Published in 9th International Symposium on Languages for Intensional Programming (ISLIP), 1-9, 1996
which should be used for any reference to this work

NETWORKING AND COMPUTING: FROM THE CHIP TO THE WEB

PETER KROPF
Département d'informatique, Université Laval
Ste-Foy (Québec), Canada G1K 7P4
E-mail: Peter.Kropf@ift.ulaval.ca

and

JOHN PLAICE
E-mail: John.Plaice@ift.ulaval.ca

Abstract

There are two fundamental trends in the development of computers: the miniaturization of components and the increase in communication capacities. The combination of these two trends is leading to a qualitatively new situation, in which the same techniques will be applicable at all scales of computing, be they at the chip level or at the level of the World Wide Web (WWW).

1. Introduction

The von Neumann model of computation[3] has been the standard model of computation since the late 1940s. Under this model, a program is stored in a memory and executed by the combined action of a control unit and an arithmetic unit, with loads from the memory and stores to the memory being effected and coordinated by the control unit.

This model of computation has shown itself to be remarkably successful. Even today, the vast majority of computers are uniprocessor machines that strongly adhere to the von Neumann model. As a result, the standard paradigm for software is imperative programming, as it is impractical to write efficient programs for von Neumann machines in languages that are not imperative.

Basically, the von Neumann model supposes that there is a CPU and a memory hierarchy, in which computation and I/O are controlled in the CPU, and that loads and stores are made through the memory hierarchy, with varying seek times and transmission delays, depending on the nature of memory.

Despite its obvious success, the von Neumann model has often been severely criticized, either as inefficient, inelegant, or both. First, claims have repeatedly been made that the limits of technology have been reached or will soon be reached, and that the speed of light limits the potential for improvement in chip design. Despite these claims, current annual uniprocessor performance improvement, around 1.52, is actually greater than it was in the 1950s and 1960s.

Second, criticisms have been made that the model is inefficient because of a perceived bottleneck between the CPU and memory, and possibly as well between the different levels of the memory hierarchy. These criticisms have often come from programming language designers, typically from the functional and dataflow programming communities.

Nevertheless, the perpetrators of these criticisms actually share the dominant worldview of computation that led to the von Neumann model: there is a task to be effected by a computer, sitting in some memory; this task must be initiated, and then it will run, normally until it terminates.

This dominant worldview is even shared in the design of massively parallel processors (MPPs), which incorporate hundreds or thousands of processors in a single computational unit, with highly complex interconnect topologies between the processors and highly parallel memory and disk arrays. Despite the massive internal parallelism inherent in these systems, they still act and are programmed as if they were von Neumann machines.

One might think from this discussion that the von Neumann model will remain with us forever. In fact, the authors claim that this model *has* reached the breaking point, but that the underlying reasons are completely different from those that have been put forward for a number of years.

More and more, computing entails the interconnection of compute and storage servers by high-speed networks in order to offer Web-like services. The demand placed on communication can be phenomenal. For example, video-on-demand at home would probably require the equivalent of a T1-line to every home,[2] which would place incredible demands on the servers and switches involved in offering this service.

It is not technological development at the architectural level that will hinder the continued success of the von Neumann model, nor is it the inelegance of programming that might be argued by members of the functional programming community. It is the development of the Internet and the large intranets of large organizations that will ultimately put into question the von Neumann model. The demand for massively parallel I/O, possibly with real-time computation to be effected on these streams, will force us once and for all to break the von Neumann bottleneck.

In effect, the von Neumann model will be replaced by a dataflow model of computation, with the streams of information flowing from one node to another, undergoing the appropriate transformations. However, these streams will be initiated by requests of queries from users, say, using Web-like browsers. As a result, the future model can only be demand-driven dataflow.

The rest of the paper will endeavor to put forward these ideas in more detail. Section 2 begins by summarizing the von Neumann model of computation, and its extensions. Section 3 will give a detailed look at some of the criticisms levelled at the von Neumann model. Section 4 will summarize technological developments that are relevant to the discussion. Section 5 will argue in favor of the 'network as the computer' model, which will support the arguments in section 6 for the demand-driven dataflow model. We conclude with a discussion.

2. The von Neumann model

The basic von Neumann computer system is composed of four functional units: a memory (storage), a control unit, an arithmetic unit, and input/output units (peripherals). Once this system has been put together, it is assumed that its structure is independent of any given problem or application.

The memory of the system is divided into cells of equal size, which are numbered consecutively, where the numbers denote *addresses*. The system is assumed to be memory programmable, so a program is described as an exchangeable content of the memory, and instructions and data (operands and intermediate results) are stored in the *same* memory. A program is represented as a sequence of elementary instructions, which are consecutively stored in memory cells with increasing addresses. Finally, changes in the execution sequence from the stored instruction sequence are implemented by special *jump* instructions.

The von Neumann model describes not only the structure and organization of computer systems, but also their behavior, as seen by the various functional units. A program is thus a passive entity, while a process denotes an active unit.

To this basic model, several extensions have been made, used in different real, existing systems.

- Multiple processes may be executed at the same time through multiple functional units.
- The memory is subdivided into hierarchical layers, thus reducing latency.
- Some units may be present in multiple copies (several arithmetic units, control units, etc.)
- Instructions may be structured (different formats, lengths, hierarchical instruction sets, multiple address instructions, a single instruction may issue a set of instructions that are executed simultaneously by different units, etc.)
- The memory is addressed using address transformations (indirect addressing, paging, different address spaces, etc.)
- The system may react to external events (interrupts).
- The system may include one or several real-time clocks.
- A system may consist of several von Neumann machines that work autonomously, and these will be connected to a central unit.

It should be clear that the fundamental assumptions found in the basic model are not put into question by the different extensions. Performance may improve, but the overall architecture remains the same, and the same form of programming, namely *imperative programming*, is still applicable.

3. Criticisms of von Neumann

Over the years, two kinds of criticism have been levelled at the von Neumann model of computation. First, predictions have been made about the possible technological developments. Here are a few such statements summarized in Hennessy and Patterson's book:[9]

- The turning away from the conventional organization came in the middle 1960s, when the law of diminishing returns began to take effect in the effort to increase the operational speed of a computer.... Electronic circuits are ultimately limited in their speed of operation by the speed of light... and many of the circuits were already operating in the nanosecond range.

- ... sequential computers are approaching a fundamental physical limit on their potential computational power. Such a limit is the speed of light....

- ... today's machines... are nearing an impasse as technologies approach the speed of light. Even if the components of a sequential processor could be made to work this fast, the best that could be expected is no more than a few million instructions per second.

Now it is clear that despite such statements, today's computer architectures still follow exactly this model. Indeed, on the surface one might get the impression that the von Neumann model of computation is victorious. Because of the introduction of RISC instruction sets, processor performance has increased at a faster annual rate than in the previous decade. Clearly, these improvements in performance are not due to any revolutionary architectural principle, but, rather, to the steady evolution and development of technology. Basically, the architectural model has remained the same since the emerging of the first electronic computer equipment in the early 1940s[9].

Another source of criticism, at least during the 1970s and 1980s, was the programming language community. The best known criticism was from John Backus in 1978[1], but there have been many others as well. The von Neumann model is not elegant, since its semantics requires the manipulation of the system state, which can sometimes be tricky, especially when one is using constructs such as pointers. Moreover, language specialists and others have claimed the model to be inefficient, because of the single communication path between the CPU and the memory, known as the von Neumann *bottleneck*. It should be understood that this criticism assumed that the bottleneck came about solely from the interaction between CPU and memory. However, considering the dramatic gains in computer performance over the last few years, one might get the impression that this bottleneck has been overcome or at least lost its impact, as a result of the tremendous technological improvements observed[9,5]. In particular, one is attempted to take this attitude in the case of traditional computer systems, such as workstations or PCs. But in fact, the bottleneck is much deeper and more general than that.

We observe for the first time today the possibility to build and integrate a balanced pyramid of computing power in scientific, technical as well as business computation, where each element consistently supports the others.[10,5] The elements of the pyramid should consist of the highest performance supercomputers at the top, networks of worstation clusters at the bottom, and mid-size or desktop supercomputers in between. Communication is a key issue, not only inside each of the diverse components of this infrastructure, but between them as well.

The infrastructure pyramid offers the user more than just the possibility to choose the appropriate technology and system: it provides a heterogeneous system capable of satisfying the diverse and heterogeneous spectrum of application requirements with equally optimal performance. In other words, we can foresee a distributed system, commonly called a *metacomputer*,[4] consisting of such diverse components as workstations, vectorcomputers or massively parallel computers. For any distributed or parallel system to be successful, we need high bandwidth and high speed communication networks at all levels: on the chip, the board and machine level.

However, the real von Neumann bottleneck is communication. All implementations of this model have a communication bottleneck, corresponding to the point where information must be sequentialized, even though conceptually it should be passing in parallel.

There is another aspect of the von Neumann model which, to the best of our knowledge, has never been put into question. The very foundation of this model is that there is a program that can be placed in a single memory. However, *the program* does not exist. Software evolves to meet the needs of those that use it. It is modified, new versions and variants are created, and not all changes are necessarily clear or coherent. As a piece of software is used on different sites around the world, it can well be modified or finetuned differently in each locale to best meet the needs of local users.

This same versioning phenomenon[11] occurs not just for software, but for all forms of data. To compound the versioning phenomenon, data can be replicated in several different sites, with different levels of reliability: some might be out of date, others inaccurate, etc.

The versioning and reliability problems lead us to a situation where it makes little sense to refer to *the global state* of a computer system. Such a state does in fact exist, but it is unknowable, in the same sense that it is impossible to attain a complete understanding of any phenomenon in nature or in society. We can only attain an approximate understanding of the state of a system, not a complete understanding.

As a result, to refer to *the program*, which is going to manipulate the global state, as is implicitly implied in the von Neumann model, is simply unrealistic in the typical case of large software systems.

Furthermore, from a philosophical perspective, the von Neumann model is an idealistic interpretation of computing. It supposes that *the program* exists, and that it will run once and for all. In fact, except in trivial situations, that program running only once does not exist: rather, only successive approximations are built, and these are modified as bugs are fixed.

When a bug is fixed in a large program that is handling large amounts of data, the bug fix might well have no influence on most of the data, so there is no need to recompute all of the results from the beginning. This idea has been used, for example, with success for years in the area of incremental or selective recompilation. In other words, we need to treat the set of successive versions of the program as one entity, similarly for the successive versions of the treated data.

As should be clear from this discussion, the very basis for the von Neumann

model is simply unrealistic. Much more realistic is the idea of a demand-driven model, where changes made to a program and to data will only provoke the necessary updates, rather than forcing complete rebuilds of systems.

4. Technological improvements

If the von Neumann architecture has survived for so long, it is because technological developments have kept it alive. Clock frequencies, the number of gates per chip (density) and the number of pin-outs per chip have been steadily increasing for the last twenty-five years, and are expected to continue to increase.[7]

At the same time, the complexity of uniprocessor chips just keeps on increasing. Many of the top-model chips available today (e.g., PowerPC 620, MIPS R10000, Intel P6, AMD K5) implement ingenious forms of instruction level parallelism (ILP), involving speculative execution and dynamic scheduling. The von Neumann model really only exists at the chip interface and interconnection level.

At the hardware level, the von Neumann communication bottleneck is being addressed with the possibility of using optical interconnects, which would allow *full connectivity* between components of a system, as well as dynamic reconfigurability of routing, given that photons do not interact and therefore carry tremendous possibilities for parallelism. Much of the hierarchical memory arrangement could be completely reconfigured if a parallel optical interconnect was used to replace electronic pins.[8] In fact, in that reference, they suggest to rip out the on-chip cache and to replace it with an optical interconnect.

By doing this, one could actually reduce the clock speeds of chips, thereby reducing power consumption, as well as to allow more effective use of the surface area of a chip, currently being wasted in greater and greater proportions.[9] Furthermore, it should be simpler to have multiple computation components on the chip, thereby turning the chip itself into a parallel computer.[6]

5. The network as computer

When the von Neumann model was invented, computer networks simply did not exist. The first real network in the modern sense was the early ARPANET, which connected several geographically removed sites at 56kbps. Today's single-mode optical fiber links can allow 2Gbps. The fibers themselves can carry much higher bandwidths, but implementation problems with the interconnects at the ends of the fibers limit the possible throughput.[12]

Such high network capacities have several implications. First, more and more people can use networks, as those can carry large amounts of information. This is exactly what is happening with the development of the Internet, of intranets for large organizations, of the Information Superhighways and World-Wide-Web-like applications.

The WWW model assumes that people request services, which are offered as soon as the demand is made. Servicing a request might provoke other requests, and in fact simultaneous multiple connections might be made to one or

to several sites to exchange, to transform or to simply download information.

Typical applications, as exemplified by Web-like systems, may be characterized as follows: information streams (data, control, voice, video, text and image sequences) are established between compute and storage servers and the (potentially multiple) participants. This requires end-to-end performance at all levels: servers, communication links and workstations. They have their equivalents at the hardware level as well as at the software level. In order to achieve high end-to-end performance, no bottleneck of whatever flavor may be allowed to occur.

But the von Neumann model collapses precisely at this moment. Its major bottleneck, communication, hangs like a ringstone around its neck. It is not that the compute nodes are not fast enough, it is simply that one cannot get information on and off a chip fast enough, even if no real computation needs to be done! And if the chip does not create the bottleneck, it will be the bus....

Another implication of such high communication speeds is that computers have less and less time to actually handle the data that is flying through, since the communication speedup is greater every year than is the processor speedup. The standard bottleneck, in which the access times of the different levels of the memory hierarchy grow further and further apart each year, just keeps on getting worse: the annual growth of processor performance has increased from 1.35 (1980) to 1.55 (since 1986), the communication speeds have increased by a factor of 1.52 over the past 25 years, but memory performance has only increased by 1.07 since 1980.[9]

6. The future of software

A general scientific principle is that if an idea is applicable and useful at widely differing scales, then it is a good idea. For example, Newton's Universal Law of Gravitation states that planetary motion is governed by the same force as is an apple is falling to Earth. The gravitational force applies at all levels of matter, although at high densities, relativistic corrections must be made. Similarly, the Maxwell-Hertz-Heaviside equations for electromagnetic forces are applicable for all known levels of matter.

The developments that we have described in the last two sections show that there is a tendency to have streams of data flowing from many different directions, with the need to place computation nodes in the appropriate places to actually do any necessary transformations. This tendency is taking place at the macro-scale (WWW), as well as on the chip, as well as at every level between.

But the software model for the WWW is *not* the von Neumann model. It is a navigate/query model, familiar to those who work in demand-driven dataflow. Demands must be propagated from node to node, possibly simultaneously pursuing multiple paths, until they can finally be met, and the appropriate data streams must then be set up.

This model is also applicable at smaller scales. As more and more processing units become available, and as massively parallel input onto computation chips becomes a possibility, then computations will also be effected in a

demand-driven manner.

For all this to be managed, a fault-tolerant distributed operating system will be required, once again at all scales of computation. It is clear that if computation is to take place on the WWW, then fault-tolerance will be fundamental, as links can break down, system loads can vary significantly, etc., which means that there is no known global state of the system anymore, and that the many different entities of a system must ensure by sufficient coordination (communication and synchronisation) a fault-tolerant, still reliable functioning.

Fault-tolerance will also be necessary at the chip level: chip densities are steadily increasing (currently at 60% per year[9]), and we can easily envisage that in a not distant future, a single chip will have a large number of computing and communication units. For simple economic reasons, most manufacturers will want to be able to market these chips, even if a small proportion of the components are known to be faulty. So, for these chips to be usable, a distributed fault-tolerant operating system will have to run on the chip itself, thereby increasing flexibility and reliability.

As for communication, it too seems to be becoming more and more scalable, with the use of Asynchronous Transfer Mode (ATM). Under ATM, 53-byte self-addressing cells, carrying 48-byte payloads, are used for communication at all levels. Currently used by telecommunication companies, they are used more and more on an experimental basis in computer systems as well.

Given the already complex means of allocating resources on a chip, building an on-chip fault-tolerant operating system is quite feasible. However, for these ideas to be realistic and usable, such an operating system must be as simple as possible. In fact, what we need is the operating system equivalent of RISC instruction sets or fixed-size ATM cells. The operating system should be able to rapidly switch demands, passing them from one system to another using high-speed eduction. In fact, Niklaus Wirth's claim that current software — particularly operating system software — is too complex[13] is in fact an understatement.

7. Conclusions

Despite the many claims made against the von Neumann model, it has survived 50 years because it is good at implementing what it intends to implement: a single program residing in some memory that is to be executed on a single computer. As a result, the standard claims made against the model do not stand up, as they suppose as basis the very reason of the von Neumann model.

Nevertheless, we feel that the von Neumann model cannot survive, simply because the communication demands placed on it by the Internet and large intranets will get stuck in the real von Neumann bottleneck, which is communication.

Furthermore, the very notion of the program which is sitting in a single memory and that is going to process the data which is sitting in the same memory is simply unrealistic, given the versioning and replication of data that is necessary when using world-wide networks.

The only solution to these problems is to develop the demand-driven dataflow

of computation that is inherent in the Web to all scales, introducing the massively parallel interconnections that are required for this model to work.

8. References

1. J. Backus. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *Comm. ACM*, 21(8):613–641, 1978.
2. G. Bell and J. Gemell. On-ramp prospects for the information superhighway dream. *Comm. ACM*, 39(7):55–61, 1996.
3. A. W. Burks, H. H. Goldstein, and J. von Neumann. Preliminary discussion of the logical design of an electronic computing instrument. In *Papers of John von Neumann*, pages 97–146. MIT Press, Cambridge, Massachusetts, 1987.
4. National Center for Supercomputing Applications. NCSA Metacomputer. `http://www.ncsa.uiuc.edu/General/CC/CCMetacomputer.html`, 5 February 1996.
5. National Science Foundation. Task force on the future of the NSF supercomputer centers program. `http://www.cise.nsf.gov/asc/W/default.html`, 23 May 1996.
6. G. Fox. Basic issues and current status of parallel computing – 1995. Technical report, NPAC, Syracuse, 1995.
7. E. E. E. Frietman. *Opto-Electronic Processing & Networking: A Design Study*. Delft University of Technology, Delft, The Netherlands, 1995.
8. E. E. E. Frietman. Optical-read/store & write devices: alternatives for the pin limitation problem. In *HPCS'96 Conference Proceedings*, Ottawa, Canada, June 1996. IEEE Canada Electronic Services. Distributed by Carleton University Press.
9. J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, San Francisco, California, 1996. 2nd edition.
10. F. Hossfeld. On the value of diversity: an insider's view of high-performance computing. In *HPCS'96 Conference Proceedings*, Ottawa, Canada, June 1996. IEEE Canada Electronic Services. Distributed by Carleton University Press.
11. J. Plaice and W. W. Wadge. A new approach to version control. *IEEE Trans. on Soft. Eng.*, 19(3), March 1993.
12. A. S. Tanenbaum. *Computer Networks*. Prentice-Hall, Upper Saddle River, New Jersey, 1996. 3rd edition.
13. N. Wirth. A plea for lean software. *IEEE Computer*, 28(2):64–68, February 1995.