An Evaluation of sDPCE language for on chip SIMD processors

4 E − 9

Linda Lu*, Hirono Tsubota*, Stanislas de Crevoisier **,
Toshiyuki Tamura*, Ken-ichi Tanaka*, Kazuo Kyuma*
* Mitsubishi Electric Corporation, Advanced Technology R&D Center
**ENST Paris
E-mail: lindalu@qua.crl.melco.co.jp

1.Introduction

The demand for more powerful computers is answered by the availability of on chip Single Instruction stream Multiple Data stream (SIMD) processors. However, because the software development environment is lacking for these type of processors, the application developers cannot take full advantage of the hardware by programming in a high level language. The introduction of the small Data Parallel C Extension (sDPCE) programming language provides an advanced development environment targeting this type of processors. This paper reports the result obtained from evaluating the language with the target Pentium Pro processor with MMX technology.

2.Evaluation result

The specification of sDPCE is on paper "A proposal of sDPCE language for on chip SIMD processors."[1] The most important issue is that the preprocessor of the sDPCE language can recognize parallel type data and replace operations on them with MMX library functions. Four vector-matrix arithmetic and two image processing testing programs involve looping on large size data are written in both C and sDPCE languages. The vector-matrix arithmetic programs include vector addition, vector inner product, matrix-vector product and matrix-matrix product using 16 bits data. The image processing algorithms include chrome keying and image subtraction tested with bitmap images composed of 8 bits per pixels. Besides the exception that sDPCE programs are preprocessed by sDPCE preprocessor, the rest of the compilation and linkage are the same for all programs. The numbers of cycles it takes to execute loops in C programs or the MMX library functions in sDPCE programs are recorded by visual tuning environment Vtune[1]. The target in test is a Pentium Pro processor with MMX technology. Programming is done by Microsoft Visual C++[2] on Windows95. The result shows that the vector-matrix arithmetic programs of sDPCE that are executed with four data in parallel have 2.5 times fewer cycle than the corresponding C programs. On the side of image processing programs, sDPCE achieves a 3.2 times fewer cycles in chrome keying and a 4.7 times fewer cycle in image subtraction. Figure 1 and figure 2 illustrate the result obtained from vector inner product and chrome keying. (Better performance can be obtained from further optimization)

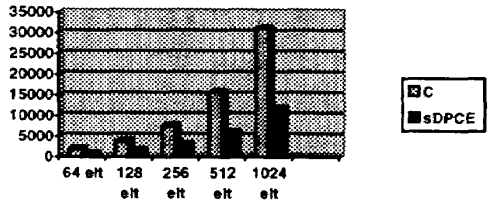

Figure 1. Cycles of vector inner product (16 bits)

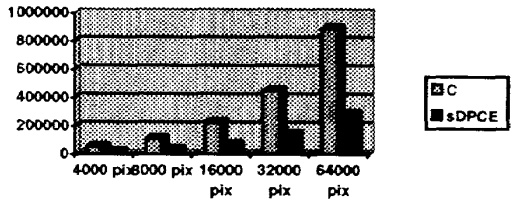

Figure 2. Cycles of chrome keying (8 bits)

3.MMX Library Functions

Several approaches have been tried to define MMX library functions to reduce overhead of function calling. The approaches are defining them as normal function, inline function and macro definition. Defining function as an inline function reduces the normal function execution cycles significantly as shown in figure 3. However, problems are found in inline functions when addressing memory pointers that are passed as arguments. By using macro definition, this problem is solved and the number of cycles is furthered reduced. Therefore, the evaluation is done on macro defined MMX library functions.

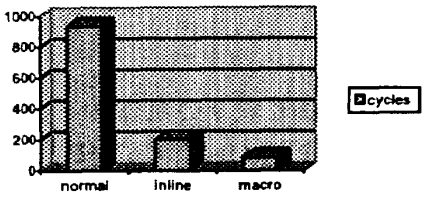

Figure 3. Cycles of MMX library functions definitions

[1] Product of Intel Corporation.
[2] Product of Microsoft Corporation.

4. Preprocessing

sDPCE avoids developers from programming directly with loops and large size arrays of data. The syntax of the language provides abstraction of parallel data structure while on the other hand, insulates users from the complexity of parallel operations. Figure 4 shows part of the source code from chrome keying program. Variables *Foreground* and *Background* are defined as parallel type data. Black area in the bitmap Foreground is going to be replaced by the image of *Background*. The procedure is to compare every pixel in the *Foreground* with black pixel and set it to *Background* if it is of the color black. After preprocessing this program, the process is passed to a library function called *MMX_ushort_cwrite_EQU* shown in figure 5, which uses MMX instructions including *pcmpeqw*, *pand*, *pandn* and *por* to process pixels in parallel.

```
#define size 4000
#define black 0
main (){
        shape [size]bitmap;
        unsigned short int:bitmap
                Foreground, Background;

        /* initialization omitted */

        where (Foreground = = black)
                Foreground=Background;
}
```

Figure 4. sDPCE source code

```
main (){
        unsigned short Foreground[4000];
        unsigned short Background[4000];

        /*initialization omitted*/

        MMX_ushort_cwrite_EQU(Foregrou
nd, Background, Foreground, 4000);
}
```

Figure 5. sDPCE preprocessed output

5. Simplicity

When compare the source code with a corresponding standard C source program, the simplicity of sDPCE becomes apparent. Figure 6 is a part of C program to find the maximum among elements of a vector. In sDPCE, it can be expressed by a single operator '>?=', which will then be replaced with a MMX library function by the preprocessor(figure 7). Similarly, the calculation of the vector inline product in figure 8 can be expressed by operator '**' as in figure 9, which will then be replaced with a MMX library function either for the calculation of vector inner product, vector-matrix product or matrix-matrix product depending on the dimensions of the operands.

```
int vector[size];
int index, max;

/* initialization omitted */

max=vector[0];
for (index=1; index <size; index++){
        if (max < vector[index])
                max=vector[index];
}
```

Figure 6. Finding maximum in a vector in C

```
shape [size]v;
unsigned short int:v vector;
int max;

/* initialization of vector omitted */

max >?= vector;
```

Figure 7. Finding maximum in a vector in sDPCE

```
int product;
int v1[size], v2[size];

/*initialization omitted */

product = v1[0]*v2[0] +v1[1]*v2[1]
        +v1[2]*v2[2]+v1[3]*v2[3]...
```

Figure 8. Vector inline product in C

```
shape [size]v;
unsigned short int:v v1,v2;

/*initialization omitted */

product= v1 ** v2;
```

Figure 9. Vector inline product in sDPCE

6.Conclusion

In view of execution cycles and ease of usage, the performance of sDPCE shows its advantage on processing large size data that has the potential to make use of SIMD target. The preprocessor takes into account the parallel data recognition and library functions replacement. As a result, it serves as an efficient tool for the software developers to take full advantage of the SIMD processor.

References
[1] Tsubota Hirono: A proposal of sDPCE language for on chip SIMD processors.
[2] Data Parallel C Extensions, DPCE Subcommittee Technical Report, Version 1.6.
[3] Rafael C. Gonzalez, Richard E. Woods: Digital Image Processing, Addison Wesley, 1992.
[4] MMX Technology Technical Overview, Intel Corporation.
[5] MMX Technology programmers reference manual, Intel Corporation.
[6] Jeff Prosise: Programming Wondows95 with MFC, Microsoft Press, 1996.