A PARALLEL LOOP SCHEDULING ALGORITHM BASED ON THE SMARANDACHE *f*-INFERIOR PART FUNCTION

Tatiana Tabirca* Sabin Tabirca**

*Department of Computer Science, Manchester University

**Department of Computer Science, "Transilvania" University of Brasov

Abstract. This article presents an application of the inferior Smarandache *f*-part function to a particular parallel loop-scheduling problem. The product between an upper diagonal matrix and a vector is analysed from parallel computation point of view. An efficient solution for this problem is given by using the inferior Smarandache *f*-part function. Finally, the efficiency of our solution is proved experimentally by presenting some computational results.

Parallel programming has been intensely developed in order to solve difficult problems that contain either a big number of computation or a large volume of data. These often occur both in real word applications (*e.g.* Weather Prediction) or theoretical problems (*e.g.* Differential Equations). Unfortunately, there is not a standard for writing parallel programs; this depends on the parallel language used or the parallel platform on which the computation is performed. A common fact of this diversity is represented by easiness to parallelise loops. Loops represent an important source of parallelism occurring in at most all the scientific applications. Many algorithms dealing to the scheduling of loop iterations to processors have been proposed so far.

1.Introduction

Consider that there are p processors denoted in the following by P_1, P_2, ..., P_p and a single parallel loop (see Figure 1.).

```
DO PARALLEL I=1,N
      CALL LOOP_BODY(I)
END DO
```

Figure 1. Single Parallel Loop

We also assume that the work of the routine loop_body(i) can be evaluated and is given by the function $w: N \to R$, where $w(i) = w_i$ represents either the number of routine's operations or its running time (presume that $w(0)=0$). The total amount of work for the parallel loop is $\sum_{i=1}^{N} w(i)$. The efficient loop-scheduling algorithm distributes equally this total amount of work on processors such that a processor receives a quantity of work equal to $\frac{1}{p} \cdot \sum_{i=1}^{N} w(i)$.

Let l_j and h_j be the lower and upper loop iteration bounds, $j = 1,2,..., p$, such that processor j executes all the iteration between l_j and h_j. These bounds are found distributing equally the work on processors by using

$$\sum_{i=l_j}^{h_j} w(i) \approx \frac{1}{p} \cdot \sum_{i=1}^{N} w(i) \; (\forall j = 1,2,..., p). \tag{1}$$

Moreover, they satisfy the following conditions

$$l_1 = 1. \tag{2.a}$$

$$\text{if we know } l_j, \text{ then } h_j \text{ is given by } \sum_{i=l_j}^{h_j} w(i) \approx \frac{1}{p} \cdot \sum_{i=1}^{N} w(i) = \overline{W} \; . \tag{2.b}$$

$$l_{j+1} = h_j + 1. \tag{2.c}$$

Suppose that Equation (2.b) is computed by a less approximation. This means that if we have the value l_j, then we find h_j as follows:

$$h_j = h \;\Leftrightarrow\; \sum_{i=l_j}^{h} w(i) \le \overline{W} < \sum_{i=l_j}^{h+1} w(i) \; . \tag{3}$$

In the following, we present an optimal parallel solution for the product between an upper diagonal matrix and a vector. This is an important problem that occurs in many algorithms for solving linear systems. The Smarandache inferior part function is used to distribute equally the work on processors.

2. The Smarandache Inferior Part Function

The inferior part function (sometime is named the floor function) $[,]: R \to Z$, defined by $[x]=k \Leftrightarrow k \le x < k+1$, is one of the most used elementary functions. The Smarandache inferior part function represents a natural generalisation of the floor function [Smara1]. Smarandache proposed and studied this generalisation especially in connection to Number Theory functions [Smara1, Smara2]. In the following, we present equation for some Smarandache inferior part functions.

Consider $f: Z \to R$ a function that is strict increasing and satisfies $\lim_{n\to-\infty} f(n) = -\infty$ and $\lim_{n\to\infty} f(n) = \infty$. The Smarandache *f*-inferior part function denoted by $f_{[]}: R \to Z$ is defined by

$$f_{[]}(x) = k \Leftrightarrow f(k) \le x < f(k+1). \tag{4}$$

The function $f_{[]}$ is well defined because of the good properties of *f*. When $f(k)=k$ the floor function [x] is obtained. In the following we study the Smarandache *f*-inferior part function when $f(k) = \sum_{i=1}^{k} i^a$.

Remark. Sometime, we will study only the positive inferior part by considering function $f: N \to R, f(0)=0$. In this case, we only consider $f_{[]}: [0,\infty) \to Z$.

Theorem 1. *If* $f(k) = \sum_{i=1}^{k} i$, *then the Smarandache f-inferior part is given by*

$$f_{[]}(x) = \left[\frac{-1+\sqrt{1+8\cdot x}}{2}\right] \forall x \ge 0. \tag{5}$$

Proof The proof is obtained by starting from the double inequality $\sum_{i=1}^{k} i \le x < \sum_{i=1}^{k+1} i$.

Observe that the equation $\frac{k\cdot(k+1)}{2} = x > 0$ has only one positive root given by $k = \frac{-1+\sqrt{1+8\cdot x}}{2} > 0$. The following equivalences prove Theorem 1

$$\sum_{i=1}^{k} i \le x < \sum_{i=1}^{k+1} i \Leftrightarrow \frac{k\cdot(k+1)}{2} \le x < \frac{(k+1)\cdot(k+2)}{2} \Leftrightarrow$$

$$\Leftrightarrow k \le \frac{-1+\sqrt{1+8\cdot x}}{2} < k+1 \Leftrightarrow k = \left[\frac{-1+\sqrt{1+8\cdot x}}{2}\right].$$

Thus, the equation for the Smarandache *f*-inferior part is $f_{\sqcup}(x)=\left[\frac{-1+\sqrt{1+8\cdot x}}{2}\right]$.

♦

Theorem 2. *If* $f(k)=\sum_{i=1}^{k} i^2$, *then the Smarandache f-inferior part is given by*

$$f_{\sqcup}(x)=\left[-\frac{1}{2}+\sqrt[3]{\frac{3\cdot x}{2}-\sqrt{\left(\frac{3\cdot x}{2}\right)^2+\frac{1}{1728}}}+\sqrt[3]{\frac{3\cdot x}{2}+\sqrt{\left(\frac{3\cdot x}{2}\right)^2+\frac{1}{1728}}}\right]\forall x\geq 0. \quad (6)$$

Proof We use the Cardano equation for solving $x^3+px+q=0$. A real root of this equation is given by

$$x=\sqrt[3]{-\frac{q}{2}-\sqrt{\left(\frac{q}{2}\right)^2+\left(\frac{p}{3}\right)^3}}+\sqrt[3]{-\frac{q}{2}+\sqrt{\left(\frac{q}{2}\right)^2+\left(\frac{p}{3}\right)^3}}. \quad (7)$$

The equation $\frac{k\cdot(k+1)\cdot(2\cdot k+1)}{6}=x>0$ is transformed as follows:

$$\frac{k\cdot(k+1)\cdot(2\cdot k+1)}{6}=x \Leftrightarrow 2\cdot k^3+3\cdot k^2+k-6x=0 \Leftrightarrow$$

$$\Leftrightarrow \text{(apply the transformation } k=y-\frac{1}{2}) \Leftrightarrow y^3-\frac{1}{4}\cdot y-3\cdot x=0.$$

Applying Equation (7), we find that

$$y=\sqrt[3]{\frac{3\cdot x}{2}-\sqrt{\left(\frac{3\cdot x}{2}\right)^2+\frac{1}{1728}}}+\sqrt[3]{\frac{3\cdot x}{2}+\sqrt{\left(\frac{3\cdot x}{2}\right)^2+\frac{1}{1728}}} \text{ and}$$

$$k=-\frac{1}{2}+\sqrt[3]{\frac{3\cdot x}{2}-\sqrt{\left(\frac{3\cdot x}{2}\right)^2+\frac{1}{1728}}}+\sqrt[3]{\frac{3\cdot x}{2}+\sqrt{\left(\frac{3\cdot x}{2}\right)^2+\frac{1}{1728}}}.$$

The Smarandache *f*-inferior part is given by:

$$\sum_{i=1}^{k} i^2\leq x<\sum_{i=1}^{k+1} i^2 \Leftrightarrow \frac{k\cdot(k+1)\cdot(2\cdot k+1)}{6}\leq x<\frac{(k+1)\cdot(k+2)\cdot(2\cdot k+3)}{6} \Leftrightarrow$$

$$\Leftrightarrow k\leq -\frac{1}{2}+\sqrt[3]{\frac{3\cdot x}{2}-\sqrt{\left(\frac{3\cdot x}{2}\right)^2+\frac{1}{1728}}}+\sqrt[3]{\frac{3\cdot x}{2}+\sqrt{\left(\frac{3\cdot x}{2}\right)^2+\frac{1}{1728}}}<k+1\Leftrightarrow$$

$$\Leftrightarrow k=\left[-\frac{1}{2}+\sqrt[3]{\frac{3\cdot x}{2}-\sqrt{\left(\frac{3\cdot x}{2}\right)^2+\frac{1}{1728}}}+\sqrt[3]{\frac{3\cdot x}{2}+\sqrt{\left(\frac{3\cdot x}{2}\right)^2+\frac{1}{1728}}}\right]. \quad ♦$$

3. An Efficient Algorithm for the Upper Diagonal Matrix-Vector Product

In this section, we present an efficient algorithm for the product $y = a \cdot x$ between an upper diagonal matrix $a = (a_{i,j})_{i,j=\overline{1,n}} \in M_n(R)$ and a vector $x \in R^n$. This problem is quite important occurring in several other important problems such us solving linear systems or LUP matrix decomposition.

Because a is an upper diagonal matrix, the product $y = a \cdot x$ is given by

$$y_i = \sum_{j=1}^{i} a_{i,j} \cdot x_j \ \forall i = 1,2,...n. \tag{8}$$

The product can be computed in parallel by using a simple computation shown below.

DO PARALLEL i=1,n
    $y_i = 0$
    DO j=1,i
        $y_i = y_i + a_{i,j} \cdot x_j$
    END DO
END DO

Figure 2. Parallel Computation for the Upper Matrix – Vector Product.

For this parallel loop we have the following elements:

- The work of iteration i is $w(i) = i, i = 1,2,...,n$; the total work is $\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}$.
- The quantity of work received by a processor should be approximately equal to $\overline{W} = \frac{n \cdot (n+1)}{2 \cdot p}$

The difficult problem for the efficient loop scheduling algorithm is how Equation (1) is implemented. To find the upper bounds from this is quite expensive and can be done in $O(\log n + \frac{n}{p})$ [Jaja]. But, we want to find the upper bounds in at most $O(p)$ complexity and we show that this is possible for our problem. For that we use the following theorem

Theorem 3. *The parallel computation for the upper matrix-vector product can efficiently be scheduled on processors (with respect of Equation 1) by using the following upper bounds:*

$$h_j = \left\lfloor \frac{-1+\sqrt{1+4\cdot j\cdot \frac{n\cdot(n+1)}{p}}}{2} \right\rfloor, j=1,2,...,p. \quad (9)$$

Proof The Smarandache *f*-inferior part function presented in Theorem 1 is used to obtain the proof. We found that if $f(k)=\sum_{i=1}^{k} i$ then $f_{\lfloor\rfloor}(x)=\left\lfloor \frac{-1+\sqrt{1+8\cdot x}}{2} \right\rfloor \forall x \geq 0$.

Since each processor receives a quantity equal to $\overline{W}=\frac{n\cdot(n+1)}{2\cdot p}$, we find that the first j-1 processors have received approximately $(j-1)\cdot\overline{W}$. Thus, the upper bound of processor j is the biggest number k such that all the previous work done by processors 1,2,...j should be approximately equal to $j\cdot\overline{W}$. Mathematically, this can be written as follows

$$1+2+...+h_j \leq j\cdot\overline{W} < 1+2+...+h_j+(h_j+1) \Leftrightarrow$$

$$\Leftrightarrow h_j = f_{\lfloor\rfloor}(j\cdot\overline{W}) = \left\lfloor \frac{-1+\sqrt{1+8\cdot j\cdot\overline{W}}}{2} \right\rfloor \Leftrightarrow$$

$$\Leftrightarrow h_j = \left\lfloor \frac{-1+\sqrt{1+4\cdot j\cdot \frac{n\cdot(n+1)}{p}}}{2} \right\rfloor, j=1,2,...,p$$

A more rigorous and technical explanation can be found in [Tabi]. ♦

According to this theorem, the efficient scheduling is obtained using the upper bound from Equation (9). These bounds certainly give the better approximation of Equation 1. Thus, the part of parallel loop scheduled on processor j is presented in Figure 3. This processor computes all the sums of Equation (8) between $h_{j-1}+1$ and h_j.

$$\textbf{DO } i = \left[\frac{-1+\sqrt{1+4\cdot(j-1)\cdot\frac{n\cdot(n+1)}{p}}}{2}\right]+1, \left[\frac{-1+\sqrt{1+4\cdot j\cdot\frac{n\cdot(n+1)}{p}}}{2}\right]$$

$y_i = 0$

DO j=1,i

$y_i = y_i + a_{i,j}\cdot x_j$

END DO

END DO

Figure 3. Computation of Processor j.

4. Computational Results and Final Conclusions

This section presents some computational results of scheduling the parallel loop from Figure 3. In order to find that the proposed method is efficient from the practical point of view, two other scheduling algorithms are used. The first scheduling algorithm named *uniform scheduling,* divides the parallel loop into p chunks with the same size $\left[\frac{n}{p}\right]$. Obviously, this represents the simplest scheduling strategy but is inefficient because all the big sums are computed on processor p. The second scheduling algorithm named *interleaving*, distributes the work on processors from p to p, such that a processor does not compute two consecutive works. This scheduling distributes the large work equally on processors. All the algorithms have been executed on SGI Power Challenge 2000 parallel machine with 16 processors for a upper diagonal matrix of dimension 300. The running time are presented in Table 1.

	P=1	*P=2*	*P=3*	*P=6*	*P=8*
Balanced	1.878	1.377	0.974	0.760	0.472
Interleaving	2.029	1.447	1.041	0.803	0.576
Uniform	2.028	2.122	1.660	1.335	0.970

Table 1. Computational Times for three Scheduling Algorithms.

The first important remark that can be outlined is that there is no way to develop efficient methods in Computer Science without Mathematics and this article is a prove for that. Using a special function named the Smarandache inferior part, it has

been possible to find an efficient scheduling algorithm for the upper diagonal matrix-vector product.

The second important remark is that the scheduling proposed in this article is efficient in practice as well. Table 1 shows that the times for the line **balanced** are smallest. It can be seen that the interleaving strategy also offers good times. Table 1 also shows that the uniform strategy gives the largest times.

References

[Jaja] J.Jaja, (1992) Introduction to Parallel Algorithms, Adison-Wesley.

[Smara1] F. Smarandache, A Generalisation of The Inferior Part Function, http://www.gallup.unm.edu/~smarandache.

[Smara2] F.Smarandache, (1993) Only Problems ... Not Solutions, Xiquan Publishing House, Phoenix-Chicago.

[Tabi]T. Tabirca and S. Tabirca, (2000) Balanced Work Loop-Scheduling, Technical Report, Department of Computer Science, Manchester University. [*In Preparation*]