Implementing BLAS libraries in SPARC V8 assembly language (with 64 bit extensions)

Gauri Patrikar[1] and Siddharth Rao Deb[1]

[1]Project Research Engineer, IITB

January 27, 2020

Abstract

The BLAS (basic linear algebra subprograms) and C string are widely used libraraies. The BLAS are written in fortran, and the string libraries in C. Here, we will write some of the programs in assembly, specifically, SPARC V8 assembly with the 64 bit extensions. This we do as due to the 64 bit extensions, which uses vector instructions will help in increasing the speed of the program as the number of instructions executed will be less.

We start with giving a brief description of the Blas library and also some details on the 64 bit extensions used. We will then proceed to explain the assembly language program written.

At the end the difference in performance is given.

1 Introduction

Basic Linear Algebra Subprograms (BLAS) is a specification that prescribes a set of low-level routines for performing common linear algebra operations such as vector addition, scalar multiplication, dot products, linear combinations, and matrix multiplication. They are the de facto standard low-level routines for linear algebra libraries; the routines have bindings for both C and Fortran.

The BLAS (Basic Linear Algebra Subprograms) routines provide standard building blocks for performing basic vector and matrix operations. The Level 1 BLAS perform scalar, vector, and vector-vector operations. These are the libraries we will be concentrating on. In this document the dot product program is discussed in detail.

2 BLAS libraries

Here, details on the blas level 1 libraries and their implementation is discussed in length. We will first take a look at how the blas libraries are implemented in fortran, then some details of the 64 bit extensions and then go on to discuss the AJIT impllementaion

2.1 BLAS Routine

2.2 AJIT 64 bit extensions

2.3 Some differences

1. The increments in memory can only be 1, 2, 4 or 8 for 8, 16, 32 or 64 bits of element length.

2. Signed and unsigned operations will need different programs.

2.4 AJIT implementation

Here we will show the implementation of unsigned dot product

2.4.1 Input and output

Here the input to the assembly program will be -

1. The two arrays.

2. Length of the array(n).

3. The increment of memory space between each element of the array for both arrays(incx, incy).

The output will be the final dot product.

2.4.2 Program Flow

1. Initialize the mask and the register which will hold intermediate output.
2. Set the amount of decrease to be done for memory
3. Load values from memory
4. Vector multiply the output(unsigned).
5. Byte reduce it.
6. Store output in a different register for further iterations.
7. Decrease the memory for the new input.
8. Decrease n (number of elements in array).
9. If n is greater than 0 then go to next iteration.
10. Else end and give final output.

Note : We have used word load instead of doubleword load due to alignment issues.

2.4.3 The assembly program

```
        save %sp,-96,%sp
        mov 0xff, %l0
        mov 0x0,%l2


eight:  set 0x80008,%l3
eight_p:ld [%i0],%g4
        ld [%i0 + 4],%g5
        ld [%i1],%g2
        ld [%i1 + 4],%g3
        vumuld8 %g2,%g4,%g6
        addbyter %g6,%l0,%l1
        add %l1,%l2,%l2
```

```
        addd %l3,%i0,%i0
        subcc %i2,8,%i2
        bg eight
        nop
        ba end
        nop

end:    mov %l2,%i0
        restore
        retl
        nop
```

3 Conclusion

Here, we have given the assembly language implementation of the blas level 1 libraries, mainly the dot product function. The assembly was specifically of Sparc V8 with 64bit extensions.