Programming Languages and Translators
COMS W4115

Department of Computer Science
Columbia University
Fall 2008

Fast vector processing language

Gowri Kanugovi <gk2263@columbia.edu>
Pratap Prabhu <pvp2105@columbia.edu>
Ravindra Babu Ganapathi <rg2547@columbia.edu>

Table of Contents

1. Introduction

1.1. Motivation

Image processing, databases and cryptography are among those domains which commonly deal with Operation on large sequential data, but most of the current C/C++ compilers produce slow native code for the x86 family of processors. These processors include a vector processing unit which has the capability to execute instructions in parallel. Such instructions known as **S**ingle **I**nstruction **M**ultiple **D**ata (SIMD) operate on 128-bits of data at a time. Utilizing the power of these SIMD instructions acted as the main motivation for the development of Fast Vector Processing Language (FVPL). FVPL allows programmers to transparently and efficiently utilize the SIMD instructions to compute large amount of sequential data at higher speeds.

Vectors are among the most important data structures used in any language. However, the programmer is forced to treat vectors differently from scalars in most languages. Scalars are allowed to be programmed transparently but vectors are always manipulated in loops. In FVPL we aim to provide the same transparency to vectors. Programmers manipulate with vectors just like the way they would with scalars. The addition, subtraction, logical operations on two or more arrays need not be in a loop anymore. Hence in FVPL we aim to provide both utility and speed to the programmers.

1.2. Description

FVPL allows the Operation of vectors as primitive data types. In the sense, that the operations on vectors similar to operations on fundamental types, like say the way C/C++ treats the Operation of integers. This provides the programmer not only ease of programming but also the motivation to use vectors without having to worry about the large amount of code which would have otherwise been required. Allowing the use of vectors succinctly and clearly taking advantage of new generation processors to decrease the time of execution is the goal of FVPL.

Since vectors are allowed to be used hand-in-hand with scalars, the verbosity of the entire language decreases by a great extent. The syntax of the language closely resembles C which makes the transition for programmers much easier. The data types are named just like in C: int8, int, float. FVPL also provides support for I/O operations like reading from a file and writing to a file. The constructs used are similar in syntax with the C/C++ constructs: if-else, for, while. It also provides control-flow statements such as return, break and continue to transfer control and terminate from loops respectively. Programmers also have the flexibility to print any message of variable on the standard output without having to mention its type to the compiler. The compiler for FVPL internally assumes the data type or message the user is trying to output.

The language is implemented in the Functional Programming language OCaml. Using OCaml not only reduced the size of the source code but its many powerful features helped programming the concepts in a much simpler fashion.

1.3. Example Program

The following program depicts simple vector operations written in FVPL.

```
int main(){
        int a[10], b[20], c[10];
        int8 d[10], e[10], f[20];

        a = 5;  //all 10 elements of a are initialized to 5
        b = a+a;  //the first ten elements of b now will not contain 10
        print(a);
        print (b);

        c = b-a; //all 10 elements of c will now be 5
        print(c);

        d = 5;
        e = 6;
        f = d & e;  //logical and operation on elements of d and e
        print(f);

        c = c | f;  //logical or operation on elements of c and f
        print(c);

        a = a ^ b; //logical xopr operation on elements of a and b
        print(a);

}
```

It is clear from the example that not only is the program very concise but even the array Operations are very simple. The operations on vector elements are carried out just like operations on any scalar values. Transparency and efficiency are thus achieved using FVPL.

2. Tutorial

This section gives a brief description on how to get started with programming in FVPL. The compiler runs on both Windows and Linux. However, the following description is for running it on Linux only. Consider the following example program for adding a vector to itself and displaying the result.

```
int main()
{
  int a[10], b[10];
  a = 5;
  b = a + a;
  print(b);
}
```

Let us consider this program to be named as add_vector.fvpl. To run this program one can use the ***make exec*** command provided in the make file.

To run the program, use the following command:

```
$ ARGS="add_vector.fvpl" make exec; ./a.out
```

The output generated is:

```
ocamlc -c ast.mli
ocamlyacc parser.mly
ocamlc -c parser.mli
ocamllex scanner.mll
121 states, 5751 transitions, table size 23730 bytes
ocamlc -c symbol.ml
ocamlc -c interpret.ml
ocamlc -c scanner.ml
ocamlc -c parser.ml
ocamlc -c fvpl.ml
ocamlc -o fvpl parser.cmo scanner.cmo symbol.cmo interpret.cmo fvpl.cmo
./fvpl < add_vector.fvpl > /tmp/tmp.c
g++ -DLINUX -msse2 -msse3  -I./ /tmp/tmp.c
10
10
10
10
10
10
10
10
10
10
```

You will notice that the make exec command takes care of building the source code, generating the intermediate C++ code and compiling it with g++ to produce the final result.

The basic data types of FVPL are just like in C: int8, int, float.

The description on each data type, the language constructs, the types of expressions, types of operators, scopes are explained in more detail in the Language Reference Manual in section-3 of this report.

3. Language Reference Manual

3.1. Lexical Conventions

FVPL comprises tokens such as: keywords, identifiers, comments, integer constants, floating point constants, operators and separators. It is a free form language; spaces, tabs and new lines are ignored and considered to only serve as delimiters between tokens.

3.1.1. Comments

Single line comments in FVPL are begin with the characters // like in the C-language. Multi-line comments are also supported by FVPL. Such comments begin with /* and should be terminated by */

3.1.2. Identifiers

Identifiers are sequence of letters, digits and the underscore ('_') character. The first letter however has to be only either an alphabet or the underscore character. Identifiers cannot begin with a digit. FVPL identifiers are case-sensitive. Following are some examples of FVPL identifiers

Valid identifiers: A, foo, a, _bar, bar_foo, count2
Invalid identifiers: 1A, a#, foo-bar

3.1.3. Keywords

The following table summarizes the identifiers used as keywords in FVPL. These keywords cannot be used as otherwise.

int	float	int8	while	main
for	continue	void	return	sizeof
Break	if	else		

3.1.4. Constants

Constants provide programmers the ease of initializing any of their identifiers to one of the supported primitives. The different types of constants supported by FVPL are:

Integer constants: Integers of FVPL consists of an optional '+' or '-' sign followed by any number of digits in the range of 0-9.

Floating point constants: Floating point numbers in FVPL comprises of an optional '+' or '-' sign followed by an integer of one or more digits. This is followed by a decimal point which is then followed by an integer of one or more digits.

3.1.5. Operators

FVPL supports operations on both scalars and vectors.

Operators on Scalars: A programmer can perform the following actions on a scalar type in FVPL:

- Arithmetic operators: The operators '+, '-', '*' and '/' are supported by FVPL. The semantics of the operators are similar to those of addition, subtraction, multiplication and division respectively. The multiplication and division operators are however supported only for the int and float data types in FVPL. The precedence and associativity of operators follows the same conventions as in the C-language.
- Bitwise logical operators: FVPL supports bitwise logical AND, OR, XOR and NOT operations. These are denoted by '&', '|', '^' and '~' respectively.
- Assignment operators: The assignment operator in FVPL is denoted the '=' symbol. This operator assigns the value of the right hand side expression to the left hand side expression.
- Sizeof operator: The sizeof operator returns the size of the operand in bytes. The result is an integer value and thus can be used for assigning any integer variable.

Operators on Vectors: FVPL, being a vector processing language tries to provide the programmer the maximum benefit of vector operations. These operators can be used transparently by the programmer as though he is performing the operation on a scalar. For example, there is no need for him to maintain a loop to perform operation on each of the vector member. The following operators are supported by FVPL:

- Vector arithmetic operators: The operators '+', '-', '*' and '/' are supported by FVPL. The semantics of the operators are similar to those of addition, subtraction, multiplication and division respectively. Addition of two vectors implies that each element of one vector is added to the corresponding element of the other vector.

For e.g.: Three vectors A,B and C

*The operation (A+B)*C implies that the sum of each of the element in A and B is multiplied by the corresponding element in C*

- Vector logical operators: FVPL supports bitwise logical operations on the vector elements. The logical AND, OR, XOR and NOT are represented as '|&', '|', '^' and '~' respectively. Any logical vector operation implies that the operator is applied on each of its corresponding elements.

  For e.g.: Two vectors A and B

  The operator A&B implies a bitwise logical AND between every element of A and the corresponding element of B.

- Vector assignment operator: The vector assignment operator denoted by '=' operator implies that every element of the vector is initialized with the value of the scalar element on the right hand side of the operator.

  For e.g.: Vector A

  A=5 assigns the value 5 to every element of A

- Vector initialization: At the time of the creating the vector, FVPL allows the programmer to initialize every element of the vector. This is particularly useful when the programmer is trying to create small vectors and wants to assign distinct values to it manually.

  For e.g.: Vector A

  A = {1, 2, 3, 4, 5} initializes the vector with 5 elements each with the value given in the braces.

- Vector concatenation: FVPL allows two vectors to be concatenated with each other by using the '@' operator. At the end of this operation we will have one vector which contains all the elements of the two vectors involved in the operation.

  For e.g.: Vector A = {1, 2, 3}

  Vector B = {4, 5, 6}

  A@B returns {1, 2, 3, 4, 5, 6}

- Vector copy. In FVPL, elements of one vector can be copied into the other vector by using the vector copy operator. The operator used is same as the assignment operator, but in this case the right hand side of the operation has to be a vector.

  For e.g.: Vector A, B

  A = B implies every element of B is copied into the corresponding position in A i.e. A[0]=B[0], A[1]=B[1]... A[999]=B[999]

- Vector sizeof operator: The sizeof operator on vectors returns the size in bytes of the total memory allocated to the vector. For example, if the vector contains 'n' integer elements then the sizeof operator returns (4*n) bytes.

3.1.6. Separators

The two separators supported by FVPL are comma (‘,’) and semi-colon (‘;’). The separator ‘;’ is used to indicate the end of a statement whereas the ‘,’ separator separates two identifiers.

3.1.7. Block delimitation

In FVPL, the opening flower braces ‘{‘ indicates the beginning of a block of code and the closing flower braces ‘}’ indicate the end of the block of code. Blocks play a major role in defining the scope of variables. All variables defined within a block are visible only to the code in that block. Scoping is explained in more detail in the next section.

3.1.8. Scoping

Scope of variables can be defined in two different contexts in FVPL:

- *Local scope*: Variables declared in a block of code are visible only within that block. This is the local scope of the variable. Accessing the variable anywhere outside its scope will result in a compilation error indicating that the variable is undefined.
- *Global scope*: A global variable is declared outside all functions. These variables can be accessed by any part of the program. If any function alters the value of this variable, then the altered variable is seen by all other functions. However, local variables with the same name override the global variable within that block.

3.2. Data Types

Every identifier in FVPL is associated with a type which indicates the way the identifier is interpreted by the program.

3.2.1. Basic Type

FVPL allows operations on both the scalar and vector types.

- Scalar data types: FVPL supports the following basic data types on scalars:
  int8 is a sequence of 8 bits
  int is a sequence of 32 bits
  float is a single precision floating number, having a size of 32 bits

- Vector data types: FVPL supports the following basic data types on vectors:
  int8 creates a vector such that each of its element is a sequence of 8 bits
  int creates a vector such that each of its element is a sequence of 32 bits
  float creates a vector such that each of its element is a single precision floating point number with size of 32 bits

3.3. Branching Construct

In FVPL, the following constructs are used to control the flow of code:

- *if: 'if'* is the keyword used for conditional execution of code. If the condition associated with the 'if' statement is true, then the code block associated with it is executed

  Syntax: if(condition) { /* statements to be executed */ }

- *else:* 'else' is the keyword used in conjunction with 'if'. When the condition associated with the 'if' statement evaluates to false then the code block associated with 'else' is executed.

  Syntax: else { /* statements to be executed */ }

- *return:* The 'return' keyword passes the flow of control to the statement which called for the execution of a particular function. This keyword is specifically used to return from the called function back to the calling function.

  Syntax: return;

- *break:* The “break” keyword is used to terminate the execution of a loop and pass control to the first statement after the loop.

  Syntax: break;

- *continue:* The “continue” keyword is used to terminate the execution of the current iteration of the loop. The control is passed to the first statement of the loop with the modified looping variable.

  Syntax: continue;

3.4. Looping Construct

The looping construct in FVPL are the keywords *'for'* and *‘while’*. The semantics of FVPL *'for'* and *‘while’* is same as the C language 'for' and ‘while’ loop. It is used to execute the same piece of code till some condition is met.

```
Syntax: for(initialization; condition; looping)
              {
                    /* statements to be executed till
                       the termination condition is
                       reached */
               }
Syntax: while(condition)
            {
                    /*statements to be executed till
                      the condition evaluates to true */
            }
```

3.5. Declarations

Any variable used in the FVPL program needs to be declared before it is used in the program. The declaration of a variable should include the data type and the identifier name.

Syntax: data-type identifier;

Two identifiers of the same type can be declared in a single line and separated with a comma. Declaration statements can optionally include the initialization of the variable.

3.6. Functions

FVPL supports function calls like the C-language. Functions are written to perform a particular sub-task which can be reused. In FVPL, functions need to be declared like any other variable before they are actually used.

The starting point of an FVPL program, like in C, is the *main* function. Therefore, any functions that are used in the program need to be either defined or declared before it.

Functions can optionally return some value to the caller. After returning from the function, the calling function resumes execution. Given below is the syntax of function declaration

Syntax: return-type function-name(function-parameters) { //body of the function }

Return-type: It can be any one of the data types supported by FVPL i.e. int, float, int8. If the function is not returning a value to the caller, then the return type should be “void”.

Function-name: It is the identifier of the function. This name is used as the reference to the function when the function is to be called.

Function-parameters: A list of variables that are passed to the function body by the caller. The parameters can be of any data type but the order in which these data types are passed to the function is important.

Function-body: A block of code, which performs the task of the function. Any variable declared within the function exist only within the function and cannot be viewed anywhere outside. But the function can access and modify the global variables in the program.

3.7. Grammar

```
program:
          { [],[] }
          | g_decl  program
          | fun_def program

fun_def : INT ID LPARAN fun_def_args_list RPARAN LBRACE decl_list stmt_list
RBRACE

fun_def_args_list : fun_def_args_list COMMA  fun_def_args
                  | fun_def_args
                  | {[]}


fun_def_args:   datatype ID
          | datatype ID LBRACKET DIGIT RBRACKET

stmt_list :
    { [] }
    |stmt stmt_list

stmt:   IF LPARAN expr RPARAN LBRACE stmt_list RBRACE elsetag
  | ID ASN stmt
  | FOR LPARAN init_stmt SEP opt_expr SEP init_stmt RPARAN LBRACE stmt_list
  RBRACE
  | WHILE LPARAN expr RPARAN LBRACE stmt_list RBRACE
  | expr SEP
  | PRINT LPARAN  STRING  RPARAN SEP
  | PRINT LPARAN ID RPARAN SEP
  | PRINT LPARAN DIGIT RPARAN SEP
  | PRINT LPARAN FDIGIT RPARAN SEP
  | ID LPARAN arg_list RPARAN SEP
  | RETURN opt_bracket_l expr opt_bracket_r SEP
  | CONTINUE SEP
  | BREAK SEP

opt_expr: expr
    |

opt_bracket_l: LPARAN
          |

opt_bracket_r: RPARAN
          |

elsetag: ELSE LBRACE stmt_list RBRACE
```

```
arg_list:  expr COMMA arg_list
     |expr
     |

expr:
     expr PLUS term
     | expr MINUS term
     | expr OR term
     | expr AND term
     | expr XOR term
     | expr LT term
     | expr GT term
     | expr LTEQ term
     | expr GTEQ term
     | expr EQ term
     | expr NOTEQ term
     | ID PLUSPLUS
     | ID MINUSMINUS
     | LBRACE int_id_list RBRACE
     | LBRACE f_id_list RBRACE
     | term
     | STRING

term: term MUL atom
| term DIV atom
| atom

f_id_list: FDIGIT COMMA f_id_list
     | FDIGIT

int_id_list: DIGIT COMMA int_id_list
     | DIGIT

atom: DIGIT
| FDIGIT
| ID

init_stmt: ID ASN expr
     | expr
     |

g_decl:  INT var_decl SEP
     |FLOAT var_decl SEP
     |INT8 var_decl SEP

decl_list: decl decl_list
```

```
    |{[]}
decl: datatype var_decl SEP

var_decl : var COMMA var_decl
    | var

var : ID { Var_Decl
    | ID LBRACKET DIGIT RBRACKET

datatype: INT
    | FLOAT
    | INT8
```

4. Project Plan

4.1. Process

The FVPL team followed a variant of **Extreme Programming** methodology. We followed weekly schedules to identify tasks for the week. The identified task pool was divided among the three members with weekly deliverables. We used mails for quick daily communication during the week to discuss on any issues, possible solutions for the issue, and also to take up new unplanned tasks. The turnaround time for solving minor issues was less than 24 hours and hence there was no slack period during development.

Each member of the team worked on part of every module of the project at some stage. This was planned on purpose which helped all members having complete understanding of the project.

4.2. Programming Style

We followed simple programming style giving importance to readability of the code. We used the following guidelines throughout the code.

- Easy to read function names
- Proper Indentation
- Line break after every function
- Grouping commonly used functions for reusability
- Used helper functions to take advantage of the power of recursion

Sample Code:

```
(*check if availble, if yes, then return the tuple*)
let symbol_detail sym =
        let rec find_sym = function []-> (NULL,"",0)
                                |(typename,var,size)::l -> if (var = sym) then
                                (typename,var,size) else find_sym l in
let k = find_sym !sym_table in
```

```
if get_valid k = "" then find_sym !g_sym_table
else k

let check_lhs v1 =
        let a = symbol_detail v1 in
if get_size a = -1 && !v_flag = 1 then
                        raise_error "Mismatch in LHS and RHS type"
                else if get_type a <> !cur_type then
                        if get_type a = Int8 && !cur_type= Int then ignore("")
                        else   raise_error "Mismatch in LHS and RHS data type"
```

4.3. Project Timeline

Date	**Module Completed**
24 Sep 2008	Language proposal
19 Oct 2008	LRM
19 Oct 2008	Lexer
30 Oct 2008	Parser
18 Nov 2008	AST/Symbol Table
29 Nov 2008	Code Generation
09 Dec 2008	Testing
09 Dec 2008	Project Report

4.4. Roles and Responsibilities

Gowri Kanugovi	LRM, action part of the parser, symbol table, Semantic analysis for local/global variable and function declarations, error handling, test case generation and documentation
Pratap Prabhu	CVS, action part of the parser, symbol table, AST, Semantic analysis for local/global variable and function declarations, function call validation, test script, test case generation and documentation
Ravindra Babu	Lexer, grammar part of parser, type checking in expressions, integrating compiler with external library and code generation, test case generation and documentation

4.5. Development Environment

Operating System:
We used Linux as our primary development environment but our compiler was tested on both Windows and Linux.

Language Used:
OCaml Lex is used to implement the scanner, OCaml Yacc is used to implement the parser, and the rest of the compiler is implemented in OCaml. The optimized library for vector operations is implemented using C++.

4.6. Tools Used

Version Control
CVS is used for version control and backing up our source code. The CVS repository resided on *cvs.sourceforge.net.*

Integrated Development Environment
We used the Eclipse IDE with OCaml plugin (OCalIDE) for development.

Testing
Shell script was written to automate the regression test suite.

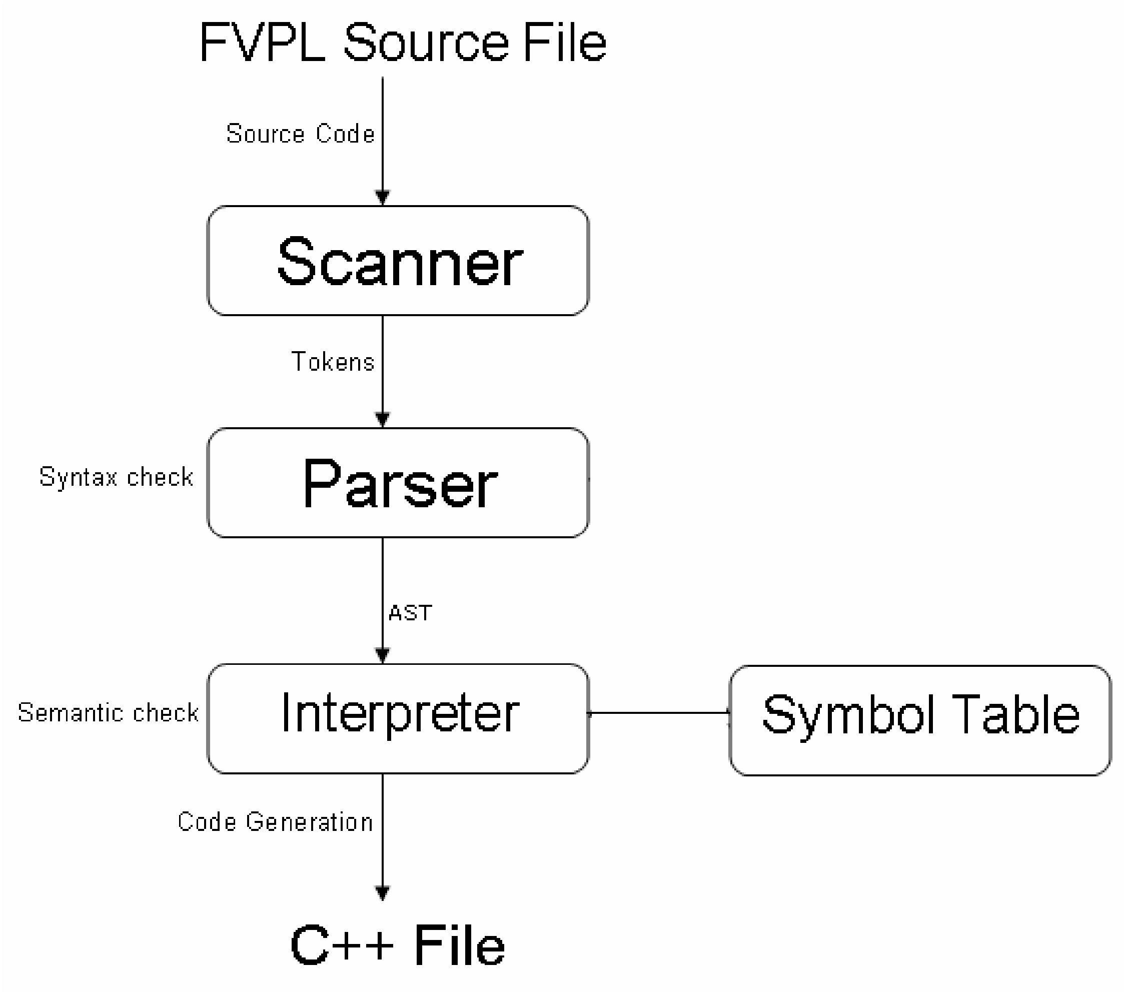
4.7. Project Log

Date	Tasks	Contributor
21 Sep 2008	Language Proposal	Gowri, Pratap, Ravindra
21 Sep 2008	CVS Setup	Pratap
05 Oct 2008	Initial Planning	Gowri, Pratap, Ravindra
19 Oct 2008	LRM	Gowri, Pratap, Ravindra
30 Oct 2008	Lexer and Grammar portion of Parser(Syntax)	Ravindra
07 Nov 2008	Actions added to grammar, Symbol Table added. Support for local/Global variables and functions. Check for undefined variables	Gowri, Pratap
12 Nov 2008	Data type check, Scalar/Vector type check. Different conventions in code generation for scalar and vector operations	Ravindra
18 Nov 2008	Function call check, matching actual arguments with formal parameters, break, continue and while added, support for initializing vectors added	Gowri, Pratap
19 Nov 2008	Integrating the C code generated, with SSE code, allocating memory for temporary buffer and passing the right size for vector operations	Ravindra
29 Nov 2008	SSE code added for arithmetic and logical operations	Ravindra
02 Nov 2008 -9 Dec 2008	Test Cases added	Gowri, Pratap, Ravindra
28 Nov 2008 -9 Dec 2008	Documentation	Gowri, Pratap, Ravindra
09 Dec 2008	Final Testing and Release	Gowri, Pratap, Ravindra

5. Architectural Design

5.1. Block Diagram

Given below is the graphical representation of the FVPL compiler. The description of each stage is given in the next section



5.2. Architecture Details

The FVPL compiler design consists of the following four main components

Lexer

The lexer scans the given input FVPL source code and converts into tokens based on a set of rules. This part of the module takes care of removing comments and white spaces.

Parser
The parser parses the input sequence of tokens and checks for syntax to be in accordance with the FVPL grammar thus producing the Abstract Syntax Tree (AST). The verified syntactically correct statements are passed for semantic analysis.

Interpreter
This is the abstract syntax tree walker which walks through the constructs verified by the parser and inserts the identified constants and identifiers into symbol table along with additional information to identify the type and size of buffer for later usage. The statements are checked for data type match, function call validation, identifier declaration before usage and various other semantics.

Code generation
This section includes 2 parts. The first part generates appropriate translation of vector operation into function calls, and passes on the other verified C++ statements unchanged to output. The second part of code generation is our C++ library consisting of optimized vector operation routines. The optimization primarily includes using SSE intrinsics to boost performance.

The code generated from FVPL is passed on to an optimizing C++ compiler to compile into executable file. Due to the usage of intrinsic instead of inline assembly or generating direct assembly instructions, the code is portable and can be compiled on any x86 based platform using a compiler with intrinsic support. The code is tested on windows using VC++ compiler and on Linux using g++ compiler but it should also work with minimal changes on Sun Solaris using sun studio compiler.

6. Testing

Testing is an important phase in software development. Not only does it reveal hidden bugs but also enables the developers to ensure that the program is behaving as designed. Any inconsistencies in FVPL during programming were unveiled in this phase and thus helped us create a robust compiler for the language.

Testing was done during all phases of the development. Whenever a new feature was added we first ensured that it works as designed. For this we created test cases i.e. example programs to test some part of the compiler. Only when the compiler generated the desired C++ code under all valid inputs, did we move to the next step of development.

6.1. Input program and generated code

In this section, we will see three example programs and the generated C++ code for each of them.

Input program-1
In this program we tested one of the most common programs, the *Fibonacci series* generation. This test also helped us test one of the looping constructs used in FVPL, *while* loop.
Consider the following program as the input program to the FVPL compiler to compute the first ten numbers in the Fibonacci series:

```
int
main () {
int i, j, tmp;
i=1;
j=1;

while(i<10){
        print (i);
        print (j);
        tmp = j;
        j = j+i;
        i = tmp;
        }

return 1;
}
```

Generated C++ for program-1
When the above program was passed to the FVPL compiler, it generates the following C++ code

```
#include <stub-print.h>
#include <stub.h>

int main ( void) {
 int i,j,tmp;
i = 1;
j = 1;
while ((i < 10)) {
print(i);
print(j);
tmp = j;
j = (j + i);
i = tmp;

}
return 1;
}
```

Since the syntax of FCPL closely resembles C/C++, the generated C++ code looks very similar to the source program itself.

Output of program-1

```
1
1
1
2
2
3
3
5
5
8
8
13
```

Input program-2

The following test case was used to test the addition of floating point vectors.

```
int main()
{
        float a[10], b[10];
        a = 5.0;
        b = a + a;
        print(a);
        print(b);
}
```

Generated C++ for program-2

Given below is the corresponding C++ code generated

```
#include <stub-print.h>
#include <stub.h>

int main ( void) {
        float a[10],b[10];
        copy(a, 5.0,10);
        copy(b, add((float*)alloca(10*sizeof(float)), a, a, 10),10);
        print(a,10);
        print(b,10);
}
```

This generated code reveals the working of the compiler behind the scene. The programmer had the ease to assign the sum of one vector with itself to another vector by simply saying *b=a+a,* but the compiler would interpret this statement to produce *copy(b,* *add((float*)alloca(10*sizeof(float)), a, a, 10),10),* the methods add and copy are part of the stub code included in the file *stub.h* which is given as a part of the include statement in the generated code.

The above test case helped us test the high level working of the addition of two vectors and also the internal working of the compiler with respect to the stub function copy and add.

Output of program-2

```
5.000000
5.000000
5.000000
5.000000
5.000000
5.000000
5.000000
5.000000
5.000000
5.000000
10.000000
10.000000
10.000000
10.000000
10.000000
10.000000
10.000000
10.000000
10.000000
10.000000
```

Input program-3

This program tests the initialization of vectors and also the various print statements.

```
int
main () {
        int a[10], b[20], c[10];
        float x[10];
        int k;

        k = 10;

        x = {1.0,2.0};
        a = {1,2,3,4,5};
        b=a;
        print (23);
```

```
            print (23.5);
            print (k);
            print (b);
            print ("hello world");

            print (x);
            return 1;
      }
```
Initialization of arrays with user defined values is also allowed in FVPL like shown above.

Generated C++ for program-3
The C++ code generated for program 3 is as below

```
      int main ( void) {
            int a[10],b[20],c[10];
            float x[10];
            int k;
            k = 10;
            float tmp[] = {1.0, 2.0,-999923.0};copy_vector(x, 10,tmp);}
            int tmp[] = {1, 2, 3, 4, 5,-999923};copy_vector(a, 10,tmp);}
            copy(b, a,10);
            print(23);
            print(23.5);
            print(k);
            print(b,20);
            print_string ("hello world");
            print(x,10);
            return 1;
      }
```

Output of program-3
```
      23
      23.5
      10
      1
      2
      3
      4
      5
      hello world
      1.000000
      2.000000
```

In addition to the above three test cases, we created **45** positive tests and **20** negative test cases to ensure testing of every aspect implemented and supported by our language. The test scripts provide a complete coverage of the compiler code.

6.2. Negative test cases and output

Error handling is an important task of any compiler. It is not only sufficient to test it the program is behaving as expected but it is also important to test that it is capable of detecting erroneous conditions in the program. Since we have emphasized on the input program, the generated code and also the desired output, we will not consider test cases where we introduced deliberate errors to ensure that the compiler catches those exceptions and prompts to the programmer the appropriate error message.

Negative test case-1

The following test case was used to ensure that the programmer has only one function by the name *main.* There is another test case which ensures that the program has at least one function by the name *main.*

```
int main () {
        int i;
        i=2+3+4;
        }

        int
        main () {
        int j;
        j=2+3+4;
}
```

Running the above program would yield the following message to the programmer:

Fatal error: exception Failure("Cannot have more than one main function")

This test case helped us conclude that our compiler produced the correct error message if it sees more than one main method in the program

Negative test case-2

This test case depicts the case when the program tries to use a variable which has not been defined yet. This is an important task for most compilers. The compiler should look up the symbol table whenever a variable is accessed to ensure that the variable has been declared before it is being used.

```
int main(){
        int a, b;
        a=2;
        c=a+1;
        b=3;
}
```

Running the above program would yield the following message to the programmer:

```
Fatal error: exception Failure("Encountered undefined symbol c")
```

Thus our compiler is able to handle cases as the one described above.

Negative test case-3
The following test case is related to type checking i.e. ensuring that the variable is assigned a value that it is capable of holding only. Assigning invalid values would result in a compiler error, making sure that our compiler does strict type-checking.

```
int main(){
        int i[10],k;
        k=0;
        i=k;
        k=i;
}
```

The above case depicts a scenario when a scalar variable (k) is assigned a vector (i). This is an invalid assignment and the compiler detects it by displaying the following message

```
Fatal error: exception Failure("Mismatch in LHS and RHS type")
```

Negative test case-4
This test case depicts the case when a function is defined to take a certain number of arguments but is invoked with a different number of arguments. The types of arguments passed to the function are also checked by the compiler and this is tested in a different test case.

```
int foo(int a, int b){
        return 1;
}

int main(){
        foo(2);
}
```

It is clear that the function foo is expecting two arguments but it is invoked with only one argument. The compiler throws the following exception:

```
Fatal error: exception Failure("number of params different")
```

Negative test case-5
This test case was written to ensure that in the global scope an identifier and function cannot have the same name. The FVPL compiler also checks for cases when the local identifier has the same name as a defined function. This is also an invalid declaration and is tested in a different test case.

```
int foo;
int foo() {
        return 1;
        }

int
main () {
        int foo1;
        foo1 = 0;
        return 1;
}
```

In the above program, clearly the name foo is used for both the global variable and a function. The compiler generates the following error message:

Fatal error: exception Failure("Cannot have function name same as global var")

6.3. Test automation

Given below is the test script used to run all the test cases automatically when invoked with the *make test* command. The script runs all the positive and negative tests from the respective folders, compares the produced output with the expected output. If they are the same then it displays SUCCESS for all the positive test cases and FAIL for all the negative test cases. This script was used for regression testing whenever any new feature was incorporated.

```
#!/bin/sh
# Author Pratap Prabhu
echo "------------ Testing FVPL success cases ------------"
for i in tests/success/*.input; do
  ARGS="$i" make exec 1>/dev/null; ./a.out > /tmp/out
  k=`basename $i .input`

  diff -w -i /tmp/out tests/success/$k.output > /dev/null
  if [ $? -eq 0 ]
   then
        echo "$i --------------- SUCCESS"
  else
        echo "$i --------------- FAIL"
  fi
```

```
done

 echo "\n\n------------ Testing FVPL failure cases ------------"
 for i in tests/failure/*.input; do
   ./fvpl < $i > /dev/null
   if [ $? -eq 0 ]
    then
         echo "$i --------------- SUCCESS"
   else
         echo "$i --------------- FAIL"
   fi
done
```

Below is an excerpt of the output produced by running the automated test script.

```
------------ Testing FVPL success cases ------------
tests/success/test-comments.input --------------- SUCCESS
tests/success/test-continue-break.input --------------- SUCCESS
tests/success/test-fibonacci.input --------------- SUCCESS
tests/success/test-file-input.input --------------- SUCCESS
tests/success/test-for.input --------------- SUCCESS
tests/success/test-func1.input --------------- SUCCESS
tests/success/test-func2.input --------------- SUCCESS
tests/success/test-global.input --------------- SUCCESS
tests/success/test-if.input --------------- SUCCESS
….
….
….

------------ Testing FVPL failure cases ------------
Fatal error: exception Failure("Cannot have more than one main function")
tests/failure/test-2mains.input --------------- FAIL
Fatal error: exception Failure("number of params different 2")
tests/failure/test-func1.input --------------- FAIL
Fatal error: exception Failure("Type of arguments not matching defined function3")
tests/failure/test-func2.input --------------- FAIL
Fatal error: exception Failure("Return type should be an integer")
tests/failure/test-func3.input --------------- FAIL
Fatal error: exception Failure("Cannot have function name same as global var")
tests/failure/test-id-same-as-function.input --------------- FAIL
Fatal error: exception Failure("Unable to find main function")
tests/failure/test-nomain.input --------------- FAIL
Fatal error: exception Failure("Mismatch in LHS and RHS data type")
tests/failure/test-type-checking1.input --------------- FAIL
….
….
```

6.4. Results

We performed two performance tests with FVPL vs. GCC compiler.

The first sample adds a small number indicating brightness factor to a 5 MB image. This is a common operation used while increasing or decreasing brightness of an image in a image editor.

The second sample performs two operations, the first one is to mask off a part of the image based on the given mask value, followed by multiply each pixel in the image with another input image.

Both the above examples are written in C++ and FVPL and both the versions are compiled with g++, turning on the optimization flags. The results are as follows.

Time Taken by C++ code to add brightness to 5 MB image=416.994ms

Time Taken by FVPL to add brightness to 5 MB image= 212.091ms

Time Taken by C++ code to mask and multiply pixel(5 MB image)=568.638ms

Time Taken by FVPL code to mask and multiply pixel(5 MB image)=259.936ms

It is clear from the above example that the performance boost is around 2X. Although, this is good improvement with respect to the C++ program, the performance obtained by hand optimized version is clearly higher due to a variety of reasons as explained in the next section.

7. Lessons Learned

Gowri Kanugovi
Functional programming is a powerful tool to design a compiler in a fast, concise and efficient manner. Decide on the semantics of the language in the early stages and stick to it throughout the implementation. We effectively met all the deadlines we set and this helped us finish the project on time implementing all the features we proposed to design.

Pratap Prabhu
An important aspect which led to the smooth development of FVPL was the division of tasks in an independent manner. This way we did not have to depend on the completion of some task to carry on with individual development. Overall all the features we proposed to implement were implemented in the available time. Good co-ordination among the team memebers helped complete the project without any major hurdles.

Ravindra Babu
FVPL is a well planned and executed project, we planned to implement only features that are possible to implement within the available time and hence we proposed only those features in our initial proposal. We did not slip any intermittent milestones and completed the project on time. Although, the performance of our compiler is better than GCC in our tests, this performance is not as good as the hand optimized SSE code. This is due to the allocation of temporary buffers and redundant copy of data to avoid corrupting the input buffers.

Team
Given unlimited time, we would have gone a step ahead and implemented features like auto 16 byte alignment of the data and inter procedural optimization to eliminate redundant loads and stores which would have given further boost in performance. The other aspects that are missing in our compiler are support for multi-dimensional arrays for matrix operations and operating on partial arrays. With support for these two features our compiler will be useful in various domains like matrix operations, string processing and cryptography.

We chose performance as a metric for our compiler since this is the one key metric to create demand and sell a new compiler while retaining the popular C programming style with an added advantage of writing concise code for vector operations.

8. Appendix

8.1. Source Code

```
/*    Scanner code-scanner.mll
    Author: Ravindra Babu   */

{ open Parser
let fullstr = ref "";;
}

let digit=['0'-'9']
let float_digit = ['-''+']?['0'-'9']+['.']['0'-'9']* (['e''E'](['-''+']?(['0'-'9']+)?))?
| ['-''+']?['0'-'9']*['.']['0'-'9']+ (['e''E'](['-''+']?(['0'-'9']+)?))?
| ['-''+']?['0'-'9']*['.']?['0'-'9']+ (['e''E']['-''+']?(['0'-'9']+)?)

let string = "(['0'-'9''a'-'z''A'-'Z'])*"

rule token = parse
| eof               { EOF }
| "print"           { PRINT }
| "if"              { IF}
| "else"            { ELSE}
| "for"             { FOR}
| "while"           { WHILE }
| "return"          { RETURN}
| "continue"        { CONTINUE}
| "break"           { BREAK}
| "int"             { INT}
| "float"           { FLOAT}
| "int8"            { INT8}
| "void"            { VOID}
| '"'               { fullstr:=""; str_quote lexbuf }
| '('               { LPARAN}
| ')'               { RPARAN}
| '{'               { LBRACE}
| '}'               { RBRACE}
| '['               { LBRACKET}
| ']'               { RBRACKET}
| "++"              { PLUSPLUS }
| "--"              { MINUSMINUS }
| '+'               { PLUS}
| '-'               { MINUS}
| '*'               { MUL}
| '/'               { DIV}
| '|'               { OR}
| '&'               { AND}
| '^'               { XOR}
| '~'               { NOT}
| '='               { ASN}
| ';'               { SEP}
| ','               { COMMA}
| '<'               { LT}
| '>'                   { GT}
| "<="                  { LTEQ}
| ">="                  { GTEQ}
| "=="                  { EQ}
| "!="                  { NOTEQ}
| float_digit as f_digit { FDIGIT (f_digit) }
| digit+  as digi   { DIGIT(int_of_string(digi)) }
| ['a'-'z' 'A'-'Z']+(digit|['a'-'z''A'-'Z''_'])* as id_str { ID(id_str)}
| "/*"                  { multi_comment lexbuf }
| "//"                  { single_comment lexbuf }
| _                     { token lexbuf }

and single_comment = parse
 "\n" {token lexbuf }
 |_   { single_comment lexbuf }

and multi_comment = parse
  "*/" { token lexbuf }
| _    { multi_comment lexbuf }

and str_quote = parse
  '"' { STRING (!fullstr) }
  | _ as str { fullstr:= !fullstr^(String.make 1 str); str_quote lexbuf }


{

}
```

/*   Parser Code-parser.mly
Authors: Gowri Kanugovi, Pratap Prabhu, Ravindra Babu */

```
%{ open Ast
   let parse_error msg =
      print_endline (msg);
      flush stdout
 %}

%token EOF  IF  ELSE  WHILE  FOR  RETURN  LBRACKET   RBRACKET
  CONTINUE  BREAK INT  FLOAT INT8
  VOID  LPARAN  RPARAN  LBRACE  RBRACE  PLUS  MINUS  MUL  DIV
  OR  AND  XOR  NOT  ASN  SEP  COMMA  LT  GT  LTEQ  GTEQ  EQ  NOTEQ  RETURN
      PRINT QUOTES PLUSPLUS MINUSMINUS WHILE

%token PLUS MINUS TIMES DIVIDE EOF SEQ ASN
%token <string> ID
%token <int> DIGIT
%token <string> FDIGIT
%token <string> STRING

%left COMMA
%right ASN
%left OR AND XOR
```

```
%left EQ NOTEQ
%left LT GT LTEQ GTEQ
%left PLUS MINUS
%left TIMES DIVIDE
%left PLUSPLUS MINUSMINUS

%start program
%type <Ast.program> program


%%

program:
                    { [],[] } /* pass a tuple to interpreter, first = global decs, second = fun defs */
                    | g_decl  program  { ($1::fst $2), snd $2 }
                    | fun_def program { fst $2, ($1::snd $2) }


fun_def : INT ID LPARAN fun_def_args_list RPARAN LBRACE decl_list stmt_list RBRACE {
                              { fun_name = $2;
                                        arg_list = $4;
                                        decl_list = $7;
                                        stmt_list = $8};}

fun_def_args_list : fun_def_args_list COMMA  fun_def_args {$3::$1}
                    | fun_def_args {[$1]}
                    | {[]}


fun_def_args:   datatype ID {FunArg ($1,$2,-1)}
          | datatype ID LBRACKET DIGIT RBRACKET{ FunArg ($1,$2,$4) } /* For arrays */


/* first part of func decl list */

stmt_list :
     { [] }
     |stmt stmt_list{$1::$2}


stmt:   IF LPARAN expr RPARAN LBRACE stmt_list RBRACE elsetag { If ($3,$6,$8) } /* If
     statement */
          | ID ASN stmt { Assign($1,$3,1) } /* assignment */
          | FOR LPARAN init_stmt SEP opt_expr SEP init_stmt RPARAN LBRACE stmt_list
     RBRACE {  For($3,$5,$7,$10) } /* looping (**)*/
          | WHILE LPARAN expr RPARAN LBRACE stmt_list RBRACE {While ($3,$6) }
          | expr SEP{ Expr($1,1) } /* Just an expression, i++ */
          | PRINT LPARAN  STRING  RPARAN SEP {Print_string ($3) } /* Print string */
          | PRINT LPARAN ID RPARAN SEP {  Print ($3) } /* print identifiers */
          | PRINT LPARAN DIGIT RPARAN SEP { Print_const (string_of_int($3), Int) } /* print
     numbers */
          | PRINT LPARAN FDIGIT RPARAN SEP { Print_const (($3), Float) } /* print numbers */
          | ID LPARAN arg_list RPARAN SEP {  FunCall ($1, $3) } /* Function call . Arguments are
     expr list */
          | RETURN opt_bracket_l expr opt_bracket_r SEP { Return ($3) }
          | CONTINUE SEP { Continue }
```

```
            | BREAK SEP { Break }

opt_expr: expr { $1 }
      | { ExprEmpty}

opt_bracket_l: LPARAN {}
                        | {}

opt_bracket_r: RPARAN {}
                        | {}

elsetag: ELSE LBRACE stmt_list RBRACE { $3 }
                        | {[]}

arg_list:  expr COMMA arg_list { $1::$3 }
                                    |expr{[$1]}
                                    | {[]}

/* handle expressions */
expr:
      expr PLUS term { Binop($1,Plus,$3) }
      | expr MINUS term {Binop ($1,Minus,$3) }
      | expr OR term {Binop ($1, Or,$3)}
      | expr AND term {Binop ($1, And,$3)}
      | expr XOR term {Binop ($1, Xor,$3)}
      | expr LT term {Binop ($1, Lt,$3)}
      | expr GT term {Binop ($1, Gt,$3)}
      | expr LTEQ term{Binop ($1, Lteq,$3)}
      | expr GTEQ term{Binop ($1, Gteq,$3)}
      | expr EQ term{Binop ($1, Eq,$3)}
      | expr NOTEQ term{Binop ($1, NotEq,$3)}
      | ID PLUSPLUS {Uop ($1,PlusP)}
      | ID MINUSMINUS {Uop($1,MinusM)}
      | LBRACE int_id_list RBRACE { IntArray ($2) }
      | LBRACE f_id_list RBRACE {FloatArray ($2) }
      | term { $1 }
      | STRING { String($1) }


term: term MUL atom { Binop($1, Mul, $3) }
   | term DIV atom { Binop($1, Div, $3) }
   | atom { $1 }

f_id_list: FDIGIT COMMA f_id_list { $1::$3 }
      | FDIGIT {[$1]}

int_id_list: DIGIT COMMA int_id_list { $1::$3 }
      | DIGIT {[$1]}

atom: DIGIT {Literal($1) }
   | FDIGIT { FloatL ($1) }
   | ID { Id($1) }


init_stmt: ID ASN expr {Assign($1,Expr($3,0),0)} /* comma separated values */
      | expr { Expr ($1,0) }
      | {Empty}

/* Handle decl_list (& globals) and derivatives from now on */

g_decl:  INT var_decl SEP { Var_list (Int,$2) }
      |FLOAT var_decl SEP { Var_list(Float, $2) }
      |INT8 var_decl SEP { Var_list(Int8,$2) }

decl_list: decl decl_list { $1::$2 } /* variable declaration */
      |{[]}

decl: datatype var_decl SEP { Var_list ($1,$2) }

var_decl : var COMMA var_decl { $1::$3 }
      | var{ [$1] }


var : ID { Var_Decl ($1,-1) } /* -1 indicates no dimension */
      | ID LBRACKET DIGIT RBRACKET{ Var_Decl ($1,$3) } /* For arrays */


datatype:          INT{Int }
      | FLOAT{Float }
      | INT8{Int8 }
```

/* AST walker and code generation – interpreter.ml
Authors: Gowri Kanugovi, Pratap Prabhu, Ravindra Babu */

```
open Ast
open Symbol

exception SemanticError

let error_msg = ref ""

let v_flag = ref 0;;
let loop_flag = ref 0;;
let cur_type = ref NULL;; (*to keep track of the data type in the current expr*)

(* function used to handle error messages *)
let raise_error msg =
   error_msg := msg;
   raise SemanticError

let get_type = function (t, v, s)-> t
let get_vector = function (t, v, s)-> (s > 0)
let get_size = function (t,v,s)-> s
let size = ref 0

(* Function to determine the data type of the current symbol *)
let dtype_eval = function
   Int -> "int"
   | Int8 -> "unsigned char"
   | Float -> "float"
   | _ -> raise_error "Invalid data type"

(* Function for type checking of two variables *)
let eval_type v1 v2 f op=
   let a = symbol_detail v1
      in let b = symbol_detail v2
            in if(get_vector a = false &&
           get_vector b = false &&
           get_type a = get_type b) then
                   "(" ^ v1 ^ " "^op^" " ^ v2 ^")"
            else if (get_type a = get_type b) then
                   begin
                   v_flag:=1;
                   if(get_size a = -1 || get_size a = -2)
               then size:=get_size b
                   else if(get_size b = -1 || get_size b = -2)
               then size:=get_size a
                   else if( get_size a > get_size b)
               then size:= get_size b
                   else size := get_size a;
                   f^"(("^dtype_eval (get_type a)^
              "*)alloca("^string_of_int !size
                   ^"*sizeof("^dtype_eval (get_type a)
              ^")), " ^ v1 ^ ", "
                   ^ v2 ^", "^string_of_int !size^")"
                   end
            else
            raise_error "data type mismatch"

(* Function to check that a scalar is not assigned a vector or if there is a mismatch of the data
      types *)
let check_scalar v1 v2 op=
   let a = symbol_detail v1
      in let b = symbol_detail v2
            (*Remove the type check for allowing data type conversion or promotion*)
            in if(get_vector a = false &&
           get_vector b = false &&
           get_type a = get_type b)
           then "(" ^ v1 ^ " "^op^" " ^ v2 ^")"
            else
            raise_error "Invalid operation or data type mismatch"

(* Function for type checking variables on the lhs and rhs of an expression *)
let check_lhs v1 =
   let a = symbol_detail v1
      in if get_size a = -1 && !v_flag = 1 then
            raise_error "Mismatch in LHS and RHS type"
      else if get_type a <> !cur_type then
            if get_type a = Int8 && !cur_type= Int then ignore("")
            else
            raise_error "Mismatch in LHS and RHS data type"

(* Function which sets the vectro flag if the argument is a vector *)
let check_var v1 =
   let a = symbol_detail v1
```

```
      in if get_size a > 0 then
            begin
            v_flag:=1;
            cur_type:= get_type a
            end
      else
            v_flag := 0;
            cur_type:= get_type a

(* Function for generating C++ code for printing floating point vectors *)
let print_float_array x =
   let rec p xlist str = match xlist with
      [] -> ""
      |[x] -> (str^ (x)^"}")
      |hd::tl -> let x1 = p tl (str^(hd)^", ") in x1;
   in
      p x "{"
      ;;

(* Function for generating C++ code for printing integer vectors *)
let print_int_array x =
   let rec p xlist str = match xlist with
      [] -> ""
      |[x] -> (str^string_of_int (x)^"}")
      |hd::tl -> let x1 = p tl (str^string_of_int(hd)^", ") in x1;
   in
      p x "{"
      ;;
let array = ref 0;;

let const_size = -2

(* Function to walk expressions and output relevant code*)
let rec expr_eval = function
        ExprEmpty -> "";
      | Literal(x) -> insert_symbol_const Int (string_of_int(x)) const_size;
                      v_flag:=0 ; cur_type:= Int; string_of_int(x);
      | FloatL (x) -> insert_symbol_const Float x const_size;
                      v_flag:=0;cur_type:=Float ; x;
      | String (x) -> v_flag:=0; cur_type:=StrType; ("\""^x^"\"");
      | Id(x) -> lookup_symbol(x); check_var x; x
      | IntArray (x) -> array:=1; cur_type:= Int;
                        v_flag:= 1; let x1 = print_int_array (x) in x1
      | FloatArray (x) -> array:=1; cur_type:= Float; v_flag:= 1;
                          let x1 = print_float_array (x) in x1
      | Uop (id, op1) -> lookup_symbol id; let a = symbol_detail id
                         in let vf = get_vector a in
                                        if vf = true then
                                              raise_error "Cannot use unary operators on vectors";
                                        if op1 = PlusP then
                                              (id^"++")
                                              else
                                              (id^"--");
      | Binop (x,op,y) ->
            let v1 = expr_eval x and v2 = expr_eval y in
            match op with
```

```
                 Plus -> eval_type v1 v2 "add" "+"
                |Minus -> eval_type v1 v2 "sub" "-"
                | Mul -> eval_type v1 v2 "mul" "*"
                | Div -> eval_type v1 v2 "div" "/"
                | Or -> eval_type v1 v2 "bit_or" "|"
                | And -> eval_type v1 v2 "bit_and" "&"
                | Xor -> eval_type v1 v2 "bit_xor" "^"
                | Lt -> check_scalar v1 v2 "<"
                | Gt -> check_scalar v1 v2 ">"
                | Lteq -> check_scalar v1 v2 "<="
                | Gteq -> check_scalar v1 v2 ">="
                | Eq -> check_scalar v1 v2 "=="
                | NotEq -> check_scalar v1 v2 "!="



(* Function to evaluate statements *)
let expr_to_tuple expr = (*print_string "CALL";*)
  ignore (expr_eval expr);
  if !v_flag = 1 then
    (0, !cur_type)
  else
    (-1, !cur_type)
    ;;

  let rec stmt_eval =
    let rec exprlist_eval = function
    [x] -> let v1 = expr_eval x in v1
    |x::y -> let v1 = exprlist_eval y in
          let v2 = expr_eval x in
          let v3 = (v2 ^ "," ^ v1)in v3
    | [] -> "" in
  function
  Dummy_str(x) ->  print_endline (x);
  | Assign (x,y1, iflag) ->  (match y1 with
                                  Expr (y, _) ->
                                  lookup_symbol x;  v_flag := 0;
               array:= 0; cur_type:= NULL;
               let v1 = expr_eval y in
                if !v_flag = 0 then
                begin
                        check_lhs x;
                        let rec a =symbol_detail x
                        in
                         if (get_size a > 0) then
                           print_string ("copy(" ^ x ^ ", "^v1^","^string_of_int(get_size a)^");\n")
                        else
                        begin
                           print_string(x ^" = "^ v1 );
                           if iflag = 1 then
                           print_string (";\n")
                        end

                end
                else
                begin
```

```
                                check_lhs x;
                                if !array = 1 then
                                        if !cur_type = Int then
                                                let a = symbol_detail x in print_string ("{int tmp[] = " ^
                                                    v1 ^ ";" ^"copy_vector(" ^ x ^ ", "
                                                                ^string_of_int(get_size a) ^",tmp);}\n");
                                        else
                                        let a = symbol_detail x in print_string ("{float tmp[] = " ^
                                                        v1 ^ ";" ^"copy_vector(" ^ x ^ ", "
                                                        ^string_of_int(get_size a) ^",tmp);}\n");

                                else
                                begin
                                        let a = symbol_detail x
                                        in
                                        let b = symbol_detail v1
                                        in
                                        if ( get_size b > 0) then size:= (get_size b);
                                        if(!size < 1) then size:=(get_size a)
                                        else if( (get_size a) > (!size)) then size:= !size
                                        else size := (get_size a);
                                        print_string ("copy(" ^ x ^ ", "^v1^","^string_of_int(!size)^");\n");
                                end
                        end
                        | FunCall (_,_) -> lookup_symbol x; let a = symbol_detail x in
                                if get_type a = Int && get_vector a = false then
                                begin
                                        print_string (x^" = ");
                                        stmt_eval y1;
                                end
                                else
                                        raise_error (x^" should be of type Int");
                        | _ -> raise_error ("Invalid assignment");)
        | Return (x) -> let v1 = expr_eval x in
                        if !cur_type = Int then
                                print_string ("return " ^ v1 ^";")
                        else
                                raise_error "Return type should be an integer"
        | FunCall (fname, expr) -> if f_find_sym fname = false then
                                        raise_error ("Cannot find function "^fname)
                                  else
                                        let k1 = List.map expr_to_tuple expr in
                                        fun_arg_type_eval fname (List.length expr) k1;
                                        let v1 = exprlist_eval expr in print_string (fname^"("^v1^");\n");
        | Expr(x,iflag) -> let v1 = expr_eval x in print_string (v1);
                                        if iflag = 1 then
                                                        print_string (";\n")
        | If (expr1, stmt1, stmt2) -> let v1 = expr_eval expr1 in print_endline ("\nif (" ^ v1 ^ ") {");
                                                                List.iter stmt_eval stmt1;
                                                                print_endline ("}");
                                                                if List.length stmt2 != 0 then
                                                                begin
                                                                        print_string(" else {" );
                                                                        List.iter stmt_eval stmt2;
                                                                        print_endline ("}\n");
                                                                end
```

```
  | For (stmt1, expr1, stmt2, stmt_lst) -> loop_flag:=1;print_string ("for (");
                                            stmt_eval stmt1; print_string ";";
                                            let e1 = expr_eval expr1 in print_string (e1 ^ ";");
                                            stmt_eval stmt2; print_string (")\n {");
                                            List.iter stmt_eval stmt_lst;
                                            print_endline ("}\n"); loop_flag:=0;
  | While (expr1, stmt_list) -> loop_flag := 1;print_string ("while (");
                          let e1 = expr_eval expr1 in print_string (e1^") {\n");
                          List.iter stmt_eval stmt_list;
                          print_string ("\n}\n");loop_flag:=0;
  | Print (x) -> lookup_symbol x; let x1 = symbol_detail x in
                          let x3 = string_of_int (get_size x1) in
                          if get_vector x1 = false  then
                                  print_string ("print("^x^");\n")
                          else
                                  print_string ("print("^x^"," ^ x3 ^");\n")
  | Print_const (x, y) -> ignore(dtype_eval y);
                                  print_string ("print("^x^");\n");
  | Print_string (x) -> print_string ("print_string (\""^x^"\");");
  | Continue -> if !loop_flag = 1 then
                          print_endline("continue;")
                  else
                          raise_error "Misplaced continue statement. Not within a loop";
  | Break -> if !loop_flag = 1 then
                          print_endline("break;")
                  else
                          raise_error "Misplaced break statement. Not within a loop";
  | Empty ->                      ignore();
;;

(* Function to evaluate declarations *)
let rec decl_eval =
  let vdecl_str dtype arg = match arg with
  Var_Decl (x, y) -> if y != -1 then
                                  begin
                                    insert_symbol dtype x y;
                                          x ^ "[" ^ string_of_int(y) ^ "]";
                                  end
                          else
                          begin
                                  insert_symbol dtype x y;
                                  x
                          end
  in
  let rec vdecl_eval dtype = function
  [x] -> let v1 = vdecl_str dtype x in v1
  |x::y -> let v1 = vdecl_eval dtype y in let v2 = vdecl_str dtype x in
    let v3 = (v2 ^ "," ^ v1)in v3
  | [] -> "" in
  function
  Var_list (dtype,var_list) -> let dstr = dtype_eval dtype in
                           let vstr = vdecl_eval dtype var_list in
                           print_endline (dstr ^" "^ vstr ^ ";")

let fun_size = 0
let find_size = function
      (dtype, id, size)-> size

let fun_type_list = ref [];;

(* Function to evaluate function argument *)
let rec fun_eval_list_print =
   let fun_eval = function
   FunArg (dtype, id, size) -> (*print_string "FUNARG ";*) let vbool = if (size > 0) then 0
                                 else -1 in fun_type_list := (vbool,dtype)::!fun_type_list;
                                 insert_symbol dtype id size; let dstr = dtype_eval dtype in
                                 if size = -1 then (dstr ^ " " ^ id ^ " ")
                                 else (dstr ^ " " ^ id ^ "["^string_of_int size^"] ") in
   function
   [x] -> let v1 = fun_eval x in print_string (v1)
   |hd::tl -> let v1 = fun_eval hd in fun_eval_list_print tl; print_string ("," ^ v1)
   | [] -> print_string ("void")


(* Function to eval functions *)
let fun_put fundecl =
  clear_symtable();  global_scope:= 0; fun_type_list := [];
  print_string ("int " ^ fundecl.fun_name ^ " ( " );
   fun_eval_list_print fundecl.arg_list;
   f_insert_sym fundecl.fun_name (List.length fundecl.arg_list) !fun_type_list; (* fun name, number
      of args, list of type of args *)
   print_string (") {\n "); (* arguments *)
   List.iter decl_eval fundecl.decl_list; List.iter stmt_eval fundecl.stmt_list; (*declarations  &
      statements*)
   print_endline ("\n}\n\n");;

(* Function to find the main function *)
let find_main count fundecl =
   if fundecl.fun_name = "main" then
      count + 1
   else
      count

(* Begin semantic analysis *)
let eval (gdecl,program) =
   let c =
   List.fold_left find_main 0 program in
   if c = 0 then
      raise_error "Unable to find main function"
   else if c > 1 then
      raise_error "Cannot have more than one main function"
   else
      begin
            global_scope:= 1;
            print_endline "#include <stub-print.h>";
            print_endline "#include <stub.h>\n";
            List.iter decl_eval gdecl; global_scope:= 0;
            List.iter fun_put program
      end
```

```
/*    Symbol Table – symbol.ml
      Authors: Gowri Kanugovi, Pratap Prabhu */

      open Ast

      exception SemanticError

      let global_scope = ref 0;; (* 0 = local scope, 1 = global scope *)
      let error_msg = ref "";;

      let raise_error msg =
        error_msg := msg;
        raise SemanticError

      let sym_table = ref [(NULL,"",0)];;
      let g_sym_table = ref [(NULL,"",0)];;

      (* (-1, type) -> -1 indicates scalar *)
      let f_sym_table = ref [ ("fread_float", 3, [(-1,StrType);(0,Float);(-1,Int)]);
                              ("fread_int", 3, [(-1,StrType);(0,Int);(-1,Int)]);
                              ("fwrite_float", 3, [(-1,StrType);(0,Float);(-1,Int)]);
                              ("fwrite_int", 3, [(-1,StrType);(0,Int);(-1,Int)]);
                              ("copy_vector", -1, [(-3,NULL)]); ("",0,[(-3,NULL)])];;

      let dtype_eval_d = function
        Int -> "int"
        | Int8 -> "int8"
        | Float -> "float"
        | _ -> raise_error "Invalid data type"
        ;;

      let debug_tuple (vflag, dtype) =
        let x1 = dtype_eval_d dtype in
              print_string ("VLAG="^string_of_int(vflag)^"  "^x1^" \n")
        ;;

      let fun_arg_type_eval fname alen exprlist =
        let rec fun_arg_type_eval_each = function
                        [] -> raise_error "EXCEPTION!!"
                        | (funame, asize, typelist)::l ->

                        if (fname=funame) then
                          begin
                          if (asize != -1) then (* variable number of args*)
                            begin
                            if (asize != alen) then
                              raise_error ("number of params different " ^
                              string_of_int (List.length typelist));
                              if (( typelist) <> (exprlist)) then
                              begin
                              (*List.iter debug_tuple exprlist;*)
                              raise_error ("Type of arguments not matching defined function" ^
                              string_of_int(List.length exprlist))
                              end
                          end (* asize!= -1*)
                        end
                      else fun_arg_type_eval_each l
                      in
                    fun_arg_type_eval_each !f_sym_table
      ;;

let f_find_sym fname =
  let rec find_sym = function [] -> false
        |(funame,_,_)::l -> if (fname = funame) then true else find_sym l
        in
    find_sym !f_sym_table
    ;;


let f_insert_sym fname num_arg type_arg_list =
  let rec g_find_sym = function [] -> false
        |(typename,var,size)::l -> if (var = fname) then true else g_find_sym l
  in
        if g_find_sym !g_sym_table = true then
                raise_error "Cannot have function name same as global var";
        if f_find_sym fname = true then (* Check if function already exists *)
                ignore(raise_error ("Redeclaration of id similar to global function "^fname))
        else
                ignore (f_sym_table:= (fname, num_arg, type_arg_list)::!f_sym_table)




let insert_symbol dtype sym size =
let rec find_sym = function [] -> false
        |(typename,var,size)::l -> if (var = sym) then true else find_sym l
        in
  if f_find_sym sym = true then
    raise_error "Cannot have ID same name as function";
  if !global_scope = 0 then (* local sym table *)
    begin

        if find_sym !sym_table = true then (* Check if symbol already exists *)
        ignore(raise_error ("Redeclaration of local id "^sym))
    else
        ignore(sym_table:=  (dtype,sym,size)::!sym_table);
    end
  else
    begin
        if find_sym !g_sym_table = true then (* Check if symbol already exists in global scope *)
        ignore(raise_error ("Redeclaration of global id "^sym))
    else
        ignore(g_sym_table:=  (dtype,sym,size)::!g_sym_table);
    end


(* Insert const values into global scope, unique instance of each value *)
let insert_symbol_const dtype sym size =
let rec find_sym = function [] -> false
        |(typename,var,size)::l -> if (var = sym) then true else find_sym l
        in
        if find_sym !g_sym_table != true then (* Check if symbol already exists *)
        ignore(g_sym_table:=  (dtype,sym,size)::!g_sym_table)


let lookup_symbol sym = (* error if sym does not exist *)
  let rec find_sym = function [] -> false
        |(typename,var,size)::l -> if (var = sym) then true else find_sym l
  in
  let k = find_sym !sym_table  in
  if k = false then
     begin
        let j = find_sym !g_sym_table in
        if j = false then
                ignore (raise_error ("Encountered undefined symbol "^sym))
     end



let get_valid = function (t, v, s)-> v

(*check if availble, if yes, then return the tuple*)
let symbol_detail sym =
  let rec find_sym = function []-> (NULL,"",0)
                |(typename,var,size)::l -> if (var = sym) then (typename,var,size) else find_sym l
in let k = find_sym !sym_table
in
  if get_valid k = "" then find_sym !g_sym_table
  else k


let clear_symtable () =
  sym_table:= [(NULL,"",0)];;
```

/* Interface file- ast.mli
Author: Pratap Prabhu */

```
type uop =
  PlusP
  | MinusM


type operator =
  Plus
     |Minus
     | Mul
     | Div
     | Or
     | And
     | Xor
     | Lt
     | Gt
     | Lteq
     | Gteq
```

```
      | Eq
      | NotEq


type expr =
   Literal of int
   | FloatL of string
   | Id of string
   | IntArray of int list
   | FloatArray of string list
   | Binop of expr * operator * expr
   | Uop of string * uop
   | String of string
   | ExprEmpty

type datatype =
   Int
   |Int8
   | Float
   | NULL
   | Function (* Type with which functions are stored in global sym table *)
   | StrType

type var =
   Var_Decl of string * int


type decl =
   Var_list of datatype * var list

 type stmt =
       Expr of expr * int
      |Return of expr
      | If of expr * stmt list * stmt list
      | For of stmt * expr * stmt * stmt list
      | While of expr * stmt list
      | Dummy_str of string
      | Assign of string * stmt * int
      | FunCall of string * expr list
      | Continue
      | Break
      | Print of string
      | Print_const of string * datatype
      | Print_string of string
      | Empty

type funarg =
      FunArg of datatype * string * int


type fun_def = { (* ret_type is always an int *)
   fun_name: string;
   arg_list: funarg list;
   decl_list: decl list;
   stmt_list: stmt list;
   }
```

```
type program = decl list * fun_def list
```

/* **Compiler main file– fvp.ml***/

```
open Ast

let _ =
try
  let lexbuf = Lexing.from_channel stdin in
  let program = Parser.program Scanner.token lexbuf in
  (Interpret.eval program);
 with End_of_file -> exit 0
 | Parsing.Parse_error -> ignore(failwith "Parser syntax error"); exit 1
 | Interpret.SemanticError -> ignore(failwith !Interpret.error_msg); exit 1
 | Symbol.SemanticError -> ignore(failwith !Symbol.error_msg); exit 1
```

/* **Stub code file – stub.h**
Author: Ravindra Babu */

```
#include <iostream>
#ifdef LINUX
#include <pmmintrin.h>
#else
#include<intrin.h>
#endif

typedef unsigned char int8;


int8 sat(int a)
{
    if(a > 255) return 255;
    else if(a < 0) return 0;
    return a;
}

//SSE add API
int8* add(int8 *dst, const int8* src1, const int8 *src2, int size)
{
    int count = size / 16 ;

    __m128i* xmm1 = (__m128i*)src1;
    __m128i* xmm2 = (__m128i*)src2;
    __m128i* xmm3 = (__m128i*)dst;
    __m128i XMM1, XMM2;

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM2 = _mm_loadu_si128(xmm2);
        XMM1 = _mm_adds_epu8(XMM1, XMM2);
        _mm_storeu_si128(xmm3, XMM1);
        xmm1++; xmm2++; xmm3++;
    }
    //to handle fall back
    for(int i = count * 16 ; i < size ; i++)
    dst[i] = sat(src1[i] + src2[i]);

    return dst;
}

int8* add(int8 * dst, const int8* src1, const int8 src2, int size)
{
    int count = size / 16;

    __m128i* xmm1 = (__m128i*)src1;
    __m128i* xmm2 = (__m128i*)dst;
    __m128i XMM1, XMM2;
    XMM2 = _mm_set1_epi32(src2);

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM1 = _mm_adds_epu8(XMM1, XMM2);
        _mm_storeu_si128(xmm2, XMM1);
        xmm1++; xmm2++;
    }
    //to handle fall back
    for(int i = count * 16 ; i < size ; i++)
    dst[i] = sat(src1[i] + src2);

    return dst;
}
int8* add(int8 *dst, const int8 src1, const int8* src2, int size)
{
    return add(dst, src2, src1, size);
}
int* add(int *dst, const int* src1, const int *src2, int size)
{
    int count = size / 4 ;

    __m128i* xmm1 = (__m128i*)src1;
    __m128i* xmm2 = (__m128i*)src2;
    __m128i* xmm3 = (__m128i*)dst;
    __m128i XMM1, XMM2;

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM2 = _mm_loadu_si128(xmm2);
        XMM1 = _mm_add_epi32(XMM1, XMM2);
        _mm_storeu_si128(xmm3, XMM1);
        xmm1++; xmm2++; xmm3++;
    }
    //to handle fall back
    for(int i = count * 4 ; i < size ; i++)
    dst[i] = src1[i] + src2[i];

    return dst;
}
```

```
int* add(int * dst, const int* src1, const int src2, int size)
{
    int count = size / 4;

    __m128i* xmm1 = (__m128i*)src1;
    __m128i* xmm2 = (__m128i*)dst;
    __m128i XMM1, XMM2;
    XMM2 = _mm_set1_epi32(src2);

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM1 = _mm_add_epi32(XMM1, XMM2);
        _mm_storeu_si128(xmm2, XMM1);
        xmm1++; xmm2++;
    }
    //to handle fall back
    for(int i = count * 4 ; i < size ; i++)
    dst[i] = src1[i] + src2;

    return dst;
}

int* add(int *dst, const int src1, const int* src2, int size)
{
    return add(dst, src2, src1, size);
}

float* add(float *dst, const float* src1, const float *src2, int size)
{
    int count = size / 4 ;

    __m128 XMM1, XMM2;
    const float *s1 = src1, *s2 = src2;
    float *d = dst;

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_ps(s1);
        XMM2 = _mm_loadu_ps(s2);
        XMM1 = _mm_add_ps(XMM1, XMM2);
        _mm_storeu_ps(d, XMM1);
        s1+=4; s2+=4; d+=4;
    }
    //to handle fall back
    for(int i = count * 4 ; i < size ; i++)
    dst[i] = src1[i] + src2[i];

    return dst;
}

float* add(float *dst, const float* src1, const float src2, int size)
{
    int count = size / 4 ;
    __m128 XMM1, XMM2;
    XMM2 = _mm_set1_ps(src2);
    const float *s1 = src1;
    float *d = dst;
    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_ps(s1);
        XMM1 = _mm_add_ps(XMM1, XMM2);
        _mm_storeu_ps(d, XMM1);
        s1+=4; d+=4;
    }
    //to handle fall back
    for(int i = count * 4 ; i < size ; i++)
    dst[i] = src1[i] + src2;

    return dst;
}

float* add(float *dst, const float src1, const float *src2, int size)
{
    add(dst, src2, src1, size);
}


//SSE sub API
int8* sub(int8 *dst, const int8* src1, const int8 *src2, int size)
{
    int count = size / 16 ;

    __m128i* xmm1 = (__m128i*)src1;
    __m128i* xmm2 = (__m128i*)src2;
    __m128i* xmm3 = (__m128i*)dst;
    __m128i XMM1, XMM2;

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM2 = _mm_loadu_si128(xmm2);
        XMM1 = _mm_subs_epu8(XMM1, XMM2);
        _mm_storeu_si128(xmm3, XMM1);
        xmm1++; xmm2++; xmm3++;
    }
    //to handle fall back
    for(int i = count * 16 ; i < size ; i++)
    dst[i] = sat(src1[i] - src2[i]);

    return dst;
}

int8* sub(int8 * dst, const int8* src1, const int8 src2, int size)
{
    int count = size / 16;

    __m128i* xmm1 = (__m128i*)src1;
    __m128i* xmm2 = (__m128i*)dst;
    __m128i XMM1, XMM2;
    XMM2 = _mm_set1_epi32(src2);

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM1 = _mm_subs_epu8(XMM1, XMM2);
        _mm_storeu_si128(xmm2, XMM1);
        xmm1++; xmm2++;
    }
    //to handle fall back
    for(int i = count * 16 ; i < size ; i++)
    dst[i] = sat(src1[i] - src2);

    return dst;
}
int8* sub(int8 *dst, const int8 src1, const int8* src2, int size)
{
        int count = size / 16;

    __m128i* xmm1 = (__m128i*)src2;
    __m128i* xmm2 = (__m128i*)dst;
    __m128i XMM1, XMM2;
    XMM2 = _mm_set1_epi32(src1);

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM1 = _mm_subs_epu8(XMM2, XMM1);
        _mm_storeu_si128(xmm2, XMM1);
        xmm1++; xmm2++;
    }
    //to handle fall back
    for(int i = count * 16 ; i < size ; i++)
    dst[i] = sat(src1 - src2[i]);

    return dst;
}
int* sub(int *dst, const int* src1, const int *src2, int size)
{
    int count = size / 4 ;

    __m128i* xmm1 = (__m128i*)src1;
    __m128i* xmm2 = (__m128i*)src2;
    __m128i* xmm3 = (__m128i*)dst;
    __m128i XMM1, XMM2;

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM2 = _mm_loadu_si128(xmm2);
        XMM1 = _mm_sub_epi32(XMM1, XMM2);
        _mm_storeu_si128(xmm3, XMM1);
        xmm1++; xmm2++; xmm3++;
    }
    //to handle fall back
    for(int i = count * 4 ; i < size ; i++)
    dst[i] = src1[i] - src2[i];

    return dst;
}

int* sub(int * dst, const int* src1, const int src2, int size)
{
    int count = size / 4;

    __m128i* xmm1 = (__m128i*)src1;
    __m128i* xmm2 = (__m128i*)dst;
    __m128i XMM1, XMM2;
    XMM2 = _mm_set1_epi32(src2);

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM1 = _mm_sub_epi32(XMM1, XMM2);
        _mm_storeu_si128(xmm2, XMM1);
        xmm1++; xmm2++;
    }
    //to handle fall back
    for(int i = count * 4 ; i < size ; i++)
    dst[i] = src1[i] - src2;

    return dst;
}

int* sub(int *dst, const int src1, const int* src2, int size)
{
    int count = size / 4;

    __m128i* xmm1 = (__m128i*)src2;
    __m128i* xmm2 = (__m128i*)dst;
    __m128i XMM1, XMM2;
    XMM2 = _mm_set1_epi32(src1);

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM1 = _mm_sub_epi32(XMM2, XMM1);
        _mm_storeu_si128(xmm2, XMM1);
        xmm1++; xmm2++;
    }
    //to handle fall back
    for(int i = count * 4 ; i < size ; i++)
    dst[i] = src1 - src2[i];

    return dst;
}

float* sub(float *dst, const float* src1, const float *src2, int size)
{
    int count = size / 4 ;

    __m128 XMM1, XMM2;
    const float *s1 = src1, *s2 = src2;
    float *d = dst;

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_ps(s1);
        XMM2 = _mm_loadu_ps(s2);
        XMM1 = _mm_sub_ps(XMM1, XMM2);
        _mm_storeu_ps(d, XMM1);
        s1+=4; s2+=4; d+=4;
    }
    //to handle fall back
    for(int i = count * 4 ; i < size ; i++)
    dst[i] = src1[i] - src2[i];

    return dst;
}

float* sub(float *dst, const float* src1, const float src2, int size)
{
    int count = size / 4 ;

    __m128 XMM1, XMM2;
    XMM2 = _mm_set1_ps(src2);
    const float *s1 = src1;
    float *d = dst;
    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_ps(s1);
        XMM1 = _mm_sub_ps(XMM1, XMM2);
        _mm_storeu_ps(d, XMM1);
        s1+=4; d+=4;
    }
    //to handle fall back
    for(int i = count * 4 ; i < size ; i++)
    dst[i] = src1[i] - src2;

    return dst;
}

float* sub(float *dst, const float src1, const float *src2, int size)
{
    int count = size / 4 ;

    __m128 XMM1, XMM2;
    XMM1 = _mm_set1_ps(src1);
    const float *s2 = src2;
    float *d = dst;
    for(int i = 0; i< count; i = i++)
    {
        XMM2 = _mm_loadu_ps(s2);
        XMM1 = _mm_sub_ps(XMM1, XMM2);
        _mm_storeu_ps(d, XMM1);
        s2+=4; d+=4;
    }
    //to handle fall back
    for(int i = count * 4 ; i < size ; i++)
    dst[i] = src1 - src2[i];

    return dst;
}


//SSE "bitwise and" API
template <typename T>
T* bit_and(T *dst, const T* src1, const T *src2, int size)
{
    int count = size / (16/sizeof(T)) ;

    __m128i* xmm1 = (__m128i*)src1;
    __m128i* xmm2 = (__m128i*)src2;
    __m128i* xmm3 = (__m128i*)dst;
    __m128i XMM1, XMM2;

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM2 = _mm_loadu_si128(xmm2);
        XMM1 = _mm_and_si128(XMM1, XMM2);
        _mm_storeu_si128(xmm3, XMM1);
        xmm1++; xmm2++; xmm3++;
    }
    //to handle fall back
    for(int i = count * (16/sizeof(T)) ; i < size ; i++)
    dst[i] = src1[i] & src2[i];

    return dst;
}

template <typename T>
T* bit_and(T * dst, const T* src1, const T src2, int size)
{
    int count = size / (16/sizeof(T));

    __m128i* xmm1 = (__m128i*)src1;
    __m128i* xmm2 = (__m128i*)dst;
    __m128i XMM1, XMM2;
    XMM2 = _mm_set1_epi32(src2);

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM1 = _mm_and_si128(XMM1, XMM2);
        _mm_storeu_si128(xmm2, XMM1);
        xmm1++; xmm2++;
    }
    //to handle fall back
    for(int i = count * (16/sizeof(T)) ; i < size ; i++)
    dst[i] = src1[i] & src2;

    return dst;
}
```

```
template <typename T>
T* bit_and(T *dst, const T src1, const T* src2, int size)
{
    return bit_and(dst, src2, src1, size);
}


//SSE "bitwise or" API
template <typename T>
T* bit_or(T *dst, const T* src1, const T *src2, int size)
{
    int count = size / (16/sizeof(T)) ;

    __m128i* xmm1 = (__m128i*)src1;
    __m128i* xmm2 = (__m128i*)src2;
    __m128i* xmm3 = (__m128i*)dst;
    __m128i XMM1, XMM2;

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM2 = _mm_loadu_si128(xmm2);
        XMM1 = _mm_or_si128(XMM1, XMM2);
        _mm_storeu_si128(xmm3, XMM1);
        xmm1++; xmm2++; xmm3++;
    }
    //to handle fall back
    for(int i = count * (16/sizeof(T)) ; i < size ; i++)
    dst[i] = src1[i] | src2[i];

    return dst;
}

template <typename T>
T* bit_or(T * dst, const T* src1, const T src2, int size)
{
    int count = size / (16/sizeof(T));

    __m128i* xmm1 = (__m128i*)src1;
    __m128i* xmm2 = (__m128i*)dst;
    __m128i XMM1, XMM2;
    XMM2 = _mm_set1_epi32(src2);

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM1 = _mm_or_si128(XMM1, XMM2);
        _mm_storeu_si128(xmm2, XMM1);
        xmm1++; xmm2++;
    }
    //to handle fall back
    for(int i = count * (16/sizeof(T)) ; i < size ; i++)
    dst[i] = src1[i] | src2;

    return dst;
```

```
}

template <typename T>
T* bit_or(T *dst, const T src1, const T* src2, int size)
{
    return bit_or(dst, src2, src1, size);
}

//SSE "bitwise xor" API
template <typename T>
T* bit_xor(T *dst, const T* src1, const T *src2, int size)
{
    int count = size / (16/sizeof(T)) ;

    __m128i* xmm1 = (__m128i*)src1;
    __m128i* xmm2 = (__m128i*)src2;
    __m128i* xmm3 = (__m128i*)dst;
    __m128i XMM1, XMM2;

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM2 = _mm_loadu_si128(xmm2);
        XMM1 = _mm_xor_si128(XMM1, XMM2);
        _mm_storeu_si128(xmm3, XMM1);
        xmm1++; xmm2++; xmm3++;
    }
    //to handle fall back
    for(int i = count * (16/sizeof(T)) ; i < size ; i++)
    dst[i] = src1[i] ^ src2[i];

    return dst;
}

template <typename T>
T* bit_xor(T * dst, const T* src1, const T src2, int size)
{
    int count = size / (16/sizeof(T));

    __m128i* xmm1 = (__m128i*)src1;
    __m128i* xmm2 = (__m128i*)dst;
    __m128i XMM1, XMM2;
    XMM2 = _mm_set1_epi32(src2);

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM1 = _mm_xor_si128(XMM1, XMM2);
        _mm_storeu_si128(xmm2, XMM1);
        xmm1++; xmm2++;
    }
    //to handle fall back
    for(int i = count * (16/sizeof(T)) ; i < size ; i++)
    dst[i] = src1[i] ^ src2;

    return dst;
```

```
}

template <typename T>
T* bit_xor(T *dst, const T src1, const T* src2, int size)
{
    return bit_xor(dst, src2, src1, size);
}


//TODO: SSE implementation
template <typename T>
T* copy(T *dst, T* src, int size)
{
    int count = size / (16/sizeof(T)) ;

    __m128i* xmm1 = (__m128i*)src;
    __m128i* xmm3 = (__m128i*)dst;
    __m128i XMM1;

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        _mm_storeu_si128(xmm3, XMM1);
        xmm1++; xmm3++;
    }
    //to handle fall back
    for(int i = count * (16/sizeof(T)) ; i < size ; i++)
    dst[i] = src[i];

    return dst;
}


int8* copy(int8 *dst, const int src, int size)
{
    int count = size / (16/sizeof(int8)) ;

    __m128i* xmm3 = (__m128i*)dst;
    __m128i XMM1 = _mm_set1_epi8((int8)src);

    for(int i = 0; i< count; i = i++)
    {
        _mm_storeu_si128(xmm3, XMM1);
        xmm3++;
    }
    //to handle fall back
    for(int i = count * (16/sizeof(int8)) ; i < size ; i++)
    dst[i] = (int8)src;

    return dst;
}

float* copy(float *dst, const double src, int size)
{
    int count = size / (16/sizeof(float)) ;
    __m128 XMM1 = _mm_set1_ps((float)src);
   float *d = dst;
    for(int i = 0; i< count; i = i++)
    {
         _mm_storeu_ps(d, XMM1);
         d = d + 4;
    }
    //to handle fall back
    for(int i = count * (16/sizeof(float)) ; i < size ; i++)
    dst[i] = (float)src;

    return dst;
}


int* copy(int *dst, const int src, int size)
{
    int count = size / (16/sizeof(int)) ;

    __m128i* xmm3 = (__m128i*)dst;
    __m128i XMM1 = _mm_set1_epi32((int)src);

    for(int i = 0; i< count; i = i++)
    {
         _mm_storeu_si128(xmm3, XMM1);
         xmm3++;
    }
    //to handle fall back
    for(int i = count * (16/sizeof(int)) ; i < size ; i++)
    dst[i] = (int)src;
    return dst;
}

//template<typename T>
//void copy(T* a, double b, int size)
//{
//    for(int i = 0; i< size; i++)
//                   a[i] = (T)b;
//
//}

__m128i _mm_mul_epi32(const __m128i &src1, const __m128i &src2)
{
    __m128i hiSrc1, loSrc1;
    __m128i hiSrc2;
    __m128d tempS1, tempS2;

    hiSrc1 = _mm_srli_si128( src1, 8);
    hiSrc2 = _mm_srli_si128( src2, 8);

    tempS1 = _mm_cvtepi32_pd(src1);
    tempS2 = _mm_cvtepi32_pd(src2);
    tempS1 = _mm_mul_pd(tempS1, tempS2);

    loSrc1 = _mm_cvtpd_epi32(tempS1);
    tempS1 = _mm_cvtepi32_pd(hiSrc1);
    tempS2 = _mm_cvtepi32_pd(hiSrc2);
    tempS1 = _mm_mul_pd(tempS1, tempS2);

    hiSrc1 = _mm_cvtpd_epi32(tempS1);
    return _mm_or_si128(_mm_slli_si128( hiSrc1, 8), loSrc1);
}

__m128i _mm_mul_epi8(const __m128i &src1, const __m128i &src2)
{
    __m128i hiSrc1, loSrc1;
    __m128i hiSrc2, loSrc2;
    //__m128d tempS1, tempS2;

    hiSrc1 = _mm_unpackhi_epi8( src1, _mm_setzero_si128());
    hiSrc2 = _mm_unpackhi_epi8( src2, _mm_setzero_si128());

   hiSrc1        = _mm_mullo_epi16(hiSrc1, hiSrc2);

   loSrc1 = _mm_unpacklo_epi8( src1, _mm_setzero_si128());
    loSrc2 = _mm_unpacklo_epi8( src2, _mm_setzero_si128());

   loSrc1        = _mm_mullo_epi16(loSrc1, loSrc2);

    return _mm_packus_epi16 (loSrc1, hiSrc1);
}


int* mul(int *dst, const int* src1, const int *src2, int size)
{
    int count = size / 4 ;

    __m128i* xmm1 = (__m128i*)src1;
    __m128i* xmm2 = (__m128i*)src2;
    __m128i* xmm3 = (__m128i*)dst;
    __m128i XMM1, XMM2;

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM2 = _mm_loadu_si128(xmm2);
        XMM1 = _mm_mul_epi32(XMM1, XMM2);
        _mm_storeu_si128(xmm3, XMM1);
        xmm1++; xmm2++; xmm3++;
    }
    //to handle fall back
    for(int i = count * 4 ; i < size ; i++)
    dst[i] = src1[i] * src2[i];

    return dst;
}

int* mul(int * dst, const int* src1, const int src2, int size)
{
    int count = size / 4;
```

```
      __m128i* xmm1 = (__m128i*)src1;
      __m128i* xmm2 = (__m128i*)dst;
      __m128i XMM1, XMM2;
      XMM2 = _mm_set1_epi32(src2);

      for(int i = 0; i< count; i = i++)
      {
            XMM1 = _mm_loadu_si128(xmm1);
            XMM1 = _mm_mul_epi32(XMM1, XMM2);
            _mm_storeu_si128(xmm2, XMM1);
            xmm1++; xmm2++;
      }
      //to handle fall back
      for(int i = count * 4 ; i < size ; i++)
      dst[i] = src1[i] * src2;

      return dst;
}



int8* mul(int8 *dst, const int8 *src1, const int8 *src2, int size)
{
    int count = size / 16 ;

      __m128i* xmm1 = (__m128i*)src1;
      __m128i* xmm2 = (__m128i*)src2;
      __m128i* xmm3 = (__m128i*)dst;
      __m128i XMM1, XMM2;

      for(int i = 0; i< count; i = i++)
      {
            XMM1 = _mm_loadu_si128(xmm1);
            XMM2 = _mm_loadu_si128(xmm2);
            XMM1 = _mm_mul_epi8(XMM1, XMM2);
            _mm_storeu_si128(xmm3, XMM1);
            xmm1++; xmm2++; xmm3++;
      }
      //to handle fall back
      for(int i = count * 16 ; i < size ; i++)
      dst[i] = sat(src1[i] * src2[i]);

      return dst;
}

int8* mul(int8 *dst, const int8 *src1, const int8 src2, int size)
{
    int count = size / 16 ;

      __m128i* xmm1 = (__m128i*)src1;
      __m128i* xmm3 = (__m128i*)dst;
      __m128i XMM1, XMM2;
    XMM2 = _mm_set1_epi8(src2);

      for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_si128(xmm1);
        XMM1 = _mm_mul_epi8(XMM1, XMM2);
        _mm_storeu_si128(xmm3, XMM1);
        xmm1++; xmm3++;
    }
    //to handle fall back
    for(int i = count * 16 ; i < size ; i++)
    dst[i] = sat(src1[i] * src2);

    return dst;
}


float* mul(float *dst, const float* src1, const float *src2, int size)
{
    int count = size / 4 ;

    __m128 XMM1, XMM2;
    const float *s1 = src1, *s2 = src2;
    float *d = dst;

    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_ps(s1);
        XMM2 = _mm_loadu_ps(s2);
        XMM1 = _mm_mul_ps(XMM1, XMM2);
        _mm_storeu_ps(d, XMM1);
        s1+=4; s2+=4; d+=4;
    }
    //to handle fall back
    for(int i = count * 4 ; i < size ; i++)
    dst[i] = src1[i] * src2[i];

    return dst;
}

float* mul(float *dst, const float* src1, const float src2, int size)
{
    int count = size / 4 ;

    __m128 XMM1, XMM2;
    XMM2 = _mm_set1_ps(src2);
    const float *s1 = src1;
    float *d = dst;
    for(int i = 0; i< count; i = i++)
    {
        XMM1 = _mm_loadu_ps(s1);
        XMM1 = _mm_mul_ps(XMM1, XMM2);
        _mm_storeu_ps(d, XMM1);
        s1+=4; d+=4;
    }
    //to handle fall back
    for(int i = count * 4 ; i < size ; i++)
    dst[i] = src1[i] * src2;
    return dst;
}


template <typename T>
T* mul(T *dst, const T *src1, const T *src2, int size)
{
    for(int i = 0; i< size; i++)
                dst[i] = src1[i] * src2[i];
}

template <typename T>
T* mul(T *dst, const T *src1, const T src2, int size)
{
    for(int i = 0; i< size; i++)
                dst[i] = src1[i] * src2;
}

template <typename T>
T* mul(T *dst, const T src1, const T* src2, int size)
{
    return mul(dst, src2, src1, size);
}

template <typename T>
T* div(T *dst, const T *src1, const T *src2, int size)
{
    for(int i = 0; i< size; i++)
                dst[i] = src1[i] / src2[i];
  return dst;
}

template <typename T>
T* div(T *dst, const T *src1, const T src2, int size)
{
    for(int i = 0; i< size; i++)
                dst[i] = src1[i] / src2;
  return dst;
}

template <typename T>
T* div(T *dst, const T src1, const T* src2, int size)
{
    for(int i = 0; i< size; i++)
                dst[i] = src2[i] / src1;
  return dst;
}
```

/* Stub code file – stub-print.h
Author: Gowri Kanugovi, Pratap Prabhu */

```
#include <iostream>
#include <string>
#include <fstream>

using namespace std;
```

```
template<typename T>
void print (T a) {
      std::cout << a << "\n";
}

void print (float a) {
      printf("%f\n",a);
}

void print_string (string str) {
      std::cout <<str<< "\n";
}

void copy_vector (int dst[], int dst_size, int src[]) {
      int i;

      for (i=0; i<dst_size; i++ ) {
            dst[i] = src[i];
      }
}

void copy_vector (float dst[], int dst_size, float src[]) {
      int i;
      for (i=0; i<dst_size; i++ ) {
            dst[i] = src[i];
      }
}


void copy_vector (float dst[], int dst_size, float src[], int src_size) {
      int i;
      int low = (dst_size>src_size?src_size:dst_size);
      for (i=0; i<low; i++ ) {
            dst[i] = src[i];
      }
}

void copy_vector (int dst[], int dst_size, int src[], int src_size) {
      int i;
      int low = (dst_size>src_size?src_size:dst_size);
      for (i=0; i<low; i++ ) {
            dst[i] = src[i];
      }
}

void print (int *a, int size) {
      for (int i=0; i<size; i++) {
            std::cout << a[i] << "\n";
      }
}

void print (float *a, int size) {

      for (int i=0; i<size; i++) {
            printf ("%f\n", a[i]);
```

```
        }
}

void print (unsigned char *a, int size) {
        for (int i=0; i<size; i++) {
                std::cout << (int)a[i] << "\n";
        }
}

void fread_int (string file, int *a, int size) {
        char *tmp = (char *) a;
        ifstream myfile;
        myfile.open (file.c_str(), ios::in|ios::binary);
        if (!myfile.is_open()) {
                cout << "Unable to open file " << file << "\n";
                exit (-1);
        }
        myfile.read (tmp, size*sizeof(int));

        myfile.close();
}

void fread_float (string file, float *a, int size) {
        float *tmp = (float *)a;
        ifstream myfile;

        myfile.open (file.c_str(), ios::in|ios::binary);
        if (!myfile.is_open()) {
                cout << "Unable to open file " << file << "\n";
                exit (-1);
        }

        myfile.read ((char *)tmp, size*sizeof(float));

        myfile.close();
}

void fwrite_int (string file, int *a, int size){
        int *tmp = a;
        ofstream myfile;

        myfile.open (file.c_str(), ios::out|ios::binary);
        if (!myfile.is_open()) {
                cout << "Unable to open file " << file << "\n";
                exit (-1);
        }

        myfile.write ((char *)tmp, size*sizeof(float));

        myfile.close();

}

void fwrite_float (string file, float *a, int size){
        float *tmp = a;
        ofstream myfile;
        myfile.open (file.c_str(), ios::out|ios::binary);
        if (!myfile.is_open()) {
            cout << "Unable to open file " << file << "\n";
            exit (-1);
        }

        myfile.write ((char *)tmp, size*sizeof(float));

        myfile.close();

}
```

8.2. Positive Test Cases

```
./test-file-output.input
------------------------
int main(){
float a[5], b[5];
a={25.23,45.213,12.21,678.98,6e+5};
fwrite_float("out", a, 10);
fread_float("out", b, 10);
print(b);
}
------------------------

./test-ops1.input
------------------------
int main(){
int i,j,k;
i=10;
j=12;
i=i+j;
j=i-j;
i=i*j;
j=i/j;

k=i&j;
i=k|j;
j=k^i;

print("After all operations");
print(i);
print(j);
print(k);

}
------------------------

./test-file-input.input
------------------------

int
main () {
```

```
int a[10];

fread_int ("hey", a, 10);

print (a);

return 1;
}
------------------------

./test.add_SSE2.input
------------------------

int main()
{

        int a[10], b[10];
        a = 5;
        b = a + 2;
        print(a);
        print(b);
}
------------------------

./test.add3_SSE.input
------------------------

int main()
{

        int8 a[10], b[10];
        a = 5;
        b = a + a;
        print(a);
        print(b);
}
------------------------

./test.and_SSE2.input
------------------------

int main()
{

        int a[10], b[10];
        a = 5;
        b = a & 2;
        print(a);
        print(b);
}
------------------------

./test-continue-break.input
------------------------

int main(){
```

```
int i;
i=0;
while(i<=10){
            i=i+1;
            if(i==2){
                        continue;
            }
            print(i);
            if(i==5){
                        break;
            }
            }
}
-------------------------

./test.sub_SSE2.input
-------------------------

int main()
{

            int a[10], b[10];
            a = 5;
            b = a - 2;
            print(a);
            print(b);
}
-------------------------

./test-comments.input
-------------------------

/*This test case tests single line
 *and multi line comments */

int main() {
            int i;

            //Assigning i to 0
            i=0;
            print(i);
}
-------------------------

./test.mul2_SSE.input
-------------------------

int main()
{

            int8 a[10], b[10];
            a = 5;
            b = 7;
            b = b * a;
            print(a);
            print(b);
```

```
}
-------------------------

./test.or_SSE.input
-------------------------

int main()
{

        int a[10], b[10];
        a = 5;
        b = a | a;
        print(a);
        print(b);
}
-------------------------

./test-init-vector.input
-------------------------

int
main () {
int a[5], b[5], c[10];
float x[2];
int k;

k = 10;

x = {1.0,2.0};
a = {1,2,3,4,5};
b=a;
c=8;
print (23);
print (23.5);

print (k);
print (c);
 print (b);
print ("hello world");

print (x);
return 1;
}
-------------------------

./test-if.input
-------------------------
int main(){

int i,j;
i=1;
j=1;
if(i==1){
        if(j){
                i=i+1;
                print(i);
        }
        else{
                j=j+1;
                print(j);
        }
}
else{
        i=i+2;
        print(i);
}
}
------------------------

./test-fibonacci.input
------------------------

int
main () {
int i, j, tmp;
i=1;
j=1;

while(i<100){
        print (i);
        print (j);
        tmp = j;
        j = j+i;
        i = tmp;
        }

return 1;
}
------------------------

./test-file-input2.input
------------------------

int
main () {
float a[10];

fread_float ("hey1", a, 10);

print (a);

return 1;
}
------------------------

./test.or_SSE2.input
------------------------
int main()
{
        int a[10], b[10];
        a = 5;
        b = a | 2;
        print(a);
        print(b);
}
-------------------------

./test.add2_SSE.input
-------------------------
int main()
{
        float a[10], b[10];
        a = 5.0;
        b = a + a;
        print(a);
        print(b);
}
-------------------------

./test.int8.input
-------------------------

int main()
{
        int8 a[10], b[20];
        a = 5;
        b = 10;
        b = a;
        print(b);

}
-------------------------

./test-file-output2.input
-------------------------

int main(){
        int a[10],b[10];
        a=89;
        fwrite_int("out1",a,10);
        fread_int("out1",b,10);
        print(b);
}-------------------------

./test.or2_SSE.input
-------------------------

int main()
{

        int8 a[10], b[10];
        a = 5;
        b = a | a;
        print(a);
        print(b);
}
-------------------------
```

```
./test.assign2.input
------------------------

int main()
{
            float a[10], b[20];
            a = 5.0;
            b = 10.0;
            b = a;
            print(b);

}
------------------------

./test.assign.input
------------------------

int main()
{
            int a[10], b[20];
            a = 5;
            b = 10;
            b = a;
            print(b);

}
------------------------

./test-func2.input
------------------------

int foo(int a[10], int b){
            print(b);
            print(a);
            return 2;

}

int foo2(int b,int c[9]){
            return 1;

}

int main(){
            int j[10];
            j=0;
            foo(j,3);
}
------------------------

./test.xor2_SSE.input
------------------------

int main()
{
        int8 a[10], b[10];
        a = 5;
        b = 7;
        b = b ^ a;
        print(a);
        print(b);
}
------------------------

./test.div2.input
------------------------

int main()
{

        int a[10], b[10];
        a = 2;
        b = 7;
        b = b / a;
        print(a);
        print(b);
}
------------------------

./test.assign3.input
------------------------

int main()
{
        int8 a[10], b[20];
        a = 5;
        b = 10;
        b = a;
        print(b);

}
------------------------

./test-global.input
------------------------

int i;
float f;

int f1(){
        i=10;
        f=12.0;
        print("Inside f1: i and f");
        print(i);
        print(f);
}

int f2(){
        i=12;
        f=16.0;
        print("Inside f2: i and f");
        print(i);
        print(f);
}

int main(){
        int i;
        i=3;
        print("Inside main: i and f");
        print(i);
        print(f);
        f1();
        f2();
        print("Inside main after function calls: i and f");
        print(i);
        print(f);
}
------------------------

./test.mul_SSE.input
------------------------

int main()
{

        int a[10], b[10];
        a = 5;
        b = 7;
        b = b * a;
        print(a);
        print(b);
}
------------------------

./test-func1.input
------------------------

int func1(int a, int b, float c, float d){
        int i;
        a=10;
        print(a);
        print(b);
        print(c);
        print(d);
        return 0;
}

int main(){
        int r;
        r = func1(12,23,3.0, 2.0);
        print("Return value");
        print(r);
}
------------------------

./test.and_SSE.input
------------------------
```

```
int main()
{

        int a[10], b[10];
        a = 5;
        b = a & a;
        print(a);
        print(b);
}
------------------------

./test.and2_SSE.input
------------------------

int main()
{

        int8 a[10], b[10];
        a = 5;
        b = a & a;
        print(a);
        print(b);
}
------------------------

./test.div3.input
------------------------

int main()
{

        float a[10], b[10];
        a = 2.0;
        b = 5.0;
        b = b / a;
        print(a);
        print(b);
}
------------------------

./test.xor_SSE.input
------------------------

int main()
{

        int a[10], b[10];
        a = 5;
        b = 7;
        b = b ^ a;
        print(a);
        print(b);
}
------------------------
```

./test.sub3_SSE.input

```
int main()
{

        int8 a[10], b[10];
        a = 5;
        b = a - a;
        print(a);
        print(b);
}
```

./test.mul3_SSE.input

```
int main()
{
        float a[10], b[10];
        a = 2.0;
        b = 4.0;
        b = b * a;
        print(a);
        print(b);
}
```

./test.sub_SSE.input

```
int main()
{
        int a[10], b[10];
        a = 5;
        b = a - a;
        print(a);
        print(b);
}
```

./test.div1.input

```
int main()
{
        int8 a[10], b[10];
        a = 5;
        b = 7;
        b = b / a;
        print(a);
        print(b);
}
```

./test.div4.input

```
-------------------------

int main()
{

		int a[10], b[10];
		a = 5;
		b = a / 2;
		print(a);
		print(b);
}
-------------------------

./test.xor_SSE2.input
-------------------------

int main()
{

		int a[10], b[10];
		a = 5;
		b = a ^ 2;
		print(a);
		print(b);
}
-------------------------

./test.add_SSE.input
-------------------------

int main()
{

		int a[10], b[10];
		a = 5;
		b = a + a;
		print(a);
		print(b);
}
-------------------------

./test.sub2_SSE.input
-------------------------

int main()
{

		float a[10], b[10];
		a = 5.0;
		b = a - a;
		print(a);
		print(b);
}
-------------------------

./test.mul_SSE2.input
```

```
------------------------

int main()
{

        int a[10], b[10];
        a = 5;
        b = a * 2;
        print(a);
        print(b);
}
------------------------

./test.init1.input
------------------------

int main()
{

        int a[5], b[10];
        a = 2;
        b = 1;
        print(a);
        print(b);
}
------------------------

./test-while.input
------------------------

int
main () {
int i;

i=0;
while (i<10) {
        print("Hey");
        if(i==3) {
                break;
        }
        i++;
}

return 1;
}
------------------------

./test-for.input
------------------------

int main(){
int i;


i++;
for(i=0;i<10;i++){
        print(i);
}
for(i=10;i>0;i--){
        print(i);
}
for(i=0;;){
        print(i);
        i++;
        if(i==3){
                break;
        }
}
}
------------------------
```

8.3. Negative Test Cases

```
./test-type-checking2.input
------------------------

int main(){
int i[10],k;
k=0;
i=k;
k=i;
}------------------------

./test-misplaced-continue.input
------------------------

int main(){
int i;
i=1;
if(i<10){
        if(i==3) { continue; }
        i++;
}
}
------------------------

./test-func3.input
------------------------

int foo () {
return (2.3);

}

int main () {
foo ();
return 1;
}
------------------------
```

```
./test-undeclared-variable.input
------------------------

int main(){
int a, b;
a=2;
c=a+1;
b=3;
}
------------------------

./test-id-same-as-function.input
------------------------

int foo;
int foo() {
        return 1;
        }

int
main () {
int foo1;
foo1 = 0;
return 1;
}
------------------------

./test-misplaced-break.input
------------------------

int main(){
int i;
i=1;
if(i<10){
        if(i==3) { break; }
        i++;
}
}
------------------------

./test-id-func-same-name.input
------------------------

int foo(){
return 34;
}

int main(){
int foo;
foo=9;
foo();
}
------------------------

./test-invalid-unary-operation.input
```

```
-------------------------

int main(){
int i[10];
i=4;
i++;
}
-------------------------

./test-nomain.input
-------------------------

int foo () {
int i;
i = 3+4+5;
}
-------------------------

./test-func2.input
-------------------------

int foo(int a, float b[10], int c){
return 1;
}

int main(){
float i[10];
foo(2,i,32.21);
}
-------------------------

./test-invalid-assignment.input
-------------------------

int main(){
int f;
f=break;
}
-------------------------

./test-func1.input
-------------------------

int foo(int a, int b){
return 1;
}

int main(){
foo(2);
}
-------------------------

./test-filenotfound.input
-------------------------

int main(){
```

```
int a[10];
fread_int("tmp", a, 10);
}
------------------------

./test-functionnotfound.input
------------------------

int foo(){
int i;
i=9;
i=i+9;
return i;
}

int main(){
int i;
i=foo1();
print(i);
}
------------------------

./test-nofuncs.input
------------------------

int a;
a =5;
------------------------

./test-type-checking1.input
------------------------

int main(){
int i;
float j;
i=3;
j=i;
}
------------------------

./test-invalid-declaration.input
------------------------

int main(){
int a;
string f;
a=10;
f="hello";
}
------------------------

./test-2mains.input
------------------------

int main () {
int i;
```

```
i=2+3+4;
}

int
main () {
int j;
j=2+3+4;
}
------------------------

./test-invalid-return.input
------------------------

int main(){
float f;
f=23.4;
return f;
}
------------------------

./test-invalid-return-assignment.input
------------------------

int foo(){
int i;
i=9;
return i;
}

int main(){
float f;
f=foo();
}
------------------------
```

8.4. Makefile

```
CCFLAGS = -DLINUX -msse2 -msse3
all: build

exec: build
	./fvpl < $(ARGS) > /tmp/tmp.c
	g++ $(CCFLAGS) -I./ /tmp/tmp.c

build: ast.cmo parser.cmo scanner.cmo symbol.cmo  interpret.cmo  fvpl.cmo
		ocamlc -o fvpl parser.cmo scanner.cmo symbol.cmo interpret.cmo fvpl.cmo

scanner.cmo: scanner.mll
		ocamllex scanner.mll


parser.cmo: parser.mly
		ocamlyacc parser.mly
		ocamlc -c parser.mli
```

```
ast.cmo: ast.mli
	ocamlc -c ast.mli

interpret.cmo: interpret.ml
	ocamlc -c interpret.ml

symbol.cmo: symbol.ml
	ocamlc -c symbol.ml

fvpl.cmo: fvpl.ml scanner.ml parser.ml
	ocamlc -c scanner.ml
	ocamlc -c parser.ml
	ocamlc -c fvpl.ml

test: build
	chmod u+x testscript.sh
	./testscript.sh both

clean:
	rm -f *.cmo *.cmi parser.mli parser.ml scanner.ml fvpl a.out
```