

$f_{1/2}$: A parameter to characterize memory and communication bottlenecks

Roger W. HOCKNEY

Computer Science Department, Reading University, Whiteknights, Reading, Berks, United Kingdom RG6 2AX

Ian J. CURINGTON

Floating Point Systems, Apex House, London Road, Bracknell, Berks, United Kingdom, RG12 2TE

Received August 1987

Abstract. When the arithmetic performance of a computer is greater than the memory or communication bandwidth, the overall speed of the computer is strongly dependent on the amount of arithmetic performed, on average, per main memory reference. The performance degradation can be described by a single hardware parameter $f_{1/2}$ which is the number of floating-point operations per memory reference required to achieve half of the peak performance of the computer's arithmetic pipelines. Excellent agreement is shown between this computational model and measured timings on the FPS 5000 computer.

Keywords. $f_{1/2}$, memory bottleneck, communication bottleneck, I/O bottleneck, memory-bandwidth-limited computation, performance characterization, performance parameters.

1. Introduction

There are two principal reasons why high-performance vector computers do not reach their advertised peak performance ($\hat{f}_{\infty}$): either the vectors are not long enough, or insufficient arithmetic is performed per main memory reference. The performance parameters (r_{∞} , $n_{1/2}$) were introduced in 1977 [3–5,9] to characterize the variation of performance with vector length, and later the $n_{1/2}$ -method of algorithm analysis [6,8] was developed from these to compare the performance of vector algorithms. Measurements of these parameters have been reported for a number of computers in [2,4,5,7,9,10]. In this characterization, r_{∞} is the asymptotic performance that is theoretically possible for infinite vector length (hence the subscript), and $n_{1/2}$ is the vector length required to achieve half of the asymptotic performance. If the vector length $n > 9n_{1/2}$, then more than 90% of the asymptotic performance r_{∞} is obtained, and the vector length cannot be said to be constraining the computer performance. However, the asymptotic performance r_{∞} is often substantially less than the theoretical peak performance $\hat{f}_{\infty}$ of the arithmetic pipelines, because of delays associated with transferring data between main memory and the arithmetic pipelines. This paper therefore introduces the performance parameter $f_{1/2}$ to take into account the degradation of performance due to such memory and communication bottlenecks.

By analogy with $n_{1/2}$, the parameter $f_{1/2}$ is defined as the number of floating-point operations per memory reference (a new variable f) that are required, in order that the

asymptotic performance reach one half of the peak performance. Since the number of floating-point operations and main memory references are known for any algorithm, f can be calculated; and by comparing f to $f_{1/2}$, it is possible to estimate the asymptotic performance that should be achieved. After this the actual performance for a finite vector length can be estimated in the usual way by comparing n to $n_{1/2}$ [9, p. 60]. Thus this characterization treats computer performance as a function of two variables: vector length (n) and floating-point operations per memory reference (f). We call f the *computational intensity*, and $f_{1/2}$ the *half-performance intensity*.

Both $n_{1/2}$ and $f_{1/2}$ are hardware parameters because they can be related to characteristics of the hardware: $n_{1/2}$ is related to the number of segments in the arithmetic pipelines [3–5], and $f_{1/2}$ is related to the ratio of arithmetic speed to memory bandwidth. To find this relationship it is necessary to use a model of computation, and in Section 2 we consider a simple model of a fast arithmetic pipeline working from a local memory which is fed with data from a main memory by a slower I/O memory pipeline. We consider the two cases of *sequential I/O*, when arithmetic and I/O cannot take place simultaneously, and *overlapped I/O* when they can. Different models of computation may give different formulae for $f_{1/2}$ in terms of hardware parameters, but the fundamental definition of $f_{1/2}$ as the value of f required to make the asymptotic performance equal to half the peak performance, remains the same for all cases. The above model corresponds closely to the FPS 5000 computer system, and Section 3 shows good correspondence between theory and measurements when the $f_{1/2}$ characterization is used.

The list below summarizes the terms used in the analysis of performance that follows:

- r_{∞} = asymptotic performance of a pipeline that is approached as the vector length goes to infinity. Measured in floating-point operations per second in an arithmetic pipeline, or in words transferred per second in a memory access pipeline;
- $n_{1/2}$ = vector length required to achieve half the asymptotic performance. Called the half-performance length;
- n = vector length;
- f = average number of floating-point operations performed per main memory reference. Called the computational intensity;
- $\bar{r}_{\infty}(f)$ = asymptotic performance of a combined memory and arithmetic pipeline, for a given value of f ;
- $\bar{n}_{1/2}(f)$ = half-performance length of the combined memory and arithmetic pipeline, for a given value of f ;
- $\hat{r}_{\infty}$ = peak performance, i.e. the limit of asymptotic performance as f goes to infinity;
- $f_{1/2}$ = value of f required to raise the asymptotic performance of the combined memory and arithmetic pipelines to half the peak value. Called the half-performance intensity;
- $r_{\infty}^{(m)}$ = asymptotic performance of the memory pipeline. Commonly called the memory bandwidth in words transferred per second;
- $n_{1/2}^{(m)}$ = half-performance length of the memory pipeline. Equal to the number of words that could have been transferred during the memory latency time;
- $r^{(a)}$ = asymptotic performance of the arithmetic pipeline;
- $n_{1/2}^{(a)}$ = half-performance length of the arithmetic pipeline;
- t_1 = time to transfer a block of n data between main and local memories;
- t_2 = time for one vector operation of length n on data stored in local memory;
- $\bar{t}(f)$ = average time for one vector operation of length n , when there are f floating-point operations for every memory reference.

2. Model of computation

Most current and proposed computers have a hierarchical memory organization: the arithmetic pipelines can run at their peak rates if the data is taken from a relatively small local memory or group of registers, whereas the bulk of the problem data must be stored in a much larger and necessarily much slower main memory. Usually the arithmetic pipelines can process data in the local memories faster than data can be transferred between the main and local memories; consequently the speed of calculations on such computers can become *memory bandwidth limited*. A well-known example of this phenomenon is the CRAY-1, in which the main memory bandwidth was one third of that required to support one arithmetic pipeline. This problem was removed in the CRAY X-MP by increasing the memory bandwidth threefold. However, the problem has returned with the CRAY-2 and ETA¹⁰ in which access to the massive common (or shared) memory is much slower than access to the local memories of the component CPUs.

In order to make a simple model of memory-limited computation, we consider a single memory pipeline linking a main memory to a local memory, and an arithmetic pipeline working entirely on data in the local memory. The model calculation proceeds by transferring a block of n data between the main and local memories, and then performing f vector operations of length n upon the data. The I/O time to transfer a block of n data between memories is modelled by

$$t_1 = (n + n_{1/2}^{(m)})/r_{\infty}^{(m)} \quad (1)$$

and the time to perform a vector operation on data in the local memory is modelled by

$$t_2 = (n + n_{1/2}^{(a)})/r_{\infty}^{(a)}. \quad (2)$$

The time *per vector operation* for the combined memory and arithmetic pipeline is modelled by

$$\bar{t} = (n + \bar{n}_{1/2})/\bar{r}_{\infty}. \quad (3)$$

There are two cases to consider, depending on whether the I/O takes place *sequentially* with the arithmetic, or whether the I/O and arithmetic are *overlapped* in time.

(a) *Sequential I/O*. The total time for f vector operations is $t_1 + ft_2$, hence the time per vector operation is

$$\bar{t} = t_1/f + t_2. \quad (4)$$

Representing this in the standard form (3) we find that both $\bar{r}_{\infty}$ and $\bar{n}_{1/2}$ are functions of f :

$$\bar{r}_{\infty}(f) = \hat{r}_{\infty}/(1 + x^{-1}), \quad (5)$$

$$\bar{n}_{1/2}(f) = (n_{1/2}^{(m)} + xn_{1/2}^{(a)})/(1 + x) \quad (6)$$

where

$$\hat{r}_{\infty} = r_{\infty}^{(a)}, \quad x = f/f_{1/2} \quad \text{and} \quad f_{1/2} = r_{\infty}^{(a)}/r_{\infty}^{(m)}. \quad (7)$$

In the above expressions $\hat{r}_{\infty}$ is seen to be the peak performance, which is obtained in the limit as $f \rightarrow \infty$, i.e. when the I/O overhead of transferring data to local memory is negligible compared with the time to perform arithmetic. In this limit the performance is at its peak and equal to the performance of the arithmetic pipeline alone. It is the value of $\hat{r}_{\infty}$ that is usually quoted by manufacturers to describe the performance of their computers.

For finite values of f , equation (5) describes how the asymptotic performance $\bar{r}_{\infty}$ increases towards its peak $\hat{r}_{\infty}$ as f , the amount of arithmetic per memory reference increases. This

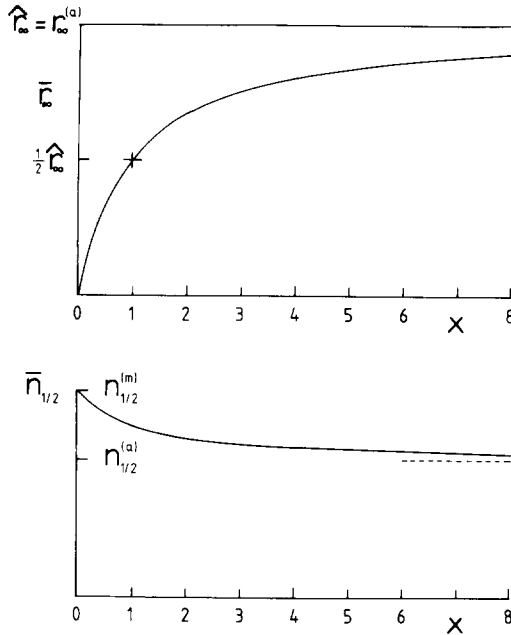


Fig. 1. The variation of asymptotic performance $\hat{r}_\infty$ and half-performance length $\bar{n}_{1/2}$ with the amount of arithmetic per memory reference, f , for the case of a combined memory I/O and arithmetic pipeline, when the I/O and arithmetic take place sequentially. The graphs are drawn for $\nu = \bar{n}_{1/2}^{(m)}/\bar{n}_{1/2}^{(a)} = 1.5$.

approach to the peak is characterized by the value of the parameter $f_{1/2}$, which is seen to be the value of f required to obtain half of the peak performance. We note that $f_{1/2}$ is a hardware parameter, in this case the ratio of arithmetic to memory speeds; whereas f describes the nature of the algorithm that is being executed on the hardware, being the ratio of arithmetic operations to memory references. Equations (5) to (7) show that the performance of the algorithm is determined by the ratio $f/f_{1/2}$ which we have called x . This predicted behaviour is shown in Fig. 1, and is borne out very clearly in the lower curve of Fig. 4.

We note that the above variation of performance with f that has been obtained for sequential I/O, has the same functional form as the variation of the performance, r , of a single vector pipeline with vector length n [4,5,9], namely from (3)

$$r = n/\bar{t} = \hat{r}_\infty / (1 + x^{-1}) \quad (8)$$

where now

$$x = n/\bar{n}_{1/2}.$$

Combining equations (8) and (5), we obtain the performance as a function of the two variables n and f :

$$r = \frac{\hat{r}_\infty}{(1 + \bar{n}_{1/2}/n)(1 + f_{1/2}/f)} \quad (9)$$

where the first factor in the denominator gives the degradation of performance due to finite vector length, and the second factor the degradation due to memory bottleneck. For small values of $n \ll \bar{n}_{1/2}$ and $f \ll f_{1/2}$ one obtains

$$r \approx \hat{r}_\infty \left(\frac{n}{\bar{n}_{1/2}} \right) \left(\frac{f}{f_{1/2}} \right) \quad (10)$$

which shows that it is not the absolute values of n and f but their ratios to $\bar{n}_{1/2}$ and $f_{1/2}$ that

give the factors by which the peak performance is degraded. On the other hand for large values of $n \gg \bar{n}_{1/2}$ and $f \gg f_{1/2}$,

$$r \approx \hat{r}_\infty. \quad (11)$$

Expression (9) suggests that the performance of a wide range of algorithms on a wide range of computers, might be predicted by describing the computers with the three performance parameters ($\hat{r}_\infty$, $\bar{n}_{1/2}$, $f_{1/2}$) and characterizing the algorithms by appropriate average values of (n , f). In making use of equation (9), it must be remembered that $\bar{n}_{1/2}$ is not, strictly speaking, a pure hardware parameter as its value does depend on the f of the algorithm via equation (6). However, if $n_{1/2}^{(m)} \approx n_{1/2}^{(a)}$ this variation is negligible, and $\bar{n}_{1/2} \approx n_{1/2}^{(m)} \approx n_{1/2}^{(a)}$.

In the above model of computation, we have assumed for simplicity that all memory transfers are block transfers of size n , and that input to local memory can be treated in the same way as output from local memory. Although an accurate representation of any particular computer system may need a more complex representation, we feel that the above model includes the essence of memory-limited computation, and can therefore be used as an approximate simple model for more complex systems. For example, in using equations (5)–(11) we would expect to get a reasonable approximate result in a more complex case by treating f as the overall ratio of arithmetic to I/O operations in an algorithm, even though this ratio may change in different parts of the algorithm; and to take the ratio of total arithmetic to number of vector operations as the vector length, n , in (9), even though the vector length may change during the algorithm. Also we would regard $r_\infty^{(m)}$ as the total memory bandwidth, and $r_\infty^{(a)}$ as the total arithmetic performance; even though these rates may be achieved with multiple I/O pipelines and multiple arithmetic pipelines. Similarly $n_{1/2}^{(m)}$ and $n_{1/2}^{(a)}$ are the effective half-performance lengths of the multiple pipelines, when they are regarded as an equivalent single pipeline. Note, particularly, that when a pipeline is replicated c times, both its effective performance rate *and* its effective half-performance length are c times that of a single pipeline. This replication effect is a simple consequence of the standard timing equation (3) as is shown in [9, p. 181].

(b) *Overlapped I/O.* When I/O between the main and local memories can take place simultaneously with arithmetic, we say that the I/O and arithmetic are overlapped. In this case the average time per vector operation is the maximum of the I/O time and the arithmetic time

$$\bar{t} = \max\{t_1/f, t_2\}. \quad (12)$$

If the I/O time (first term) dominates,

$$\bar{t} = t_1/f = (n + n_{1/2}^{(m)})/(fr_\infty^{(m)}). \quad (13a)$$

Comparing with the standard form (3) we obtain

$$\bar{r}_\infty = fr_\infty^{(m)}, \quad \bar{n}_{1/2} = n_{1/2}^{(m)}. \quad (13b)$$

If, however, the arithmetic time (second term) dominates, then

$$\bar{t} = t_2 = (n + n_{1/2}^{(a)})/r_\infty^{(a)}. \quad (14a)$$

Hence, comparing again with (3) we have

$$\bar{r}_\infty = r_\infty^{(a)}, \quad \bar{n}_{1/2} = n_{1/2}^{(a)}. \quad (14b)$$

The behaviour of overlapped I/O is rather more complicated than that of sequential I/O, and is illustrated in Fig. 2. The dotted lines give the parameters when the I/O time dominates, and the full lines apply when the arithmetic dominates. The figure also assumes the usual case that the arithmetic pipeline is faster, and has a smaller half-performance length than the I/O pipeline (i.e. $r_\infty^{(a)} > r_\infty^{(m)}$, and $n_{1/2}^{(a)} < n_{1/2}^{(m)}$).

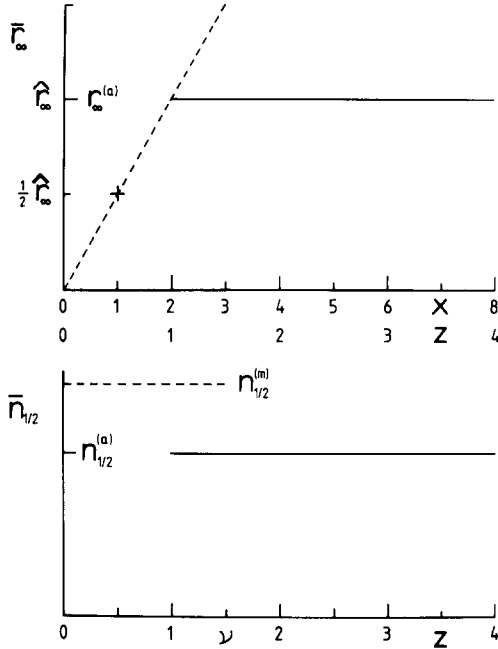


Fig. 2. The variation of $(\hat{r}_\infty, \bar{n}_{1/2})$ with f for the case of a combined memory I/O and arithmetic pipeline, when the I/O and arithmetic can be overlapped. Full lines: parameters when arithmetic dominates, equations (14b); dotted lines: parameters when I/O dominates, equations (13b). Notation as Fig. 1, and $\nu = 1.5$.

In order to find out whether I/O or arithmetic dominates, one must examine the breakeven vector length, n_1 , at which I/O and arithmetic take equal times. This occurs when $t_1/f = t_2$, whence

$$n_1 = (n_{1/2}^{(m)} - zn_{1/2}^{(a)})/(z - 1) \quad (15)$$

where $z = fr_\infty^{(m)}/r_\infty^{(a)}$.

The variation of n_1 with z is drawn in Fig. 3 for the case $\nu = n_{1/2}^{(m)}/n_{1/2}^{(a)} = 1.5$, and the regions of the (z, n_1) -plane corresponding to I/O or arithmetic dominance are shown.

If $z > \nu$, the arithmetic time dominates for all vector lengths, because n_1 is negative. Equations (14) apply, and the asymptotic performance is constant and equal to the $r_\infty^{(a)}$ of the arithmetic pipeline. Since this is the situation when $f \rightarrow \infty$ we have, by definition, the peak performance

$$\hat{r}_\infty = r_\infty^{(a)}, \quad (16a)$$

the same as for the sequential I/O case.

If, however, $z < 1$, I/O dominates for all vector lengths (because n_1 is again negative) and equations (13) apply. The total computation time, t_1 , for f vector operations is constant, hence the asymptotic performance rises linearly with f , reaching the peak performance $\hat{r}_\infty = r_\infty^{(a)}$ when $f = r_\infty^{(a)}/r_\infty^{(m)}$. The asymptotic performance reaches half the peak performance when f reaches half this value, hence by definition

$$f_{1/2} = \frac{1}{2}r_\infty^{(a)}/r_\infty^{(m)}. \quad (16b)$$

Thus overlapping halves the value of $f_{1/2}$ from that obtained for sequential I/O.

Between $1 < z < \nu$, either I/O or arithmetic may dominate depending on the vector length (because, now, n_1 is positive). Figure 2 shows that if I/O dominates $n < n_1$ the asymptotic $\hat{r}_\infty$ performance can exceed the asymptotic performance of the arithmetic pipeline $r_\infty^{(a)}$, and it might appear that this is absurd and against physical intuition. However, this is not the case, because $\hat{r}_\infty$ is a theoretical asymptotic performance (for $n \rightarrow \infty$) which in this case can never be

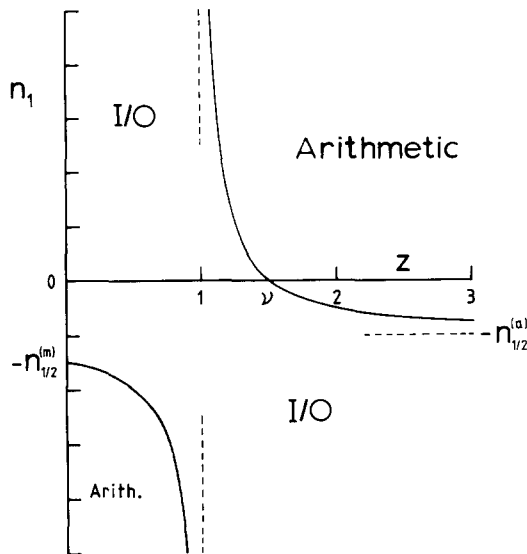


Fig. 3. The breakeven vector length, n_1 , as a function of $z = fr_{\infty}^{(m)}/r_{\infty}^{(a)}$, showing the regions of the (z, n_1) -plane in which either I/O or arithmetic dominates the calculation time. The graph is drawn for $\nu = 1.5$.

reached, because n must be less than n_1 for the dotted line to apply. If n were increased towards infinity in order to try to attain the higher asymptotic rate, n would exceed n_1 , arithmetic would dominate, and the solid curve of Fig. 3 would apply, with $r_{\infty} = r_{\infty}^{(a)}$. It must be remembered that *two* parameters are always necessary to measure the performance of a pipeline, namely both $\bar{r}_{\infty}$ and $\bar{n}_{1/2}$; and that the actual performance, r , is related to the asymptotic performance by equation (8), i.e.

$$r = \bar{r}_{\infty} / (1 + \bar{n}_{1/2}/n). \quad (17)$$

If the effect of the term $\bar{n}_{1/2}/n$ is included, one finds that the actual performance is *always* less than the asymptotic performance of the arithmetic pipeline, as is intuitively obvious.

The above theoretical prediction of the variation of performance with f for overlapped I/O, namely a linear rise to the peak value as shown in Fig. 2, is borne out by the upper curve of measured values in Fig. 4.

3. Measurements of $f_{1/2}$

The Floating Point System 5000 computer [1] provides a good example of a hierarchical memory system which should be well represented by the simple theoretical model of computation that has been described above. Computation takes place within the XP32 *arithmetic coprocessor* which contains a floating-point multiply pipeline and two floating-point add pipelines. These pipelines take their data from, and return results to a local memory (LM) of 16 kwords contained within the XP32. The bulk data for the problem being solved, is however held in a separate *system common memory* (SCM) which can have a capacity of 1 Mword. Calculation proceeds by transferring blocks of data from SCM to LM, performing arithmetic upon it, then returning the results from LM back to SCM. It is easy to vary the ratio of arithmetic to memory references, f , and to execute the arithmetic sequentially with the I/O between SCM and LM, or time-overlapped with it.

The measured results for $\bar{r}_{\infty}(f)$ as a function of f are shown in Fig. 4, in which the circles are measured values, and the full lines the theoretical predicted curves. The lower curve is for sequential I/O, and shows the characteristic shape shown in Fig. 1. There is good agreement

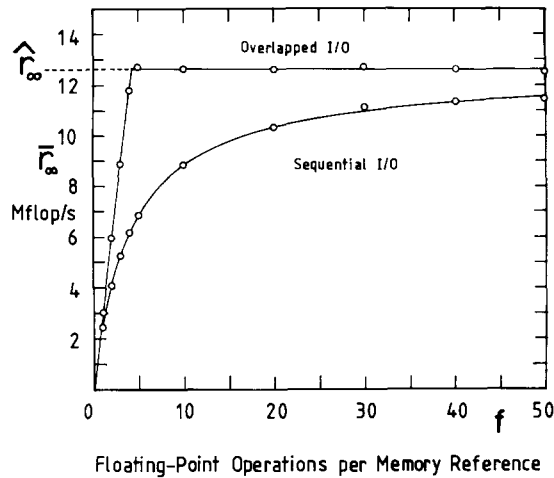


Fig. 4. Comparison of theoretical predicted performance $\hat{r}_\infty$ (Mflop/s) as a function of f (solid lines) with measured values (circles), for the FPS-XP32 arithmetic coprocessor processing data in system common memory. There are f floating-point operations performed within the XP32 for every common memory reference.

between the curve predicted by equation (5) and the measured values. The results for overlapped I/O show the linear rise to the peak value as predicted in Fig. 2, and equations (13b) and (14b). The values for peak performance and $f_{1/2}$ which are in the predicted curves of Fig. 4 are given in Table 1, and we see, as was predicted in Section 2, that overlapping I/O with arithmetic approximately halves the value of $f_{1/2}$.

Figure 5 gives the XP32 program that was used in the above measurements. The 'DO 200' loop is a repeat to increase the accuracy of the timing which is performed to microsecond accuracy on a separate GPIOP processor that does not interfere with the operation of the XP32. The time is measured immediately before and after the loop with the CALL to the routine APSEC. Within the 'DO 200' loop, the input vector A is transferred from the SCM to the XP memory by the first CALL XPDMAR, the arithmetic is performed by the CALL ZVSASM, and the result vector D is transferred from the XP memory to the SCM memory by the second CALL XPDMAR. The arithmetic kernel used in this experiment, vector scalar add scalar multiply (VSASM), adds a scalar to each element of the input vector and then multiplies each element by another scalar. There are thus two floating-point operations per element for every CALL ZVSASM. Since each element is read once from SCM and stored once to SCM, there are two memory references per element, and the value of f is equal to the number of CALLs to ZVSASM, which is varied in the 'DO 150' loop where $f = \text{IMEMREF}$. The two CALLs to XPISNC ensure that the input vector is loaded completely before the arithmetic starts, and that the arithmetic completes before the output vector is stored. This is the case of sequential I/O. The timing for overlapped I/O is simulated by commenting-out the two calls to XPISNC. In this case the loading and storing of the vectors takes place simultaneously with the arithmetic, as it would in a triple buffering scheme. In each loop of such a scheme, the arithmetic on one segment of the vector is performed simultaneously with the storage of the result of the previous segment, and with the loading of the next segment. Although the program

Table 1

Values of peak performance, $\hat{r}_\infty$, and $f_{1/2}$ for a single FPS XP32 arithmetic coprocessor when processing data in system common memory; these values were used in the theoretical curves of Fig. 4

Case	$\hat{r}_\infty$ (Mflop/s)	$f_{1/2}$
Sequential I/O	12.5	4.2
Overlapped I/O	12.6	2.2

```

C----- Memory Bandwidth test program
C----- benchmark segment -----
C
      PROGRAM TIMEST
C
      INTEGER NCHAN,NLOOP,CLEN,IMEMREF
      PARAMETER (MAXNCHAN = 8000) ! number of channels per scan
C
      REAL A(MAXNCHAN)           ! input vector, one scan of channels
      REAL B(MAXNCHAN)           ! output / input
      REAL NEWTIME
      INTEGER IXPA,IXPB,I,J,K,KK
      COMMON /PARS/ NCHAN,NLOOP,CLEN,IMEMREF,NEWTIME
C
      NEWTIME = 0.
C
      IA =      0                ! XP32 LMD address of input vector
      IB =     NCHAN             ! XP32 LMD address of first scalar
      IC = 2 * NCHAN             ! XP32 LMD address of second scalar
      ID = 3 * NCHAN             ! XP32 LMD address of output vector
C
      IXPA = 1                   ! use XP32 number 1
C
      J = XPWAIT (-1) ! wait for all to stop
      CALL APSEC (NEWTIME)
      DO 200 K=1,NLOOP
C
C === ask for DMA transfers for input
      CALL XPDHAR (NCHAN, IA,1,A,1,2,1) ! load input vector
C
C === Synchronise the I/O with the arithmetic section (SEE NOTES)
      CALL XPISNC
C
C === Ask arithmetic section for Vector Scalar Add, Scalar Multiply
      DO 150 I=1,IMEMREF
150      CALL ZVSASM (IA,IB,IC,ID,NCHAN) ! xp32 call
C
C === Synchronise the I/O with the arithmetic section (SEE NOTES)
      CALL XPISNC
C
C === ask for DMA transfers for output
      CALL XPDHAR (NCHAN, ID,1,B,1,2,-1) ! send results to SCM
C
      J = XPSEL (IXPA)
      CALL XPRUN ! dispatch channel list
C
C === wait for all to stop
      J = XPWAIT (-1) ! wait for channel list to complete
200      CONTINUE
C
      CALL APSEC (NEWTIME)
      STOP
      END

```

Fig. 5. The XP32 arithmetic coprocessor program used in the measurement of $\bar{n}_{1/2}$ and $f_{1/2}$. See text for explanation.

in Fig. 5 does not contain all the necessary details for this buffering, the timing obtained by omitting the XPISNC calls should not be essentially different from that obtained from a properly buffered program. The CALL XPRUN initiates the execution of program described above, and XPWAIT ensures that each instantiation of the 'DO 200' loop is completed before the next starts.

4. Conclusions

With the move to large hierarchical memories in many new computer systems, there is a need for a parameter to quantify the seriousness of the memory bottleneck problem. In this

paper we have introduced a new performance parameter, $f_{1/2}$, for this purpose, and shown that it can be used to predict accurately the performance of the FPS-5000 computer system. We conclude that $f_{1/2}$ could prove to be a very useful parameter for quantifying memory-bandwidth limited computation, and hope that similar measurements will be made on other computer systems.

Acknowledgment

The authors wish to thank the management of Floating Point Systems U.K. Ltd., Bracknell, for use of their FPS-5000 computer system, and the support staff for their assistance.

References

- [1] S.F. Cannon, VLSI chips increase processing power in array processors, *Comput. Technology Rev.* (Winter 1983) 294–295.
- [2] I.J. Curington and R.W. Hockney, Synchronization and pipeline overhead measurements on the FPS 5000 MIMD computer, in: M. Feilmeier, G. Joubert and U. Schendel, eds., *Parallel Computing 85* (North-Holland, Amsterdam, 1986) 469–476.
- [3] R.W. Hockney, Supercomputer architecture, in: F. Summer, ed., *Infotech State of the Art Conference: Future Systems* (Infotech International, Maidenhead, 1977) 277–305.
- [4] R.W. Hockney, Characterization of parallel computers and algorithms, *Comput. Phys. Commun.* **26** (1982) 285–291.
- [5] R.W. Hockney, Characterizing computers and optimizing the FACR(1) Poisson-solver on parallel unicomputers, *IEEE Trans. Comput.* **32** (1983) 933–941.
- [6] R.W. Hockney, The $n_{1/2}$ -method of algorithm analysis, in: B. Engquist and T. Smedsaas, eds., *PDE Software: Modules, Interfaces and Systems* (North-Holland, Amsterdam, 1984) 429–444.
- [7] R.W. Hockney, (r_{∞} , $n_{1/2}$, $s_{1/2}$) measurements in the 2-CPU CRAY X-MP, *Parallel Comput* **2** (1985) 1–14.
- [8] R.W. Hockney, Algorithmic phase diagrams, *IEEE Trans. Comput.* **36** (1987) 231–233.
- [9] R.W. Hockney and C.R. Jesshope, *Parallel Computers 2 – Architecture, Programming and Algorithms* (Adam Hilger, Bristol, 1981).
- [10] R.W. Hockney and D.F. Snelling, Characterizing MIMD computers: e.g. the Denelcor HEP, in: M. Feilmeier, G. Joubert and U. Schendel, eds., *Parallel Computing 83* (North Holland, Amsterdam, 1984) 521–526.