Programming Languages and Translators
COMS W4115

Department of Computer Science
Columbia University
Fall 2008

Fast vector processing language

Ganapathi, Ravindra Babu <rg2547@columbia.edu>
Kanugovi, Gowri <gk2263@columbia.edu>
Musuvathy, Deepika <dkm2123@columbia.edu>
Prabhu, Pratap V <pvp2105@columbia.edu>

1. Overview

Large sequential data computations are common in various domains like image processing, databases and cryptography. Most of the current compilers for C, C++ generate slow native code for the x86 family of processors.

New x86 processors include vector processing unit which are capable of computing data in parallel. These instructions are SIMD (single instruction multiple data) instructions which operate on 128 bits of data at a time. For example, 4 integers or floating point operations can be performed with a single instruction.

Though these instructions boost performance for computation intensive applications, they are not popular among programmers. The reasons are:

1) Backward compatibility with old processors
2) Not accustomed to think parallel
3) Lack of awareness

Hence we propose to create a language which would allow programmers to transparently and efficiently utilize the power of SIMD instructions (such as SSE, SSE2) to compute large amount of sequential data at higher speeds.

The compiler will generate C/C++ code from our language and pass it to a C/C++ compiler.

2. Specification

1) The following interfaces will be provided

    - To create arrays (static and dynamic).
    - To free memory in case of dynamic allocation
    - Print/read Arrays to/from a file/stdout

2) Data types
    a. Scalar:
        i. int
        ii. float
        iii. double
    b. Vector:
        i. vint
        ii. vfloat
        iii. vdouble

3) Operations on scalars
   a. Arithmetic operations (A + B, A - B, Mul and Div are supported for Float only)
   b. Bitwise logical operations (A | B, A & B, A ^ B, ~A)
   c. Assignment and initialization operations

4) Operations on vectors (arrays)
   a. Vector concatenation (A @ B)
   b. Vector casting (float to integer, integer to float)
   c. Vector copy (A = B)
   d. Scalar copy into a vector (A = 5)
   e. Vector arithmetic operations (A + B, A - B, Mul and Div are supported for Float only)
   f. Vector logical operations (A | B, A & B, A ^ B, ~A)
   g. Vector comparisons (A == B, A != B)
   h. Vector initialization(A = {1,2,3,4,5,6})

3. Examples

```
vint A = create_array(10000000,…);
vint B = create_array(10000000,…);
vint C = create_array(10000000,…);
// A, B and C are large arrays of integer

A = 5; //initialize every element in the array A to 5

A = B; // Copy elements in B to A, A[0] = B[0], A[1] = B[1]........ A[9999999] = B[9999999]
C = (A + B) * 10; //Add each element of A with each element of B and multiply the result by 10

B = A ^ 0xFFFFFFFF; //Xor each element of A with max unsigned integer
                    //In image processing this results in a negative of the original image
```

4. Uniqueness of our language

We guarantee you high performance along with easy usage. Code generated by our compiler will either match or beat the best commercially available compilers.