# D2X: an eXtensible conteXtual Debugger for modern DSLs

Ajay Brahmakshatriya

Saman Amarasinghe

CSAIL, MIT

CGO 2023, February 28th



# Writing A Debugger is Hard!

# Developers love debuggers

## 1. Debuggers provide context

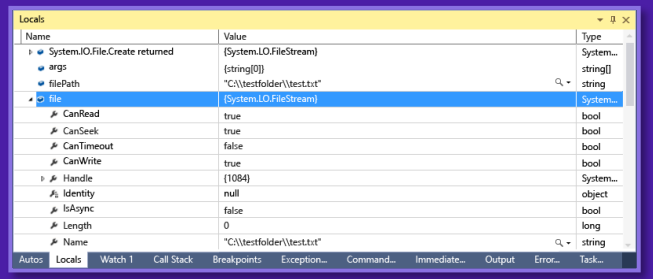
```
(gdb) bt
#0  0x0000555558696970 in chameleon::cc::create_new_constant_value(llvm::Type*
#1  0x00005555586982d0 in chameleon::cc::process_function_with_args(llvm::Func
#2  0x000055555869b9bd in chameleon::cc::ConstPropPass::run(llvm::Function&, l
#3  0x0000555558663416 in llvm::detail::PassModel<llvm::Function, chameleon::c
#4  0x000055555701ac2f in llvm::PassManager<llvm::Function, llvm::AnalysisMana
#5  0x0000555557b74da6 in llvm::detail::PassModel<llvm::Function, llvm::PassMa
#6  0x0000555557019806 in llvm::ModuleToFunctionPassAdaptor::run(llvm::Module&
#7  0x0000555557b75896 in llvm::detail::PassModel<llvm::Module, llvm::ModuleTo
#8  0x0000555557017758 in llvm::PassManager<llvm::Module, llvm::AnalysisManage
#9  0x0000555557b85258 in (anonymous namespace)::EmitAssemblyHelper::RunOptimi
#10 0x0000555557b87917 in clang::EmitBackendOutput(clang::DiagnosticsEngine&,
#11 0x000055555854bc22 in clang::BackendConsumer::HandleTranslationUnit(clang:
```

Call stack

```
linked_list.h
105             end->create(last_free_pos, args...);
106         }
107
108         template <typename Cond>
109         int remove_if (Cond&& condition) {
110             linked_list_node* current = begin;
B+ 111             linked_list_node* next;
 >  112             linked_list_node* prev = nullptr;
113             int removed = 0;
114             bool block_empty;
115
116             while (current != nullptr) {
117                 block_empty = true;
118                 for (int i = 0; i < count; i++) {
119                     if (current->used(i)) {
120                         if (condition(current->to_const_reference(i))) {
multi-thre Thread 0x7fffff7b07 In: run_test<1, 1, 1>                    L112  PC: 0x8004622
Breakpoint 2 at 0x800377a: linked_list.h:111. (12 locations)
(gdb) c
Continuing.

Breakpoint 2, linked_list<test_struct<1>, 1>::remove_if<void run_test<1, 1, 1>(std::vector<
int, std::allocator<int> >&)::{lambda(test_struct<1> const&)#1}>(void run_test<1, 1, 1>(std
::vector<int, std::allocator<int> >&)::{lambda(test_struct<1> const&)#1}&&) (
    this=0x7fffffedf00, condition=...) at linked_list.h:112
(gdb)
```

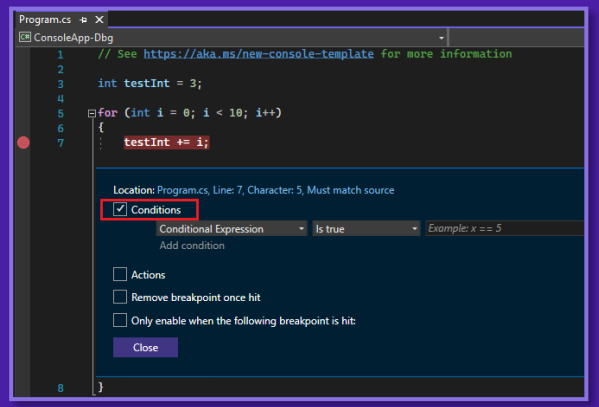
Source listing



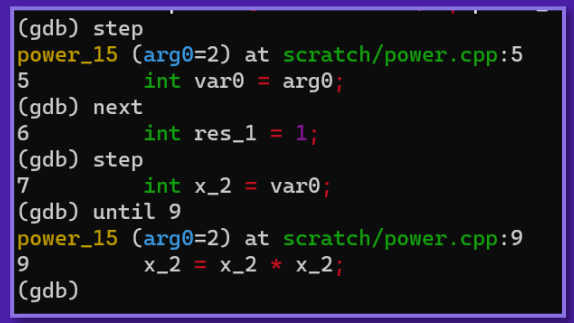

Inspect variables and expressions

# Developers love debuggers

1. Debuggers provide context
2. Debuggers can control execution




Setting breakpoints and watchpoints




Single stepping execution


Time travel debugging

# Developers love debuggers

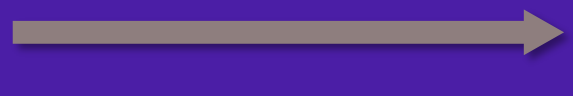
1. Debuggers provide context
2. Debuggers can control execution
3. Debuggers can manipulate state
4. Debuggers work where printing doesn't

# DSLs need specialized Debuggers

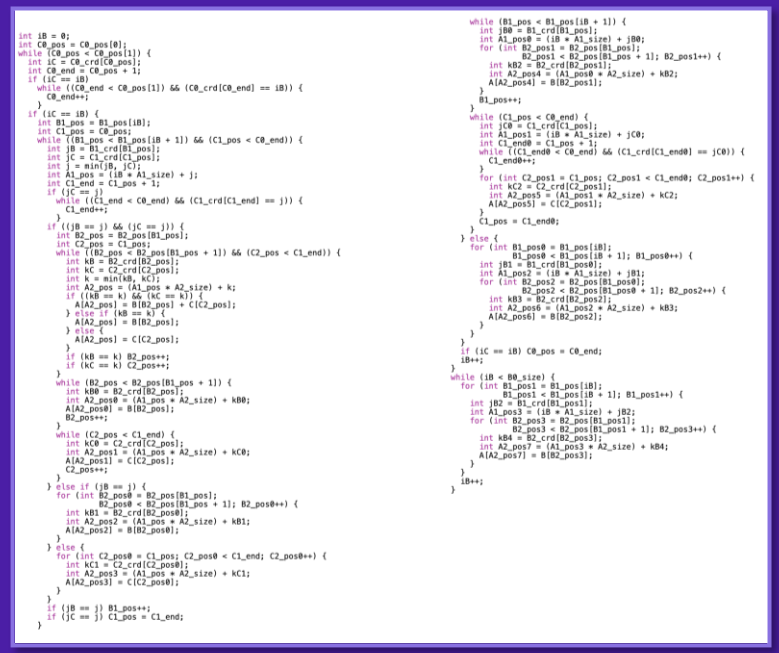
## 1. Single operator generates 100s of lines of code

$$A_{ijk} = B_{ijk} + C_{ijk}$$

TACO DSL



Compiler



# DSLs need specialized Debuggers

1. Single operator generates 100s of lines of code
2. DSL compilers perform complex transformations
3. DSLs have complex data types that don't support printing

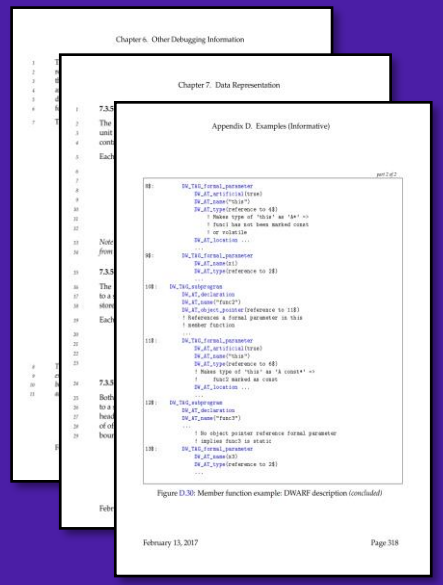
Debug information standards are complex

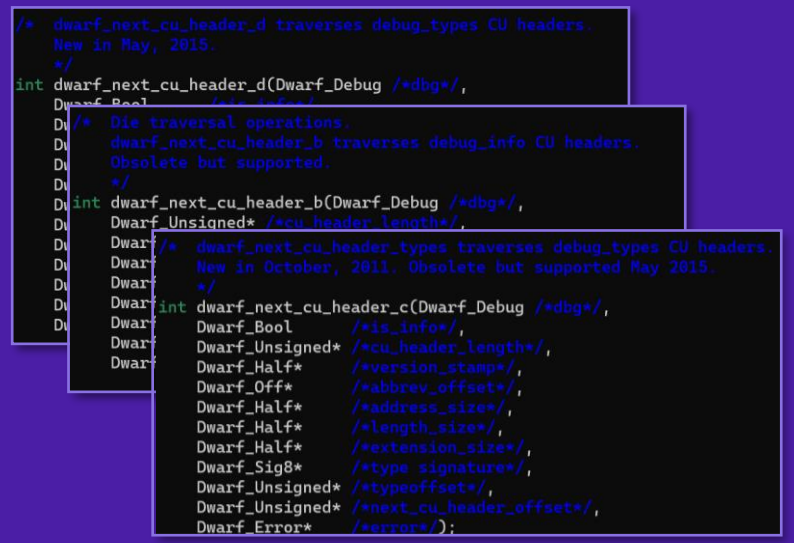
Implementing/ retrofitting debuggers is time consuming

Reimplement compiler tooling to support debugging

# Debug information standards are complex

## DWARF-5 standard is 475 pages




## Standard libraries are hard to use

# Implementing/retrofitting debuggers is time consuming


Plugins supported by debuggers are not portable

# Reimplement compiler tooling to support debugging

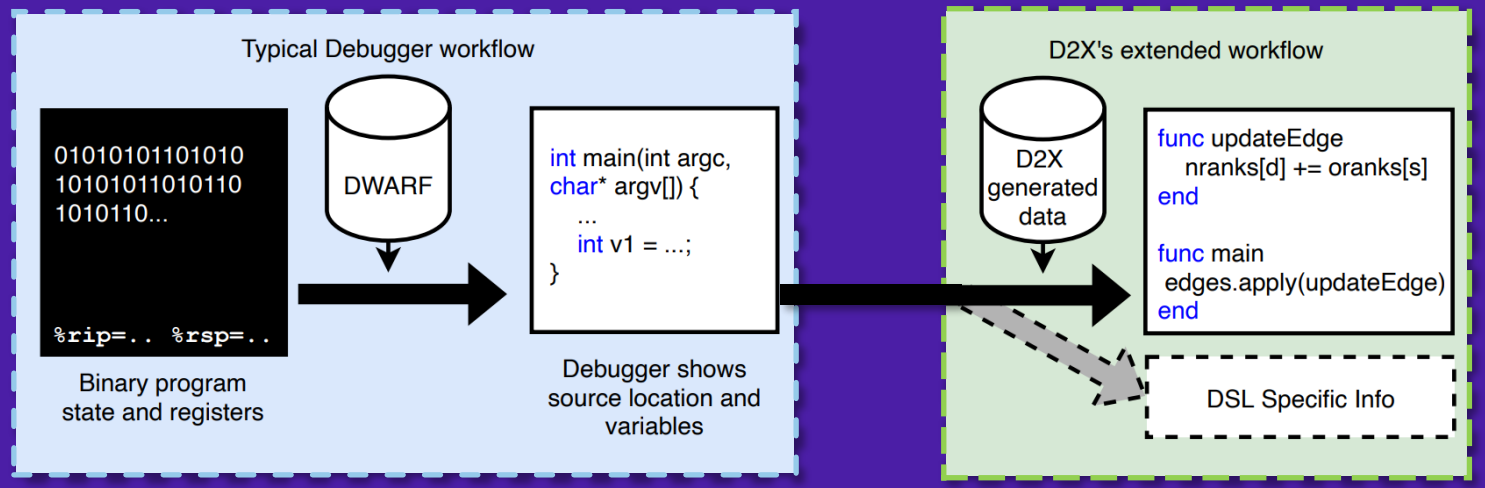
Not enough support to extract and generate debug information

DSL specific debugging information cannot be represented

# D2X


https://buildit.so/d2x

# D2X: Overview




1. Extended Call Stack

3. DSL Breakpoints

2. Extended Vars

4. DSL Source View

# D2X: Debug Information

## 1. Source Maps: C++ arrays alongside generated code

```
int power_15 (int arg0) {
  int var0 = arg0;
  int res_1 = 1;
  int x_2 = var0;
  res_1 = res_1 * x_2;
  x_2 = x_2 * x_2;
  res_1 = res_1 * x_2;
  x_2 = x_2 * x_2;
  res_1 = res_1 * x_2;
  x_2 = x_2 * x_2;
  int var3 = res_1 * x_2;
  return var3;
}
/*  Begin debug information for section: 0 */
static struct d2x::runtime::d2x_source_stack d2x_0_source_table[] = {
        {0, 0}, //0
        {0, 0}, //1
        {3, 0}, //2
        {4, 3}, //3
        {4, 7}, //4
        {4, 11}, //5
        {4, 15}, //6
        ...
};
static struct d2x::runtime::d2x_source_loc d2x_0_source_list[] = {
        {0, 61, 1, -1}, //0
        {0, 154, 2, -1}, //1
        {3, 290, 4, -1}, //2
        {5, 15, 6, -1}, //3
        {0, 61, 1, -1}, //4
        {0, 154, 2, -1}, //5
        {3, 290, 4, -1}, //6
        {5, 15, 6, -1}, //7
        ...
```

Each line of generated code is mapped to a stack of DSL source

```
func updateEdge(s: Vertex, d: Vertex)
   nrank[d] += orank[s]
end
func main()
#s1# edges.apply(updateEdge) // PUSH Schedule
#s2# edges.apply(updateEdge) // PULL Schedule
end
```

```
void updateEdge_1(int s, int d) {
    atomicAdd(&nrank[d], orank[s]); // For #s1#
}
void updateEdge_2(int s, int d) {
    nrank[d] += orank[s];        // For #s2#
}
```

DSLs compile the same function differently based on context

# D2X: Debug Information

## 2. Variable Maps: Key-value pairs attached to source lines

```
static struct d2x::runtime::d2x_var_stack d2x_0_var_table[] = {
        {0, 0},
        {0, 0},
        {0, 0},
        {0, 0},
        {6, 0},
        {6, 18},
        {6, 24},
        {6, 30},
        ...
};
static struct d2x::runtime::d2x_var_entry d2x_0_var_list[] = {
        {14, 15, 0},
        {16, 15, 0},
        {17, 15, 0},
        {18, 15, 0},
        {19, 15, 0},
        {20, 15, 0},
        {14, 15, 0},
        {16, 15, 0},
        ...
};
static const char* d2x_0_string_table[] = {
        "index",
        "matrix_vector_multiplication",
        "a.constant_val",
        "0",
        "a.is_constant",
        "b.constant_val",
        "b.is_constant",
        "1",
};
```

a. Constant values as strings

```
std::string d2x_resolver_0 (std::string arg0) {
  VertexSubset<int>** var2 = d2x::runtime::rtv::find_stack_var(arg0);
  VertexSubset<int>* var3 = var2[0];
  std::string var4 = ("is_dense(" + std::to_string(var3->is_dense)) + ") [";
  if (!(var3->is_dense)) {
    for (int var5 = 0; var5 < var3->num_vertices_; var5 = var5 + 1) {
      var4 = (var4 + std::to_string(var3->dense_vertex_set_[var5])) + ", ";
    }
  } else {
    for (int var6 = 0; var6 < var3->vertices_range_; var6 = var6 + 1) {
      if (var3->bool_map_[var6]) {
        var4 = (var4 + std::to_string(var6)) + ", ";
      }
    }
  }
  var4 = var4 + "]";
  return var4;
}
```

b. Runtime values are small snippets of C++ code that can be run when debugging

# D2X: Debug Information

Easy to understand and trivial to extend!

# D2X: Interactive Debugger Support

**Goal 1:** Use popular debuggers with 0 modifications

**Goal 2:** Portable

# D2X: Interactive Debugger Support

```c
#include <stdio.h>

void print_hello(void) {
        printf("Hello World!\n");
}


int main(int argc, char* argv[]) {
        return 0;
}
```

```
$ gcc -g hello.c -o hello
$ gdb ./hello
```

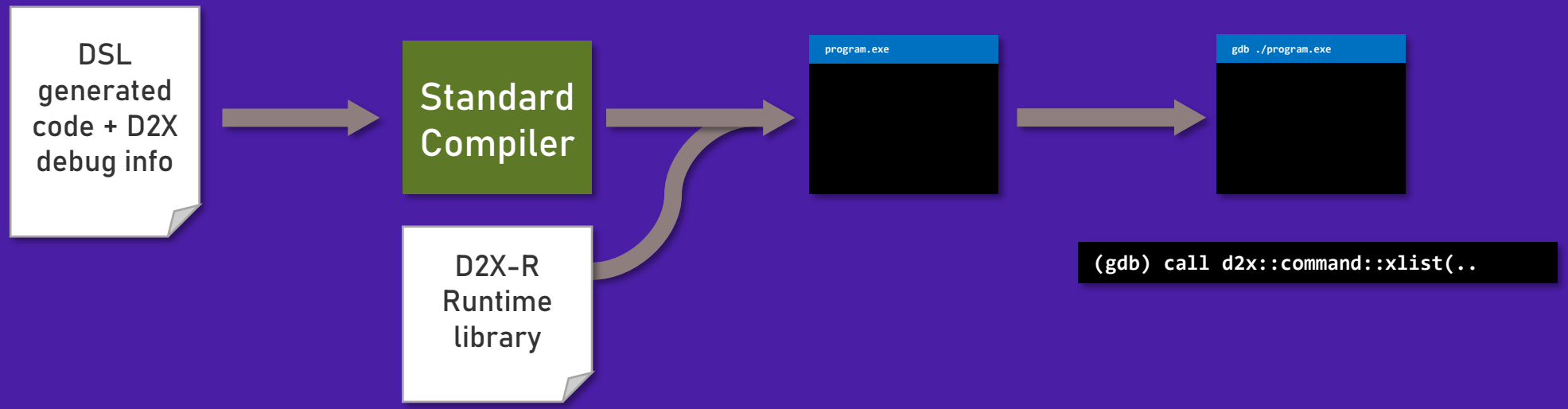
```
Reading symbols from ./hello...
(gdb)

(gdb) break main
Breakpoint 1 at 0x1177: file hello.c, line 9.
(gdb) run
Starting program:
Breakpoint 1, main (argc=1, argv=..) at hello.c:9
9               return 0;

(gdb) call print_hello()
Hello World!
(gdb)
```

# D2X: Interactive Debugger Support

## Implement D2X "commands"



# D2X: Interactive Debugger Support

## Printing source information

```
(gdb) call d2x::command::xlist($rip, $rsp)
```

```
void xlist (uint64_t rip, uint64_t rsp) {
    std::string filename, dsl_filename;
    int line_no, dsl_line_no;

    find_source_info(rip, rsp, filename, line_no);

    translate_source_info(filename, line_no, dsl_filename, dsl_line_no);

    print_source_from_file(dsl_filename, dsl_line_no);
}
```

**Implementation in the D2X Runtime library**

# D2X: Interactive Debugger Support

## Convenience Macros

```
define xlist
        call d2x::runtime::cmd::xlist($rip, $rsp)
end

define xvars
        if $argc == 0
                call d2x::runtime::cmd::xvars($rip, $rsp, "")
        end
        if $argc == 1
                call d2x::runtime::cmd::xvars($rip, $rsp, "$arg0")
        end
end
```

### GDB Specific helpers

```
gdb --command=helper.init ./program
...
(gdb) xlist
...
(gdb) xvars foo
```

# D2X: Interactive Debugger Support

## Managing DSL Breakpoints

One line of DSL maps to 10s of lines of generated code

+

Passively "printing" information in the debugger is not sufficient

## How can D2X take control?

# D2X: Interactive Debugger Support

## Managing DSL Breakpoints

```
(gdb) eval “print %d”, 3
$1 = 3

(gdb) eval “break %s:%d”, “foo.cpp”, 10
Breakpoint 1 (foo.cpp:10)
```

“printf”-style eval for commands

```
(gdb) eval “%s”, d2x::command::xbreak(..
```

# D2X: Interactive Debugger Support

# Managing DSL Breakpoints

```
(gdb) eval “%s”, d2x::command::xbreak($rip, $rsp, “file.gt:10”)
```

```
void xbreak (uint64_t rip, uint64_t rsp, std::string break_spec) {
    std::vector<std::pair<std::string, int>> locs = find_source_locations(break_spec);

    std::string ret = “”;
    for (auto loc: locs) {
        ret += “break ” + loc.first + “:” + std::to_string(loc.second) + “\n”;
    }
    std::cout << “Inserted ” << locs.size() << “ breakpoints for ” << break_spec << “\n”;
    return ret;
}
```

```
define xbreak
        eval “%s”, d2x::runtime::cmd::xlist($rip, $rsp, “$arg0”)
end
define xdel
        eval “%s”, d2x::runtime::cmd::xdel($rip, $rsp, “$arg0”)
end
```

# D2X: Interactive Debugger Support

## All D2X Commands

| Command | Description |
| --- | --- |
| (gdb) xbt | Print the extended stack for the current line |
| (gdb) xlist | Print the source code for the current frame in the extended stack |
| (gdb) xframe [frame_id] | Display the current extended frame or switch the current frame |
| (gdb) xbreak [breakspec] | Insert a new breakpoint at the specified filename and line number or display all the current extended breakpoints |
| (gdb) xdel [breakspec] | Delete the breakpoint at the specified filename and line number |
| (gdb) xvars [varname] | Display all live vars at the current line or compute and print the value of the var name specified |

XVARS allow developers to run arbitrary code and extended D2X

# D2X: Compiler library

## D2X-C helps generate debugging info

1. library functions to attach source information to generated lines of code
2. Create and maintain live variables and their values + generate code for runtime values
3. Extract source information for embedded DSLs using DWARF info

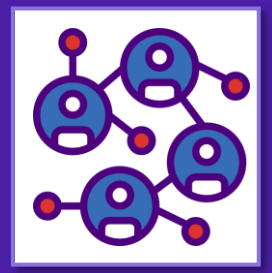
Debugging added to 45,966 loC GraphIt compiler with 1.4% change!

# D2X + BuildIt

Rapid prototyping DSL compilers made easy!

# BuildIt[1]

- Multi-Stage C++ library that helps domain-experts to write DSLs without *any* compiler knowledge
- Developers write a library/interpreter for their abstraction. BuildIt creates a compiler



DSL for graph analytics on GPUs in 2021 LoC[2]


High-performance regular-expressions compiler

Generate custom network protocols and implementations

...

1 - A. Brahmakshatriya and S. Amarasinghe, "BuildIt: A Type-Based Multi-stage Programming Framework for Code Generation in C++," 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), Seoul, Korea (South), 2021

2 - A. Brahmakshatriya and S. Amarasinghe, "GraphIt to CUDA Compiler in 2021 LOC: A Case for High-Performance DSL Implementation via Staging with BuilDSL," 2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), Seoul, Korea, Republic of, 2022

# D2X + BuildIt

## first stage variables and code are erased from the generated code

```
template <typename T>
struct Tensor {
    dyn_var<T*> buffer;
    static_var<bool> is_constant;
    static_var<T> constant_val;
};

...

dyn_var<T> Tensor::get_value(dyn_var<int> index) {
    if (is_constant) return constant_val;
    return buffer[index];
}
```

```
b[j] = 5.0;

c[i] = 2 * a[i][j] * b[j];
```

```
for (i_3 = 0; i_3 < 1024; i_3 = i_3 + 1) {
  arg0[i_3] = 0;
  for (j_4 = 0; j_4 < 512; j_4 = j_4 + 1) {
    arg0[i_3] = arg0[i_3] + var15 * arg1[i_3 * 512 + j_4] * 5.0;
  }
}
```

```
(gdb) xvars
1. a.constant_val
2. a.is_constant
3. b.constant_val
4. b.is_constant
(gdb) xvars b.is_constant
b.is_constant = 1
(gdb) xvars b.constant_val
b.constant_val = 5.0
(gdb) xframe 7
#7 in matrix_vector_multiplication at einsum.cpp:360
360             c[i] = 2 * a[i][j] * b[j];
```

## DSLs written with BuildIt get Debuggers for free!

# D2X + BuildIt

## Demo

# D2X + BuildIt

looking to write a DSL or add debugging support to existing DSLs?


https://buildit.so

**We would love to hear:** Ajay (ajaybr@mit.edu)

MIT CSAIL