Explaining the Gap between Theoretical Peak Performance and Real Performance for Supercomputer Architectures*

W. SCHÖNAUER AND H. HÄFNER

Rechenzentrum der Universität Karlsruhe, Postfach 6980, D-76128 Karlsruhe, Germany

ABSTRACT

The basic architectures of vector and parallel computers and their properties are presented followed by a discussion of memory size and arithmetic operations in the context of memory bandwidth. For a single operation micromeasurements of the vector triad for the IBM 3090 VF and the CRAY Y-MP/8 are presented, revealing in detail the losses for this operation. The global performance of a whole supercomputer is then considered by identifying reduction factors that reduce the theoretical peak performance to the poor real performance. The responsibilities of the manufacturer and of the user for these losses are discussed. The price-performance ratio for different architectures as of January 1991 is briefly mentioned. Finally a user-friendly architecture for a supercomputer is proposed. © 1994 by John Wiley & Sons, Inc.

1 BASIC ARCHITECTURES

In Figure 1 the prototype vector supercomputer is illustrated. The technology is usually based on ECL and characterized by the cycle time τ in nanoseconds. The main feature is an add/logical and a multiplication pipeline that are referred to as a pipe group. The pipelines operate with operands from the vector registers. Data are transferred between memory and vector registers by one or several load/store pipelines. Pipelines can be "chained" to form one large pipeline. If the add and multiply pipes operate in parallel supervector speed is obtained with two results per cycle and pipe group. The pipelines may be multitrack pipelines, e.g., a four-track pipeline is combined internally from four parallel pipes and delivers four results per cycle. The pipes may operate under the control of a mask.

Critical points of such a supercomputer are (see Fig. 1): (1) The finite length of the vector registers (e.g., 64 elements) requiring that longer vectors are processed in "sections" (strip-mined) with a restart of the pipelines for each section. Usually the compiler generates a "section loop." (2) The most critical point is the memory bandwidth in words (64 bits) per cycle and pipe group. Usually the performance of a supercomputer is limited by the memory bandwidth and not by the speed of the pipelines. Unfortunately most supercomputers have a one-word memory bottleneck. Laudable

* This is an extended version of a paper printed in H. Küsters, E. Stein (Eds.), *Proceedings of the Joint International Conference on Mathematical Methods and Supercomputing in Nuclear Applications*. Kernforschungszentrum Karlsruhe, Karlsruhe, Germany, 1993, vol. 2, pp. 75–88.

Received June 1993
Revised November 1993
Accepted January 1994

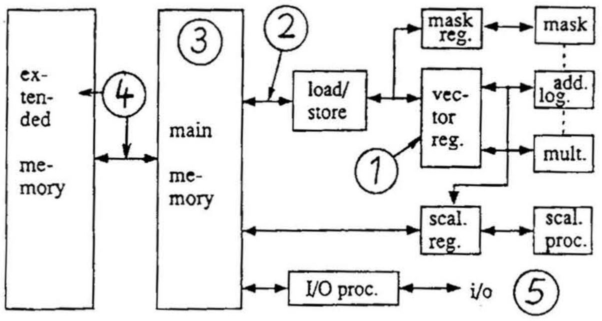

FIGURE 1 Prototype vector supercomputer.

exceptions are the CRAY Y-MP and C90 supercomputers with two loads and one store per cycle and pipe group. (3) The memory size and (4) the size and bandwidth of the extended memory ultimately limit the problem size that can be solved on the supercomputer. (5) The input/output of even the fastest disks is much too slow to be used to deliver data for the extremely fast pipelines.

Figure 2 represents a prototype superscalar node of a parallel computer. Its core is a superscalar RISC processor that can execute several instructions simultaneously. The technology usually is CMOS characterized by the frequency f in MHz. The floating-point units for add and multiply usually are pipelined and can operate in parallel (superscalar speed).

Critical points of such a superscalar node are (see Fig. 2): (1) The size of the data cache has significant influence on the performance because (2) the memory bandwidth between the processor and the memory is one word or even half a word per cycle. If data must be taken from the memory, the combined effect of memory bandwidth and "cache stumbling" (reload of data by cache lines, possibly with "dirty" cache misses) leads to a significant drop in performance. Thus reuse of data in the cache is essential. The size of (3) the memory determines the size of the problems that can be solved. (4) The communication bandwidth in MB/s, the startup time for sending a message in μsec (latency), and the type of interconnection limit the global performance of the parallel computer. The influence of communication, however, depends strongly on the type of problem and/or the selected algorithm. If it is possible to hide completely the communication behind the computation (latency hiding) there is no influence of the communication on the performance. Therefore we must look at the ratio of computation to communication.

Figure 3 shows the two basic types of parallel computers based on p processors. For the shared memory parallel computer the problem must be split into p "independent" tasks. The number of processors p is limited by the memory bandwidth and the memory contention of the p processors. Idling processors can be used easily for other jobs because of the shared data. Large parallel vector supercomputers like the CRAY Y-MP C90 with 16 processors are of this type.

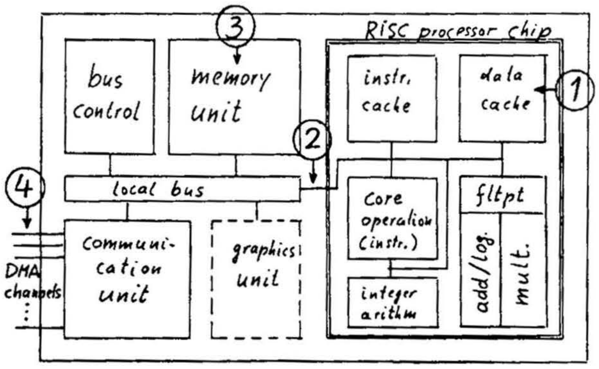

FIGURE 2 Prototype superscalar node.

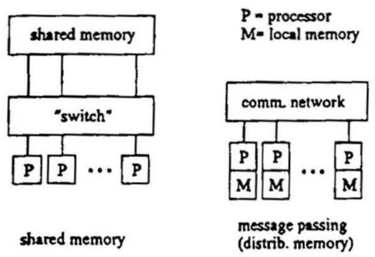

FIGURE 3 Shared memory and distributed memory parallel computer.

The distributed memory/message passing parallel computer has only local memories. A processor can operate only on data in "its" memory. If it needs data from nonlocal memories, the data has to be transferred by the communication network. Here the problem must be split into p independent tasks with independent data. This means that ultimately all problems are shifted to the user. The use of an idling processor for another job is not possible for many reasons. Thus "space sharing" is used where each user gets a fixed partition of the computer, his/her GFLOPS-PC. The number of processors is in theory unlimited, but in practice limited by the properties of the communication network. Currently based on a mixture of hardware and software, manufacturers are implementing "virtual shared memory" with a global address space. The Kendall Square Research KSR1 [1] parallel computer is the first computer that realizes this concept, other manufacturers will follow soon. Unfortunately this does not solve the basic problem of communication and availability of data at the right moment in the right local memory, but results in an easier programming style. Experience has to show how much must be paid for this convenience as compared with the explicit message passing.

From recent preannouncements of some manufacturers, hybrid parallel computers are emerging where several microprocessors are clustered in a node and access the (shared) node memory. Thus a node is a shared memory parallel computer and several nodes are combined into a distributed memory parallel computer. A future version of the Intel Paragon with four i860 XP arithmetic processors or the Convex SPP-1 with eight HP 7100 processors in a node are such hybrid parallel computers.

This discussion implies that there are four basic methods to make a supercomputer faster: (1) Reduce the cycle time τ or increase the frequency f, this is limited by the hardware technology. (2) Use pipelining or simulate a pipeline in a superscalar processor, this is limited by the number of stages. (3) Use parallel pipelines (multitrack pipelines), this is limited by the memory bandwidth. (4) Use multiprocessors, i.e., parallel computers (external parallelism). This is limited for the shared memory computers by the memory bandwidth and memory contention and for the message passing computers by the properties of the communication network. Usually supercomputers use all four possibilities up to their limits.

2 MEMORY BANDWIDTH AND MEMORY SIZE

Trivially data processing means moving data and storing data. Because of the memory bottleneck the number of necessary memory references for the different types of basic operations are required to be known: For example in the following statements s is a scalar, a, b, c, d are vectors.

$$a_i = b_i + c_i, \tag{1}$$

$$a_i = b_i + c_i * d_i, \tag{2}$$

$$a_i = b_i + s * c_i, \tag{3}$$

$$a_i = a_i + b_i * c_i, \tag{4}$$

$$a_i = a_i + s * b_i, \tag{5}$$

$$s = s + a_i * b_i. \tag{6}$$

Equation 1 is a dyadic operation that needs two loads and one store per cycle and arithmetic group (* and +). For triadic operations with three operands of the type shown in Equations 2 to 6 supervector or superscalar speed can be achieved by the parallel execution of multiplication and addition. Equation 2 is the vector triad or general triad: this is the most important operation. The performance ratio of engineering benchmarks for different supercomputers corresponds closely to the performance ratio of the vector triad. This is the most general operation for supervector speed. It needs three loads and one store, i.e., four memory references.

The linked triad (Equation 3) with one scalar operand needs two loads and one store. In a repeated contracting general (Equation 4) or linked (Equation 5) triad and if a_i can be fixed in a vector register, then only two loads for Equation 4 or one load for Equation 5 is required. Equation 5 is the kernel operation of the matrix multiplication in columnwise form and the solution of linear equations in a block-oriented form (e.g., in the LINPACK benchmark [3]) and encounters the one-word memory bottleneck. For a supercomputer with a memory bandwidth of one word per cycle and arithmetic group the performance for Equation 5 is four times that for Equation 2. The general triad (Equation 2) is representative of engineering applications whereas the contracting linked triad is representative of problems in theoretical physics that are mainly based on matrix multiplication. A count of the types of operations

Table 1. Actual and Expected Price for 1 TB of DRAM Memory Chips in M$

Type of Chip	Number of Chips	Price in M$ in Year			
		1992	1993	1994	1995
1 Mbit	8 M	28	28	29	29
4 Mbit	2 M	22	15	12.5	11.2
16 Mbit	0.5 M	63	33	15.5	10.5
64 Mbit	0.125 M	—	—	54	28

and not only + and * is necessary. Equation 6 is the scalar product that can be vectorized by the wheel method [2], it needs two loads.

The size of the memory determines the size of the problems that can be solved on a supercomputer. and it also affects its price. For a 1-GFLOPS theoretical peak performance for engineering applications (finite element. finite volume. finite difference methods) roughly 0.1 to 1 Gword (1 word = 64 bits = 8 bytes) of memory is needed. This is the sum of main and extended memory for the ECL-based large and fast technology supercomputers. For the lower limit at least 25% and for the upper limit at least 10% should be main memory. For the CMOS microprocessor-based parallel computers this represents the sum of the node memories over all nodes. Manufacturers presently have begun to develop the TFLOPS (teraFLOPS. 10^{12}) machine. For 1 TFLOPS. 0.1 to 1 Tword of memory is required. i.e.. 0.8 to 8 TB. How many chips and how many M$ are needed for 1 TB of distributed random access memory (DRAM) memory? There is no significant difference between the (selected) chips for main memory and the chips for extended memory or for the memory of microprocessors. The speed ranges from 50 to 80 ns access time. The results according to a private communication of a manufacturer is presented in Table 1. Up to 1994 the 4-Mbit chip yields the lowest price, but 2 million chips is an extremely large number. Thus we will have to wait until we can afford the 64-Mbit chip. If we take the upper limit of 8 TB the number of chips and dollars increases eightfold and faster static RAM (SRAM) chips are four times the price of DRAM chips.

Memory is the critical component of all supercomputers. Only the CRAY Y-MP and C90 have a memory bandwidth of two loads and one store per cycle and arithmetic group (note: per arithmetic group means per multiplication and addition floating-point unit). but even this is not sufficient to support the vector triad (Equation 2). Unfortunately most other supercomputers have the one-word memory bottleneck. The Fujitsu VP 2600 has eight words per cycle load or store. but it has also eight pipe groups. thus it has the one-word memory bottleneck. The size of the memories of supercomputers is usually below the lower limit mentioned above. which means that their usability is restricted by memory size and that intolerable input/output times result if large problems are solved on such a supercomputer.

3 SINGLE OPERATIONS

The most significant operation for a supercomputer is the vector triad (Equation 2). where eventually only 10% of the theoretical peak performance is obtained and thus 90% of the performance is lost. Where are the lost cycles [4–7]?

The key to the investigation of a single operation is micromeasurements that carefully analyze the timing behavior and make all influencing parameters visible. For this purpose the assembler code of a vector operation is investigated and the timing of all instructions is determined. taking into account the overlapping of operations using chaining. The resulting time to process a vector of n elements is defined as $t_{mic}(n)$.

The result of the micromeasurement for the vector triad on one processor of an IBM 3090S VF (Vector Facility) are the following timing formulae. it is necessary to distinguish whether the data is in the cache or in main storage:

data in cache

$$t_{mic}(n) = \tau \left(\underbrace{2\left(n + \left\lceil \frac{n}{vrl} \right\rceil * 28\right)}_{\text{load b,c}} + \underbrace{n + \left\lceil \frac{n}{vrl} \right\rceil * (\overset{\substack{\text{load d}\\ \downarrow}}{28} + \overset{\substack{\text{sectioning}\\ \downarrow}}{22}) + 5}_{\substack{\text{multiply and add.}\\ \text{data from cache}}} + \underbrace{n + \left\lceil \frac{n}{vrl} \right\rceil * 31 + \left\lceil \frac{n}{cll} \right\rceil}_{\text{store a}} \right). \tag{7a}$$

data in main storage

$$t_{mic}(n) = \tau \Big(\underbrace{2\left(n + \left\lceil \frac{n}{vrl} \right\rceil * 38 + \left\lceil \frac{n}{cll} \right\rceil * 6\right)}_{\text{load b, c from m.s.}} + \underbrace{n + \left\lceil \frac{n}{vrl} \right\rceil (\overset{\overset{\text{d}}{\downarrow}}{38} + \overset{\overset{\text{sect.}}{\downarrow}}{22}) + \left\lceil \frac{n}{cll} \right\rceil * 6 + 5}_{\text{multiply and add, d from m.s.}}$$

$$+ \underbrace{n + \left\lceil \frac{n}{vrl} \right\rceil * 32 + \left\lceil \frac{n}{cll} \right\rceil * 18}_{\text{store a into m.s.}} + \underbrace{\left(\left\lceil \frac{n}{ps} \right\rceil - 1\right) * 95}_{\text{3 ld. 1 st.over ps}} \Big). \qquad (7b)$$

In Equation 7 the hardware cycle time τ is outside the parentheses, with the cycles inside the parentheses. This formula highlights the influence of the architectural parameters and the sources of lost cycles. The cycle time $\tau = 15$ ns, the length of the vector registers vrl = 256 words (word 64 bits = 8 bytes), the cache line length cll = 16 words, the page size ps = 512 words (4 KB). $\lceil a \rceil$ denotes the ceiling function, i.e., the next higher integer larger or equal to a. $t_{mic}(n)$ is a step function that jumps to the next step if one of the ceiling functions jumps.

If we replace the step functions by a straight line through the center of the steps we get a "continuous model" $t_{mean}(n)$. In the case of the data in main storage we get

$$t_{mean}(n) = \underbrace{\underbrace{\left(4 + \frac{146}{vrl} + \frac{36}{cll} + \frac{95}{ps}\right)}_{W} \tau}_{\tau_{eff.mean}} \left(n + \underbrace{\frac{48.5}{W}}_{n_{1/2.eff.mean}}\right). \qquad (8)$$

This formula has been written so that within the second parentheses we have the n elements of the vector and an absolute term, a fictitious number of "elements," which is Hockney's half performance length [8, 9] $n_{1/2.eff.mean}$. The subscript eff indicates an effective mean value that will not hold for an individual operation. The ideal time would be $t_{ideal}(n) = \tau n$. Instead of τ we have in Equation 8 $\tau_{eff.mean} = W\tau$ and instead of n we have n + $n_{1/2.eff.mean}$. W increases the hardware cycle time and could be called a waste factor. Architectural efficiency can be defined as $\eta_a = 1/W$. If W has the ideal value 1, $\eta_a = 1$. Hockney's half performance length results in a natural and tranparent way. From Equation 8 the following performance parameters emerge (in parentheses are the corresponding values for data in cache that result from Equation 7a in the same way):

W = 7.01 (4.60), $\tau_{eff.mean}$ = 105.1 (69.0) ns,

η_a = 0.143 (0.212), $n_{1/2.eff.mean} \simeq$ 7 (16).

These numbers speak for themselves. The low value of $n_{1/2.eff.mean}$ results from the large value of W and is not the merit of short startup.

Figure 4 shows the mean model $t_{mean}(n)/\tau$ in cycles for data in main storage and in cache. The measurements are produced by repeated executions of the vector triad. If the data do not fit into the cache the cache starts rolling (or better stumbling) and the transition from data in cache to

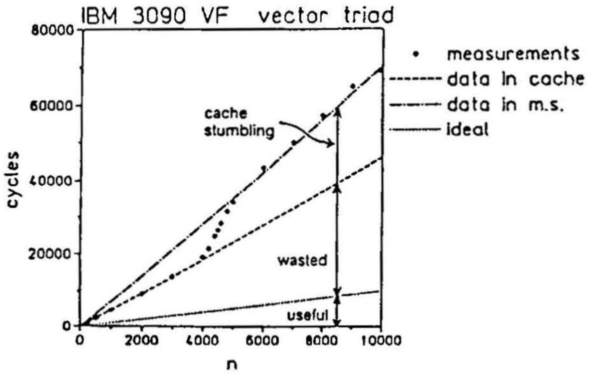

FIGURE 4 Mean model for data in main storage and data in cache for the vector triad for the IBM VF.

data in main storage takes place. It would be much better to regain the cycles that are lost.

The measurement of the vector triad for a CRAY Y-MP/8 on a single processor using assembler code indicates that two sections of 64 elements are processed in three chimes (here chime means the number of cycles needed to process one vector register length) because the linked triad (Equation 2) needs three loads and one store whereas the CRAY has only two loads and one store. Thus the six loads and two stores for two sections are delivered by the six loads and three stores of three chimes, one store is lost. The result of the micromeasurement is:

$$t_{mic}(n) = \tau\left(n + \underbrace{\left\lceil\frac{n}{2vrl}\right\rceil * \min(n, vrl) + 6\left\lceil\frac{n}{2vrl}\right\rceil}_{\substack{\textit{additional cycles because}\\ \textit{of third chime per}\\ \textit{128 elements}}} + \overset{\substack{\text{section loop organization}\\ \uparrow}}{6\left\lceil\frac{n}{vrl}\right\rceil} + \underset{\substack{\downarrow\\ \textit{programming of a port for}\\ \textit{load and termination of}\\ \textit{add and store pipe}}}{23} + \overset{\substack{\text{startup}\\ \uparrow}}{29}\right) \qquad (9)$$

Replacing the step functions in $t_{mic}(n)$ in Equation 9 by mean straight lines gives the mean timing formula

$$t_{mean}(n) = \underbrace{\underbrace{\left(\frac{3}{2} + c_s + c_e + \frac{9}{vrl}\right)}_{W}\tau}_{\tau_{eff.mean}}\left(n + \underbrace{\frac{58 + \frac{vrl}{2}}{W}}_{n_{1/2,eff.mean}}\right). \qquad (10)$$

The resulting performance parameters are:

$W = 1.96, \quad \tau_{eff.mean} = 11.76$ ns,

$\eta_a = 0.51, \quad n_{1/2,eff.mean} \simeq 46.$

The parameters are $\tau = 6$ ns, vrl = 64 words. However, these measurements have been made under ideal conditions on a dedicated computer so that there has been no memory access contention between the two load ports and the store port by an optimal placement of the operands and of the result in the memory. In general self-induced memory access conflicts of the ports obey a certain pattern, depending on the relative location of the operands in the memory banks (no additional bank conflicts occur). For general location of the operands "in the mean" (mean value over many positions) for n useful operations $n * c_s$ cycles are wasted for self-induced memory access conflicts. The defined value of c_s must be measured on a dedicated computer.

On a Y-MP/8 with eight processors there is memory contention by the seven other processors. This influence clearly depends on the local job profile, in our case that at the KFA Jülich. Again in the mean for n useful cycles $n * c_e$ cycles is wasted for external memory contention, thus defining the loss coefficient c_e. The following values for the vector triad were measured: $c_s = 0.1$, $c_e = 0.22$. These are the values published by Häfner and Schönauer [6], due to more extensive measurements they are significantly better than those reported by them [5, 7].

These values are essentially better than those of the IBM 3090S VF. For this most significant operation the loss for the IBM is 85.7% or 78.8% if the data is in the main storage or in the cache, whereas for the CRAY the loss is "only" 49%. These losses are primarily due to memory bottlenecks, their sources can be seen directly from the micromeasurements. Thus an explanation of the gap between theoretical peak performance and real performance for a single operation has been obtained.

4 COMPLETE COMPUTER: ESTIMATE OF GLOBAL EFFICIENCY

In the preceding section we have investigated the details of a single operation. However, the user and above all the manager of a computer center are more interested in an evaluation of the whole supercomputer. They may resort to benchmarks and if carefully interpreted, may be of great use, but they may eventually tell more about the quality of the software than the tested supercomputer. In [11] all relevant aspects of benchmarking are discussed in 21 articles by highly reputated specialists, also the very difficult problem of benchmarking parallel computers is covered. These articles clearly show the inherent difficulties of benchmarking. Another possibility is to analyze the individual reduction factors [7, 10] that reduce the theoretical peak performance of a supercomputer to the possibly very poor real performance. These reduction factors are much more instructive than benchmarks and above all they identify who is responsible for the individual loss: the manufacturer, the user, or both. These factors clearly explain the gap between theoretical peak and real performance as seen from a user's as well as a computer center manager's point of view. The most important property of these reduction factors, however, is the possibility to identify the main sources of the losses and thus to discuss how to remedy them.

For the evaluation of the real performance of a supercomputer we use the following formula:

$$\bar{r}_{real} = \underbrace{\frac{1000}{\tau[ns]} * 2 * P}_{\text{theoretical peak}} * \underbrace{f_1 * f_2 * \cdots * f_N}_{\text{reduction factors}} \; [MFLOPS] \quad (11)$$

$$\eta_g = f_1 * f_2 * \cdots * f_N. \quad (12)$$

The value $1{,}000/\tau$, τ is ns, gives the theoretical peak in MFLOPS of a single pipeline or a superscalar processor for a single floating-point unit that delivers one result per cycle in pipelined mode, the factor 2 accounts for supervector or superscalar speed. P is the total number of arithmetic groups in the whole system. The f_i are the individual reduction factors. The product of the f_i can be interpreted as a global efficiency η_g that reduces the theoretical peak performance to the real performance $\bar{r}_{real}$. It has been explained how the individual losses can be factorized [7]. This factorization makes apparent a highly nonlinear relation between the losses (continued fraction). Therefore the "obvious" factors f_i are not independent but dependent on the order in which they are applied. Thus, they should be defined in the same sequence in which the losses appear in the execution of a program. In the following we discuss models for individual reduction factors f_i.

4.1 Reduction Factors for Single Processor Vector Computers

The factors discussed in this article hold also for a superscalar node processor where a pipeline is simulated by a mixture of hardware and software; the individual factors may be different, the global effect, however, is the same.

For register machines longer vectors are stripmined in sections equal in length to the vector registers. We assume that we need $n * c_{slo}$ wasted cycles for n useful cycles for the section loop organization. Then the corresponding reduction factor is

$$f_{slo} = \frac{n}{n + n * c_{slo}} = \frac{1}{1 + c_{slo}}. \quad (13)$$

Instead of directly giving a value for f_{slo} it can be defined by an easily accessible loss coefficient, in this case c_{slo}, and correspondingly for the following reduction factors. Later we discuss the combined effect of all reduction factors. For this purpose plausible values are assumed for the loss coefficients with the additional purpose of illustrating how these values result in a corresponding reduction factor. Here we assume $c_{slo} = 0.1$, so that $f_{slo} = 0.91$.

A significant cause for the loss of cycles is the memory bottleneck. For a key operation in a program of m_{op} memory references, the computer makes m_{hw} references. For n operations this requires $n * \min(m_{hw}, m_{op})$ cycles to get the $n * m_{op}$ necessary memory references. Thus, the corresponding reduction factor is using the ceiling function $\lceil\,\rceil$

$$f_{mb} = \frac{n}{n\left\lceil\frac{m_{op}}{\min(m_{hw}, m_{op})}\right\rceil} = \frac{1}{\left\lceil\frac{m_{op}}{\min(m_{hw}, m_{op})}\right\rceil}. \quad (14)$$

For the vector triad (Equation 2) $m_{op} = 4$, and on a computer with the (usual) one-word memory bottleneck, $m_{hw} = 1$, giving $f_{mb} = 0.25$.

Memory access problems cause lost cycles, e.g., cache stumbling for the IBM VF or the factors c_s and c_e for the CRAY. If we assume that the price for n useful cycles is $n * c_{map}$ lost cycles for memory access problems, the reduction factor is

$$f_{map} = \frac{n}{n + n * c_{map}} = \frac{1}{1 + c_{map}}. \quad (15)$$

Assume $c_{map} = 0.3$, i.e., $f_{map} = 0.77$.

Another cause for loss of cycles is noncontiguous memory access by indirect addressing, here also the index vectors must be loaded. If we assume the following simplified model, namely, the execution time of an operation is doubled by indirect addressing, a part c of our operations is with contiguous elements and the remainder part 1 − c is with indirect addressing. Then the reduction factor is

$$f_{ia} = \frac{1}{c + 2(1 - c)}. \tag{16}$$

Assume c = 0.9, then $f_{ia} = 0.91$.

Using the finite vector length n together with the half performance length $n_{1/2,\text{eff.mean}}$ leads to the reduction factor

$$f_{hp} = \frac{n}{n + n_{\frac{1}{2},\text{eff.mean}}} = \frac{1}{1 + \frac{n_{\frac{1}{2},\text{eff.mean}}}{n}}. \tag{17}$$

Assume n = 1,000, $n_{1/2,\text{eff.mean}} = 50$, then $f_{hp} = 0.95$.

For a complete program assume that only a part v of the n operations is vectorizable and the part 1-v must be executed in scalar mode, then a reduction factor is the ratio of the reduced performance to the full performance which yields (we omit the intermediate calculation)

$$f_{vp} = \frac{1}{v + (1 - v)w}, \tag{18}$$

where w is the ratio of vector to scalar performance for the key operation. This is Amdahl's law expressed as a reduction factor. For v = 0.95, w = 10 we get $f_{vp} = 0.69$.

At this point we can identify the responsibilities of the manufacturer (m) and user (u) for the losses identified. Define a vectorization efficiency η_v that combines the above reduction factors as part of the global efficiency η_g (Equation 12):

$$\begin{array}{lllllll}
\eta_v = & f_{slo} & * f_{mb} & * f_{map} & * f_{ia} & * f_{hp} & * f_{vp} \\
& \text{sect.} & \text{mem.} & \text{mem.} & \text{indir.} & \text{half} & \text{vector.} \\
& \text{loop} & \text{bottl.} & \text{acc.} & \text{addr.} & \text{perf.} & \text{part} \\
& \text{organ.} & & \text{probl.} & & & \\
= & 0.91 & * 0.25 & * 0.77 & * 0.91 & * 0.95 & * 0.69 \\
& \text{m} & \text{m,u} & \text{m,u} & \text{u} & \text{m,u} & \text{u(respons.)} \\
= & \multicolumn{6}{l}{0.104\ (10.4\%,\ \text{i.e., } 89.6\%\ \text{lost!}).}
\end{array} \tag{19}$$

Thus the accumulated effect of these six reduction factors causes 89.6% loss of performance under quite common assumptions. In the second row of Equation 19 the responsibilities of the manufacturer and of the user are indicated by m and u, respectively. f_{slo} is doubtlessly the responsibility of the manufacturer. The most severe reduction factor f_{mb} is clearly the responsibility of both: The manufacturer is responsible for the hardware bottleneck, but the user is responsible for the choice of operation types that can reuse data in the vector registers, e.g., choosing the matrix multiplication algorithms that have as kernel operation the repeated contracting linked triad (Equation 5). For this operation we get $f_{mb} = 1$, even for the one-word memory bottleneck.

f_{map} is again the responsibility of both: The manufacturer has to design a memory access and bank structure that minimize conflicts and the user has to design algorithms without (even) stride or reusing data in a cache. f_{ia} is the responsibility of the user. He/she has to avoid indirect addressing by presorting of data (the principle of the separation of the selection and of the processing of the data [2]. f_{hp} is again the responsibility of both: The manufacturer has to provide for low startup losses, the user should use long vectors. f_{vp} is the responsibility of the user. He/she has to choose data structures (this is the key) and algorithms that allow for a large vectorizable part.

4.2 Additional Reduction Factors for Parallel Computers

This section discusses the reduction factors for distributed memory/message passing parallel computers. The multiprocessor losses for shared memory parallel computers that result from memory contention can be included into f_{map} (Equation 13), where c_{map} now depends on the number of processors. In the following assume that there are p processors.

For massively parallel computers Amdahl's law is decisive for the efficiency. For a whole program with part q of the operations being parallelizable (fine-grain parallelism) and the remaining part (1 − q) being sequential, the reduction factor is the ratio of the reduced performance to the full performance

$$f_{pp} = \frac{1}{q + (1 - q)p}. \tag{20}$$

This is in complete analogy to vectorization, if we replace v, w in Equation 18 by q, p. To discuss the combined effect of all reduction factors, also for the parallelization, we assume plausible values of the loss coefficients and resulting reduction factors. These values also illustrate how the loss coefficients transform into reduction factors. As an example for Amdahl's law we take $q = 0.98$ and $p = 256$, thus $f_{pp} = 0.16$.

A processor of a distributed memory/message passing parallel computer can operate only with data in its (local) memory. If it needs data from remote memories the necessary communication should be hidden as far as possible "behind" the computation. i.e., executed asynchronously. But there may remain for n useful operations in the mean $n * c_c$ lost operations as a result of waiting for nonoverlapping communication. This waiting is perhaps the most complicated loss in these considerations. On the hardware side it is dependent on the architecture and topology of the communication network, on the startup time (latency) of a message, and on the transfer rate. But it also depends essentially on the problem to be solved and the quality of the parallel software. If the communication can be hidden completely behind the computation, there are no losses at all by communication. This is possible only by asynchronous communication. The other extreme is completely synchronous communication where the processors have to wait during each communication step. Thus n_c is a rather complicated loss coefficient. This gives a reduction factor

$$f_c = \frac{n}{n + n * c_c} = \frac{1}{1 + c_c}. \tag{21}$$

For later use assume $c_c = 0.05$. $f_c = 0.95$.

The next source of losses is load balancing. i.e., coarse-grain parallelism. Processors are usually distributed to users by fixed subarrays, i.e., by space sharing. This is then the user's "PC" (personal computer). Users are responsible for the usage of their processors. If there is a sequential part, all but one processor are idling and performance is lost, leading to a reduction factor

$$f_{lb} = \frac{\sum_{i=1}^{p} t_i}{p * (t_{end} - t_{start})}. \tag{22}$$

where t_i is the time that processor i is busy and t_{start}, t_{end} denote the starting and ending elapsed time of "my" processor array. For later use assume $f_{lb} = 0.8$, i.e., in the mean 80% of my processors are busy. Note that the fine-grain parallelism for a single processor is included in Equation 20.

The last source of losses discussed here is the global utilization of a parallel computer. Because of the necessary space sharing the processors are usually divided into an interactive pool and into a batch pool under the control of a network queuing system (NQS). A user requires a certain number of processors that may not fit into available free groups of processors, thus it will not be possible even by the best administrative measures to distribute all the processors to the users all the time even if there are long batch waiting queues. This results in a reduction factor, e.g., on a weekly base

$$f_{gu} = \frac{\sum_{i=1}^{p} u_i}{p * 168 * 3600} \tag{23}$$

where u_i are the seconds in a week that a processor is dedicated to a user, the denominator is formed by the available "processor seconds" in a week. Assume $f_{gu} = 0.7$, which would be an excellent global utilization. Note that a single user is not aware of f_{gu}.

Combining all reduction factors for parallel computer whose processors are vector or superscalar processors, with Equations 12, 19 for the global efficiency gives

$$\begin{array}{llllll}
\eta_g = & \eta_v & * f_{pp} & * f_c & * f_{lb} & * f_{gu} \\
 & (19) & \text{paral.} & \text{com-} & \text{load} & \text{global} \\
 & & \text{part} & \text{munic.} & \text{bal.} & \text{utilization} \\
= & 0.104 & * 0.16 & * 0.95 & * 0.8 & * 0.7 \\
 & & \text{u} & \text{m,u} & \text{u} & \text{m,u(respons.)} \\
= & \multicolumn{5}{l}{0.00885 \text{ (0.885\%, i.e. 99.115\% lost!).}}
\end{array} \tag{24}$$

The global efficiency η_g now explains the gap between theoretical peak performance and real performance for the whole supercomputer. The expert may not be astonished by this result, which indicates that only 0.885% of the theoretical peak performance is sustained performance, i.e., a loss of 99.115% of the performance of the supercomputer. For 1 TFLOPS theoretical peak this means 8.85 GFLOPS sustained performance over a week.

The responsibilities for the parallel processing reduction factors are indicated below the second row of Equation 24. Here the most severe reduction factor f_{pp} for Amdahl's law ultimately means that massive parallel processing is reasonable only if a sufficiently large part q is parallelizable and thus f_{pp} is the sole responsibility of the user. The meaning of this may be that the problem to be solved has to be large enough so that the sequential part is small in relation to the parallel part. The reduction factor f_c for communication is the responsibility of both manufacturer and user: The manufacturer has to ensure by hardware and software that there is high communication bandwidth and a reasonable ratio of computation speed to communication speed. The user has to design data structures and algorithms that reduce communication and allow hiding of communication behind computation. The reduction factor f_{lb} for load balancing is clearly the responsibility of the user. The user has to ensure that all processors of a partition are continuously busy. The final factor f_{gu} for global utilization of the processor array is again the responsibility of both: The manufacturer has to design hardware and software for a flexible allocation of processors to a user and the manager of the computer center (not the individual user) has to apply adequate administrative measures that allow an efficient use of the whole computer without too severe restrictions for the individual user.

This explains why the performance of a supercomputer may be so poor for a particular problem. The benefit of this knowledge should be to investigate each factor that is under the user's influence and analyze how it can be improved. The consequence of such an investigation may well be that this type of computer is not suited for the problem or for the existing software.

4.3 The Price/Performance Relationship

For economic reasons it is important to know the price/performance relationship of different supercomputers. This is dependent on the price and the performance of the job profile. Measurements by means of a benchmark can provide the answer for the performance only if we can project future use of a program package, the components of which are fully adapted to the different architectures to be measured. Thus the performance will depend significantly on the quality of the software. Therefore, we have considered the simplified question: "How many millions of DM (Deutschmark, German currency) per year do I have to pay in order to obtain 1 GFLOPS of vector triad sustained over the year?" [10]. As the vector triad has a close correspondence to the performance ratio of engineering benchmarks, the answer may be quite realistic. It can be adapted more accurately if we consider in detail the reduction factors explained above.

The answer to the question based on the situation in January 1991 is presented [10] and is summarized below. The CRAY Y-MP 8/8128 and Fujitsu VP 2600/10 are in the range of 7 MDM/year (million DM per year), the NEC SX-3 Model 24 in a special version with the memory bandwidth of the Model 44 is 3.5 MDM/year (the standard model 24 is again 7). The IBM ES9000/720 (14.5 ns) is 70 MDM/year, the Convex C240 is 32 MDM/year. The IBM RS6000 workstation is 18 to 56 MDM/year depending on f_{gu} 0.75 to 0.24. For the NCUBE2, Model 10 (512 processors) the range is 3 to 22 MDM/year for f_{gu} 0.75 to 0.1 and for the Intel iPSC/860 Model 128 (128 processors) the range is 4 to 31 MDM/year for f_{gu} 0.75 to 0.1. This means that in January 1991 general purpose computers with vector add-on units, mini supercomputers, and workstations were expensive number crunchers and that massively parallel computers could only compete with vector supercomputers with high global utilization. Much has changed since January 1991 and will be changing continuously. Each investigation will be obsolete within a few months. This work is presented as an exemplary investigation and a new investigation is required when we are planning to purchase a new supercomputer. We have renounced to transform the DM into US$ because this relation also changes continuously. What is important in this investigation are only the relations, i.e., a relative price in whatever scale.

5 AN IDEAL SUPERCOMPUTER ARCHITECTURE

"What should the ideal supercomputer look like?" The basic ideas for a user-friendly supercomputer have been presented [12] where user-friendly means that the architecture of a supercomputer is designed in such a way that the bottlenecks of the hardware of present supercomputers disappear. A programmer's task would then be much easier because the problems that today have to be solved at the software level would have been solved at the hardware level. Cycles that are lost at the hardware level can never be regained at the software level. Thus a user-friendly supercomputer achieves under rather general conditions a sustained real performance close to the theoretical peak performance. Assume a 1-TFLOPS theoretical peak performance is required. In the context of the discussion of Table 1 we must wait until we can afford 64 Mbit chips to have a 1-TFLOP supercomputer with adequate memory. This will be possible from 1995 on. With ECL technology in the near future 2 ns cycle time will be possible, thus for supervector speed 1,000 pipe groups are required, i.e., a 1,000-fold parallelism. The question is how to organize this parallelism. From these investigations and measurements of many supercomputers for high arithmetic performance only pipelines can be used. Microprocessors with their caches and explicit programming of vector operations will not be able to compete for sustained high performance.

For the user a monoprocessor with 1,000 pipe groups would be ideal, but the present technology does not permit the construction of such a computer. Therefore we have to split the 1,000 pipe groups into, e.g., 16 processors, each with a bundle of 64 pipe groups. The basic ideas of how to design such a processor with 64 pipe groups have been presented [12]. The main requirements are a main memory bandwidth of three loads and one store per cycle and pipe group, across the bundle of 64 pipes it is single instruction multiple data (SIMD); over the operations executed in the bundle of pipes it is multiple instruction multiple data (MIMD) with a sufficient number of control units: for initial or final results it is memory-to-memory, for intermediate results it is register-to-register with 90% of the total memory extended memory, but with one-word-per-cycle-and-pipe-group transfer rate to the main memory. A proposal has been made on how to organize the main memory and corresponding buses and also how to access data in such a memory using indirect addressing. This would give an ideal 64-GFLOPS supercomputer that can come close to this performance for real applications. The internal parallelism is transparent to the user. Nevertheless such a computer will deliver high performance only if the problems to be solved are sufficiently large and suited for pipelining. The user has to design the data structures for a continuous flow of data through the bundle of pipelines. Any other use of such a supercomputer would be inefficient.

For 1 TFLOPS we have to combine 16 of these processors using external parallelism. For efficient utilization the communication bandwidth must be the same as that of the extended memory, i.e., one word per cycle and pipe group. The interconnect network must be a crossbar or a pipe switch with multistage pipelined register transfers [12]. Thus in principle an external main memory must be accessible like the local extended memory (the bus to the local extended memory should be used also for this purpose). Then one operand per cycle and pipe group can be obtained either from the local extended or from a remote main memory. The topology of the communication network must be sufficiently simple to allow such extremely high transfer rates. Most probably a two-dimensional mesh or torus would be the best solution for the 16 processors.

The programming model should be explicit message passing, not a virtual global shared memory although there may be a global address space by processor number and local address as for example the Meiko CS2. The user has to know and to determine which data is where at what time if high performance is required. An idling processor can be used in a multiprogramming style for other jobs, but there should be preferably only one job with maximum user priority. If it has terminated, the next job gets this priority. All other jobs execute with very low priority so that they do not interfere with the "priority job."

The most critical components of this "ideal" supercomputer are the memory bandwidth and communication bandwidth. Much more research and development cost should be devoted to this area. "Who is going to build this computer?" Unfortunately such a supercomputer will never be built because the necessary development cost must be raised by one manufacturer, whereas a multiple of that cost to develop software for poor architectures is dispersed over thousands of institutions and does not become visible in a proper way and thus is paid without complaint. And those people who earn this futile money are clearly delighted with this situation. Therefore "cheap" supercomputers are put together from mass production processors and off-the-shelf technology. What we get then are TFLOPS supercomputers that may deliver 9 GFLOPS as explained in the context of Equation 24.

6 CONCLUDING REMARKS

The purpose of a supercomputer is the very fast processing of very large amounts of data by its arithmetic units. Unfortunately, between the theoretical peak performance—if all data were available immediately—and the real performance is a large gap because there are severe bottlenecks in the transfer of data to and from the arithmetic units. We have discussed the basic architecture of vector processors and microprocessors and of parallel computers that are combined from these building blocks, together with their bottlenecks. In addition we have discussed the memory references for triadic operations, and seen that there is a factor of 4 in references between the general triad and the contracting linked triad. The former is typical for engineering applications, the latter is typical for matrix multiplication and the solution of linear equations.

We have presented micromeasurements that show up all the lost cycles of a distinct operation, such as the general triad and shown how the one-word memory bottleneck and cache stumbling reduce the performance for an IBM 3090 VF. The larger memory bandwidth and the cache-free memory access of the CRAY Y-MP result in a much smaller performance loss. We have defined a waste factor W and architectural efficiency $\eta_a = 1/W$ that illustrate these losses. Hockney's half performance length $n_{1/2}$ results in a natural way from these investigations.

For the estimate of the global efficiency of a supercomputer for a complete program or even throughput we identify and factorize the losses that reduce the theoretical performance to the real performance. This is presented for a vector supercomputer—it could be done in a similar way for a microprocessor-based computer. Then the losses for a distributed memory parallel computer are discussed. Using quite plausible assumptions of the loss coefficients and the resulting reduction factors we find that such a supercomputer may deliver a sustained rate of only 0.885% of its theoretical peak performance, i.e., a 1-TFLOPS supercomputer would deliver a ridiculous 9 GFLOPS. The main purpose of this analysis is so that the manufacturers and the users may recognize where performance is lost and take the corresponding measures to reduce the losses.

This was followed by consideration of the price/performance relationship for different types of supercomputers such as workstations, general purpose computers with vector units, classical vector computers, and massively parallel computers as of January 1991. This has led to the discussion of an ideal supercomputer—ideal that is from a user's point of view. Avoiding the bottlenecks of present supercomputers means keeping the data flowing on "data highways" to and from the arithmetic units. A proposal is then made on how this could be achieved for a 1 TFLOPS supercomputer with sustained performance close to the theoretical peak performance. Finally it is explained why such a supercomputer will never be built.

ACKNOWLEDGMENTS

The authors wish to express their sincere gratitude to the three anonymous referees for their valuable comments.

REFERENCES

[1] Kendall Square Research. Company and Product Overview, Kendall Square Research. Waltham. Massachusetts. USA. July 1992.

[2] W. Schönauer, *Scientific Computing on Vector Computers.* Amsterdam: North-Holland. 1987.

[3] J. J. Dongarra. "Performance of various computers using standard linear equations software. Computer Science Department. University of Tennessee. CS-89-85. 1993.

[4] W. Schönauer and H. Häfner, "A critical discussion of supercomputer architectures for engineering applications," in *Applications of Supercomputers in Engineering II, Computational Mechanics Publications, C. A. Brebbia, D. Howard, and A. Peters, Eds.* London: Southampton and Elsevier Applied Science. 1991. pp. 151–162.

[5] W. Schönauer and H. Häfner. "Supercomputers: Where are the lost cycles?" in *Proceedings of the 1991 ACM International Conference on Supercomputing.* New York: ACM Press. 1991. pp. 353–364.

[6] H. Häfner and W. Schönauer. "Micro-measurements of a supercomputer and models for memory contention." in *High Performance Computing II.* Amsterdam: North-Holland. 1991. pp. 169–180.

[7] W. Schönauer and H. Häfner. "Performance estimates for supercomputers: The responsibilities of the manufacturer and of the user." *Parallel Comput.* vol. 17. pp. 1131–1149. 1991.

[8] R. Hockney. "Performance parameters and benchmarking of supercomputers." in *Computer Benchmarks.* Amsterdam: North Holland. 1993. pp. 41–63.

[9] R. Hockney and C. Jesshope. *Parallel Computers 2.* Bristol: Adam Hilger. 1988.

[10] W. Schönauer and H. Häfner, "Supercomputers: The hardware, the architecture," in *Proceedings of the 13th IMACS World Congress on Computation and Applied Mathematics.* New Brunswick: IMACS, 1991, pp. 725–727.

[11] J. J. Dongarra and W. Gentzsch. *Computer Benchmarks.* Amsterdam: North-Holland. 1993. pp. 77–95.

[12] W. Schönauer and R. Strebler. "Could user-friendly supercomputers be designed?" in *Scientific Computing on Supercomputers II.* New York: Plenum Press, 1990. pp. 99–122.