International Journal of Software Innovation
Volume 5 • Issue 2 • April-June 2017

Parallel Online Exact Summation of Floating-point Numbers by Applying MapReduce of Java8

Naoshi Sakamoto, Tokyo Denki University, Tokyo, Japan

ABSTRACT

Java8 introduced the notion of streams that is a new data structure and supports multi-core processors. When the sum method is called for a stream of floating-point numbers, the summation is calculated at high-speed by applying MapReduce, which distributes computations to cores. However, since floating-point calculation causes an error, simple adaptation of this method cannot determine the result uniquely. Then, in this study, the authors develop a summation program that can be applied to a stream with MapReduce. Their method can calculate at high-speed with keeping correctly rounded.

KEYWORDS

Floating-point Numbers, Floating-point Summation, Java8, MapReduce

1. INTRODUCTION

Java8 introduces the notion of streams, a new data structure. This is for high-speed calculation technology brought by introducing MapReduce, which applies to multi-core processors, and FunctionalInterface, a new syntax.

FunctionalInterface is an extension of the former anonymous class. An interface is called FunctionalInterface when it only contains a single abstract method. A "lambda formula", a new syntax, is a literal of a FunctionalInterface. This is corresponding to an instance of the class implemented a single method to an anonymous class. By this extension, while methods are not still the first class object in Java, we can deal with the object of FunctionalInterface as a function object (Figure 1, l.5).

For a set of data and a given function, a *Map* process generates a set of results by applying the function for each data (Figure 1, ll. 5–7). By assuming that the function has no adverse reaction, each calculation of the function can be distributed to cores. Note that the processing order to the data is not determined uniquely.

On the other hand, for a set of data and a given binary operator, a *Reduce* process calculates aggregation for the operator by parallel processing (Figure 1, ll. 9–11). This is not so executed that each data and the intermediate result are binary-operated, and the result is let to be the new intermediate result one by one. But, this is so executed as making each data correspond to a leaf of a complete binary tree, then binary-operates at each position of medium nodes.

It has been well-known that summation for floating-point numbers causes a rounding error. It is also known that adding numbers in order from the smallest number to the largest number reduces the rounding error. That is, the order of adding for floating-point numbers affects the results. Thus, adding floating-point numbers does not satisfy the associative law.

DOI: 10.4018/IJSI.2017040102

Figure 1. Examples of processing streams

```
1. import java.util.function.DoubleUnaryOperator;
2. import java.util.stream.DoubleStream;
3. import java.util.function.DoubleBinaryOperator;
4.
5. DoubleUnaryOperator func1 = x->x+0.1;
6. DoubleStream stream11 = Arrays.stream(new double[]{1.0,
   2.0, 3.0, 4.0});
7. DoubleStream stream12 = stream11.map(func1);
8.
9. DoubleBinaryOperator func2 = (x,y)->x*y;
10. DoubleStream stream2 = Arrays.stream(new double[]{1.0,
   2.0, 3.0, 4.0});
11. double value2 = stream2.reduce(func2).get();
```

Nevertheless, adding floating-point numbers used to be executed with paying less attention to the error. Fortunately, since formerly adding has been executed in order, the results are the same for any conditions of any computers. On the other hand, the sum method of java.util.stream.DoubleStream of Java8 is also implemented with paying less attention to the error. However, by introducing the notion of stream, *Reduce* causes the discrepancy of the summation result. That is, the result of sum method of java.util.stream.DoubleStream may cause the discrepancy of the result for the environment and the condition of computers.

On the other hand, Java is equipped with BigDecimal for arbitrary-precision arithmetic. Then, we can calculate without errors by applying BigDecimal to stream (Figure 2). However, in spite of the benefits of multi-core, it spends quite long time.

In this study, we develop a stable fast program for summation of floating-point numbers by using Stream. The proposal program is independent from the order of numbers, has the same precision as BigDecimal, is at most three times slower than sum method of DoubleStream. Since basic technique called AddTwo contains six additions, the measured running time is seemed be quite good.

This paper is organized as follows: Section 2 gives basic notions, and introduces AddTwo, iFastSum, and Online Exact Sum as the essential basic notions. Section 3 explains the proposal program. Section 4 shows the result of experiments, and evaluation. Finally, Section 5 concludes.

2. PRELIMINARIES

2.1. Floating Point

"Double" type is used by many programming languages. This is a binary format of IEEE 754 that occupies 64 bits. Double type of Java8 also occupies 64bit, and consists of the sign (1 bit), the significand (52 bits), and the exponent (11 bits). Let each of values interpreted these as an unsigned integer denote s, d, and e, respectively. There are the normal number representation and the subnormal number representation. When $e \neq 0$ then the representation denotes a normal number, where the value is given by $v = \left(-1\right)^{s} \cdot \left(1 + d \cdot 2^{-53}\right) \cdot 2^{e-1023}$. On the other hand, when $e=0$, the representation denotes a subnormal number, where the value is given by $v = \left(-1\right)^{s} \cdot \left(d \cdot 2^{-53}\right) \cdot 2^{-1023}$. Normal numbers have

Exploring the Antecedents for Continuance Intention of Electronic Litigation Systems

Donghyuk Joand Hyeon Cheol Kim (2022). *International Journal of Software Innovation (pp. 1-12)*.
www.irma-international.org/article/exploring-the-antecedents-for-continuance-intention-of-electronic-litigation-systems/309730