Information Processing Letters 32 (1989) 89–93
North-Holland

24 July 1989

USING FEWER PROCESSORS TO REDUCE TIME COMPLEXITIES OF SEMIGROUP COMPUTATIONS

Y.C. CHEN and Z.C. YEH

Institute of Information Engineering, Tatung Institute of Technology, Taipei, Taiwan, Rep. China

G.H. CHEN

Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, Rep. China

Communicated by K. Ikeda
Received 6 January 1988

For parallel algorithms, their execution time consists of two parts: the computation time and the communication time. The execution time reaches a minimum when the other two are balanced. In this paper, we propose a strategy to reduce the execution time when it is dominated by the *communication time*. The strategy is to use fewer processors to decrease the communication time and therefore reduce the execution time. Based on this strategy, we have successfully reduced time complexities of semigroup computations. To be more precise, any parallel algorithm performing semigroup computations of N data items can be improved if it uses N processors (each holds one data item) and has time complexity $O(N^q \log^r N)$, $q \geqslant 0$ and $r \geqslant 0$ (both are not zero). The improved algorithm is designed on the same parallel architecture as the original one, but using only $N^{1/(q+1)}/\log^r N$ processors; the time complexity is $O(N^{q/(q+1)} \log^r N)$, which achieves the optimal speedup. Some previous algorithms can be improved using this strategy, although they were declared to be optimal. Also, our result generalizes both Carlson's and Miller and Stout's results, which are all restricted to 2-dimensional mesh-connected computers.

Keywords: Parallel processing, computational complexity

1. Introduction

A semigroup computation can be described by a tuple $(*, S)$, where $*$ is an associative operator and $S = \{a_1, a_2, \ldots, a_N\}$ is a set of data items. The problem is to compute $a_1 * a_2 * \cdots * a_N$. Some well-known semigroup computation problems are to compute the sum, product, maximum, and minimum of N data items. Recently, drastic advances in hardware technology have made it possible to design various parallel architectures. As a result, many parallel algorithms have been proposed to perform semigroup computations of N data items [1,4,5,7,8] (see Table 1). These algorithms are designed on some specific parallel architectures and need N processors, each holding one data item. Moreover, they were declared to be optimal with respect to the number of processors used on the designated parallel architectures.

For parallel algorithms, their execution time consists of two parts: the computation time and the communication time. Often, the communication time dominates the execution time. To decrease the communication time, one possible way is to reduce the machine size, that is, to use fewer processors. However, the computation time will be increased at the same time. Consequently, if the communication time and the computation time are not balanced, we can then adjust the number of processors used to reduce the execution time.

Many researchers have shown that by using fewer processors the execution time does not increase [2,11] or even does decrease [3,6]. In this paper, we show that by using fewer processors the

Table 1
Previous results about semigroup computations

Parallel Architectures	Number of Processors Used	Execution Time Required
Ring Structure	N	$O(N)$
Broadcasting Protocol Multiprocessor (BPM)	N	average $O(\log N)$ worst $O(N)$
K-dimensional Array Processor	N	$O(N^{1/K})$
K-dimensional Array Processor With A Global Bus	N	$O(N^{1/(K+1)})$
K-dimensional Array Processor With Multiple Broadcasting	N	$O(N^{1/K(K+1)})$
Tree Machine	N	$O(\log N)$
Hypercube Machine	N	$O(\log N)$

time complexities of semigroup computations can be reduced. Our result generalizes both Carlson's and Miller and Stout's results [3,6], which are all restricted to 2-dimensional mesh-connected computers. Based on our result, some previous algorithms [1,7,8] can be improved accordingly. The improved algorithms also achieve the optimal speedup. The remainder of this paper is organized as follows. In Section 2, we describe a strategy to reduce the execution time of semigroup computations. Then, in Section 3, the relationship between the execution time and the number of processors used are discussed. In Section 4, we give some improved parallel algorithms for semigroup computations. Finally, in Section 5, concluding remarks are given.

2. A strategy to reduce time complexities of semigroup computations

There exist several parallel algorithms [1,7,8] for performing semigroup computations. In these algorithms, the communication time dominates the execution time. Besides, the communication time decreases quite a little when fewer processors are used (therefore need more memory in every processor). For such algorithms, we have a method to reduce the execution time. This method can be stated formally as follows.

Theorem 1. *Assume that there exists an algorithm performing semigroup computations. If this algorithm uses N processors (every processor holds one data item) and takes $O(N^q)$ time, $q > 0$, to perform a semigroup computation of N data items, then this algorithm can be improved. The improved algorithm uses only $N^{1/(q+1)}$ processors and requires $O(N^{q/(q+1)})$ time, which achieves the optimal speedup.*

Proof. We prove this theorem by proposing a new algorithm. The new algorithm uses p $(p < N)$ processors (and therefore N/p data items are placed into every processor) and consists of the following two steps.

Step 1. Perform a semigroup computation of N/p data items in every processor. This step takes $O(N/p)$ time. After Step 1 is completed, an intermediate result is obtained in every processor.

Step 2. Use the original algorithm to perform a semigroup computation of the p intermediate results. This step takes $O(p^q)$ time.

The time complexity of the new algorithm is $O(\max\{N/p, p^q\})$, which is a function of p. The minimum occurs when $O(N/p) = O(p^q)$. Thus, the new algorithm has the minimum execution time $O(N^{q/(q+1)})$ when $p = N^{1/(q+1)}$ processors are used. □

There exist other parallel algorithms performing semigroup computations. In these algorithms, the communication time still dominates the execution time. But the communication time decreases just a little, while the computation time increases quite a lot, when fewer processors are used. Thus, the execution time can hardly be reduced, although the number of processors can be reduced. However, the optimal speedup can still be achieved. This fact can be stated formally as follows.

Theorem 2. *Assume that there exists an algorithm performing semigroup computations. If this al-*

gorithm uses N processors (every processor holds one data item) and takes $O(\log^r N)$ *time,* $r > 0$, *to perform a semigroup computation of N data items, then this algorithm can be improved. The improved algorithm uses only* $N/\log^r N$ *processors and requires the same time, which achieves the optimal speedup.*

The proof of Theorem 2 is almost the same as that of Theorem 1. So it is omitted. Tang and Lee [11] have a similar result for $r = 1$, but their result is restricted to the shared-memory computer. Combining the two theorems, we have the following corollary.

Corollary. *Assume that there exists an algorithm performing semigroup computations. If this algorithm uses N processors (every processor holds one data item) and takes* $O(N^q \log^r N)$ *time,* $q \geqslant 0$ *and* $r \geqslant 0$ *(both are not zero), to perform a semigroup computation of N data items, then this algorithm can be improved. The improved algorithm uses only* $N^{1/(q+1)}/\log^r N$ *processors and requires* $O(N^{q/(q+1)} \log^r N)$ *time, which achieves the optimal speedup.*

From the above discussion, we conclude that "more processors" is not always favorable for parallel computations.

3. Discussion

From the proof of Theorem 1, it is known that the execution time of the new algorithm depends on the number of processors used (therefore depends on the memory size required in every processor). In this section, we discuss their relationships. For easy discussion, we define the following notations:

P: the number of processors used;
M: the memory size required in every processor;
T: the execution time required.

It is clear that the new algorithm proposed in the proof of Theorem 1 has

$$P = p, \qquad M = N/p, \tag{1, 2}$$

and

$$T = O(\max\{N/p, p^q\}). \tag{3}$$

It is not difficult to derive the following equations

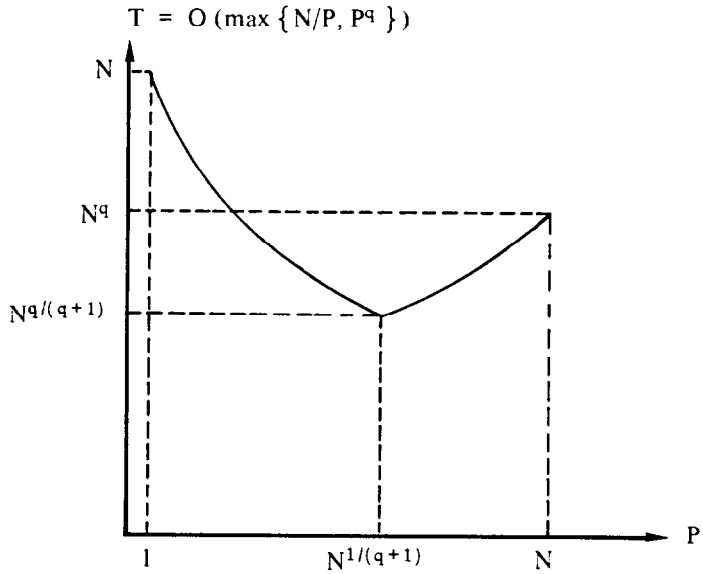

Fig. 1. The relationship between T and P.

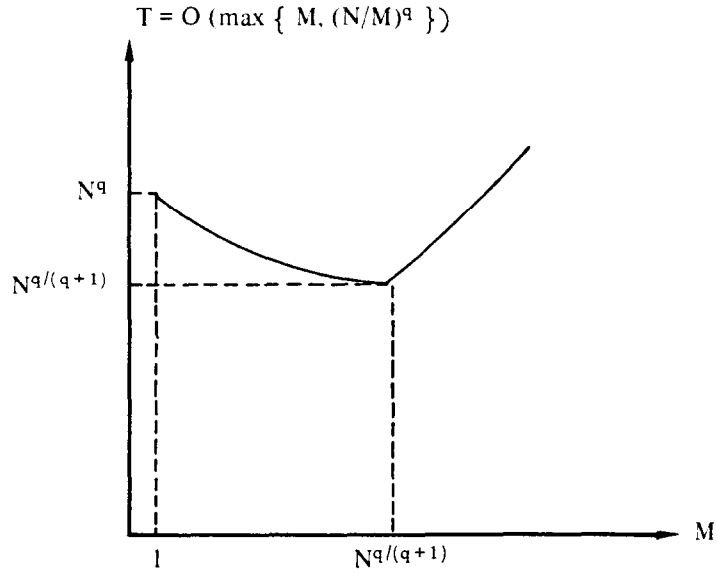

Fig. 2. The relationship between T and M.

for P and T (see Fig. 1):

$$PT = N \quad \text{for } 1 \leqslant P \leqslant N^{1/(q+1)}, \tag{4}$$

$$T = P^q \quad \text{for } N^{1/(q+1)} \leqslant P \leqslant N. \tag{5}$$

When P is smaller than $N^{1/(q+1)}$, the computation time dominates the execution time. If P increases gradually, then more parallelism can be obtained. So, the computation time decreases, which results in less execution time. Since the increased processor can fully share the computation loads, the optimal speedup can be achieved. When P increases to $N^{1/(q+1)}$, the computation time is balanced with the communication time. Thus, the minimum execution time is attained. When P is greater than $N^{1/(q+1)}$, the communication time dominates the execution time. Therefore, increasing P will result in more execution time, as a consequence of increasing communication time.

Moreover, it is clear that M and P have the following relationship:

$$MP = N. \tag{6}$$

Substituting N/M for P, (4) and (5) become

$$T = M \quad \text{for } N^{q/(q+1)} \leqslant M \leqslant N, \tag{7}$$

and

$$M^q T = N^q \quad \text{for } 1 \leqslant M \leqslant N^{q/(q+1)} \tag{8}$$

respectively. Equations (7) and (8) represent the relationship between T and M which is also shown in Fig. 2.

4. Examples

In this section, we consider two parallel algorithms which have been proposed for performing semigroup computations. Both algorithms use N processors, where N is the number of data items. Initially, every processor holds one data item. The first algorithm [8] was designed on a k-dimensional mesh-connected computer (k-MCC) with single broadcasting. It takes $O(N^{1/(k+1)})$ time to complete the execution, which has been proved to be optimal [9]. The second algorithm [7] was also designed on a k-MCC, but with multiple broadcasting. It takes $O(N^{1/k(k+1)})$ time to complete the execution, which was also proved to be optimal [7].

Using the proposed method, we can improve these two algorithms. For the first algorithm, we use only $N^{(k+1)/(k+2)}$ processors, each holding $N^{1/(k+2)}$ data items. First, each processor performs the semigroup operations on their own $N^{1/(k+2)}$ data items. This step takes $O(N^{1/(k+2)})$ time. After this step is completed, an intermediate

result is held in every processor. Then, the final result can be obtained by applying the original algorithm to the $N^{(k+1)/(k+2)}$ intermediate results. This step takes $O(N^{1/(k+2)})$ time. So, the total execution time required for the improved algorithm is

$$O(\max\{N^{1/(k+2)}, N^{1/(k+2)}\}) = O(N^{1/(k+2)}).$$

Following the same way, the second algorithm can be improved to $O(N^{1/(k(k+1)+1)})$ time, while $N^{k(k+1)/(k(k+1)+1)}$ processors are used. Both the two improved algorithms achieve the optimal speedup.

Besides, on a k-MCC without broadcasting, a semigroup computation of N data items can be performed in $O(N^{1/k})$ time [8]. Similarly, this result can be further improved to $O(N^{1/(k+1)})$ time using $N^{k/(k+1)}$ processors. The optimal speedup is also achieved for this example.

5. Concluding remarks

In this paper, we have shown that by using fewer processors the time complexities of semigroup computations can be reduced. At the same time, we extend both Carlson's, and Miller and Stout's results. We also discussed the relationship between the execution time and the number of processors used. It can be seen that several previous algorithms are improved using the proposed method, although they were declared to be optimal. All the improved algorithms achieve the optimal speedup. One possible research problem is to prove that the execution time for the improved algorithms is minimal when N or fewer processors are used. We strongly believe that this is true.

References

[1] S.H. Bokhari, Finding maximum on an array processor with a global bus, *IEEE Trans. Comp.* **33** (1984) 133–139.

[2] D.A. Carlson, Parallel processing of tree-like computations, in: *Proc. 4th Internat. Conf. on Distributed Computing Systems* (1984) 192–200.

[3] D.A. Carlson, Performing tree and prefix computations on modified mesh-connected parallel computers, in: *Proc. Internat. Conf. on Parallel Processing* (1985) 715–718.

[4] S.P. Levitan, Algorithms for a broadcast protocol multiprocessor, in: *Proc. 3rd Internat. Conf. on Distributed Computing Systems* (1982) 666–671.

[5] B. Lint and T. Agewala, Communication issues in the design and analysis of parallel algorithms, *IEEE Trans. Software Eng.* **7** (1981) 174–188.

[6] R. Miller and Q.F. Stout, Varying diameter and problem size on mesh-connected computers, in: *Proc. Internat. Conf. on Parallel Processing* (1985) 697–699.

[7] V.K. Prasanna Kumar and C.S. Raghavendra, Array processor with multiple broadcasting, *J. Parallel Distrib. Comput.* **4** (1987) 173–189.

[8] Q.F. Stout, Broadcasting in mesh-connected computers, in: *Proc. Conf. on Information Sciences and Systems* (1982) 85–90.

[9] Q.F. Stout, Mesh-connected computers with broadcasting, *IEEE Trans. Comput.* **32** (1983) 826–830.

[10] Q.F. Stout, Meshes with multiple buses, in: *Proc. 27th IEEE Symp. on Foundations of Computer Science* (1986) 264–273.

[11] C.Y. Tang and R.C.T. Lee, Optimal speeding up of parallel algorithms based upon the divide-and-conquer strategy, *Inform. Sci.* **32** (1984) 173–186.