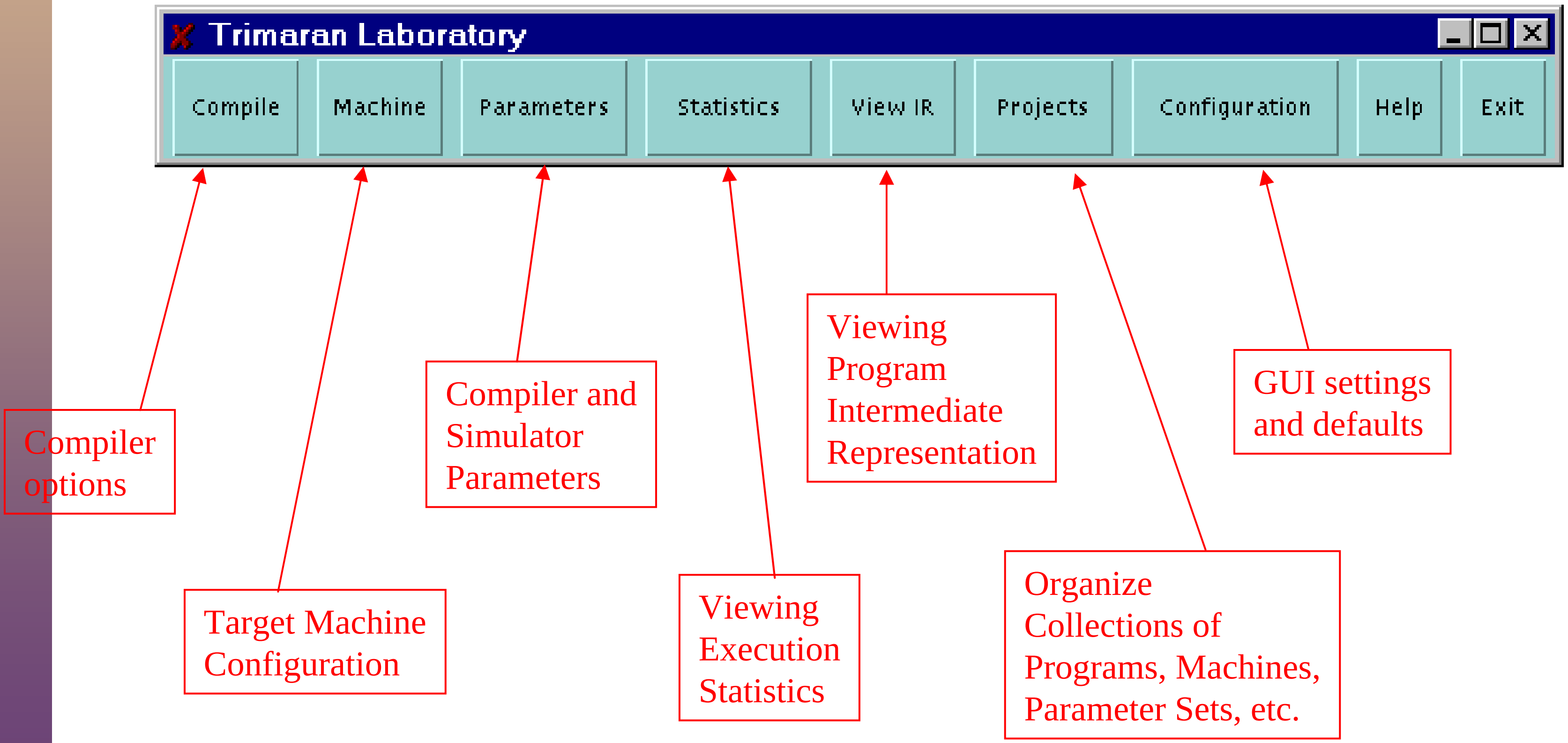
Customizing and Running the Trimaran System

The Trimaran GUI

- The Trimaran system is configured and run via a Graphical User Interface
  - Choose program to compile
  - Configure target machine
  - Configure compilation stages
  - View graphical program representations at various stages of compilation
  - View execution statistics (graphs, pie charts, etc.)
  - View extensive on-line help and documentation
- If desired, the system can also be run from the command line and be invoked from shell scripts

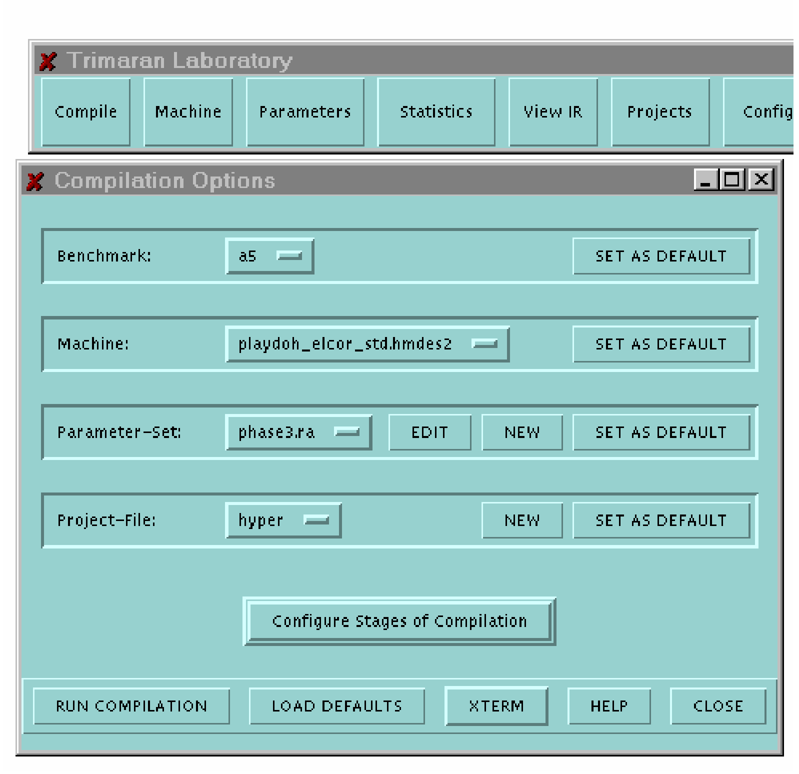
The Control Panel

- The GUI is operated from this main control panel.


Trimaran Laboratory
Compile
Machine
Parameters
Statistics
View IR
Projects
Configuration
Help
Exit
Compiler options
Target Machine Configuration
Compiler and Simulator Parameters
Viewing Execution Statistics
Viewing Program Intermediate Representation
Organize Collections of Programs, Machines, Parameter Sets, etc.
GUI settings and defaults

The Compiler Panel

- The compiler panel allows you to choose
  - Benchmark program to compile
    - you can add your own as well.
  - Target machine configuration
  - Parameter set (for the compiler and simulator)
  - Project file



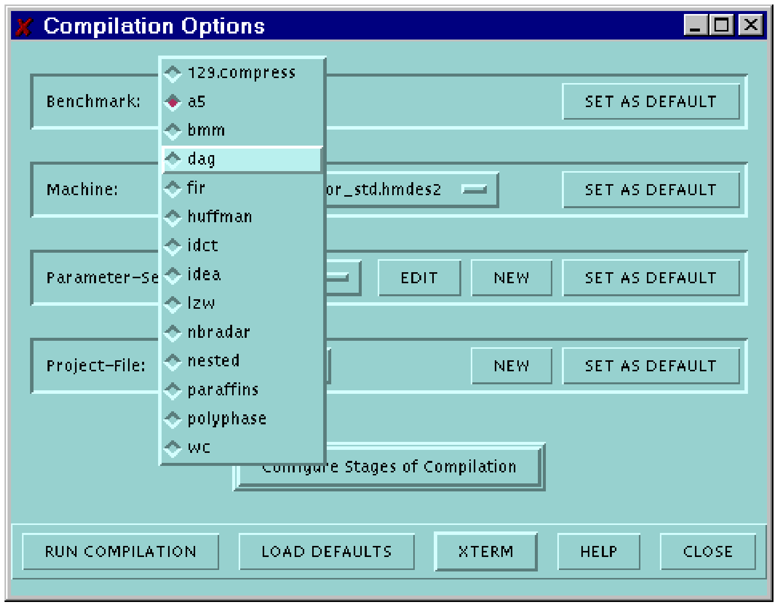
Benchmark: a5

Machine: playdoh_elcor_std.hmdes2

Parameter-Set: phase3.ra

Project-File: hyper

Choosing a Benchmark and Machine



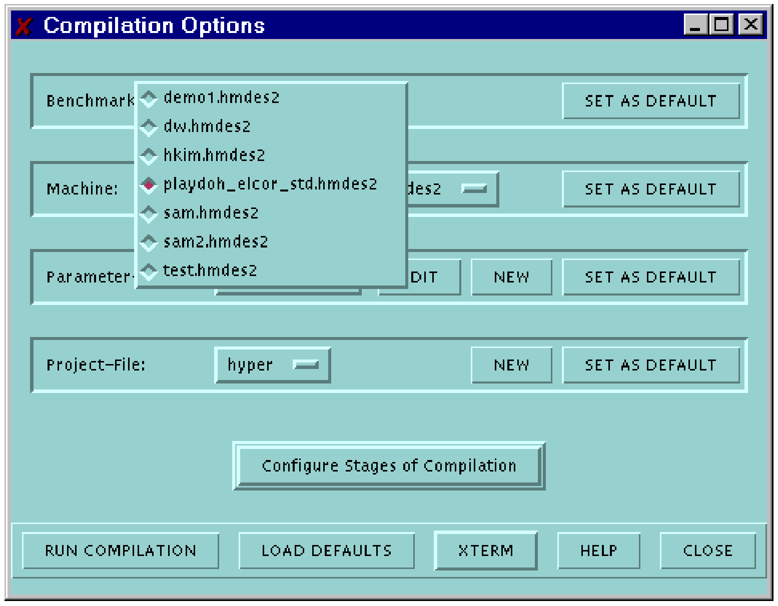
Compilation Options

Benchmark: a5

Machine: …or_std.hmdes2

Parameter-Se…

Project-File:



Compilation Options

Benchmark

Machine: playdoh_elcor_std.hmdes2

Parameter-

Project-File: hyper

Choosing a Benchmark

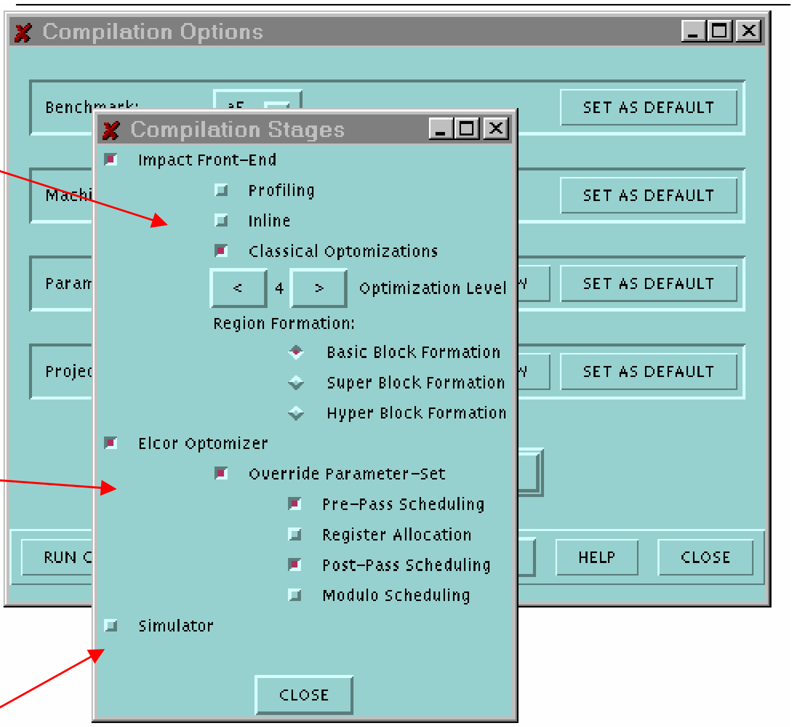
Choosing a Machine

Configuring the Compiler

Front end features

Back end features

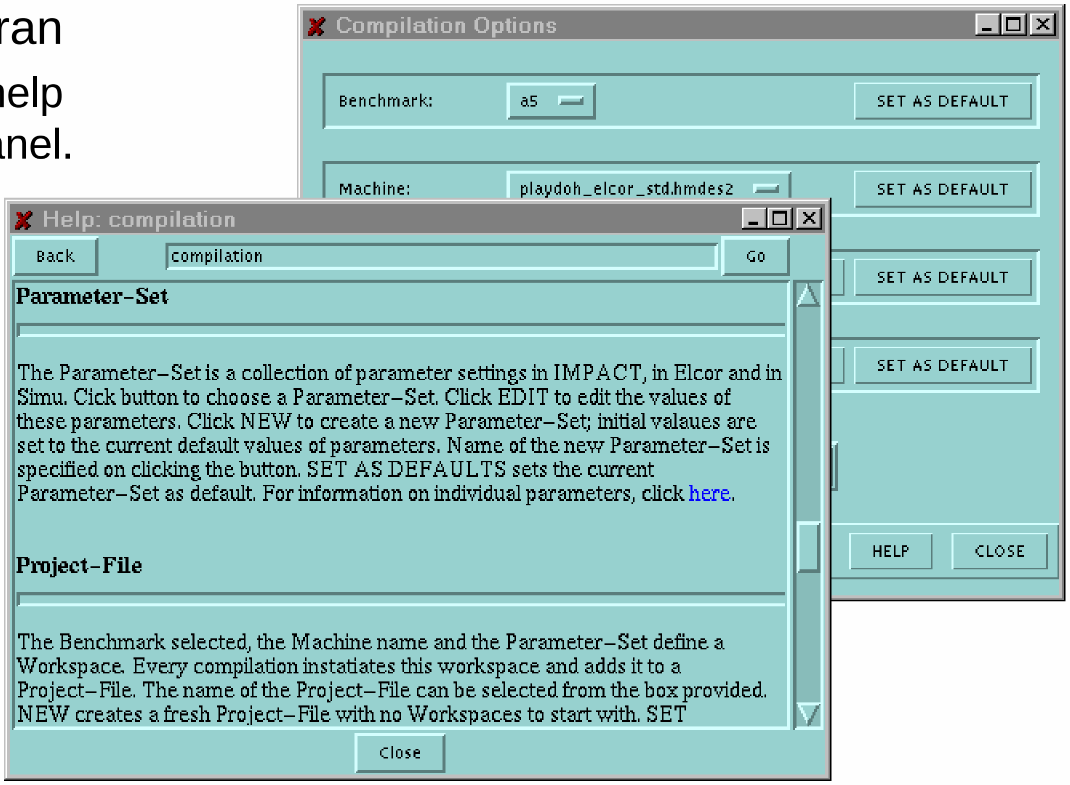
Simulator on/off



- [✓] Impact Front-End
  - [ ] Profiling
  - [ ] Inline
  - [✓] Classical Optomizations
  - < 4 > Optimization Level
  - Region Formation:
    - [✓ Basic Block Formation]
    - Super Block Formation
    - Hyper Block Formation
- [✓] Elcor Optomizer
  - [✓] Override Parameter-Set
    - [✓] Pre-Pass Scheduling
    - [ ] Register Allocation
    - [✓] Post-Pass Scheduling
    - [ ] Modulo Scheduling
- [ ] Simulator

On-line Documentation

- On-line documentation is available for each component of Trimaran
  - this is the on-line help for the compiler panel.



Parameter–Set

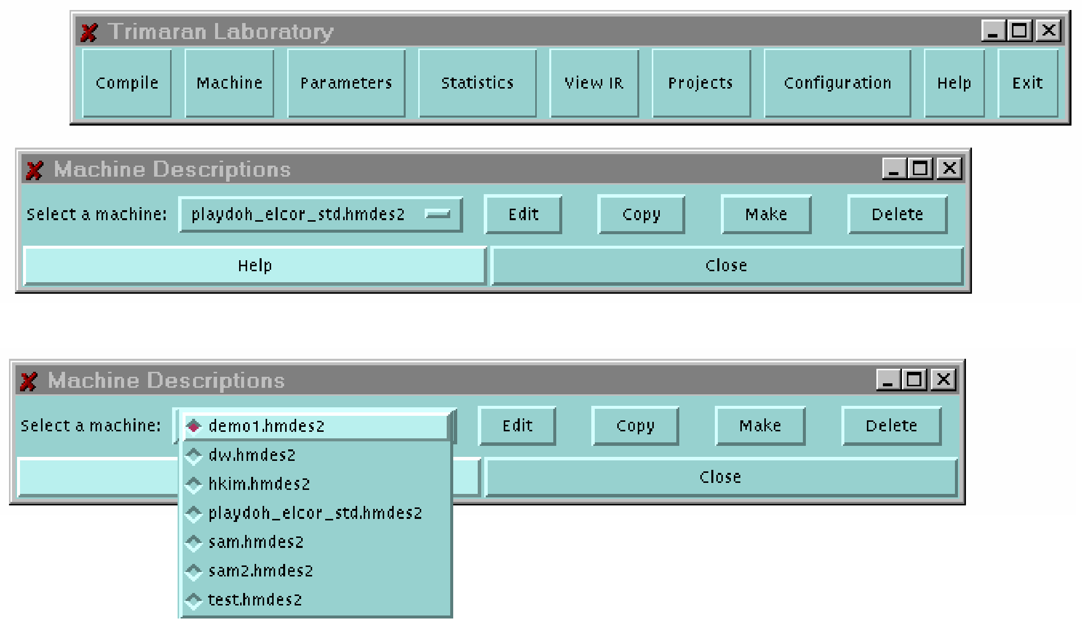
The Parameter–Set is a collection of parameter settings in IMPACT, in Elcor and in Simu. Click button to choose a Parameter–Set. Click EDIT to edit the values of these parameters. Click NEW to create a new Parameter–Set; initial valaues are set to the current default values of parameters. Name of the new Parameter–Set is specified on clicking the button. SET AS DEFAULTS sets the current Parameter–Set as default. For information on individual parameters, click here.

Project–File

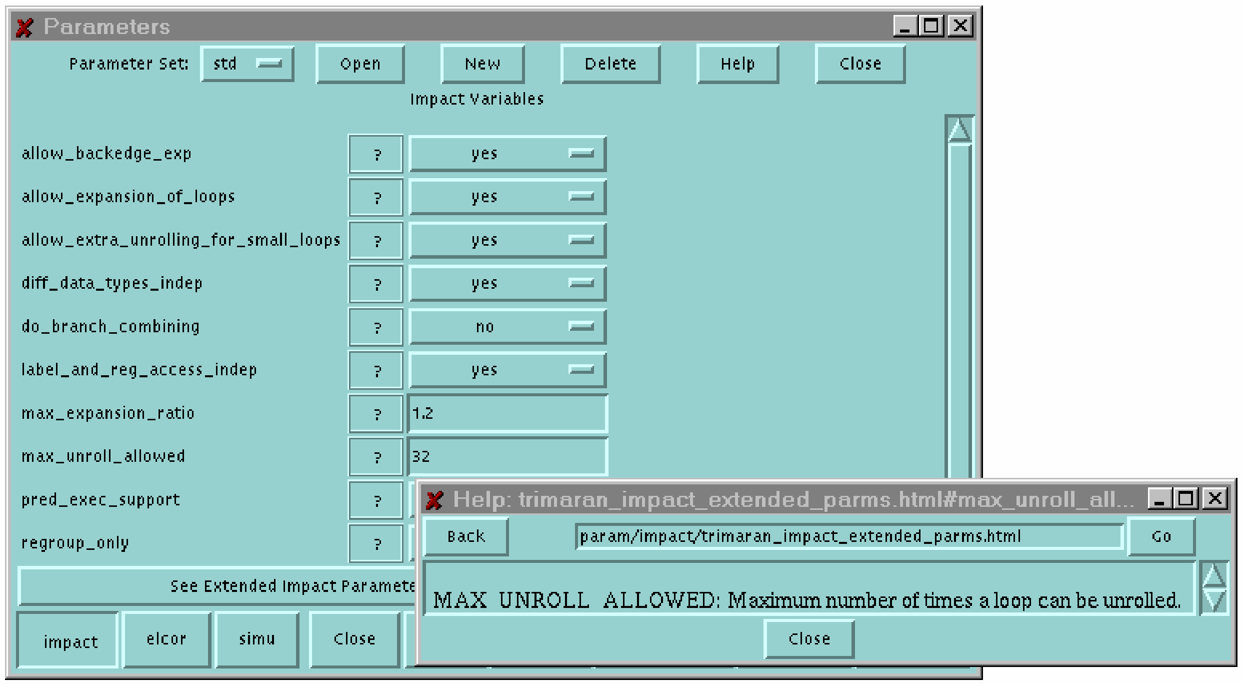
The Benchmark selected, the Machine name and the Parameter–Set define a Workspace. Every compilation instatiates this workspace and adds it to a Project–File. The name of the Project–File can be selected from the box provided. NEW creates a fresh Project–File with no Workspaces to start with. SET

The Machine Panel

- The machine panel is used create new target machines and modify existing ones



Modifying Parameters



Parameter Set: std

allow_backedge_exp	?	yes
allow_expansion_of_loops	?	yes
allow_extra_unrolling_for_small_loops	?	yes
diff_data_types_indep	?	yes
do_branch_combining	?	no
label_and_reg_access_indep	?	yes
max_expansion_ratio	?	1.2
max_unroll_allowed	?	32
pred_exec_support	?	
regroup_only	?	

Help: trimaran_impact_extended_parms.html#max_unroll_all...

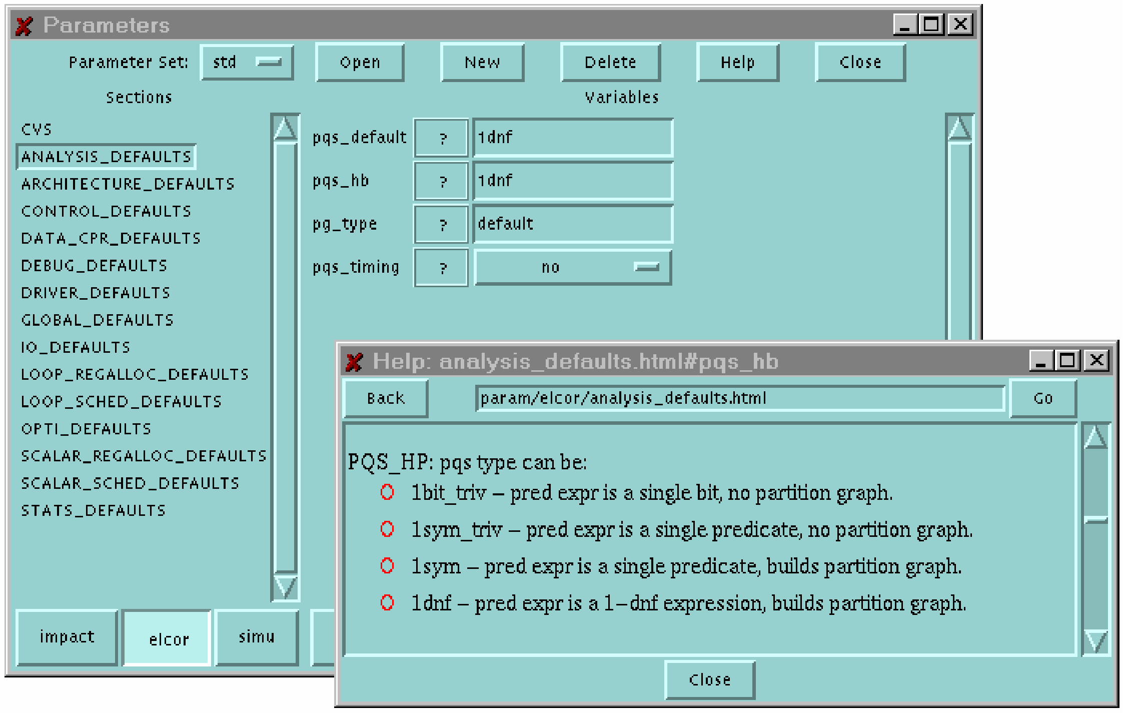
param/impact/trimaran_impact_extended_parms.html

MAX UNROLL ALLOWED: Maximum number of times a loop can be unrolled.

- Upon clicking ‘open’, the parameters are displayed
  - Here, the compiler front end parameters are displayed, along with their current values
  - Clicking a ‘?’ button opens a help window for that parameter
- Parameters can also be modified by editing text files, if desired

Parameters for the Back End

- The compiler back end has the largest number of parameters
- The parameters are organized into groups according to their use
  - Analysis
  - Optimizations
  - Register Allocation
  - Etc.

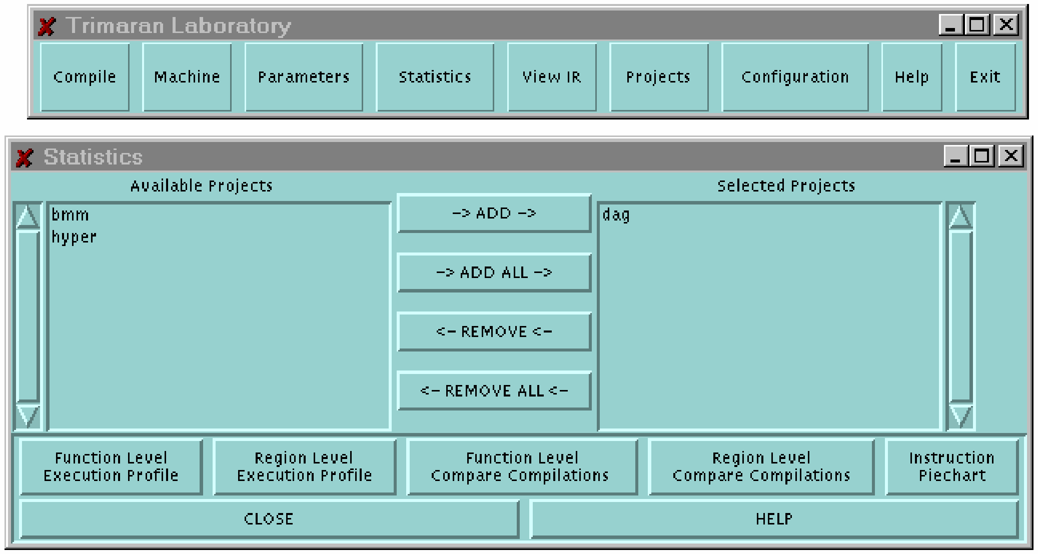


pqs_default	?	1dnf
pqs_hb	?	1dnf
pg_type	?	default
pqs_timing	?	no

PQS_HP: pqs type can be:
- 1bit_triv – pred expr is a single bit, no partition graph.
- 1sym_triv – pred expr is a single predicate, no partition graph.
- 1sym – pred expr is a single predicate, builds partition graph.
- 1dnf – pred expr is a 1–dnf expression, builds partition graph.

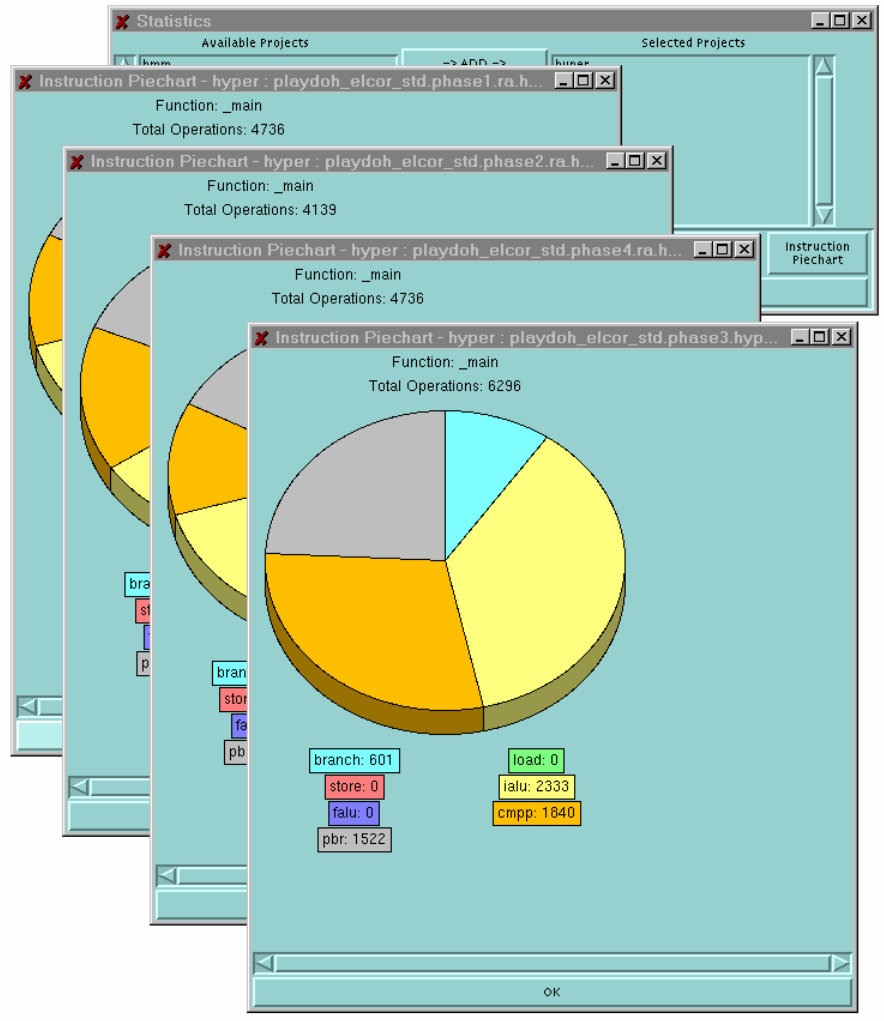
The Statistics Panel

- The statistics panel allows you to choose what statistics are displayed for the programs in one's project file
  - Function level execution profile
  - Region level profile
  - Instruction usage
  - Etc.



Viewing Statistics

- For each program in the project file, a separate graph is displayed
  - Here, pie charts show the dynamic instruction distribution.



The View IR Panel

- The IR viewer provides five kinds of views of a program.

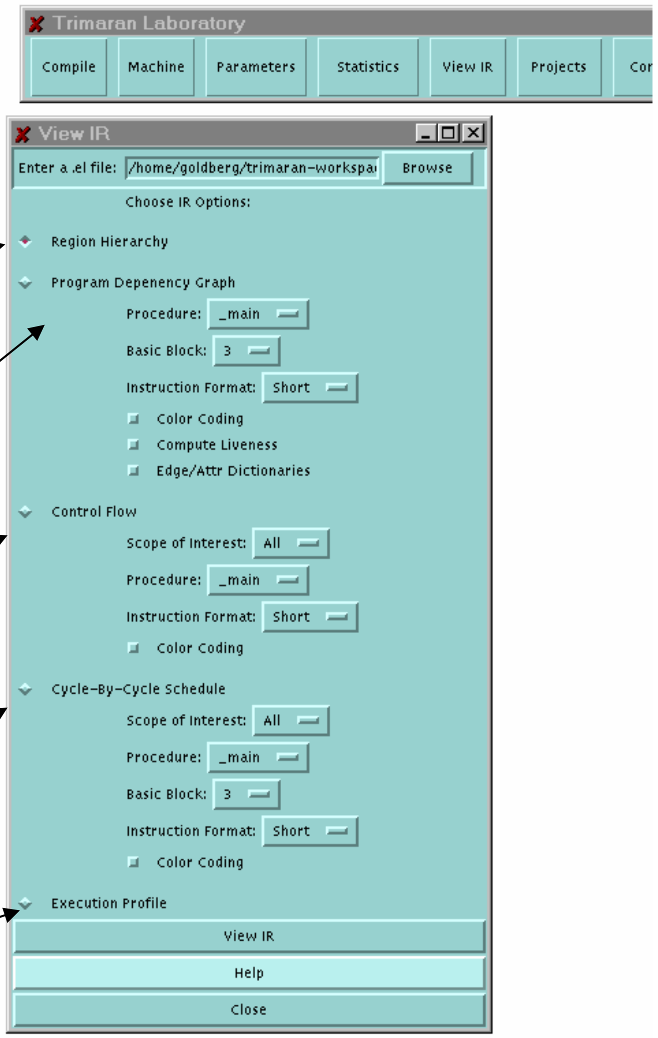
The program regions (hyperblocks, loops, etc.)

Dependence Graph

Control Flow Graph (CFG)

ILP Instruction Schedule

Profile Information



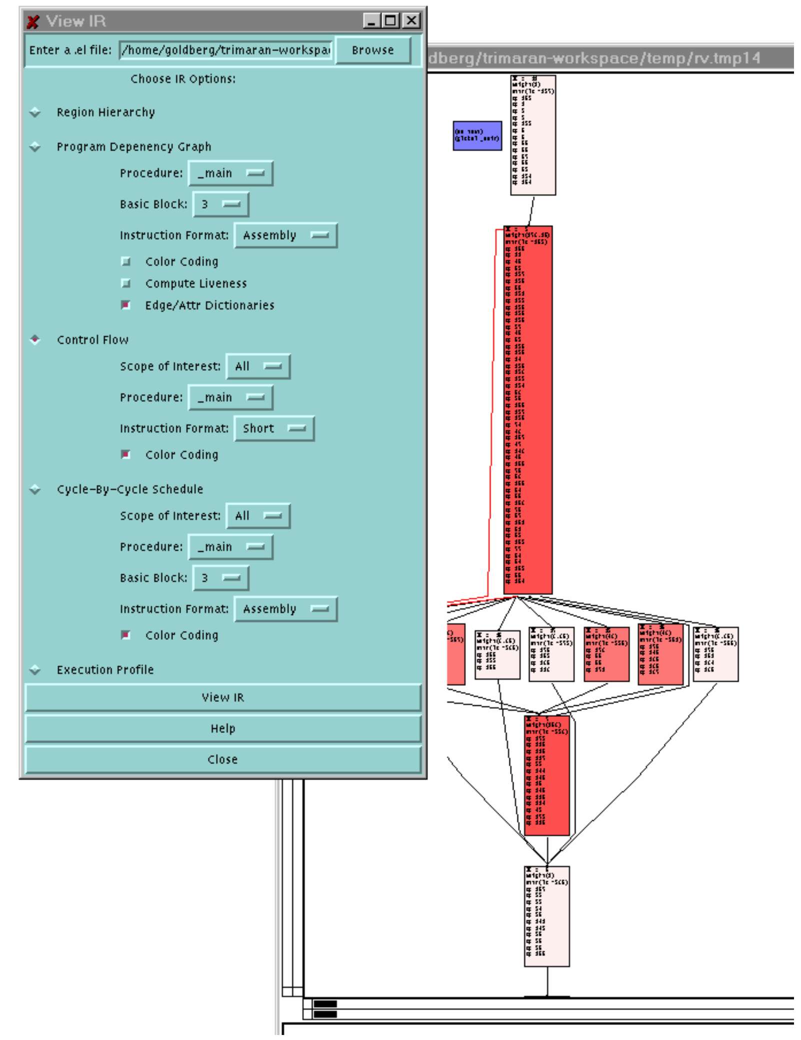
Enter a .el file: /home/goldberg/trimaran-workspa

Choose IR Options:

- Region Hierarchy
- Program Depenency Graph
  - Procedure: _main
  - Basic Block: 3
  - Instruction Format: Short
  - Color Coding
  - Compute Liveness
  - Edge/Attr Dictionaries
- Control Flow
  - Scope of Interest: All
  - Procedure: _main
  - Instruction Format: Short
  - Color Coding
- Cycle-By-Cycle Schedule
  - Scope of Interest: All
  - Procedure: _main
  - Basic Block: 3
  - Instruction Format: Short
  - Color Coding
- Execution Profile

Control Flow View

- Here is a portion of the control flow graph for a program.
- The user can specify a portion of the program to display.
- The viewer has zoom in, zoom out, scroll, etc.
  - other IR views are also present



View IR

Enter a .el file: /home/goldberg/trimaran-workspa Browse

Choose IR Options:

Region Hierarchy

Program Depenency Graph

Procedure: _main

Basic Block: 3

Instruction Format: Assembly

Color Coding

Compute Liveness

[✓ Edge/Attr Dictionaries]

[✓ Control Flow]

Scope of Interest: All

Procedure: _main

Instruction Format: Short

[✓ Color Coding]

Cycle-By-Cycle Schedule

Scope of Interest: All

Procedure: _main

Basic Block: 3

Instruction Format: Assembly

[✓ Color Coding]

Execution Profile

View IR

Help

Close

dberg/trimaran-workspace/temp/rv.tmp14

Summary

- The Trimaran GUI provides a natural interface for configuring and running the Trimaran system
  - Lowers barrier to entry for new user
    - No learning makefiles, searching parameter files, etc.
  - Provide interface to powerful visualization tools
    - IR viewer, execution statistics

http://www.trimaran.org