Bulgarian Academy of Sciences. Space Research and Technology Institute.
Aerospace Research in Bulgaria. 34, 2022, Sofia

DOI: https://doi.org/10.3897/arb.v34.e05

DIGITAL SIGNAL PROCESSING IN RADIOSOLARIZ PROJECT USING SSE2

Svetoslav Zabunov

Space Research and Technology Institute — Bulgarian Academy of Sciences

Keywords: *Radio Telescope Signal Processing, Digital Signal Processing, Streaming Single Instruction — Multiple Data Extensions 2*

Abstract

This paper aims at elaborating on the digital signal processing techniques used in data manipulation in the radioSolariz solar radio-telescope project.

Focus is drawn on the implementation of different digital signal processing algorithms through the use of streaming single instruction — multiple data extensions 2. This complementary instruction set to general purpose personal computer microprocessors offers increased computational power by realizing parallel processing. The benefit is a higher data throughput while lowering the electrical power consumption of the digital signal processing computer.

Optimized code fragments are shown along with original code snippets and these are discussed and analysed. Future work and implementation of other modern parallel processing technologies are envisaged.

Introduction

The radioSolariz project was conceived in 2019 [1] and the first prototype has been developed a year later to start collecting radio data starting from late 2020. The telescope called radioSolariz is a solar radio telescope that collects data from radio waves emitted by the Sun in the meter and decameter radio bands. The telescope's station general block diagram is shown in fig. 1. The station consists of antennas, a radio receiver and a general purpose personal computer. The radio receiver digitizes the received signal from the antennas and transfers it to the personal computer. There, using the software of radioSolariz, the signal data is processed by means of digital signal processing (DSP) techniques [2–5].

All digital processing tasks may be executed on the general purpose processor, the central processing unit (CPU) of the utilized computer, by employing standard instructions. This approach was implemented in the first software prototype. Later, for the need of improving the performance of the system, a new approach was devised — implementation of the streaming single instruction — multiple data extensions 2 (SSE2) [6–7].

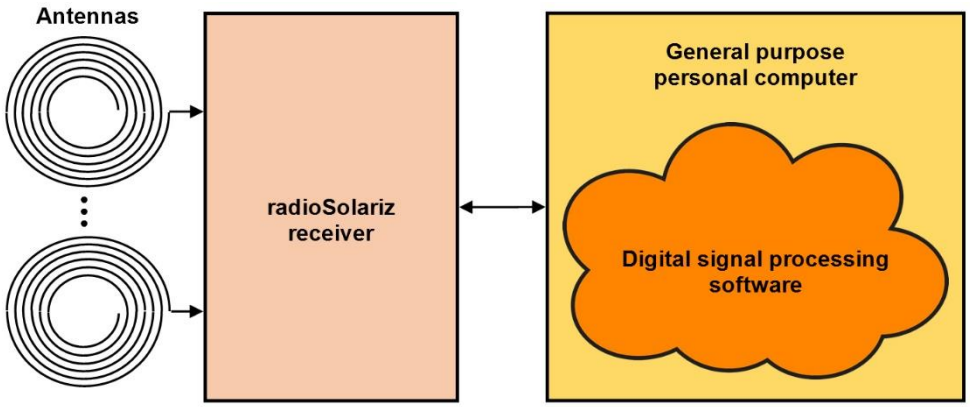

Fig. 1. RadioSolariz station general block diagram

Digital signal processing using streaming single instruction — multiple data extensions 2

What is SSE2? It is an instruction set, or more precisely an extension to the standard Intel architecture IA-32 instruction set of the CPU of the IBM-PC family personal computers. This instruction set, as its name suggests, is meant for parallel processing of data realized through the technique called single instruction, multiple data (SIMD). What this means is that a single instruction may be executed on an array of similar data, thus avoiding the process of decoding multiple instructions and saving power and transistors in the CPU by doing so. Another benefit is that the instruction set controls a parallel processing co-processor inside the main processor that can perform several operations in parallel.

SSE2 was first introduced by Intel with the initial version of the Pentium 4 processor in 2000. This instruction set is not the first parallel processing instruction set introduced by Intel. It is an improvement of the earlier SSE instruction set, and completely replaces the MMX instruction set (MMX is officially an initialism that has no meaning and is trademarked by Intel). Later in 2004 Intel introduced an extension of SSE2 called SSE3, which never reached the popularity of its predecessor. There is a SSE4 version also.

SSE2 extends the 70 instructions of the SSE model by 144 new instructions. SSE2 was implemented in the processors of the competing processor manufacturer Advanced Micro Devices (AMD). This happened in 2003 when the company introduced the Opteron and Athlon 64 AMD64 64-bit CPUs.

Digital signal processing in radioSolariz involves data preconditioning, spectral decomposition, filtering, signal power level extraction, data compression, etc. All these calculations involve processing of large amounts of data using the same operations, hence they are good candidates for parallelization. Nevertheless, the initial variant of the software relied on the classical instruction set x87 floating point

unit (FPU) that is programmatically scalar and maybe parallelized implicitly by the CPU to some extent, depending on the underlying processor architecture and algorithm structure.

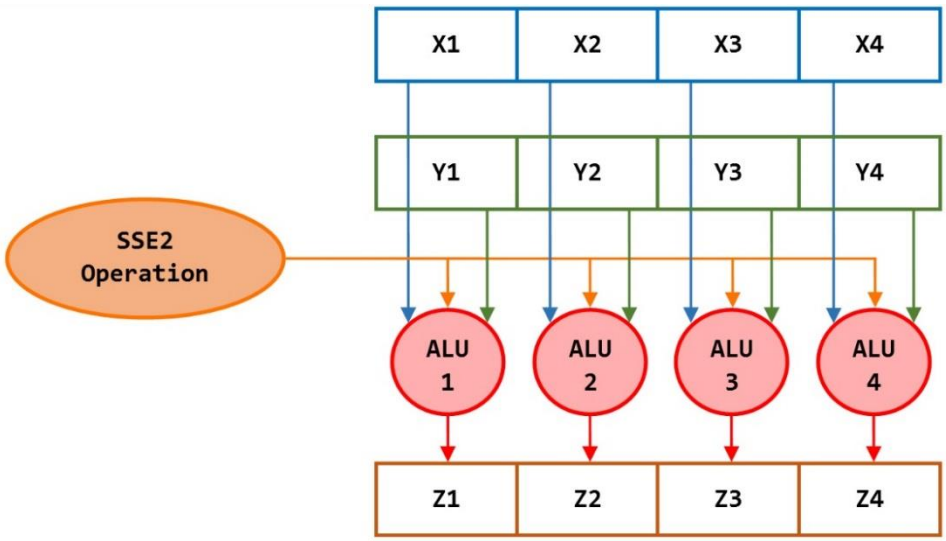

Fig. 2. Structure of streaming single instruction — multiple data extensions 2 operations on floating point 32-bit data

FPU (x87) instructions calculate intermediate results with 80 bits of precision. Such a precision is required only by numerically unstable algorithms that were not used in the radioSolariz software.

On the other hand, SSE2 floating point instructions offer the capability to perform four operation in parallel on 32-bit floating point data due to the presence of four SSE2 arithmetic and logical units (ALUs) for each processor core (see Fig. 2) or two operation in parallel on 64-bit floating point data. For the purposes of radioSolariz digital signal processing 32-bit floating point data suffices. Thus theoretically a fourfold increase in data throughput can be achieved. Due to implicit parallelism realized by the CPU on regular x87 instructions the improvement in performance is lower, but still meaningful. For this reason the second variant of the software uses extensively SSE2 instructions to perform calculations on large datasets.

Fig. 3 shows an implementation both using x87 and SSE2 instructions of a summation function that finds the sum of all elements of an array. The code in the top section of Fig. 3 is the standard x87 code while the bottom section of the same figure represents the SSE2 code. Both code snippets are representative of the respective instruction sets implementations in C++ programming language. It is visible that both codes are short, clear to read and require no comments. There are

no cumbersome code constructs when implementing SSE2 instructions in C++. All these benefits let the author translate most of the computationally intensive code to SSE2 and still keep it well readable and understandable, yet easy to debug.

```
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);
```

Fig. 3. C++ code for realizing an array sum calculation using x87 instruction set (top) and SSE2 instruction set (bottom)

Another example of an optimization using SSE2 instructions is the function used to calculate the base 10 logarithm of the signal power. The two code snippets of the original code and the optimized code are shown in Fig. 4. Here we can observe that the original code is shorter. More complex calculations require more local temporary variables to store the intermediate results. This can be avoided, but the code expression would become so hard to read that the maintenance of the code would be compromised.

In both examples a theoretical maximum improvement of the performance is four times. Tests showed real improvement of performance close to this estimation — 3.5 times. This figure varies over different processors the code is tested on, because each processor family realizes super scalar parallelism to different extent.

Many other functions that operate on large arrays of data in the radioSolariz software were optimized in a similar way and showed similar levels of performance benefits.

It was possible to realize a second level of parallelization by running as many programme threads as the number of cores were in the microprocessor, because SSE2 ALUs are present in each core of the CPU. Our final tests were executed on an 8 core processor. The final maximum theoretical performance improvement in this case is 32 times.

```
  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);
  }
```

Fig. 4. C++ code for base 10 logarithm of the signal power calculated using x87 instruction set (top) and SSE2 instruction set (bottom)

Conclusions

A several times increase in performance was observed by implementing SSE2 instructions instead of standard x87 instructions in the software. The author is encouraged to continue improving the software of the telescope through the implementation of new modern parallel processing hardware and software techniques in the next versions of the radioSolariz telescope, such as Field programmable gate arrays (FPGA) implementation [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.

ЦИФРОВА ОБРАБОТКА НА СИГНАЛИ В RADIOSOLARIZ ЧРЕЗ SSE2

С. Забунов

Резюме

Настоящата публикация цели да доразвие темата за цифровата обработка на сигнали в проекта radioSolariz. Проектът radioSolariz представлява радиотелескоп, предназначен за наблюдение на Слънцето в метровия и декаметровия радиообхвати. Цифровата обработка на сигнали в телескопа се използва за първична и вторична обработка на данните в radioSolariz.

Обърнато е внимание на приложението на различни алгоритми за цифрова обработка на сигнали чрез използване на streaming single instruction – multiple data extensions 2. Този допълнителен набор инструкции предлага подобрена изчислителна производителност, реализирана чрез паралелна обработка на данните. Предимствата са както в увеличената производителност, така и в намалената консумация на електроенергия.

Представени са алгоритмични програмни фрагменти едновременно от оригиналния и от оптимизирания код. Примерите са дискутирани и анализирани. Направен е преглед на идеите за бъдеща работа чрез приложение на модерен хардуер и софтуер за паралелна обработка на данни.