A Parallel Algorithm of Accurate Dot Product

N. Yamanaka [a] T. Ogita [b,c] S. M. Rump [d] S. Oishi [c]

[a] *Graduate School of Science and Engineering, Waseda University, Tokyo 169-8555, Japan*

[b] *CREST, Japan Science and Technology Agency (JST)*

[c] *Faculty of Science and Engineering, Waseda University, Tokyo 169-8555, Japan*

[d] *Institute for Reliable Computing, Hamburg University of Technology, Hamburg 21071, Germany*

Abstract

Parallel algorithms for accurate summation and dot product are proposed. They are parallelized versions of fast and accurate algorithms of calculating sum and dot product using error-free transformations which are recently proposed by the authors (Ogita, Rump and Oishi) in [*Accurate sum and dot product*, SIAM J. Sci. Comput., 26:6 (2005), 1955–1988]. They have shown their algorithms are fast in terms of measured computing time. However, due to the strong data dependence in the process of their algorithms, it is difficult to parallelize them. Similarly to their algorithms, the proposed parallel algorithms in this paper are designed to achieve the results as if computed in K-fold working precision with keeping the fastness of their algorithms. Numerical results are presented showing the performance of the proposed parallel algorithm of calculating a dot product.

Key words: Parallel algorithm, Accurate dot product, Accurate summation, Higher precision.

1 Introduction

Let $\mathbb{F}$ be a set of floating-point numbers. Let $x, y \in \mathbb{F}^n$. In this paper, we present a parallel algorithm to compute a dot product $x^T y$. Since dot product is a most basic task in numerical analysis, there are a number of algorithms for

Email address: naoya_yamanaka@suou.waseda.jp (N. Yamanaka).

that. Accurate dot product algorithms have various applications in numerical analysis. Excellent overviews can be found in [4,6].

Recently, the authors (Ogita, Rump and Oishi) presented an accurate dot product algorithm **DotK** [7] using so-called "error-free transformations". Their algorithm is fast, not only in terms of floating-point operations but also in terms of measured computing time on a serial computer. However, it is difficult to parallelize **DotK** because of the strong data dependence.

To overcome this, we develop a parallelizing method for **DotK** and present a new algorithm **PDotK** of calculating a dot product whose result is aimed to be as accurate as that by **DotK**. It is shown that an error bound on the result by **PDotK** is less than or equal to that by **DotK**.

This paper is organized as follows: In Section 2, we briefly review the error-free transformations and the algorithms of accurate summation and dot product (**SumK** and **DotK**) in [7]. In Section 3, we present parallel algorithms **PSumK** and **PDotK**, which are the parallelized versions of **SumK** and **DotK**, respectively. We also analyze the proposed algorithms and present the theorems for **PSumK** and **PDotK**, which confirm that the proposed parallel algorithms achieve the results as in K-fold precision. Finally, we present results of numerical experiments showing the performance of the proposed algorithm **PDotK** in Section 4.

The developed parallel algorithms can be applied not only to shared memory systems but to distributed memory systems.

2 Accurate Sum and Dot Product

In this section, we briefly review some algorithms used in the accurate dot product algorithm **DotK** [7].

Throughout this paper, we assume floating-point arithmetic adhering to IEEE standard 754. Let $fl(\cdots)$ be the result of floating-point operations, where all operations inside parentheses are executed by ordinary floating-point arithmetic in rounding-to-nearest. We donote by $\mathbf{u}$ the machine epsilon; in IEEE standard 754 double precision $\mathbf{u} = 2^{-53}$. We assume that neither overflow nor underflow occur.

Following [4], we define γ_n as

$$\gamma_n := \frac{n\mathbf{u}}{1 - n\mathbf{u}} \quad \text{for} \quad n \in \mathbb{N}.$$

When using γ_n, we implicitly assume that $n\mathbf{u} < 1$.

Let $p = (p_1, \ldots, p_n)^T \in \mathbb{F}^n$. Then it holds that [4]

$$\tilde{s} := fl\left(\sum_{i=1}^{n} p_i\right) \quad \Longrightarrow \quad \left|\tilde{s} - \sum_{i=1}^{n} p_i\right| \leq \gamma_{n-1} \sum_{i=1}^{n} |p_i| \,. \tag{1}$$

Note that (1) is valid for any order of addition in the summation.

The algorithm **DotK** is based on the error-free transformations of addition and multiplication of two floating-point numbers. Following [7], we call the algorithms **TwoSum** and **TwoProduct**, respectively.

First, we introduce the addition algorithm **TwoSum**. Knuth [1] presented the following algorithm [1] which transforms a pair (x, y) with $x, y \in \mathbb{F}$ into a new pair (a, b) with $a, b \in \mathbb{F}$ satisfying $x + y = a + b$ with $a = fl\,(x + y)$, $|b| \leq \mathbf{u}\,|a|$.

Algorithm 2.1 (Knuth [1]) *Error-free transformation of the sum of two floating-point numbers.*

$$\begin{aligned} &\textbf{function } [a, b] = \textbf{TwoSum}(x, y) \\ &\quad a = fl\,(x + y) \\ &\quad c = fl\,(a - x) \\ &\quad b = fl\,((x - (a - c)) + (y - c)) \end{aligned}$$

Next, we will see the multiplication algorithm **TwoProduct**, which transforms a pair (x, y) with $x, y \in \mathbb{F}$ into a new pair (a, b) with $a, b \in \mathbb{F}$ satisfying $x \cdot y = a + b$, $|b| \leq \mathbf{u}\,|a|$.

The multiplication routine needs to split the input arguments into two parts. For the number t given by $\mathbf{u} = 2^{-t}$, we define $s := \lceil t/2 \rceil$; in IEEE 754 double precision we have $t = 53$ and $s = 27$. The following algorithm by Dekker [2] splits a floating point number $x \in \mathbb{F}$ into two parts x, y, where both parts have at most $(s - 1)$ nonzero bits.

Algorithm 2.2 (Dekker [2]) *The algorithm* **Split** *splits a t-bits floating-point number $x \in \mathbb{F}$ into $x_h, x_t \in \mathbb{F}$ such that $x = x_h + x_t$.*

$$\begin{aligned} &\textbf{function } [x_h, x_t] = \textbf{Split}(x) \\ &\quad c = fl\,(\textbf{factor} \cdot x) \qquad\qquad \% \ \textbf{factor} = 2^{\lceil t/2 \rceil} + 1 \\ &\quad x_h = fl\,(c - (c - x)) \\ &\quad x_t = fl\,(x - x_h) \end{aligned}$$

[1] Throughout the paper, we use Matlab-style notation for describing algorithms.

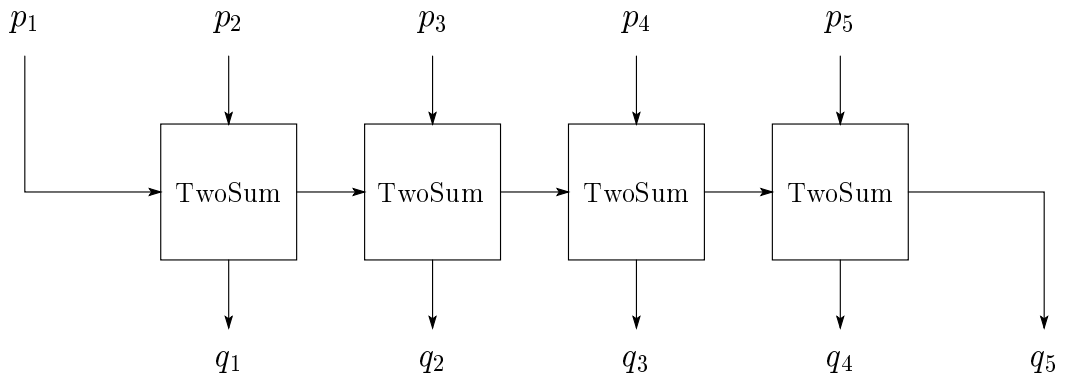

Fig. 1. Outline of **VecSum** for $n = 5$

Using **Split**, the following multiplication routine by G.W. Veltkamp (see [2]) can be formulated.

Algorithm 2.3 (Veltkamp (see [2])) *Error-free transformation of the product of two floating-point numbers.*

$$
\begin{aligned}
&\textbf{function } [a, b] = \textbf{TwoProduct}(x,\ y)\\
&\quad a = fl\,(x \cdot y)\\
&\quad [x_1, x_2] = \textbf{Split}(x)\\
&\quad [y_1, y_2] = \textbf{Split}(y)\\
&\quad b = fl\,(x_2 \cdot y_2 - (((a - x_1 \cdot y_1) - x_2 \cdot y_1) - x_1 \cdot y_2))
\end{aligned}
$$

The algorithm **TwoSum** can be extended to an error-free *vector* transformation with respect to the summation:

Algorithm 2.4 (Ogita et al. [7]) *The algorithm* **VecSum** *transforms a vector* $p = (p_1, p_2, \ldots, p_n)^T \in \mathbb{F}^n$ *into a new vector* $q = (q_1, q_2, \ldots, q_n)^T \in \mathbb{F}^n$ *satisfying* $\sum_{i=1}^{n} p_i = \sum_{i=1}^{n} q_i$.

$$
\begin{aligned}
&\textbf{function } q = \textbf{VecSum}(p)\\
&\quad q_1 = p_1\\
&\quad \textit{for } i = 2 : n\\
&\qquad [q_i, q_{i-1}] = \textbf{TwoSum}(p_i, q_{i-1})\\
&\quad \textit{end}
\end{aligned}
$$

Here, the input vector p is transformed into the output vector q without changing the sum, and q_n is replaced by $fl\,(\sum p_i)$. Kahan [5] calls this a "distillation algorithm". Figure 1 illustrates an outline of **VecSum** for $n = 5$.

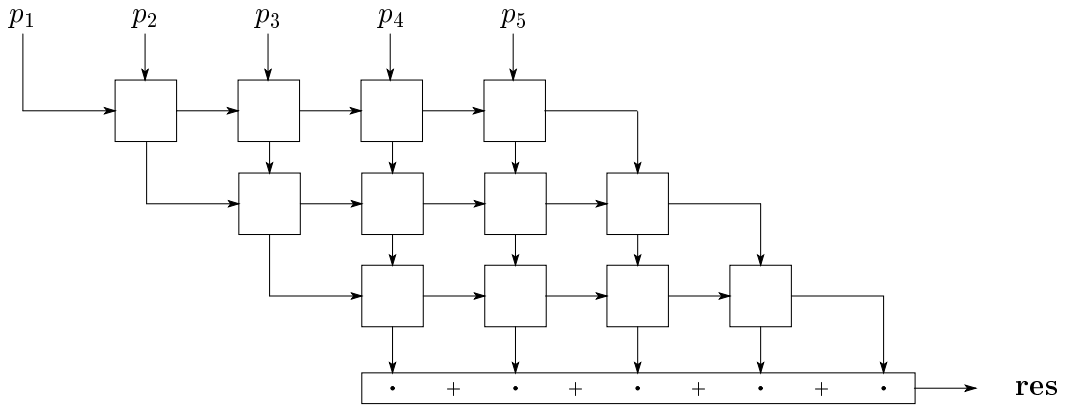

Fig. 2. Outline of **SumK** for $n = 5$ and $K = 4$

By (1), the algorithm **VecSum** satisfies [7, (4.4)]

$$\sum_{i=1}^{n-1} |q_i| \leq \gamma_{n-1} \sum_{i=1}^{n} |p_i| \,. \tag{2}$$

In [7], the authors (Ogita, Rump and Oishi) developed an algorithm **SumK**, which calculates the summation using **VecSum** iteratively. By **SumK**, we can obtain the result as if computed in K-fold precision.

Algorithm 2.5 (Ogita et al. [7]) *Summation $\sum_{i=1}^{n} p_i$ for $p \in \mathbb{F}^n$ as in K-fold precision by $(K-1)$-fold error-free vector transformation.*

$$\begin{aligned}
&\textbf{function res} = \textbf{SumK}(p,\, K) \\
&\quad \textit{for } i = 1 \ : K-1 \\
&\qquad p = \textbf{VecSum}(p) \\
&\quad \textit{end} \\
&\quad \textbf{res} \ = fl\left(\sum_{i=1}^{n} p_i\right)
\end{aligned}$$

Figure 2 illustrates an outline of **SumK** for $n = 5$ and $K = 4$.

Denote s and S by

$$s := \sum_{i=1}^{n} p_i, \qquad S := \sum_{i=1}^{n} |p_i| \tag{3}$$

Condition number of the summation of the vector p is defined by

$$\mathrm{cond}\left(\sum_{i=1}^{n} p_i\right) := \frac{S}{|s|}, \quad s \neq 0.$$

Then error bounds of the result **res** by **SumK** are given as follows [7]:

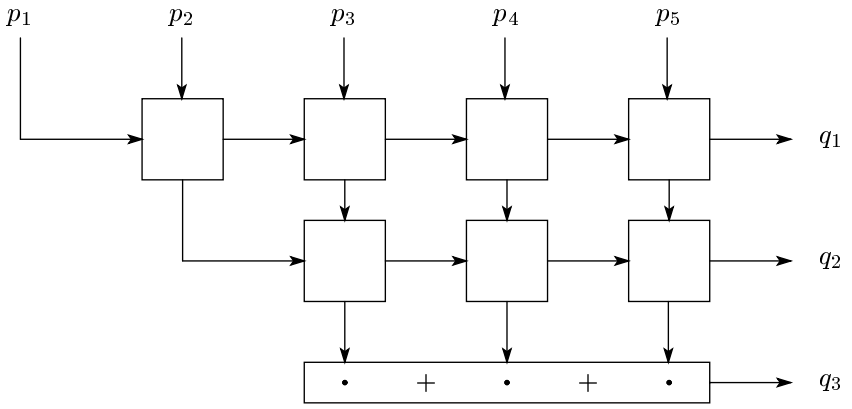

Fig. 3. Outline of **SumL** for $n = 5$ and $L = 3$

Theorem 2.6 (Ogita et al. [7]) *Let* **res** *be the result obtained by* **SumK**, *then*

$$|\mathbf{res} - s| \leq (\mathbf{u} + 3\gamma_{n-1}^2)|s| + \gamma_{2(n-1)}^K S \tag{4}$$

and

$$\frac{|\mathbf{res} - s|}{|s|} \leq \mathbf{u} + 3\gamma_{n-1}^2 + \gamma_{2(n-1)}^K \cdot \mathrm{cond}\left(\sum_{i=1}^n p_i\right). \tag{5}$$

Basically, the theorem says

$$\frac{|\mathbf{res} - s|}{|s|} \lesssim \mathbf{u} + \mathcal{O}(\mathbf{u}^K) \cdot \mathrm{cond}\left(\sum_{i=1}^n p_i\right),$$

which means **res** is computed in internally K-fold working precision.

We present here a summation algorithm **SumL** whose result is represented by a sum of L floating-point numbers.

Algorithm 2.7 *Summation* $\sum_{i=1}^n p_i$ *for* $p \in \mathbb{F}^n$ *as in* L*-fold precision whose result is represented by a sum of* L *floating-point numbers.*

function $q = \mathbf{SumL}(p,\, L)$

for $k = 1 : L - 1$

$p(1 : n - k + 1) = \mathbf{VecSum}(p(1 : n - k + 1))$

$q_k = p_{n-k+1}$

end

$q_L = \mathrm{fl}\left(\sum_{i=1}^{n-L+1} p_i\right)$

Figure 3 illustrates an outline of **SumL** for $n = 5$ and $L = 3$.

For later use, we present the following theorem for **SumL**.

Theorem 2.8 *Algorithm 2.7* (**SumL**) *satisfies the following inequalities:*

$$\left|\sum_{k=1}^{L} q_k - \sum_{i=1}^{n} p_i\right| \le \gamma_{n-1}^{L} \sum_{i=1}^{n} |p_i| \tag{6}$$

$$\sum_{k=1}^{L} |q_k| \le (1+\gamma_{2(n-1)}) \sum_{i=1}^{n} |p_i| \tag{7}$$

Proof. We first prove (6). Let $p^{(j)} = \left(p_1^{(j)},\ p_2^{(j)}, \ldots, p_{n-j+1}^{(j)}\right)^T$, $1 \le j \le L-1$, denote the intermediate vector as the result of **VecSum** of j-th loop in **SumL**. Let $p^{(0)} := p$. Then **SumL** satisfies

$$\sum_{i=1}^{n} p_i = \sum_{k=1}^{j} q_k + \sum_{i=1}^{n-j} p_i^{(j)} \quad \text{for} \quad 1 \le j \le L-1.$$

For $j = L-1$, it follows by (1) that

$$\begin{aligned}
\left|\sum_{k=1}^{L} q_k - \sum_{i=1}^{n} p_i\right| &= \left|\sum_{k=1}^{L} q_k - \left(\sum_{k=1}^{L-1} q_k + \sum_{i=1}^{n-L+1} p_i^{(L-1)}\right)\right| = \left|q_L - \sum_{i=1}^{n-L+1} p_i^{(L-1)}\right| \\
&= \left|fl\left(\sum_{i=1}^{n-L+1} p_i^{(L-1)}\right) - \sum_{i=1}^{n-L+1} p_i^{(L-1)}\right| \\
&\le \gamma_{n-L} \sum_{i=1}^{n-L+1} \left|p_i^{(L-1)}\right|.
\end{aligned}$$

Applying (2) to this inductively, we have

$$\left|\sum_{j=1}^{L} q_j - \sum_{i=1}^{n} p_i\right| \le \gamma_{n-L}\gamma_{n-L+1} \sum_{i=1}^{n-L+2} \left|p_i^{(L-2)}\right| \le \cdots \le \gamma_{n-1}^{L} \sum_{i=1}^{n} |p_i|.$$

The inequality (6) is proved.

We next prove (7). Using the fact that $q_k = fl\left(\sum_{i=1}^{n-k+1} p_i^{(k-1)}\right)$ and (1) yield

$$|q_k| \le (1+\gamma_{n-k}) \sum_{i=1}^{n-k+1} \left|p_i^{(k-1)}\right| \quad \text{for} \quad 1 \le k \le L.$$

Again applying (2) to this inductively, we have

$$\begin{aligned}
|q_k| &\le (1+\gamma_{n-k})\gamma_{n-k+1} \sum_{i=1}^{n-k+2} \left|p_i^{(k-2)}\right| \\
&\le (1+\gamma_{n-k})\gamma_{n-k+1} \cdots \gamma_{n-1} \sum_{i=1}^{n} |p_i| \\
&\le (1+\gamma_{n-k})\gamma_{n-1}^{k-1} \sum_{i=1}^{n} |p_i|.
\end{aligned}$$

Using $\gamma_n < 1$, we obtain

$$\begin{aligned}\sum_{k=1}^{L} |q_k| &= \sum_{k=1}^{L} (1+\gamma_{n-k})\gamma_{n-1}^{k-1} \sum_{i=1}^{n} |p_i| \\ &\le (1+\gamma_{n-1}) \sum_{k=1}^{L} \gamma_{n-1}^{k-1} \sum_{i=1}^{n} |p_i| \\ &\le \frac{1+\gamma_{n-1}}{1-\gamma_{n-1}} \sum_{i=1}^{n} |p_i| = (1+\gamma_{2(n-1)}) \sum_{i=1}^{n} |p_i| \,,\end{aligned}$$

which is the desired formula (7). □

We can see from (6) that the resultant vector q of **SumL** satisfies

$$\frac{\left|\sum_{k=1}^{L} q_k - s\right|}{|s|} \lesssim \mathcal{O}(\mathbf{u}^L) \cdot \mathrm{cond}\left(\sum_{i=1}^{n} p_i\right),$$

where $s = \sum_{i=1}^{n} p_i$, which means q is computed in L-fold working precision.

We proceed to the dot product. Let $x, y \in \mathbb{F}^n$. Now, we name the following algorithm **VecProduct**, which is used in **DotK** and transforms a dot product $x^T y$ into a summation $\sum_{i=1}^{2n} t_i$ such that $x^T y = \sum_{i=1}^{2n} t_i$.

Algorithm 2.9 *The algorithm* **VecProduct** *transforms two vectors* $x, y \in \mathbb{F}^n$ *into a new vector* $t \in \mathbb{F}^{2n}$ *satisfying* $x^T y = \sum_{i=1}^{2n} t_i$.

function $t = \mathbf{VecProduct}(x,\ y)$
  $r = 0$
  for $i = 1 : n$
    $[h_i,\ t_i] = \mathbf{TwoProduct}(x_i,\ y_i)$
    $[r,\ t_{n+i-1}] = \mathbf{TwoSum}(r,\ h_i)$
  end
  $t_{2n} = r$

Figure 4 illustrates an outline of **VecProduct** for $n = 4$.

For later use, we present the following theorem for **VecProduct**.

Theorem 2.10 *Algorithm 2.9* (**VecProduct**) *satisfies the following inequal-*

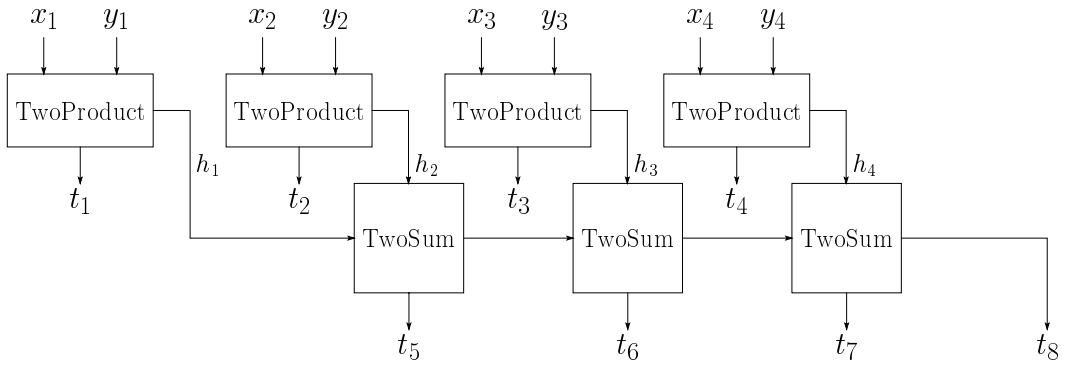

Fig. 4. Outline of **VecProduct** for $n = 4$

ities:

$$\sum_{i=1}^{2n-1} |t_i| \leq \gamma_n \sum_{i=1}^{n} |x_i y_i| \tag{8}$$

$$|t_{2n}| \leq (1+\gamma_n) \sum_{i=1}^{n} |x_i y_i| \,. \tag{9}$$

Proof. We first prove (8). The left-hand side of (8) can be separated as follows:

$$\sum_{i=1}^{2n-1} |t_i| = \sum_{i=1}^{n} |t_i| + \sum_{i=n+1}^{2n-1} |t_i| \,. \tag{10}$$

By the property of **TwoProduct**, it holds that $|t_i| \leq \mathbf{u}|h_i|$ for $1 \leq i \leq n$, so that

$$\sum_{i=1}^{n} |t_i| \leq \mathbf{u} \sum_{i=1}^{n} |h_i| \,. \tag{11}$$

An n-vector $(t_{n+1}, t_{n+2}, \ldots, t_{2n})^T$ is obtained by the iterative use of **TwoSum**, which is identical to the result of **VecSum**(h) for $h = (h_1, h_2, \ldots, h_n)^T$. Therefore, (2) yields

$$\sum_{i=n+1}^{2n-1} |t_i| \leq \gamma_{n-1} \sum_{i=1}^{n} |h_i| \,. \tag{12}$$

Moreover, it follows by $h_i = fl\,(x_i y_i)$ that

$$|h_i| \leq (1+\mathbf{u})\,|x_i y_i|. \tag{13}$$

Inserting (11), (12) and (13) into (10), we have

$$\begin{aligned} \sum_{i=1}^{2n-1} |t_i| &\leq \mathbf{u} \sum_{i=1}^{n} |h_i| + \gamma_{n-1} \sum_{i=1}^{n} |h_i| = (\mathbf{u} + \gamma_{n-1}) \sum_{i=1}^{n} |h_i| \\ &\leq (1+\mathbf{u})\,(\mathbf{u} + \gamma_{n-1}) \sum_{i=1}^{n} |x_i y_i|. \end{aligned} \tag{14}$$

Here, it holds that

$$(1+\mathbf{u})(\mathbf{u}+\gamma_{n-1}) = (1+\mathbf{u})\frac{n\mathbf{u}-(n-1)\,\mathbf{u}^2}{1-(n-1)\,\mathbf{u}} \le \frac{n\mathbf{u}}{1-n\mathbf{u}} = \gamma_n. \tag{15}$$

Inserting (15) into (14) proves (8).

We next prove (9). As mentioned before, the vector $(t_{n+1}, t_{n+2}, \ldots, t_{2n})^T$ is identical to the result of **VecSum**(h), so that $t_{2n} = fl\,(\sum_{i=1}^{n} h_i)$. By (1), we have

$$\left| t_{2n} - \sum_{i=1}^{n} h_i \right| \le \gamma_{n-1} \sum_{i=1}^{n} |h_i|. \tag{16}$$

It follows by (13) and (16) that

$$\begin{aligned} |t_{2n}| &\le \left| \sum_{i=1}^{n} h_i \right| + \gamma_{n-1} \sum_{i=1}^{n} |h_i| \le (1+\gamma_{n-1}) \sum_{i=1}^{n} |h_i| \\ &\le (1+\gamma_{n-1})(1+\mathbf{u}) \sum_{i=1}^{n} |x_i y_i| \\ &\le \frac{1}{1-n\mathbf{u}} \sum_{i=1}^{n} |x_i y_i| = (1+\gamma_n) \sum_{i=1}^{n} |x_i y_i|, \end{aligned}$$

which proves (9). □

Utilizing **VecProduct** and **SumK**, the authors (Ogita, Rump and Oishi) developed the algorithm **DotK**, which calculates the dot product as in K-fold precision.

Algorithm 2.11 (Ogita et al. [7]) *Dot product $x^T y$ for $x, y \in \mathbb{F}^n$ as in K-fold precision.*

$$\begin{aligned} &\textbf{function res} = \textbf{DotK}\,(x,\ y,\ K) \\ &\quad t = \textbf{VecProduct}(x,\ y) \\ &\quad \textbf{res} = \textbf{SumK}(t,\ K-1) \end{aligned}$$

Condition number of the dot product $x^T y$ is defined by

$$\mathrm{cond}\left(x^T y\right) := 2\frac{|x^T||y|}{|x^T y|}, \quad x^T y \neq 0.$$

Then error bounds of the result **res** by **DotK** are given as follows [7]:

Theorem 2.12 (Ogita et al. [7]) *Let* **res** *be the result obtained by* **DotK**, *then*

$$|\mathbf{res} - x^T y| \le \left(\mathbf{u} + 2\gamma_{4n-2}^2\right)|x^T y| + \gamma_{4n-2}^K |x^T||y| \tag{17}$$

and

$$\frac{|\mathbf{res} - x^T y|}{|x^T y|} \leq \mathbf{u} + 2\gamma_{4n-2}^2 + \frac{1}{2}\gamma_{4n-2}^K \cdot \mathrm{cond}\left(x^T y\right). \tag{18}$$

Similarly to Theorem 2.6, Theorem 2.12 says

$$\frac{|\mathbf{res} - x^T y|}{|x^T y|} \lesssim \mathbf{u} + \mathcal{O}(\mathbf{u}^K) \cdot \mathrm{cond}\left(x^T y\right),$$

which means the result **res** by **DotK** is also computed in internally K-fold working precision.

3 Parallel Algorithms

In this section, we will develop a parallel algorithm for calculating dot product in K-fold precision. First, we will present an algorithm of parallelizing **SumK**, which is named **PSumK**. Based on **PSumK**, we will present an algorithm of parallelizing **DotK**, which is named **PDotK**.

Suppose the number of CPUs to be $M \geq 2$ on a shared memory system. Then the CPUs are numbered as id from 1 to M, so $1 \leq id \leq M$. Although we presume that the shared memory system to be used for shortness' sake, our algorithm can also be applied to a distributed memory system.

3.1 Parallelizing Method of **SumK** (**PSumK**)

We first present an outline of **PSumK** which is a parallelized version of **SumK**. Let $p \in \mathbb{F}^n$.

Procedure 1: Distribute sub-vectors of p which are divided into M pieces to all CPUs. Every CPU with $2 \leq id \leq M$ has $c := \lceil n/M \rceil$ components. The CPU with $id = 1$ has $n - c\,(M-1)$ components. Let $p^{(id)}$ denote the sub-vector on the id-th CPU. Let c_{id} denote the number of components of $p^{(id)}$, then it holds that

$$c_1 \leq c_2 = c_3 = \cdots = c_M = c.$$

Procedure 2: Use $\mathbf{SumL}(p^{(id)},\ K)$, whose output is denoted by $q^{(id)}$. Then $q^{(id)}$ with K components is obtained on every CPU.

Procedure 3: Gather $q^{(id)}$ for all id into a vector q on the CPU with $id = 1$. Then the length of the gathered vector q becomes MK.

Procedure 4: Use $\mathbf{SumK}(q,\ K)$ on CPU with $id = 1$.

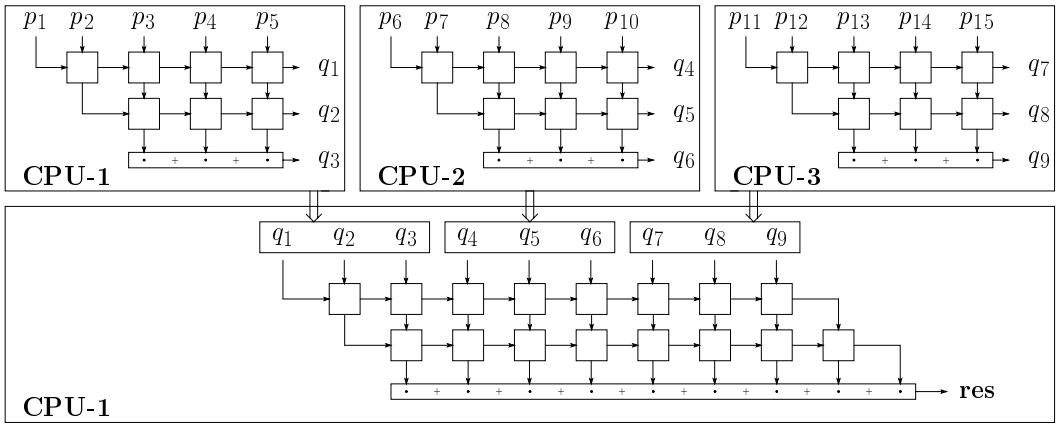

Fig. 5. Outline of **PSumK** for $n = 15$, $K = 3$ and $M = 3$

An important point is that the output $q^{(id)}$ with K components is necessary using **SumL** in Procedure 2. Otherwise, it is not possible to achieve the result with K-fold precision.

Next, we present here the concrete algorithm of **PSumK**.

Algorithm 3.1 (PSumK) *A parallelized version of* **SumK**.

function $\mathbf{res} = \mathbf{PSumK}\,(p,\ K,\ M)$

$c = \lceil \frac{n}{M} \rceil, \quad c_1 = n - c\,(M-1)$

% parallel private (index1, index2)

if $id == 1$

index1 $= 1 : c_1$

else

index1 $= c_1 + c\,(id-2) + 1 : c_1 + c\,(id-1)$

end

index2 $= K\,(id-1) + 1 : K \cdot id$

$q\,[\text{index2}] = \mathbf{SumL}\,(p\,[\text{index1}],\ K)$

% end parallel

$\mathbf{res} = \mathbf{SumK}\,(q,\ K)$

Figure 5 illustrates an outline of **PSumK** for $n = 15$, $K = 3$ and $M = 3$.

As in (3), s and S are defined by

$$s := \sum_{i=1}^{n} p_i, \qquad S := \sum_{i=1}^{n} |p_i|\,.$$

Then we present the following theorem for **PSumK**.

Theorem 3.2 *Let* **res** *be the result obtained by Algorithm 3.1* (**PSumK**). *Suppose* $\max\{n, 2(MK-1)\}\mathbf{u} < 1$. *Then the following inequalities hold:*

$$|\mathbf{res} - s| \leq (\mathbf{u} + 3\gamma_{MK-1}^2)|s| + P_1\left(\gamma^K\right) S \tag{19}$$

and

$$\frac{|\mathbf{res} - s|}{|s|} \leq \mathbf{u} + 3\gamma_{MK-1}^2 + P_1\left(\gamma^K\right) \mathrm{cond}\left(\sum p_i\right), \tag{20}$$

where

$$\begin{aligned} P_1\left(\gamma^K\right) &:= (1 + \mathbf{u} + 3\gamma_{MK-1}^2)\gamma_{c-1}^K + (1 + \gamma_{2(c-1)})\gamma_{2(MK-1)}^K \\ &= \mathcal{O}\left(\gamma_{\max\{c-1,2(MK-1)\}}^K\right). \end{aligned}$$

Proof. From the definition of vectors p, $p^{(id)}$, q and $q^{(id)}$, we collect the following equalities:

$$\sum_{i=1}^{n} p_i = \sum_{id=1}^{M} \sum_{i=1}^{c_{id}} p_i^{(id)} = s \tag{21}$$

$$\sum_{i=1}^{n} |p_i| = \sum_{id=1}^{M} \sum_{i=1}^{c_{id}} |p_i^{(id)}| = S \tag{22}$$

$$\sum_{j=1}^{MK} q_j = \sum_{id=1}^{M} \sum_{k=1}^{K} q_k^{(id)} \tag{23}$$

$$\sum_{j=1}^{MK} |q_j| = \sum_{id=1}^{M} \sum_{k=1}^{K} \left|q_k^{(id)}\right|. \tag{24}$$

Then

$$|\mathbf{res} - s| = \left|\mathbf{res} - \sum_{j=1}^{MK} q_j + \sum_{j=1}^{MK} q_j - s\right| \leq \left|\mathbf{res} - \sum_{j=1}^{MK} q_j\right| + \left|\sum_{j=1}^{MK} q_j - s\right|. \tag{25}$$

Here, using $\mathbf{res} = \mathbf{SumK}(q,\ K)$ and (4) yields

$$\begin{aligned} \left|\mathbf{res} - \sum_{j=1}^{MK} q_j\right| &\leq \left(\mathbf{u} + 3\gamma_{MK-1}^2\right) \left|\sum_{i=1}^{MK} q_i\right| + \gamma_{2(MK-1)}^K \sum_{i=1}^{MK} |q_i| \\ &\leq \left(\mathbf{u} + 3\gamma_{MK-1}^2\right) \left(\left|\sum_{i=1}^{MK} q_i - s\right| + |s|\right) + \gamma_{2(MK-1)}^K \sum_{i=1}^{MK} |q_i|. \end{aligned} \tag{26}$$

Inserting (26) into (25), we have

$$|\mathbf{res} - s| \leq \mathbf{u}'|s| + (1 + \mathbf{u}') \left|\sum_{i=1}^{MK} q_i - s\right| + \gamma_{2(MK-1)}^K \sum_{i=1}^{MK} |q_i|, \tag{27}$$

where $\mathbf{u}' := \mathbf{u} + 3\gamma_{MK-1}^2$. It follows by (21) and (23) that

$$\left|\sum_{j=1}^{MK} q_j - s\right| = \left|\sum_{id=1}^{M}\sum_{k=1}^{K} q_k^{(id)} - \sum_{id=1}^{M}\sum_{i=1}^{c_{id}} p_i^{(id)}\right| \le \sum_{id=1}^{M}\left|\sum_{k=1}^{K} q_k^{(id)} - \sum_{i=1}^{c_{id}} p_i^{(id)}\right|,$$

and using (6),

$$\left|\sum_{j=1}^{MK} q_j - s\right| \le \sum_{id=1}^{M}\left(\gamma_{c_{id}-1}^{K}\sum_{i=1}^{c_{id}}\left|p_i^{(id)}\right|\right) \le \gamma_{c-1}^{K} S. \tag{28}$$

Furthermore, applying (7) to the right-hand side of (24) and using (22) yield

$$\begin{aligned}\sum_{j=1}^{MK}|q_j| &= \sum_{id=1}^{M}\left(\sum_{j=1}^{K}\left|q_j^{(id)}\right|\right) \le \sum_{id=1}^{M}\left((1+\gamma_{2(c_{id}-1)})\sum_{i=1}^{c_{id}}\left|p_i^{(id)}\right|\right)\\ &\le (1+\gamma_{2(c-1)})S. \end{aligned}\tag{29}$$

Inserting (28) and (29) into (27), we finally have

$$|\mathbf{res} - s| \le \mathbf{u}'|s| + \left\{(1+\mathbf{u}')\gamma_{c-1}^{K} + (1+\gamma_{2(c-1)})\gamma_{2(MK-1)}^{K}\right\} S,$$

which proves (19) and divided by $|s|$ also proves (20). □

Similarly to the error bound for **SumK** in Theorem 2.6, Theorem 3.2 says

$$\frac{|\mathbf{res} - s|}{|s|} \lesssim \mathbf{u} + \mathcal{O}(\mathbf{u}^K)\cdot \mathrm{cond}\left(\sum p_i\right),$$

which means the result **res** by **PSumK** is also computed in internally K-fold working precision.

3.2 Parallelizing Method of **DotK** (**PDotK**)

Next, we present an outline of **PDotK**, which is a parallelized version of **DotK**. Let $x, y \in \mathbb{F}^n$.

Procedure 1: Distribute sub-vectors of both x and y which are divided into M pieces to all CPUs. Every CPU with $2 \le id \le M$ has $c := \lceil n/M \rceil$ components corresponding to x and y, respectively. The CPU with $id = 1$ has $n - c\,(M-1)$ components as well. Let $x^{(id)}$ and $y^{(id)}$ denote the sub-vector on the id-th CPU. Let c_{id} denote the number of components of $p^{(id)}$, then it holds that

$$c_1 \le c_2 = c_3 = \cdots = c_M = c.$$

Procedure 2: Use $t^{(id)} = \mathbf{VecProduct}(x^{(id)}, y^{(id)})$ on all CPUs, i.e. transform dot product $(x^{(id)})^T y^{(id)}$ into summation $\sum_{j=1}^{2c_{id}} t_j^{(id)}$.
Procedure 3: Use $\mathbf{SumL}(t^{(id)},\ K-1)$, whose output is denoted by $q^{(id)}$. Then $q^{(id)}$ with K components is obtained on every CPU.
Procedure 4: Gather $q^{(id)}$ for all id into a vector q on the CPU with $id = 1$. Then the length of the gathered vector q becomes MK.
Procedure 5: Use $\mathbf{SumK}(q,\ K)$ on CPU with $id = 1$.

Now, we present here the concrete algorithm of **PDotK**.

Algorithm 3.3 (PDotK) *A parallelized version of* **DotK**.

$$
\begin{aligned}
&\textbf{function res} = \mathbf{PDotK}\,(x,\ y,\ K,\ M)\\
&\quad c = \lceil \tfrac{n}{M} \rceil, \quad c_1 = n - c\,(M-1)\\
&\quad \%\ \textit{parallel private}\,(\mathrm{idx1},\ \mathrm{idx2},\ \mathrm{idx3},\ \mathrm{st1},\ \mathrm{st2},\ c_{id})\\
&\quad\quad \textit{if} \quad id == 1\\
&\quad\quad\quad \mathrm{st1} = 1, \quad \mathrm{st2} = 1\\
&\quad\quad\quad c_{id} = c_1\\
&\quad\quad \textit{else}\\
&\quad\quad\quad \mathrm{st1} = c_1 + c\,(id-2) + 1, \quad \mathrm{st2} = 2c_1 + 2c\,(id-2) + 1\\
&\quad\quad\quad c_{id} = c\\
&\quad\quad \textit{end}\\
&\quad\quad \mathrm{idx1} = \mathrm{st1} : \mathrm{st1} + c_{id} - 1\\
&\quad\quad \mathrm{idx2} = \mathrm{st2} : \mathrm{st2} + 2c_{id} - 1, \quad \mathrm{idx2s} = \mathrm{st2} : \mathrm{st2} + 2c_{id} - 2\\
&\quad\quad \mathrm{idx3} = K\,(id-1) + 1 : K \cdot id - 1\\
&\quad\quad t\,[\mathrm{idx2}] = \mathbf{VecProduct}\,(x\,[\mathrm{idx1}]\,,\ y\,[\mathrm{idx1}])\\
&\quad\quad q\,[\mathrm{idx3}] = \mathbf{SumL}\,(t\,[\mathrm{idx2s}]\,,\ K-1)\\
&\quad\quad q(K \cdot id) = t(\mathrm{st2} + 2c_{id} - 1)\\
&\quad \%\ \textit{end parallel}\\
&\quad \textbf{res} = \mathbf{SumK}\,(q,\ K)
\end{aligned}
$$

Then we present the following theorem for **PDotK**.

Theorem 3.4 *Let* **res** *be the result obtained by Algorithm 3.3* (**PDotK**). *Sup-*

pose $\max\{2n, 2(MK-1)\}\mathbf{u} < 1$. *Then the following inequalities hold:*

$$\left|\mathbf{res} - x^T y\right| \leq (\mathbf{u} + 3\gamma_{MK-1}^2)|x^T y| + P_2\left(\gamma^K\right)|x^T||y| \tag{30}$$

and

$$\frac{\left|\mathbf{res} - x^T y\right|}{|x^T y|} \leq \mathbf{u} + 3\gamma_{MK-1}^2 + \frac{1}{2}P_2\left(\gamma^K\right)\mathrm{cond}\left(x^T y\right), \tag{31}$$

where

$$\begin{aligned} P_2\left(\gamma^K\right) &:= (1 + \mathbf{u} + 3\gamma_{MK-1}^2)\gamma_{2c-1}^K + (1 + 3\gamma_c)\gamma_{2(MK-1)}^K \\ &= \mathcal{O}\left(\gamma_{\max\{2c-1,2(MK-1)\}}^K\right). \end{aligned}$$

Proof. From the definition of vectors $x^{(id)}$, $y^{(id)}$, $t^{(id)}$, q and $q^{(id)}$, we collect the following equalities:

$$q_K^{(id)} = t_{2c_{id}}^{(id)} \tag{32}$$

$$\sum_{j=1}^{2c_{id}} t_j^{(id)} = (x^{(id)})^T y^{(id)} \tag{33}$$

$$\sum_{i=1}^{MK} q_i = \sum_{id=1}^{M}\sum_{j=1}^{K} q_j^{(id)} \tag{34}$$

$$\sum_{i=1}^{MK} |q_i| = \sum_{id=1}^{M}\sum_{j=1}^{K} \left|q_j^{(id)}\right|. \tag{35}$$

Then

$$\begin{aligned} \left|\mathbf{res} - x^T y\right| &= \left|\mathbf{res} - \sum_{j=1}^{MK} q_j + \sum_{j=1}^{MK} q_i - x^T y\right| \\ &\leq |r_1| + |r_2|, \end{aligned} \tag{36}$$

where

$$r_1 := \mathbf{res} - \sum_{j=1}^{MK} q_j \quad \text{and} \quad r_2 := \sum_{j=1}^{MK} q_j - x^T y.$$

Here, using $\mathbf{res} = \mathbf{SumK}(q,\ K)$ and (4) yields

$$\begin{aligned} |r_1| &\leq \mathbf{u}'\left|\sum_{i=1}^{MK} q_i\right| + \gamma_{2(MK-1)}^K \sum_{i=1}^{MK} |q_i| \\ &\leq \mathbf{u}'\left(|r_2| + \left|x^T y\right|\right) + \gamma_{2(MK-1)}^K \sum_{j=1}^{MK} |q_j|, \end{aligned} \tag{37}$$

where $\mathbf{u}' := \mathbf{u} + 3\gamma_{MK-1}^2$. Inserting (37) into (36), we have

$$\left|\mathbf{res} - x^T y\right| \leq \mathbf{u}'\left|x^T y\right| + (1 + \mathbf{u}')|r_2| + \gamma_{2(MK-1)}^K \sum_{j=1}^{MK} |q_j|. \tag{38}$$

By (32), (33) and (34), it holds that

$$\begin{aligned}
|r_2| &= \left| \sum_{id=1}^{M} \sum_{k=1}^{K} q_k^{(id)} - \sum_{id=1}^{M} (x^{(id)})^T y^{(id)} \right| \\
&\leq \sum_{id=1}^{M} \left| \sum_{k=1}^{K} q_k^{(id)} - (x^{(id)})^T y^{(id)} \right| = \sum_{id=1}^{M} \left| \sum_{k=1}^{K} q_k^{(id)} - \sum_{j=1}^{2c_{id}} t_j^{(id)} \right| \\
&= \sum_{id=1}^{M} \left| \left(q_K^{(id)} - t_{2c_{id}}^{(id)} \right) + \sum_{k=1}^{K-1} q_k^{(id)} - \sum_{j=1}^{2c_{id}-1} t_j^{(id)} \right| \\
&= \sum_{id=1}^{M} \left| \sum_{k=1}^{K-1} q_k^{(id)} - \sum_{j=1}^{2c_{id}-1} t_j^{(id)} \right|,
\end{aligned}$$

and using (6) and (8) yields

$$\begin{aligned}
|r_2| &\leq \sum_{id=1}^{M} \left(\gamma_{2c_{id}-2}^{K-1} \sum_{j=1}^{2c_{id}-1} \left| t_j^{(id)} \right| \right) \leq \sum_{id=1}^{M} \gamma_{2c_{id}-2}^{K-1} \gamma_{c_{id}} |(x^{(id)})^T| |y^{(id)}| \\
&\leq \gamma_{2c-1}^{K} |x^T| |y|. \qquad (39)
\end{aligned}$$

Furthermore, it follows by (32) and (35) that

$$\begin{aligned}
\sum_{j=1}^{MK} |q_j| &= \sum_{id=1}^{M} \left(\sum_{k=1}^{K} \left| q_k^{(id)} \right| \right) = \sum_{id=1}^{M} \left(\left| q_K^{(id)} \right| + \sum_{k=1}^{K-1} \left| q_k^{(id)} \right| \right) \\
&= \sum_{id=1}^{M} \left(\left| t_{2c_{id}}^{(id)} \right| + \sum_{k=1}^{K-1} \left| q_k^{(id)} \right| \right).
\end{aligned}$$

Using (7), (8) and (9) yields

$$\begin{aligned}
\sum_{j=1}^{MK} |q_j| &\leq \sum_{id=1}^{M} \left((1+\gamma_{c_{id}}) |(x^{(id)})^T| |y^{(id)}| + (1+\gamma_{4(c_{id}-1)}) \sum_{i=1}^{2c_{id}-1} \left| t_i^{(id)} \right| \right) \\
&\leq \sum_{id=1}^{M} \{1 + \gamma_{c_{id}} + \gamma_{c_{id}} (1+\gamma_{4(c_{id}-1)})\} |(x^{(id)})^T| |y^{(id)}| \\
&\leq (1+3\gamma_c) |x^T| |y|. \qquad (40)
\end{aligned}$$

Here, $\gamma_c + \gamma_c(1+\gamma_{4c}) \leq 3\gamma_c$ is proved as follows: Since $2n\mathbf{u} < 1$ and $M \geq 2$, $4c\mathbf{u} < 1$ and $\gamma_{4c} < \frac{1}{3}$, so that $\gamma_c + \gamma_c(1+\gamma_{4c}) \leq \frac{7}{3}\gamma_c < 3\gamma_c$.

Inserting (39) and (40) into (38), we finally have

$$\left| \mathbf{res} - x^T y \right| \leq \mathbf{u}' |x^T y| + \left\{ (1+\mathbf{u}') \gamma_{2c-1}^{K} + (1+3\gamma_c) \gamma_{2(MK-1)}^{K} \right\} |x^T| |y|,$$

which proves (30), and divided by $|x^T y|$ also proves (31). □

Again similarly to the error bound for **DotK** in Theorem 2.12, Theorem 3.4 says

$$\frac{|\mathbf{res} - x^T y|}{|x^T y|} \lesssim \mathbf{u} + \mathcal{O}(\mathbf{u}^K) \cdot \mathrm{cond}\left(x^T y\right),$$

which means the result **res** by **PDotK** is also computed in internally K-fold working precision.

If $c \ll n$ and $MK \ll n$, then the error bounds (19) and (31) become less than (4) and (18), respectively. Moreover, note that **PSumK** and **PDotK** for $K = 2$ can be specialized according to **Sum2** and **Dot2** in [7], respectively. Then the error bounds (19) and (31) slightly change.

Thus, **SumK** and **DotK** are almost ideally parallelized as **PSumK** and **PDotK**, respectively. In the next section, we will confirm the performance of **PDotK** by numerical experiments.

4 Numerical Results

In this section, we present some results of numerical experiments showing the performance of **PDotK** on our shared memory system. We use a PC with 2 processors of Intel Dual-Core Xeon 2.80GHz with 1024KB cache and Intel C++ Compiler 9.0. Therefore, there are 4 CPUs in the shared memory system.

All floating-point operations are done in IEEE standard 754 double precision. To avoid overdoing the compiler optimizations for **TwoSum** and **TwoProduct**, a compiler option `-mp` has to be used, which ensures that floating-point arithmetic conforms more closely to the IEEE standard. Parallelization is done by OpenMP supported in the Intel Compiler.

To focus our mind on the parallel efficiency of **PDotK**, we compare the elapsed time for **PDotK** only with that for **DotK**. More comparisons with other dot product algorithms can be found in [7]. We specialize **PDotK** for $K = 2$ according to **Dot2** in [7], so that the intermediate array t in Algorithm 3.3 is not necessary in case of $K = 2$.

Before testing **PDotK**, we evaluate the parallelization overhead by OpenMP in use. To do this, we measure the parallel efficiency of a usual recursive summation algorithm:

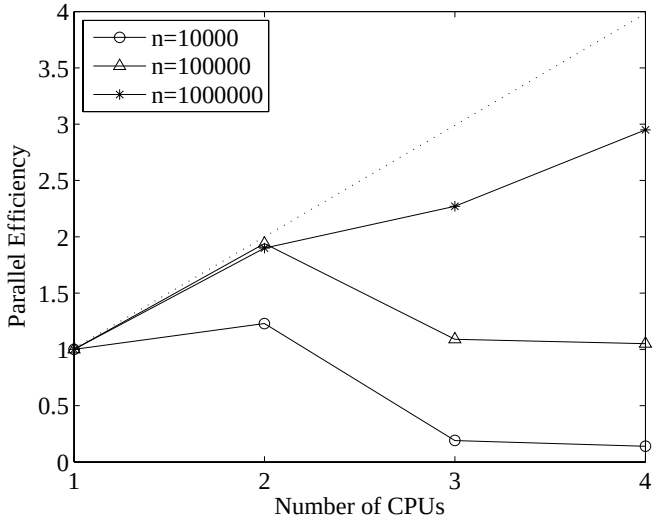

Fig. 6. Parallel efficiency of recursive summation.

$s = 0$

$\%$ *parallel for*

for $i = 1 : n$

$\quad s = fl\,(s + p_i)$

end

for calculating $\sum_{i=1}^{n} p_i$, $p_i \in \mathbb{F}$ with $n = 10^4$, 10^5 and 10^6. Here, the parallel efficiency is defined by T_1/T_{np}, where T_{np} means the elapsed time in case of using np CPUs ($1 \leq np \leq 4$). We display the result in Figure 6.

From Figure 6, it can be seen that increasing the number of processors in use need not to improve the parallel efficiency. The parallelization works well only for the largest case ($n = 10^6$). The reason is that the parallelization causes the overhead for initialization, reduction handling and so forth (cf. for example, see [3] for more details). For smaller n, the cost for the parallelization overhead becomes relatively large compared with that for pure floating-point operations. Therefore, if the floating-point operations are perfectly parallelized, then the parallel efficiency R is determined by

$$R = \frac{T_1}{T_{np}} = \frac{T_1}{C + T_1/np},$$

where C denotes the elapsed time for the parallelization overhead.

To generate extremely ill-conditioned data for testing dot product as examples, we develop the following Matlab code.

Algorithm 4.1 *Generator of extremely ill-conditioned vectors x, y for testing dot product $x^T y$.*

```
function [x,y] = gendot2(n,cnd)
% Generate extremely ill-conditioned data for dot product.
%  input
%       n: n = length(x)
%     cnd: anticipated condition number of dot product
%  output
%    x, y: generated vectors with length n
%  Note that the exact result of the dot product is cnd^-1.

  m = floor(n/2);
  Eps = 2^-24;
  L = floor(log(cnd)/-log(Eps));
  if mod(n,2) == 0
      r = mod(1:m-2,L);
      c = randn(1,m-2).*Eps.^r;
      x = [1 c 0.5*cnd^-1 -1 -c 0.5*cnd^-1]';
      b = randn(1,m-2);
      y = [1 b 1 1 b 1]';
  else
      r = mod(1:m-1,L);
      c = randn(1,m-1).*Eps.^r;
      x = [1 c cnd^-1 -1 -c]';
      b = randn(1,m-1);
      y = [1 b 1 1 b]';
  end
return
```

Using Algorithm 4.1, we generate two n-vectors x and y as follows: Let $\texttt{cnd} := 2^{400} \approx 2.58 \cdot 10^{120}$. Then $\texttt{L} = 16$. We treat the case where n is an odd number. Generate pseudo-random vectors $a, b \in \mathbb{F}^{m-1}$ whose components are uniformly distributed in $[-1, 1]$. Define $r_i := \mathrm{mod}(i, 16)$ for $1 \le i \le m-1$, i.e. $r_i = i - \lfloor i/16 \rfloor \cdot 16$, and $c_i := a_i \cdot \texttt{Eps}^{r_i}$. Then, we obtain

$$x = \left(1, c_1, c_2, \ldots, c_{m-1}, \texttt{cnd}^{-1}, -1, -c_1, -c_2, \ldots, -c_{m-1}\right)^T \in \mathbb{F}^n$$
$$y = (1, b_1, b_2, \ldots, b_{m-1}, 1, 1, b_1, b_2, \ldots, b_{m-1})^T \in \mathbb{F}^n,$$

where $n = 2m + 1$. Clearly, the exact result of dot product $x^T y$ is equal to $\texttt{cnd}^{-1}$.

Figures 7, 8 and 9 illustrate the parallel efficiency of **PDotK** compared with **DotK** for $n = 10001$, 100001 and 1000001, respectively. For each n, we set $K = 2, 4, 8$.

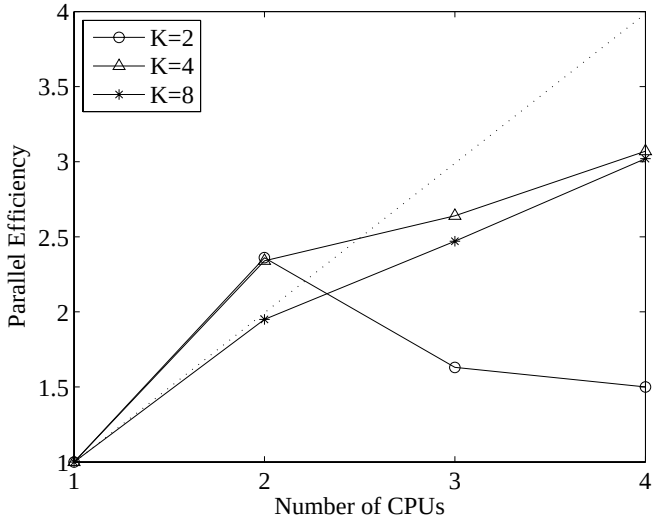

Fig. 7. Parallel efficiency of **PDotK** for $n = 10,001$.

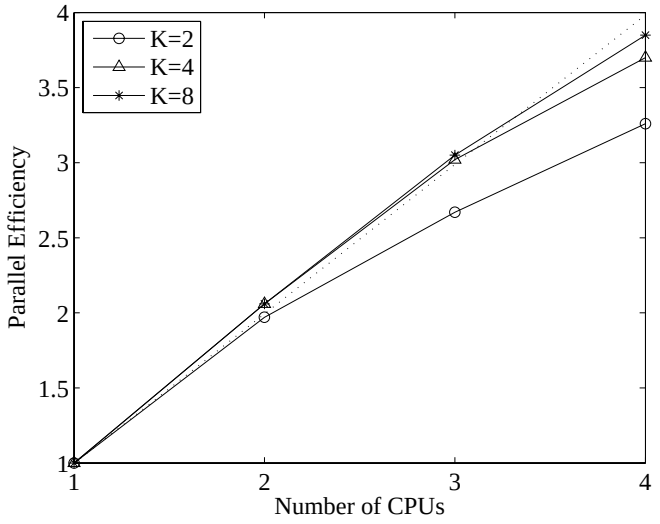

Fig. 8. Parallel efficiency of **PDotK** for $n = 100,001$.

From Figures 7, 8 and 9, we can observe the following things:

- When the number of floating-point operations increases according to an increase of n and/or K, the parallel efficiency is basically improved because the parallelization overhead becomes relatively small.
- In case of $np = 2$, **PDotK** achieves almost ideal parallel efficiency for all n, even superlinear speedups for $n = 10,001$ (Fig. 7) due to the cache effects. For large n, the cache effects seem to disappear.
- In case of $n = 1,000,001$ (Fig. 9), the parallel efficiency for $K = 2$ is much improved. This is due to an effect of the specialized **PDotK** for $K = 2$, especially the memory access time is reduced by getting rid of the intermediate array.

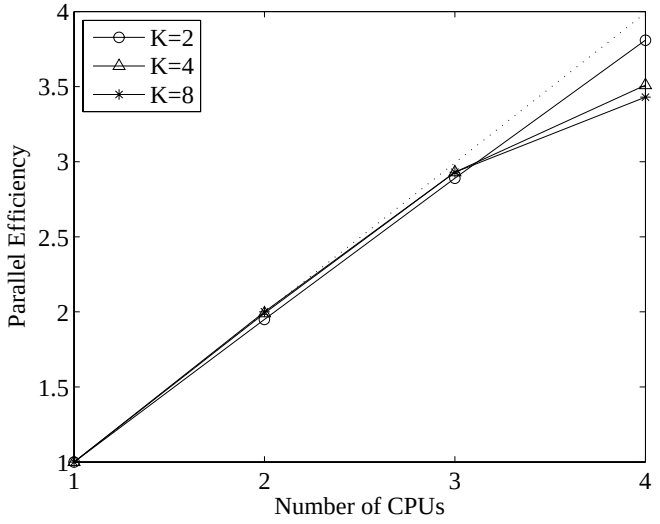

Fig. 9. Parallel efficiency of **PDotK** for $n = 1,000,001$.

It turns out that the proposed algorithm **PDotK** is very effective for $np = 2$, and meaningful for other cases, especially for larger n and K. The results with K-fold precision are ensured by Theorem 3.4.

References

[1] D. E. Knuth: The Art of Computer Programming: Seminumerical Algorithms, vol. 2, Addison-Wesley, Reading, Massachusetts, 1969.

[2] T. J. Dekker: A floating-point technique for extending the available precision, Numer. Math., 18 (1971), 224–242.

[3] M. Gerndt, B. Mohr, J. L. Träff: Evaluating OpenMP performance analysis tools with the APART test suite, Lecture Notes in Computer Science, 3149 (2004), 155–162.

[4] N. J. Higham: Accuracy and Stability of Numerical Algorithms, Second Edition, SIAM, Philadelphia, 2002.

[5] W. Kahan: Doubled-precision IEEE standard 754 floating point arithmetic, manuscript, 1987.

[6] X. Li et al.: Design, implementation and testing of extended and mixed precision BLAS, ACM Trans. Math. Softw., 28 (2002), 152–205.

[7] T. Ogita, S. M. Rump, S. Oishi: Accurate sum and dot product, SIAM J. Sci. Comput., 26:6 (2005), 1955–1988.