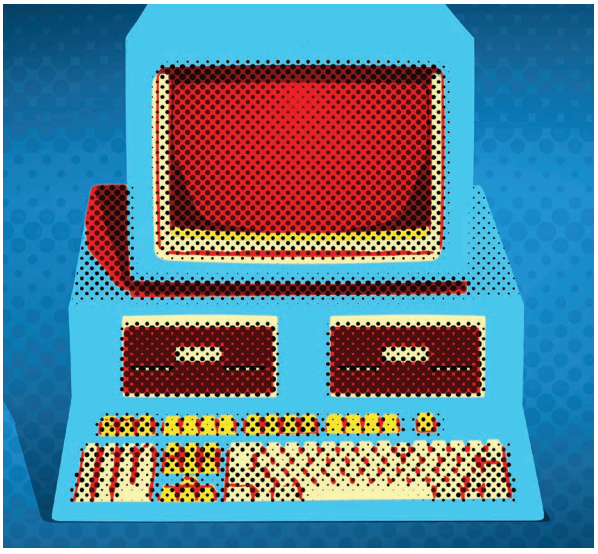

Computer Architecture and Amdahl's Law

Gene M. Amdahl, *Amdahl Corporation*

An autobiographical overview of Gene Amdahl's professional career reviews his early achievements and describes for the first time some important details of his talk at the 1967 Spring Joint Computer Conference in Atlantic City and the heated discussions that followed. The author provides much-needed clarifications, most notably his original performance formula as presented nearly 50 years ago.

My educational background has never included any training in the field of computing, so all of my design activities have been based on my experience and the necessity of solving current problems. Consequently, my computer architecture contributions are largely autobiographical.

I was raised on a farm in eastern South Dakota and attended a one-room grade school for all eight grades. We didn't have electricity until I was a freshman in high school, so my technical experience was limited primarily to mechanical equipment. I enrolled in South Dakota State College (SDSC) in the fall of 1941 in mechanical engineering, but decided it was not the field for me, opting instead for a potpourri of courses in math, chemistry, electrical engineering, and physics. World War II intervened early in my freshman year, and I joined the Navy as an electronic technician, teaching electronics. Returning to SDSC in 1947, I selected physics as my major, graduating in 1948.

I received a Wisconsin Alumni Research Foundation assistantship and began that summer in the field of theoretical physics at the University of Wisconsin in Madison. This was an unusual time in physics, for "strange particles" had just been discovered in late 1949, and the name "meson" had not yet been proposed. At that time, two other graduate students and I were assigned to determine if a force between nuclear particles that had been proposed by a Japanese physicist could adequately describe the simplest three-body nucleus, tritium (hydrogen-3). We worked for 30 days using an eight-digit desk calculator; a slide rule held two more most-significant digits. We mapped the energy of the system for all relevant ranges of the parameters, but we couldn't quite achieve a stable state. We had found the proposed force to be inadequate, but the means of calculating were even more inadequate! I then began to think about how the computing could be done better. The university had no information on computers in its library, no courses in computing, and no computers, apart from an electronic analog computer in the electrical engineering department.

THE WISCONSIN INTEGRALLY SYNCHRONIZED COMPUTER

My major professor, Robert Sachs, recognized my dilemma and arranged for me to get a two-month summer job in 1950 at the Aberdeen Proving Ground in Maryland. My assignment there was to program supersonic flow about a three-dimensional body. The instruction set was that of the EDVAC (Electronic Discrete Variable Automatic

 Published by the IEEE Computer Society

Computer), then under development.[1] I wasn't given any introduction to programming or to the structure of the computer, nor did I complete the programming during the two-month period. I heard that the development of the EDVAC was dropped because the mercury delay line was unstable due to temperature buildup when operating.

I was not enamored of the EDVAC structure because the use of fixed point with a limited word length required a lot of rescaling to maintain reasonable precision. As I returned to Wisconsin, I formulated a three-address floating-point structure, trying to make it as simple as possible and to use technologies that were commercially available. I chose a magnetic drum for main storage, with recirculating registers to minimize the use of electronics. For I/O, I planned to use paper tape with a teletypewriter, which could both punch and read paper tape and print as well.

I determined that I could use floating point exclusively if I had a way to deal with the transfer of word segments from one word to another. The three-address operation that I came up with was *Extract*, which took *n* bits, beginning at bit *j* in word one, inserted those bits starting at bit *k* in word two, the result to be stored in location three. This eliminated the need for approximately a dozen instructions in fixed point.

The complete instruction set consisted of 10 instructions: *Add*, *Subtract*, *Multiply*, *Divide*, *Compare* (and *Transfer* if the difference is zero or negative), *Transfer*, *Extract*, *Read-in*, *Read-out*, and *Halt*. *Read-in* and *Read-out* were very different from any I/O operations I observed for several years following this, until I planned the design of my second computer at IBM in 1955, the IBM 709, where I introduced the I/O channel. *Read-in* and *Read-out* specified the information source or sink, beginning at a specified point in the source or sink and at a specified location in the drum storage and continuing until completing the final specified location. The *Read-in* and *Read-out* instructions were executed concurrently and independently of computational operations. This overlap of I/O with computing was a major contributor to performance enhancement.

The magnetic drum had sufficient capacity to provide 32 tracks of storage, each containing 32 words of 50 information bits and a 5-bit-length space for track-switching time, for a total track length of 1,760-bit times. The 50-bit word was made up of 40 bits of numeric fraction: 8 bits of exponent plus 1 bit for exponent sign, and 1 bit for the sign of the fraction. The arithmetic was performed on numeric fractions by recirculating the fractions in recirculating registers, while the exponents and signs were retained in electronic registers for control purposes. The recirculating registers had the read and write heads spaced 44 bits apart: 40 bits for the fraction and 4 bits for switching time. With this spacing, the fraction would have 40 repetitions in a drum revolution, matching precisely the 1,760-bit times in the revolution.

Each of the arithmetic operations was performed in the course of one drum revolution. I thought I had invented a new way of performing division by considering the numerator fraction to be the initial value of the remainder, subtracting the denominator fraction from the remainder, adding a 1 in the leftmost quotient digit position, and then shifting the denominator fraction one bit position to the right, preparing for repetition. If at any stage of repetition the remainder became negative, the denominator fraction would be added to that remainder instead of subtracted, and a 1 would be subtracted in the corresponding quotient position rather than added. I later heard that John von Neumann had patented it.

Each operation took one drum revolution to be certain about the instruction calling for it was acquired, then a second revolution to be certain the operands were acquired, then a revolution to perform the operation, and finally a revolution to be certain the result had been stored. Since the operations were nonconflicting, there were four instructions in the pipeline at all times, one picking up its instruction, one picking up its operands, one performing its operation, and one storing its result. Consequently, the computer performed one floating-point operation per drum revolution. I believe there were several world firsts in that design: the first electronic computer to have floating-point arithmetic (and certainly the first to have only floating-point arithmetic), the first electronic computer to have pipelining, and the first electronic computer to have I/O operated concurrently and independently of computing!

The overlap of I/O with computing was a major contributor to performance enhancement in the design of IBM 709.

I told one of my fellow physics students about my computer design ideas, and he apparently was excited enough to pass the information on to the electrical engineering department, and in the late fall 1950, I was asked to give a lecture on my design ideas. I gave a seminar, and about a week later, the head of electrical engineering called my major professor and asked him to change the subject of my doctoral thesis to be a record of my computer design plan so that graduate engineers could build it and be trained in this new field. My major professor agreed, and I spent six months writing the new thesis and ordering the magnetic drum. I submitted my thesis in June 1951, expecting to graduate later that month. But no one at the university felt competent to properly evaluate it, so it was sent to scientists at the Aberdeen Proving Ground for evaluation. They approved, and I graduated the following February. The thesis was titled, "The Logical Design of an Intermediate Speed Digital Computer," and I named the computer the

WISC (Wisconsin Integrally Synchronized Computer). It was completed in 1955 and is now displayed in the Computer History Museum in Mountain View, California.

COMPUTER ARCHITECTURE WORK AT IBM

A copy of my thesis was apparently obtained by the IBM branch manager in Milwaukee and sent to IBM Poughkeepsie. Nathaniel Rochester read it and had IBM make me an offer to join the company in Poughkeepsie. I accepted and joined IBM in June 1952. My initial assignment was to simulate neural networks on the IBM 701, according to the proposed characteristics in a monograph published by Donald Hebb. I worked on it for several months and concluded that the description was inadequate. I then turned my attention to character recognition and had considerable success, even on the crude characters of wire printing.

The new "look ahead" concept consisted of fetching instructions well in advance of their execution time in order to take early actions about future branch instructions.

The 701 exhausted its market after the sale of 18 computers. The company decided that a follow-on computer, the 704, should be developed, utilizing the new magnetic core memory rather than the cathode-ray tube memory in the 701, for the capacity could be much larger. I was given the task of designing it because the other experienced IBM designers were about to be committed to a joint project with MIT to develop and produce the SAGE (Semi-Automatic Ground Environment) system.[2] I decided to double the instruction size in order to accommodate a larger address and additional instructions to provide floating-point arithmetic as well as the fixed-point arithmetic of the 701. I had heard of an English computer having a "B-box," a counter that allowed the repetition of a loop until the count reduced to zero. Any address step-changing in an array for each iteration still required separate instructions. I thought it would be more efficient if the count and the step size could be combined; then the program could be shorter and faster. I called it indexing and put three index registers in the 704 to accommodate different step sizes for different data arrays. I assigned two bits in the instruction to identify no indexing and which of the three index registers to use in this instruction. I also discovered that index register contents could be available early enough to modify the address in this instruction before fetching the data, thus having no additional execution time! It turned out that the SAGE system also had an indexing capability, but I don't know who had it first; that was a classified project, and I wasn't cleared, so I had no knowledge of it to get the dates of invention.

When the time came to price the 704 for the market, it was necessary to estimate the probable market size. Pricing people from IBM headquarters initially estimated a market of six machines (I assume they considered that the 18 701 machines had mostly satisfied the market). I was incensed and insisted that the machine had so much more capability than the 701 that it would have a larger market size. Over the next few weeks, they came back with 12, then 18, and finally 32 before I agreed. The actual number sold was 140, making it an extremely profitable program!

I was then asked to design the follow-on system, the 709, and shortly thereafter to design a supercomputer (called Stretch) to utilize a new technology: transistors. I was told it would be my project, but that I would have to get a development contract, preferably from Lawrence Livermore Labs or Los Alamos National Laboratory. I consulted a bit with John Backus in November 1954, and we agreed on the principal characteristics that Stretch should possess. I studied the capabilities of the proposed semiconductor technology, which was a new circuit type called ECL (emitter-coupled logic), sort of like the vacuum tube push-pull amplifiers. They were extremely fast circuits and very power consuming. I did some designing of a multiplier to estimate the probable performance that could be achieved if efficiently instructed. I then worked on a new concept, "look ahead," which consisted of fetching instructions well in advance of their execution time so that branch instructions could be recognized early enough to fetch an alternative instruction sequence with no delay. The design analysis was very promising, yielding several times the performance we could achieve using vacuum tubes.

Armed with my initial design results, I visited the team at Livermore first. They listened and were very cordial, but they informed me that they had already committed to contract with a competitor so they couldn't work with us. I then visited a group at Los Alamos that was very interested and would negotiate with us. I then did some more fleshing out of the Stretch design and also determined what should be done to the 704 to produce a 709. In the 709 project, I added a number of reasonably useful new instructions, one of the most interesting ones being a "history-dependent table lookup" that allowed code conversions from BCD (binary-coded decimal), IBM's preferred code, to ASCII, the newly adopted American Standard Code for Information Interchange, or vice versa. It also allowed two binary-coded decimal numbers (each decimal digit occupying a 6-bit character position) to be added or subtracted in binary, using the table lookup to convert the result to a proper BCD result. These were two examples, but many more were ultimately developed by customers.

The principal change I wanted to make was the introduction of an I/O channel, permitting the computer to specify the reading or writing of a specified amount of data to or from a magnetic tape or drum into or out of

memory without the computer having to control the data flow as it occurred, just as I did in the WISC, but also be able to continue computing with only the impact of some memory cycle delays due to conflict of memory requests. This change was a significantly costly development that required corporate approval. Elaine Boehm and I determined that we had to make an outstanding demonstration to win approval. We came up with the idea of a tape-sorting program. The IBM 703 was a sorter-collator, a fairly modestly priced machine sold to the US Department of the Treasury. We estimated the expected price for the 709 to be at least two or three times that of the 703. We programmed the sort and found that it performed so much faster than the 703 that the cost of sorting on the 709 was less than on the 703. This demonstration tipped the balance, and the I/O channel's development was approved.

At this time (mid-1955), I was surprised to find someone assigned to my Stretch project. I initially assumed he reported to me, but it became clear that he thought I reported to him. This was very disconcerting, for I had been assured that Stretch was my project before I accepted the assignment, and I had already done quite a bit of the design. This new man was uninterested in my design and had his own approach: he wanted to design a front-end computer (a commercial computer) feeding the back end (a scientific computer). To me, this seemed to prevent any possibility of creating the supercomputer that I was commissioned to design. Late that year, I was invited to meet with the laboratory manager, who showed me a plan for restructuring the lab that had several development projects feeding the technology engineering groups. The Stretch development was to be managed by the man assigned to me a few months earlier, and I was to be in charge of the Stretch detailed design. I was appalled, for I knew we could never agree, and the project would fail. I didn't respond with my reaction; I just went back to my office and wrote my letter of resignation. I did continue on until just before the end of the year, providing my best design ideas, all of which were lost, then left for South Dakota for Christmas with my extended family, and then on to Los Angeles to join Ramo Wooldridge's computer division.

Almost five years later, Emanuel Piore, IBM's chief scientist, who reported directly to Tom Watson, came to Los Angeles and invited my wife and me to dinner. He offered me the position of managing the Experimental Machines Division at IBM Research, with a requirement to be on the East Coast for a minimum of four months and a maximum of seven. My wife and I accepted and were back in New York State by November 1960. My first activities were to look at the projects in my division. I cancelled the only two hardware design ones because they had no chance of being of value to IBM. One project was a computer design that had been continually changed but was never complete enough to be evaluated; the other was a government project that utilized superconductor switches for logic, but there was no way to amplify diminishing signal levels. This left only software projects, which I retained, and the responsibility for designing a new supercomputer with insignificant funding. I believe I was given this because the Stretch project had not met its performance target, so its price had been reduced to become a loss leader. The 704, 709, and the follow-on 7090 and 7094 sustained IBM's scientific computing market.

At this time, IBM also had the SPREAD (systems programming, research, engineering, and development) committee in session: five major computer families made by various IBM divisions, each of which had generations that weren't quite compatible. Unfortunately, total development costs were growing impossibly large because any new device to be attached required that an engineering and software project be manned and funded for each member of each family. IBM's development budget was greater than most computer companies' revenue!

The System 360 family is still IBM's mainframe line and largest revenue product.

The SPREAD committee's goal was to define the data formats, I/O devices, and control, storage, and logic technologies to be standardized and to plan a new family of computers that would replace all current families. This was not only an enormous undertaking, but it was even more of a political undertaking, for it would require all of the divisions to yield their fiefdoms to the king! A major revolution was being fought, but the stakes were high: managing the costs to maintain control of the world's data processing marketplace.

I had only been back for a couple of months when I was approached by the president of the Data Systems Division, Bob Evans, to meet with him at a budgeting session that amply demonstrated the development budgeting problem. After that session, Bob and I met privately, and he asked me if I would consider designing the new family of computers. I asked him if it would be more upwardly compatible than downwardly, and he said that was the plan. I replied that I would not be willing to do that, for it would only end up creating the same budget problem that I had just observed because the generational problem would exist immediately. I told him I thought the family could be both upwardly and downwardly compatible and with virtually no cost impact; if IBM would enforce this constraint, I would be willing to accept the challenge. Bob thought for a moment, presuming I could possibly do it, and agreed. So in 1961, I moved back to Poughkeepsie, where I worked for about 10 hours a day defining data formats, the instruction set, and in several cases, the hardware structure. Each

family member was to have about a factor of three difference in performance from its neighboring members, which required registers in the smaller machines to be memory locations, but in larger machines, to be in circuitry. This System 360 family had a total of seven machines, covering a performance range of about 600 to 1. It is still the company's mainframe line, though changed through the decades, and is IBM's largest revenue product.

During this time, I had access to tape storage programs and data history for commercial, scientific, engineering, and university computing centers for the 704 through the 7094. This gave insight on relative usage of the various instructions and a most interesting statistic—each of these computing center workload histories showed 1 bit of I/O for each instruction executed! I also was able to determine the speed of computing that could be maintained for a given memory size. This was related to disk and tape speeds in a multiprocessing environment. These latter two properties I determined in 1969, when I privately estimated that System 360 would have to change the address length to exceed about 15 MIPS (million instructions per second). Researchers at Lawrence Livermore heard about the 1 bit of I/O and thought it couldn't be true, so they ran tests for a month and found that during office hours, when users were using the machines from their consoles, the number of bits of I/O averaged 1.1, and at night doing batch processing, the average was 1.0. They were surprised, but neither they nor I knew why it should have that value. When virtual

Validity of the Single-Processor Approach to Achieving Large-Scale Computing Capabilities

Gene M. Amdahl

Reprinted from the *AFIPS Conference Proceedings*, vol. 30, AFIPS Press, 1967, pp. 483–485. This article was the first publication by Gene Amdahl on what became known as Amdahl's law. Interestingly, it has no equations and only a single figure. For completeness, we print this historic paper together with the more recent core article. –eds.

For over a decade, prophets have voiced the contention that the organization of a single computer has reached its limits and that truly significant advances can be made only by interconnection of a multiplicity of computers in such a manner as to permit cooperative solution. Variously the proper direction has been pointed out as general-purpose computers with a generalized interconnection of memories, or as specialized computers with geometrically related memory interconnections and controlled by one or more instruction streams.

Demonstration is made of the continued validity of the single-processor approach and of the weaknesses of the multiple-processor approach in terms of application to real problems and their attendant irregularities.

The arguments presented are based on statistical characteristics of computation on computers over the last decade and upon the operational requirements within problems of physical interest. An additional reference will be one of the most thorough analyses of relative computer capabilities currently published—"Changes in Computer Performance," *Datamation*, September 1966, Professor Kenneth E. Knight, Stanford School of Business Administration.

The first characteristic of interest is the fraction of the computational load that is associated with data-management housekeeping. This fraction has been very nearly constant for about 10 years, and accounts for 40 percent of the executed instructions in production runs. In an entirely dedicated special-purpose environment, this might be reduced by a factor of two, but it is highly improbable that it could be reduced by a factor of three. The nature of this overhead appears to be sequential so that it is unlikely to be amenable to parallel processing techniques. Overhead alone would then place an upper limit on throughput of five to seven times the sequential processing rate, even if the housekeeping were done in a separate processor. The non-housekeeping part of the problem could exploit at most a processor of performance three to four times the performance of the housekeeping processor. A fairly obvious conclusion that can be drawn at this point is that the effort expended on achieving high parallel processing rates is wasted unless it is accompanied by achievements in sequential processing rates of very nearly the same magnitude.

Data-management housekeeping is not the only problem to plague oversimplified approaches to high-speed computation. The physical problems that are of practical interest tend to have rather significant complications. Examples of these complications are as follows: boundaries are likely to be irregular; interiors are likely to be inhomogeneous; computations required may be dependent on the states of the variables at each point; propagation rates of different physical effects may be quite different; the rate of convergence, or convergence at all, may be strongly dependent on sweeping through the array along different axes on succeeding passes; and so on. The effect of each of these complications is very severe on any computer organization based on geometrically related processors in a paralleled processing system. Even the existence of regular rectangular boundaries has the interesting property that for spatial dimension of N there are 3^N different point geometries to be dealt with in a nearest-neighbor computation. If the second nearest neighbor were also involved, there would be 5^N different point geometries to contend with. An irregular boundary compounds this problem as does an inhomogeneous interior. Computations that are dependent on the states of variables would require the processing at each point to consume approximately the same computational time as the sum of computations of all physical effects within a large region. Differences or changes in propagation rates may affect the mesh point relationships.

Ideally, the computation of the action of the neighboring points upon the point under consideration involves their values at a previous time proportional to the mesh spacing and inversely proportional to the propagation rate. Since the time step is normally kept constant, a faster propagation rate for some effects would imply interactions with more distant points. Finally, the fairly common practice of sweeping through the mesh along different axes on succeeding passes poses problems of data management that affects all processors, however, it affects geometrically related

memory came into common usage, the number of bits of I/O per instruction executed came down. Although I had limited data, I could reasonably estimate that it correlated quite closely with the reduction of the percentage of the program size, which hadn't needed to be brought in or retained during the course of its execution.

AMDAHL'S LAW

In 1967, I was asked by IBM to give a talk and present a paper[3] at the Spring Joint Computer Conference to be held on the East Coast. The purpose was for me to compare the computing potential of a super uniprocessor to that of a unique quasi-parallel computer, the ILLIAC (Illinois Integrator and Automatic Computer) IV,[4] presented at the conference by Daniel Slotnick.

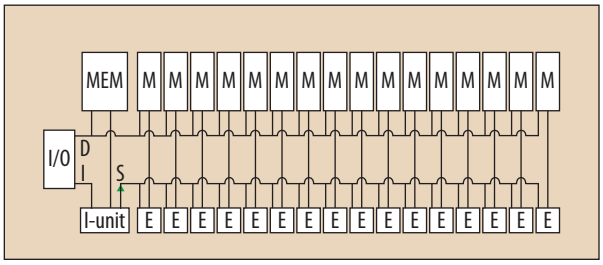

Figure 1. A diagram of the ILLIAC IV.

As shown in Figure 1, the proposed ILLIAC IV had a single instruction unit (I-unit) driving 16 arithmetic units (E-units). Each E-unit provided its own data addresses and

processors more severely by requiring transposing all points in storage in addition to the revised input-output scheduling. A realistic assessment of the effect of these irregularities on the actual performance of a parallel processing device, compared to its performance on a simplified and regularized abstraction of the problem, yields a degradation in the vicinity of one-half to one order of magnitude.

To sum up the effects of data-management housekeeping and of problem irregularities, the author has compared three different machine organizations involving approximately equal amounts of hardware. Machine A has 32 arithmetic execution units controlled by a single instruction stream. Machine B has pipelined arithmetic execution units with up to three overlapped operations on vectors of eight elements. Machine C has the same pipelined execution units, but initiation of individual operations at the same rate as Machine B permitted vector element operations. The performance of these three machines is plotted in Figure A as a function of the fraction of the number of instructions that permit parallelism. The probable region of operation is centered around a point corresponding to 25 percent data management overhead, and 10 percent of the problem operations are forced to be sequential.

The historic performance versus cost of computers has been explored very thoroughly by Professor Kenneth E. Knight. The carefully analyzed data he presents reflects not just execution times for arithmetic operations and cost of minimum of recommended configurations. He includes memory capacity effects, input-output overlap experienced, and special functional capabilities. The best statistical fit obtained corresponds to a performance proportional to the square of the cost at any technological level. This result very effectively supports the often invoked "Grosch's law." Utilizing this analysis, one can argue that if twice the amount of hardware were exploited in a single system, one could expect to obtain four times the performance. The only difficulty is involved in knowing how to exploit this additional hardware. At any point in time it is difficult to foresee how the previous bottlenecks in a sequential computer will be effectively overcome. If it were easy, they would not have been left as bottlenecks. It is true by historical example that the successive obstacles have been hurdled, so it is appropriate to quote the Rev. Adam Clayton Powell—"Keep the faith, baby!" If alternatively one decided to improve the performance by putting two processors side by side with shared memory, one would find approximately 2.2 times as much hardware. The additional two-tenths in hardware accomplish the crossbar switching for the sharing. The resulting performance achieved would be about 1.8. The latter figure is derived from the assumption of each processor utilizing half of the

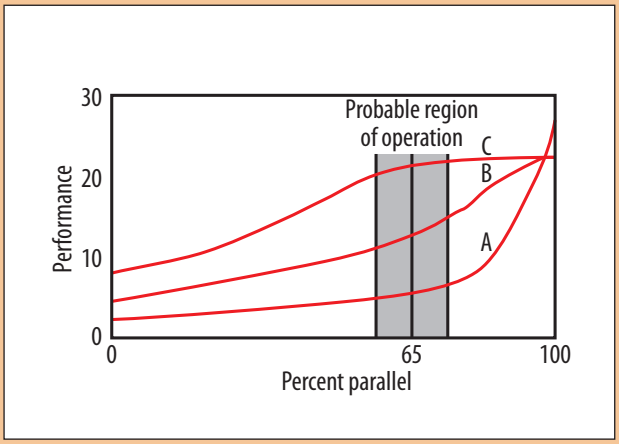

Figure A. Performance comparison of three different machine organizations.

memories about half of the time. The resulting memory conflicts in the shared system would extend the execution of one of two operations by one-quarter of the execution time. The net result is a price performance degradation to 0.8 rather than an improvement to 2.0 for the single larger processor.

Comparative analysis with associative processors is far less easy and obvious. Under certain conditions of regular formats, there is a fairly direct approach. Consider an associative processor designed for pattern recognition, in which decisions within individual elements are forwarded to some set of other elements. In the associative processor design, the receiving elements would have a set of source addresses that recognize by associative techniques whether or not they were to receive the decision of the currently declaring element. To make a corresponding special-purpose non-associative processor, one would consider a receiving element and its source addresses as an instruction, with binary decisions maintained in registers. Considering the use of thin-film memory, an associative cycle would be longer than a non-destructive read cycle. In such a technology, the special-purpose non-associative processor can be expected to take about one-fourth as many memory cycles as the associative version and only about one-sixth of the time. These figures were computed on the full recognition task, with somewhat differing ratios in each phase. No blanket claim is intended here, but rather that each requirement should be investigated from both approaches.

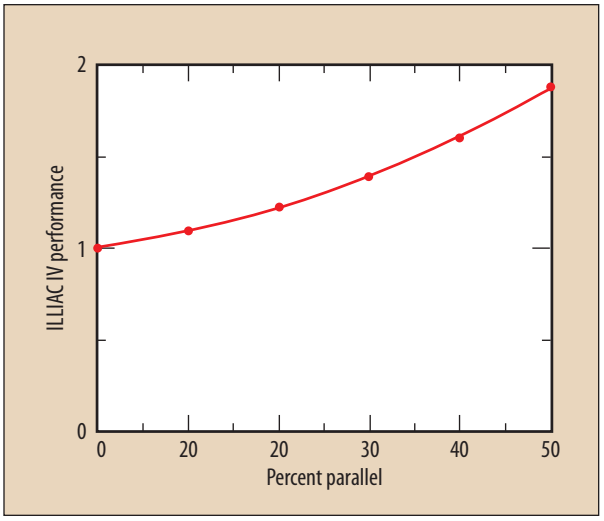

Figure 2. ILLIAC IV's performance on a problem having a varied, but reasonable, range of parallelism.

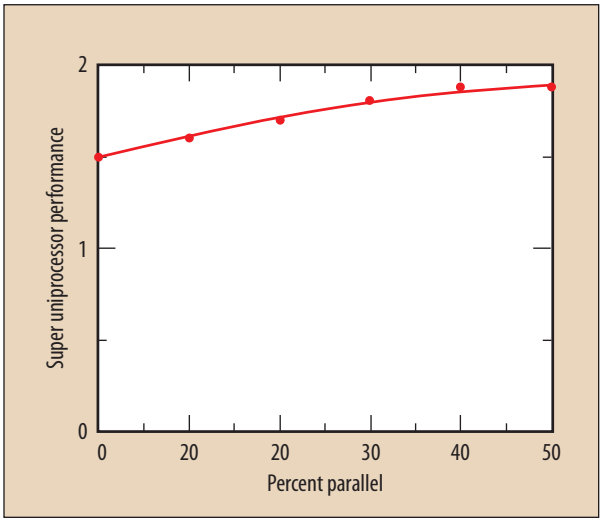

Figure 3. Super uniprocessor performance on a problem having a varied, but reasonable, range of parallelism.

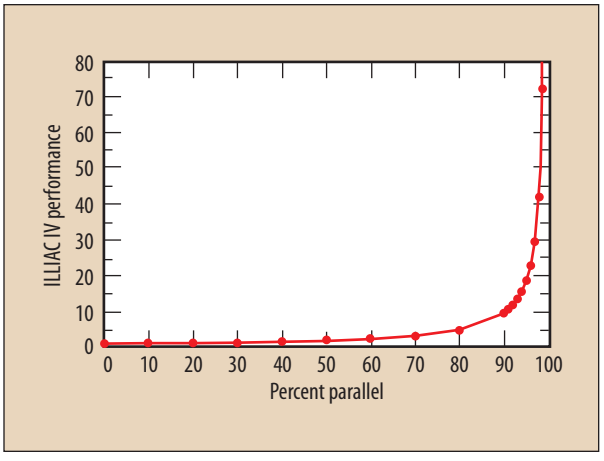

Figure 4. Performance of the ILLIAC IV with Slotnick's expected future goal of 256 E-units.

determined whether or not to participate in the execution of the I-unit's current instruction, an interesting but controversial proposal. The super uniprocessor was a design type, not a specific machine, so I had to estimate to the best of my ability what performance could reasonably be achieved by such a design. Figure 2 shows ILLIAC IV's performance on a problem having a varied, but reasonable, range of parallelism under the control of an operating system with characteristics similar to those then currently in use, having quite a bit of system management and data movement code. Figure 3 shows the performance of the super uniprocessor on that same problem and operating system.

Figure 4 shows the performance of the ILLIAC IV with Slotnick's expected future goal of 256 E-units and running a problem having a varied range of parallelism, but reaching a level of parallelism matching "America's symbol of purity," Ivory soap, at 99.44 percent.

I generated the following formula to estimate the ILLIAC IV's performance, giving it the benefit of assuming that if some parallelism existed, all processors could be usefully employed:

$$R = R_S \frac{S+P}{S+\frac{P}{M}} .$$

The numerator in the formula is $R_s \times (S + P)$, and the denominator is $S + P/16$ based on the proposed in 1967 ILLIAC IV having 16 arithmetic units. In this formula, S is the fraction of the problem that must be executed sequentially (or serially), and P is the fraction of the problem that may be executed in parallel if the computer has this capability. The sum $(S + P)$ is the aggregate workload, W to be performed. R_s is the performance of a computer that can only execute the problem in a totally sequential manner, regardless of the problem possessing the capability for parallel execution, and that has the speed of the ILLIAC IV's I-unit. The denominator reflects the capability of the ILLIAC IV to be able to execute the P component 16 times faster (this gives an advantage to the ILLIAC IV, for only in vector or matrix operations where the sizes are multiples of 16 would it perform that well). The ratio $(S + P)/(S + P/16)$ represents the speedup of the ILLIAC IV architectural feature for parallelism. This speedup times P_s is the optimistic performance of the ILLIAC IV. By the way, no one challenged this formula—my range of only up to 50 percent parallel was thought by some to be a bit low.

These figures are not quite the same as in the 1967 presentation, for they weren't published,[3] nor did I keep them, for I had no expectation of the intensity of their afterlife! I never called this formula "Amdahl's law," nor did I hear it called that for several years; I merely considered it an upper limit of performance for a computer with one I-unit and n E-units running problems under the control of that time period's operating system. The

A Few Notes on Amdahl's Law

Vladimir Getov, *University of Westminster*

Gene Amdahl's seminal paper,[1] which illustrates the sensitivity of the parallel computer's performance to the amount of code that is intrinsically sequential, is probably one of the most often cited works in the area of parallel computing. Unfortunately, this article is very short and does not include the analysis provided in the conference presentation. Therefore, for several decades, there have been various understandings, interpretations, and also speculations related to Amdahl's law. Nevertheless, one thing is clear: this article has been a corner stone in studying parallel computer performance. Following the text of the original publication, Amdahl's law states that

> the effort expended on achieving high parallel processing rates is wasted unless it is accompanied by achievements in sequential processing rates of very nearly the same magnitude.

Amdahl introduced his "law" at the 1967 Spring Joint Computer Conference, while comparing a sequential computer with performance (processing rate) of R_S and a parallel computer system having M processing units with a performance of R_P each. For application problems that allow parallelism, the corresponding workload, W, will be the sum of the sequential code, S, and the parallel code, P. Assuming there is no processing overlap between the sequential and the parallel fractions of a given workload, the execution time, T, to complete the work will be as follows:

$$T = T_S + T_P = \frac{S}{R_S} + \frac{P}{MR_P}. \quad \text{(A)}$$

If the sequential computer runs at the same rate as one of the parallel processing units (that is, $R_P = R_S$), the aggregate performance for this workload is expressed by Amdahl's original formula, as presented in 1967 but published only recently (see the other sidebar):

$$R = \frac{W}{T} = R_S \frac{S+P}{S+\frac{P}{M}}. \quad \text{(B)}$$

One of the immediate implications of Amdahl's law is the definition of the maximum possible performance for a given application problem, the so-called Amdahl's limit: if a computer has two operational rates while processing a given workload, the slow mode will limit the overall performance, even if the fast mode is infinitely fast.

This definition clearly suggests asymptotic properties for the aggregate performance. More specifically, if the number of parallel processing units becomes arbitrarily large ($M \to \infty$) and assuming that P has ideal parallelization properties, we may neglect the effect of P/M in Equation B and obtain the analytic expression of Amdahl's limit:

$$\lim_{M\to\infty} R = R_S \frac{W}{S}. \quad \text{(C)}$$

In other words, the *asymptotic parallel performance* for a specific application problem is limited by the sequential processing rate of the computer system, multiplied by the ratio of the total workload and its sequential component. The above definition and formula for Amdahl's limit can easily be generalized for many (more than two) processing rates in the case of more complex computer architectures.

The workload ratio in Equation C might or might not depend on the number of parallel processing elements, M, or on the application's problem size, N. Obviously, the best possible option for achieving high performance as the number of parallel processing elements and the application's problem size scale up is the case in which the sequential fraction of the workload is close to zero. Unfortunately, this component is usually large enough—hence it cannot be neglected—and often increases as M and N increase. A possible target, which is realistic albeit hard to achieve, would be to keep constant the sequential fraction of the workload. Clearly, the best option is when this component is relatively small and independent of M and N, such as $S = f(M, N) = const$. The difficulty in addressing and fixing these problems arises from the fact that the sequential fraction of the workload aggregates contributions from both the system software, including any coordination and synchronization overheads, and the sequential component of the specific application problem. Ensuring that all these remain small constants involves resolving a conundrum of hurdles and challenges, but if successful, leads to excellent performance results, usually known as *scaled speedup*.

The first reports about achieving scaled speedup results were published in 1988, after the completion of a set of experiments on the then brand-new 1,024-processor nCube/10 parallel machine.[2] These reports, however, also claimed that Amdahl's law had been invalidated by the team at Sandia National Laboratories, which triggered another debate in the professional community. It seems quite simple now to take Amdahl's limit formula (Equation C) and derive both the pessimistic prediction of ILLIAC IV performance, as presented by Gene Amdahl in 1967, and the excellent scaled speedup results of the Sandia experiments, published in 1988. After nearly 50 years, there is no doubt that Amdahl's law is fundamental for analyzing parallel computer performance and, as such, it is equally valid for a variety of cases, including the two cases discussed above.

References

1. G.M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," *Proc AFIPS Conf.*, AFIPS Press, 1967, pp. 483-485.
2. J.L. Gustafson, G.R. Montry, and R.E. Benner, "Development of Parallel Methods for a 1024-Processor Hypercube," SIAM J. Scientific and Statistical Computing, vol. 9, no. 4, 1988, pp. 609-638.

debate between me and Daniel Slotnick was joined by many in the audience, and it became quite heated. I felt he was trying to egg me on into attacking him rather than his computer design, but I carefully avoided that, only to be attacked by Herbert Grosch from the audience for not attacking him. It became a bit of a circus, and I was quite unhappy about being involved, for I thought of it as a rational analysis of two competing design approaches, not a bashing of another human for offering a controversial design approach.

Twenty years later, I was informed of a proof that Amdahl's law was invalidated by a team at Sandia National Labs, where researchers had used the brand-new (at that time) nCube/10 distributed memory parallel machine connecting a number of compute nodes via communication lines,[5] but with each compute node also connected to I/O devices for loading the operating system, initial data, and results. This forced all control and data movement to be carried out in parallel. I didn't enter the fray; I merely commented that what they called Amdahl's law described the ILLIAC IV, which had only one I-unit. I also heard that Amdahl's law was used to challenge the multicomputer systems developed by Massively Parallel (http://massively parallel.com), a Colorado firm whose chief system designer looked at the formula, thought it appeared to have the form of an information-theoretical statement, and used it to further enhance his system. As a result, Massively Parallel invited me to join its advisory board!

I still do not consider Amdahl's law to be so much a law as it is the relationship of memory size and computer performance to the number of bits of I/O per instruction executed or as reduced, when considered as a function of the fraction of program required to be in "virtual memory." These seem to be lost in the mist of time! I consider the WISC to be the most remarkable architectural achievement I've made, and with no input from any source other than sheer inventiveness.

Acknowledgments

An earlier version of this article was published in the *IEEE Solid-State Circuits News*, vol. 12, no. 3, 2007, pp 4-9; www.ieee.org/portal/cms_docs_societies/sscs/PrintEditions/200707.pdf.

References

1. J. von Neumann, "First Draft of a Report on the EDVAC," Moore School of Electrical Eng., Univ. Pennsylvania, 30 June 1945; www.virtualtravelog.net/wp/wp-content/media/2003-08-TheFirstDraft.pdf
2. P.N. Edwards, "Chapter 3: SAGE," *The Closed World: Computers and the Politics of Discourse in Cold War America*, MIT Press, 1996, pp. 76-114.
3. G.M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," *Proc AFIPS Conference*, AFIPS Press, 1967, pp. 483-485.
4. G.H. Barnes et al., "The ILLIAC IV Computer," *IEEE Trans. Computers*, vol. C-17, no. 8, 1968, pp. 746-757.
5. J.L. Gustafson, "Reevaluating Amdahl's Law," *Comm. ACM*, vol. 31, no. 5, 1988, pp. 532-533.

Gene M. Amdahl *began his career with IBM in 1952, where he first worked on neural network simulation and then moved on to lead the development of the IBM 704. He followed this with initial planning for the IBM 709 and Stretch (IBM 7030). In 1956, Amdahl began work at Ramo Wooldridge and later at Aeronutronic before returning to IBM in 1960. He then headed the architectural and data-flow planning for the IBM 360 product line and became IBM Fellow in 1965. He headed IBM's Advanced Computing Systems Laboratory in Menlo Park until 1970 when he founded Amdahl Corporation. Dr. Amdahl has numerous awards and patents to his credit. In 1983, he was awarded the Harry H. Goode Memorial Award by the IEEE Computer Society "in recognition of his outstanding contributions to the design, applications and manufacture of large-scale high-performance computers."*