ISSN: 2321-2152

IJMECE

International Journal of modern electronics and communication engineering

ISSN 2321-2152
www.ijmece.com
Vol 11, Issue 2, 2023

Implementation of Digital Signal Processing in the RadioSolariz Project Using SSE2

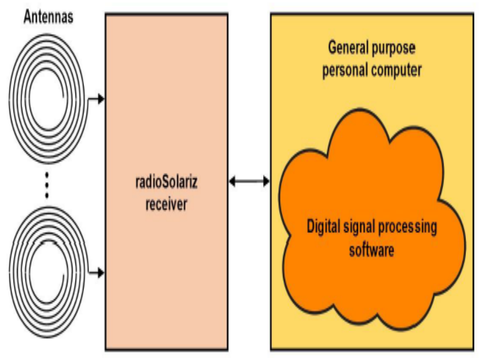
Mr.Thanniru Pavan Vinayak
Assistant Professor, Department of ECE,
Malla Reddy College of Engineering for Women.,
Maisammaguda., Medchal., TS, India

Abstract

The purpose of this article is to provide a more detailed account of the digital signal processing methods that were used to manipulate data in the radioSolariz solar radio-telescope project. The usage of streaming single instruction—multiple data extensions 2 is important to the implementation of several algorithms in digital signal processing. This supplementary set of instructions for general-purpose personal computer microprocessors allows for parallel processing, which in turn increases computing capacity. As a result, the digital signal processing computer's power consumption drops and its data throughput increases. Along with the original code snippets, we provide optimized code fragments and analyze and debate them. Additional contemporary parallel processing methods are planned for future investigation and deployment.

Introduction

The idea for radio Solariz came up in 2019 [1], and by 2020 the first prototype had been built, with the intention of beginning radio data collection in late 2020. Solar radio waves generated by the Sun in the meter and decameter bands may be detected using the radioSolariz telescope. Figure 1 shows the overall block diagram of the telescope station. Antennas, a radio receiver, and a PC are the components of the station. After receiving a signal from the antennas, the radio receiver converts it to digital format and sends it to the computer. Digital signal processing (DSP) methods are used to process the signal data there utilizing radio Solariz software [2-5]. The central processing unit (CPU) of the computer in question is a general-purpose processor that, when given standard instructions, can carry out any digital processing job. The first software prototype used this method. Streaming single instruction multiple data extensions 2 (SSE2) was later developed and implemented as a new method to enhance the system's performance [6-7].


Antennas
radioSolariz receiver
General purpose personal computer
Digital signal processing software

General block schematic of the Radio Solariz station (Fig. 1).

Processing digital signals using a single streaming command and several data extensions 2

Give me the rundown on SSE2. The central processing unit (CPU) of personal computers in the IBM-PC line uses this instruction set, which is an expansion to the regular Intel architecture IA-32 set of instructions. Following the moniker "single instruction, multiple data," this collection of instructions is designed for SIMD-based parallel data processing. So, instead of decoding numerous instructions, a single instruction may be performed on an array of comparable data, conserving power and transistors in the CPU. An additional perk is that the instruction set manages a main processor's parallel processing co-processor, which allows for the execution of many operations simultaneously. Intel first released SSE2 in 2000 with the original Pentium 4 CPU. Although Intel has released several

ISSN 2321-2152
www.ijmece.com
Vol 11, Issue 2, 2023

instruction sets for parallel computing, this one is not their first. A vast advance over its predecessor, the SSE instruction set, it supersedes Intel's branded but meaningless MMX instruction set. Intel released SSE3, an expansion of SSE2, later in 2004; nevertheless, it was never as popular as SSE2. Another version is available for SSE4. Adding 144 more instructions to the original 70 in the SSE paradigm, SSE2 expands upon it. The rival chip maker, Advanced Micro Devices (AMD), included SSE2 in their CPUs. This occurred in 2003, when AMD64 64-bit CPUs like Opteron and Athlon 64 were debuted.

Data compression, preconditioning, spectrum decomposition, filtering, signal power level extraction, and radio Solariz digital signal processing are all part of the process. These computations are ideal for parallelization since they all use the same methods to analyze huge volumes of data. However, the first version of the program used the x87 floating point unit (FPU), which is a programmable scalar and, depending on the architecture and organization of the underlying processor, may be implicitly parallelized to some degree by the central processing unit (CPU).

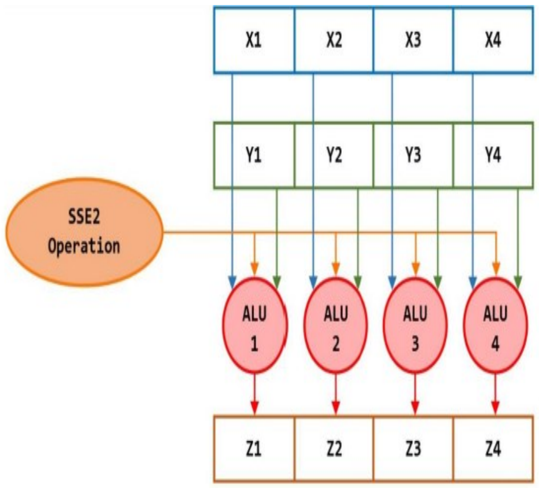
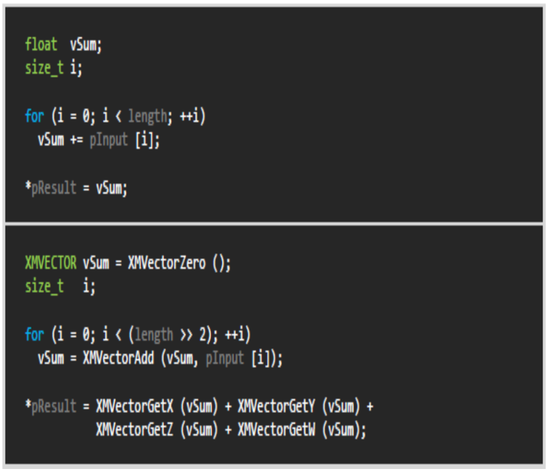


Figure 2: Streaming single instruction structure with numerous data extensions Intermediate results are calculated with 80 bits of accuracy using 2 operations on floating point 32-bit data using FPU (x87) instructions. Only numerically unstable techniques, which were not implemented in radioSolariz, need such a level of accuracy.

With four SSE2 arithmetic and logical units (ALUs) per processor core, SSE2 floating point instructions can execute two operations in parallel on 64-bit floating point data or four operations on 32-bit data simultaneously (see Fig. 2). Digital signal processing for radioSolariz only need 32-bit floating point data. In theory, this allows for a fourfold improvement in data throughput. The speed boost is lesser, but still considerable, since the CPU realizes implicit parallelism on normal x87 instructions. This is why, while doing computations on massive datasets, the second version of the program makes heavy use of SSE2 instructions. Figure 3 displays a combination of x87 and SSE2 instructions for a summing function that determines the total of an array's elements. In Figure 3, the top part shows the regular x87 code, and in the bottom part, the SSE2 code is shown. You may see examples of the two instruction sets implemented in C++ in these two pieces of code. You can see that both programs are concise, easy to understand, and comment-free. To implement SSE2 instructions in C++, no bloated code structures are required. Thanks to these advantages, the author was able to port the majority of the computationally complex code to SSE2 while maintaining readability, understandability, and debugging ease.

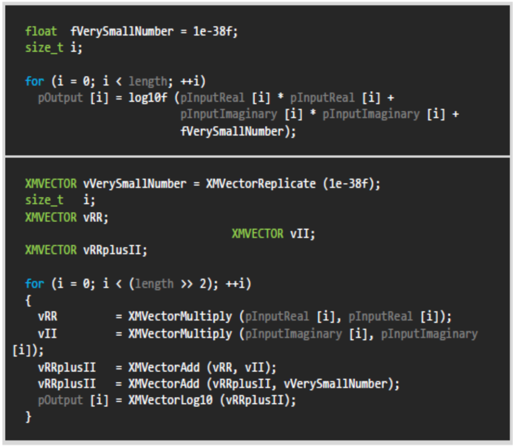

```
float  vSum;
size_t i;

for (i = 0; i < length; ++i)
  vSum += pInput [i];

*pResult = vSum;
```

```
XMVECTOR vSum = XMVectorZero ();
size_t   i;

for (i = 0; i < (length >> 2); ++i)
  vSum = XMVectorAdd (vSum, pInput [i]);

*pResult = XMVectorGetX (vSum) + XMVectorGetY (vSum) +
           XMVectorGetZ (vSum) + XMVectorGetW (vSum);
```

Figure 3: C++ code for an array sum computation utilizing the x87 and SSE2 instruction sets.

The function that determines the signal power's base-10 logarithm is another instance of an optimization that makes advantage of SSE2 instructions. If you look closely at Fig. 4, you can see bits of both the original and optimized code. The shorter original code is shown here. To hold the intermediate findings, additional local temporary variables are needed for more complicated computations. While it is possible to prevent this, doing so would result in code expressions that are so difficult to comprehend that code maintenance would suffer. There is a potential upper bound of four times performance gain in both cases. Real performance increase close to this estimate—3.5 times—was shown in the tests. This number changes depending on the tested processors, since various families of CPUs achieve super scalar parallelism to varied degrees.

There were substantial speed gains across the board for the radioSolariz software's many other functionalities that deal with massive data arrays. Due to the presence of SSE2 ALUs in every CPU core, it was feasible to achieve a second level of parallelization by executing an equal number of program threads for each core. An 8-core CPU ran our last testing. Here, a potential performance boost of up to 32 times is possible.



```
float  fVerySmallNumber = 1e-38f;
size_t i;

for (i = 0; i < length; ++i)
  pOutput [i] = log10f (pInputReal [i] * pInputReal [i] +
                        pInputImaginary [i] * pInputImaginary [i] +
                        fVerySmallNumber);
```

```
XMVECTOR vVerySmallNumber = XMVectorReplicate (1e-38f);
size_t   i;
XMVECTOR vRR;
                                XMVECTOR vII;
XMVECTOR vRRplusII;

for (i = 0; i < (length >> 2); ++i)
{
  vRR         = XMVectorMultiply (pInputReal [i], pInputReal [i]);
  vII         = XMVectorMultiply (pInputImaginary [i], pInputImaginary
[i]);
  vRRplusII   = XMVectorAdd (vRR, vII);
  vRRplusII   = XMVectorAdd (vRRplusII, vVerySmallNumber);
  pOutput [i] = XMVectorLog10 (vRRplusII);
}
```

Code in C++ for calculating the base 10 logarithm of the signal power using the x87 and SSE2 instruction sets, as shown in Figure 4.

Conclusions

Using SSE2 instructions instead of the normal x87 instructions in the program resulted in a speed boost of many times. In further updates to the radioSolariz telescope, the author is urged to further enhance the software by including new, current parallel processing hardware and software approaches, such as the installation of field programmable gate arrays (FPGAs) [8].

References

1. Zabunov, S., R. Miteva. Online Real-time Visualization of radioSolariz Spectrum and Spectrogram, Proceedings of the SES-2020 conference, ISSN 2603-3313, 2020, pp. 69–74.

2. Fog, A. Optimizing software in C++: An optimization guide for Windows, Linux and Mac platforms. Technical University of Denmark, 2021, Copyright © 2004–2021.

3. Canu, S., R. Flamary, D. Mary. Introduction to optimization with applications in astronomy and astrophysics. HAL archives-ouvertes, hal-01346134, 2016, pp. 1–36.

4. Patterson, D., J. Hennessey. Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann, 2nd edition, 1997, 916 p.

5. Knuth, D. E. The Art of Computer Programming, v. 2, Semi-numerical Algorithms. Addison-Wesley, 2nd edition, 1998, 784 p.

6. Tian, X., H. Saito, S. V. Preis, et al. Effective SIMD Vectorization for Intel Xeon Phi Coprocessors. Scientific Programming, vol. 2015, Article ID 269764, 14, 2015, pp. 1–14, https://doi.org/10.1155/2015/269764

7. Nyland, L., M. Snyder. Fast trigonometric functions using Intel's SSE2 instructions. Intel tech. rep., 2004, pp. 1–11.

8. Andraka, R. A survey of CORDIC algorithms for FPGAs. FPGA'98, the proceedings of the ACM/SIGDA sixth international symposium on Field Programmable Gate Arrays, 1998, pp. 191–200.