12-1990

Balancing a Pipeline by Folding

Per Brinch Hansen
Syracuse University, School of Computer and Information Science, pbh@top.cis.syr.edu

SU-CIS-90-41

Balancing A Pipeline By Folding

Per Brinch Hansen

December 1990

School of Computer and Information Science
Syracuse University
Suite 4-116, Center for Science and Technology
Syracuse, New York 13244-4100

SU-CIS-90-41

BALANCING A PIPELINE BY FOLDING

PER BRINCH HANSEN

December 1990

School of Computer and Information Science
Suite 4-116
Center for Science and Technology
Syracuse, New York 13244-4100

(315) 443-2368

BALANCING A PIPELINE BY FOLDING[1]

PER BRINCH HANSEN

School of Computer and Information Science
Syracuse University
Syracuse, New York 13244

December 1990

A pipeline for Householder reduction is folded several times across an array of processors to achieve approximate load balancing. The performance of the folded pipeline is analyzed and measured on a Computing Surface.

1. INTRODUCTION

Reduction of a matrix to triangular form plays a crucial role in the solution of linear equations. In this paper we analyze a pipeline algorithm for Householder reduction [1]. The pipeline is folded several times across an array of processors to achieve approximate load balancing.

The pipeline inputs, transforms, and outputs a matrix, column by column. During the computation, the columns are distributed evenly among the processors. The computing time per column decreases rapidly from the first to the last column. So the performance of the algorithm is limited mainly by the order in which the columns are distributed among the processors.

The simplest idea is to store a block of columns with consecutive indices in each processor [2]. *Block storage* performs poorly because it assigns the most time-consuming columns to a single processor and leaves much less work for other processors.

It is much better to distribute the columns cyclically among the processors, so that each processor holds a similar mixture of columns. This storage pattern is called *wrapped mapping* [2] or *scattered decomposition* [3].

A third method is *reflection storage* where the columns are distributed one at a time by going back and forth across the processors several times [2].

The *folded pipeline* combines block and reflection storage. On a Computing Surface with 25 transputers the Householder pipeline achieves an efficiency of 81% for a 1250×1250 real matrix.

The performance analysis applies not only to Householder reduction, but also to Gaussian elimination and Givens reduction.

2. PIPELINE NODES

Figure 1 shows a pipeline which transforms an $n \times n$ matrix in $n-1$ steps. Each node of the pipeline holds q columns of the matrix and performs q of the $n-1$ steps. The number of nodes is $(n-1)/q$ assuming that $n-1$ is divisible by q. We are not yet making any assumptions about how the pipeline nodes are distributed among the available processors.

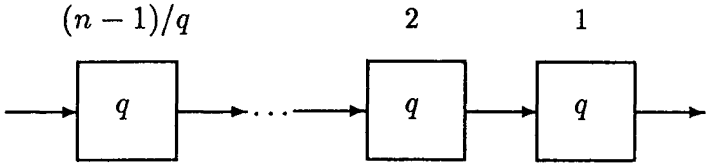

Fig. 1 Pipeline

Initially we will concentrate on the computing time of the parallel algorithm and ignore communication between the nodes. It is convenient to number the steps and nodes in reverse order as follows

$$\text{step numbers} \quad n-1\ldots,2,1$$

$$\text{node numbers} \quad (n-1)/q,\ldots,2,1$$

For Householder reduction, the computing time of the i^{th} step is approximately

$$c(i+1)^2$$

where c is a system-dependent constant.

The computing time $T(k)$ of the k^{th} node is the sum of the computing times of steps $(k-1)q+1$ through kq. For $q \gg 2$, the sum is approximately equal to the integral

$$\int_{(k-1)q}^{kq} cx^2 dx = \frac{1}{3}\, cq^3(3k^2-3k+1)$$

This formula can be rewritten as follows

$$T(k) = aq^3(3k^2-3k+1) \tag{1}$$

where $a = c/3$. The performance analysis is valid for any pipeline algorithm which satisfies Eq. (1).

When a matrix is reduced by a pipeline of 50 nodes the computing times of the first and last nodes differ by a factor of 7350. This enormous variation creates a load-balancing problem when we attempt to distribute the computation evenly among the processors.

3. A SIMPLE PIPELINE

Our goal is to predict the parallel computing time T_p when the pipeline is executed by p processors. We are still ignoring communication.

First we will consider block storage with each node running on a separate processor. For $n \gg 1$, the block length $(n-1)/p$ is approximately n/p.

Due to the computational imbalance, the first processor has more work to do than any other processor. So it determines the parallel computing time. Using Eq. (1) we find for $q \approx n/p$

$$\begin{aligned} T_p &= T(p) \\ &= a(n/p)^3(3p^2 - 3p + 1) \\ &= a(n/p)^3(p^2 + (p-1)(2p-1)) \end{aligned}$$

which can be rewritten as

$$T_p = a(1+f)n^3/p \tag{2}$$

where

$$f = (1 - 1/p)(2 - 1/p) \tag{3}$$

Notice that $0 \leq f < 2$.

If the pipeline runs on a single processor (where $p = 1$ and $f = 0$), the computing time is

$$T_1 = an^3 \tag{4}$$

The speedup

$$S_p = T_1/T_p \tag{5}$$

shows how much faster the computation runs on p processors compared to a single processor.

The efficiency of the parallel computation is

$$E_p = S_p/p \tag{6}$$

For the simple pipeline we use Eqs. (2) and (4) to obtain

$$E_p = 1/(1+f) \tag{7}$$

f is a measure of the load imbalance which reduces the processor efficiency below 100%.

Table I shows how E_p approaches 0.33 for $p \gg 1$. The load imbalance wastes two thirds of the processing capacity!

TABLE I

p	f	E_p
1	0.00	1.00
5	1.44	0.41
10	1.71	0.37
20	1.85	0.35
30	1.90	0.34

4. A FOLDED PIPELINE

To reduce the load imbalance we fold the pipeline an odd number of times m as shown in Fig. 2.

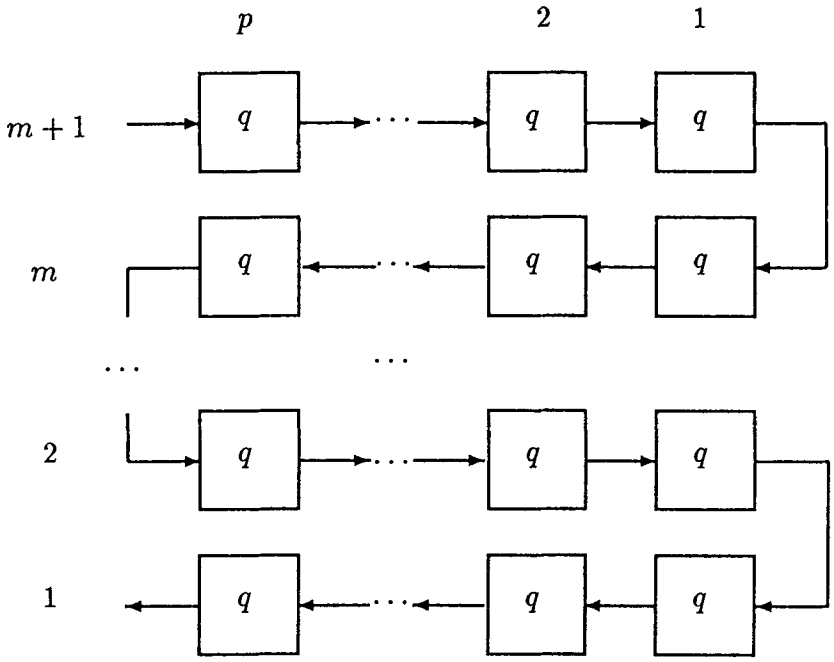

Fig. 2 Folded pipeline

The pipeline now consists of $(m+1)p$ nodes. Every processor executes $m+1$ nodes, each holding q columns where $q = (n-1)/(m+1)p$. For $n \gg 1$, the block length is approximately

$$q \approx \frac{n}{(m+1)p} \tag{8}$$

The idea is to reduce the computing time of the first node by reducing the block length q by a factor of $m+1$.

In the appendix we show that the parallel computing time T_p is

$$T_p = a(1+f)n^3/p \tag{9}$$

where Eq. (3) is replaced by

$$f = (1-1/p)(2-1/p)/(m+1)^2 \tag{10}$$

Notice how folding reduces the load imbalance f.

The processor efficiency is

$$E_p = 1/(1+f) \tag{11}$$

Table II shows f and E_p for various values of m, assuming that $p \gg 1$.

TABLE II

m	f	E_p
0	2.00	0.33
1	0.50	0.67
3	0.13	0.89
5	0.06	0.95
7	0.03	0.97
9	0.02	0.98

5. THE EFFECT OF COMMUNICATION

The remaining task is to consider how communication affects the performance of the folded pipeline.

In the single-processor case, the $n \times n$ matrix passes through $m+1$ pipeline nodes. The sequential run time is the sum of the computing and communication times.

$$T_1 = an^3 + b(m+1)n^2 \tag{12}$$

where a and b are system dependent constants. This replaces Eq. (4).

For a sufficiently large matrix the communication time is negligible compared to the computing time and we have approximately

$$T_1 = an^3 \quad \text{for } n \gg (b/a)(m+1) \tag{13}$$

If we use several processors, each of them must still transmit the matrix through $m+1$ nodes of the pipeline. The parallel run time determined by the first processor is

$$T_p = a(1+f)n^3/p + b(m+1)n^2 \tag{14}$$

This is a refinement of Eq. (9).

The grain size of a parallel computation is the ratio of the computing time to the communication time. In the appendix we show that

$$g = (a/b)(1 + f)q \tag{15}$$

According to Eq. (10), f becomes constant when $p \gg 1$. This makes the grain size proportional to the block length q.

The processor efficiency is

$$E_p = \frac{1}{(1 + f)(1 + 1/g)} \tag{16}$$

(see the appendix).

Since communication decreases the efficiency we would like to make it negligible in the parallel case as well. Equation (16) shows that this can be done only by making the algorithm coarse-grained ($g \gg 1$). This, in turn, means that the blocks must be large.

The efficiency approaches

$$E_p \approx 1/(1 + f) \quad \text{for} \quad g \gg 1 \tag{17}$$

From Eqs. (8) and (15) we conclude that if

$$\frac{n}{(m + 1)p} \gg \frac{b}{a}$$

then $g \gg 1 + f$. Since $f \geq 0$ this implies that $g \gg 1$. In other words, the problem size n must be large compared to the pipe length $(m + 1)p$. This is an example of the necessity of scaling both the problem and the parallel computer to maintain constant efficiency [4].

6. PERFORMANCE MEASUREMENTS

The Householder pipeline was programmed in occam for a Computing Surface with 45 transputers. Each transputer is connected to its two neighbors by four bidirectional channels. The channels make it possible to fold the pipeline three times.

For 64-bit real matrices, measurements show that

$$a = 2.8\mu s \qquad b = 4.2\mu s$$

According to Table II and Eq. (17) it should be possible to obtain a processor efficiency close to 0.89 for $m = 3$, provided $n/p \gg 6$.

The first experiment is Householder reduction of a 1000 × 1000 matrix. Table III shows the values of T_1, T_p, S_p, and E_p predicted by Eqs. (13) and (14). The measured run times are shown in parentheses. As the number of processors increases from 20 to 45, communication reduces the efficiency from 0.81 to 0.72.

TABLE III

p	n	$T_1(s)$	T_p (s)	S_p	E_p
20	1000	2800	173 (171)	16	0.81
25	1000	2800	142 (141)	20	0.79
30	1000	2800	121 (120)	23	0.77
35	1000	2800	106 (105)	26	0.75
40	1000	2800	95 (95)	29	0.74
45	1000	2800	87 (87)	32	0.72

In the second experiment we let $n/p = 50$ to maintain an efficiency of 0.81 which is independent of the number of processors. (With the available memory the computation can be scaled only for $p \leq 25$. See Table IV.)

TABLE IV

p	n	$T_1(s)$	T_p (s)	S_p	E_p
10	500	350	43 (42)	8	0.81
15	750	1181	97 (96)	12	0.81
20	1000	2800	173 (171)	16	0.81
25	1250	5469	271 (268)	20	0.81

7. FINAL REMARKS

We have analyzed a pipeline for Householder reduction. The algorithm illustrates the subtleties of distributing a large computation evenly among parallel processors. Load balancing is achieved by folding the pipeline several times across the array of processors. The predicted efficiency has been confirmed by experiments on a Computing Surface.

ACKNOWLEDGEMENTS

The paper has been improved in presentation by valuable advice from Nawal Copty and Jonathan Greenfield.

REFERENCES

1. Brinch Hansen, P. *The All-Pairs Pipeline.* School of Computer and Information Science, Syracuse University, 1990.

2. Ortega, J. M. *Introduction to Parallel and Vector Solution of Linear Systems.* Plenum Press, NY, 1988.

3. Fox, G., Johnson, M., Lyzenga, G., Otto, S., Salmon, J., and Walker, D. *Solving Problems on Concurrent Processors*, Vol. I, Prentice-Hall, Englewood Cliffs, NJ, 1988.

4. Gustafson, J. L. Reevaluating Amdahl's Law, *Comm. ACM* **31** (1988), 532–533.

APPENDIX

When the Householder pipeline is folded as shown in Fig. 2, the i^{th} processor from the right executes the $m+1$ nodes with indices

$$\begin{array}{l} mp+i \\ mp-i+1 \\ \quad \cdots \\ 3p+i \\ 3p-i+1 \\ p+i \\ p-i+1 \end{array}$$

The processor executes $(m+1)/2$ pairs of nodes. The k^{th} pair has the indices

$$(2k-1)p+i \qquad (2k-1)p-i+1$$

$$\text{for } 1 \le i \le p \quad \text{and} \quad 1 \le k \le (m+1)/2$$

From Eq. (1) we have

$$\begin{aligned} &T((2k-1)p+i) \\ &\quad = aq^3(3((2k-1)p+i)^2 - 3((2k-1)p+i)+1) \\ &\quad = aq^3(3(2k-1)^2p^2 + 3(2k-1)(2i-1)p + 3i^2 - 3i + 1) \end{aligned}$$

and

$$\begin{aligned} & T((2k-1)p-i+1) \\ & \quad = aq^3(3((2k-1)p-i+1)^2 - 3((2k-1)p-i+1)+1) \\ & \quad = aq^3(3(2k-1)^2p^2 - 3(2k-1)(2i-1)p + 3i^2 - 3i + 1) \end{aligned}$$

The combined computing time of the k^{th} pair of nodes is

$$\begin{aligned} T_{\text{pair}}(i,k) &= T((2k-1)p+i) + T((2k-1)p-i+1) \\ &= 2aq^3(3(2k-1)^2p^2 + 3i^2 - 3i + 1) \\ &= aq^3(24p^2k^2 - 24p^2k + 6p^2 + 6i^2 - 6i + 2) \end{aligned}$$

The total computing time of processor i is

$$T_i = \sum_{k=1}^{(m+1)/2} T_{\text{pair}}(i,k)$$

We use the standard formulas

$$\sum_{k=1}^{n} k = n(n+1)/2 \qquad \sum_{k=1}^{n} k^2 = n(n+1/2)(n+1)/3$$

to find the previous sum

$$\begin{aligned} T_i = aq^3(&p^2(m+1)(m+2)(m+3) - 3p^2(m+1)(m+3) \\ &+ (3p^2 + 3i^2 - 3i + 1)(m+1)) \end{aligned}$$

which can be reduced to

$$T_i = aq^3(m+1)(p^2(m^2+2m) + 3i^2 - 3i + 1)$$

T_i is an increasing function of the processor index i. It reaches its maximum value for $i = p$:

$$\begin{aligned} T_p &= aq^3(m+1)(p^2(m^2+2m) + 3p^2 - 3p + 1) \\ &= aq^3(m+1)(p^2(m+1)^2 + 2p^2 - 3p + 1) \\ &= aq^3(m+1)^3(p^2 + (p-1)(2p-1)/(m+1)^2) \\ &= an^3/p(1 + (1-1/p)(2-1/p)/(m+1)^2) \qquad \text{by (8)} \\ &= an^3/p(1+f) \qquad \text{by (10)} \end{aligned}$$

T_p is the computing time of the whole pipeline.

The time grain g is the ratio of the computing time and the communication time

$$\begin{aligned} g &= \frac{a(1+f)n^3/p}{b(m+1)n^2} && \text{by (14)} \\ &= \frac{a(1+f)n}{b(m+1)p} \\ &= (a/b)(1+f)q && \text{by (8)} \end{aligned}$$

The efficiency E_p is derived as follows

$$\begin{aligned} 1/E_p &= pT_p/T_1 && \text{by (5), (6)} \\ &= p(a(1+f)n^3/p + b(m+1)n^2)/(an^3) && \text{by (13), (14)} \\ &= a(1+f)n^3 \left(1 + \frac{b(m+1)p}{a(1+f)n}\right) \Big/ (an^3) \\ &= (1+f)(1+1/g) \end{aligned}$$