

# What's Ahead in Computer Design?

M. J. Flynn\*  
Stanford University  
Computer Systems Laboratory  
flynn@ee.stanford.edu

## Abstract

*CMOS technology should, over the next few years, reach lithography of under  $0.1\mu$ . This provides a die area improvement of a factor of 10 over today's technology.*

*What is the best use of this area? Multiprocessors, very high level superscalar processors, VLIW, processor-memory combinations, or simpler processors with very large caches? Some have made the case that pin bandwidth is an important limit that must be confronted by any organization. This presentation reviews the scalability of our current technology and critically analyzes the various alternatives to the best use of silicon area in the era of  $0.1\mu$  technologies.*

The rapid advance of technology has made possible a number of processor improvements which would have been thought impossible just a few years ago. These improvements include cycle time, cache size, and an extraordinary increase in the complexity of the processor; this complexity is required for the management of relatively large scales of instruction level parallelism.

## 1. The Advance of technology

Over the past five years, feature sizes have dropped from about  $f = 0.8\mu$  to about  $f = 0.35\mu$ . Indeed, it certainly seems possible to achieve somewhat less than  $f = 0.1\mu$  within the context of our current CMOS technology. Such advances provide an effective area increase of about an order of magnitude, meaning that we will be able to accommodate perhaps ten times more transistors on a fixed die size with a feature size of  $0.1\mu$  than we currently can with feature size  $f = 0.35\mu$ . Ideally, cycle time scales linearly with feature size. However, below about  $.5\mu$ , some advantage of scaling is lost. This is due to the fact that power supply voltages must be scaled down in order to avoid the

---

\*This work has been supported in part by the National Science Foundation under grant MIP-9313701.

effect of extraordinarily high electric fields which would be present in very small devices. Semiconductor Industry Associates (SIA) have developed a road map or a projection of a number of technology parameters for the next few years (Table 1) [13].

Table 1 also lists the projected SIA parameters for DRAM at comparable times. The issue for the designer, of course, is to take advantage of the advance of technology that is forseen.

## 2. The design applications

For purposes of framing the discussion of future processor development, I suggest a simple categorization of processor design environments:

- Server (time optimized).
- Client (area-time optimized).

In the server processor, we are concerned primarily with processor speed, especially the execution time of large programs. The server environment is characterized by requiring high rates of instruction execution. This could be for scientific computation, or it might be for managing large arrays of storage element disks.

The second design environment is the client. In this category I include workstations and embedded processors of all sorts. Client processors represent the overwhelming market for processors in today's application environment. However, the server processor presents important "leadership" in product design which can go a long way towards enhancing a vendor's status and product acceptance across all product offerings.

### 2.1. The server: time optimization

The architecture and the organization of the server processor is far and away the most widely discussed and argued development among processor designers. Indeed, it frequently sets the design style for processors in the future. In

**Table 1. 1994 SIA roadmap summary.**

1st DRAM Year	1992	1995	1998	2001	2004	2007
Feature Size ( $\mu m$ )	0.50	0.35	0.25	0.18	0.13	0.10
$V_{DD}$ (V)	5.0	3.3	2.5	1.8	1.5	1.2
Trans/Chip	5M	10M	20M	50M	110M	260M
Die Size ( $mm^2$ )	210	250	300	360	430	520
Freq (MHz)	225	300	450	600	800	1000
DRAM Bits/Chip	16M	64M	256M	1G	4G	16G
SRAM Bits/Chip	4M	16M	64M	256M	1G	4G

today's marketplace, there has been relatively little need to distinguish between the server processor and the client processor (except perhaps for some embedded applications). In the future, with perhaps ten times more area or transistors available on a die, some important distinctions will develop, defining a more distinct cleavage between the server and the client processors.

Server processors are limited by processor speed, and especially their speed in managing a large memory space. Building high speed processors seems to be less of a problem than effectively using them. The achievement of a gigahertz clock rate is on the SIA roadmap a few years hence. Coupling this clock rate with any of a number of performance enhancing processor architecture techniques ensures the potential for very high "raw" (or "marketing") MIPS rates. These techniques include

1. The use of wave pipeline or latchless pipelines to further increase the clock rate by factors of 2 to 3 [15].
2. Issuing 8 or more instructions per cycle either in a superscalar or a VLIW architecture format [4, 2].
3. Use of on-chip multiprocessors [11].

Wave pipelining is based on the use of delay inherent in combinatorial logic circuit implementations. Suppose a given logic unit has a maximum interlatch (clocked latches) of  $P_{\max}$  and a corresponding minimum delay of  $P_{\min}$  with clock overhead (setup, hold, skew, etc.) of  $C$ . Then the maximum achievable cycle time,  $\Delta t$ , is

$$\Delta t = P_{\max} - P_{\min} + C.$$

As with conventionally clocked systems, the system clock rate is the maximum  $\Delta t_i$  over  $i$  latched stages.

In practice, using special CAD tools [15], it is possible to arrange  $P_{\min}$  to be written about .8 to .9 of  $(P_{\max} + C)$ . While this would seem to imply clock speedup of more than 5x, environmental issues (process variation, temperature gradient across a die, etc.) limit realizable clock rate speedup to about 3x [10].

Wave pipelining works well in predictable sequencing across pipeline segments (e.g., vector unit processing, access to memory). So far, it is less successful in dealing with *decisive* logic elements—where the output of a logic stage immediately selects a new input (e.g., an instruction decoder or branch unit).

ILP (instruction level parallelism) is the basis for enhancing the performance of most current high-end microprocessors. The obvious issue is, how much ILP is available? Current processor implementations attempt to use 4- or perhaps 8-way instruction issues each cycle [5, 3, 8]. At least at this time, it seems problematic that ILP compilers will efficiently support more than these levels. An interesting question is when (and how) to find ILP? The obvious alternatives are compile time, when the entire scope of the program is available, and run time, when the true state of the machine is determined. VLIW machines (at least as implemented in the past) use only compile time ILP. Superscalar machines can use both ILP detection methods. This should give the superscalar processor an ILP advantage, since the two approaches do not detect exactly the same parallelism. But the superscalar processor also has an overhead of instruction dependency checking prior to instruction dispatch. This can add one or two cycles to overall the instruction interpretation latency, and adversely affect performance. The net conclusion at this time seems to be that the superscalar approach seems to be the paradigm of choice, since it more easily accommodates instruction set compatibility issues. However, should ILP greater than 8 prove feasible, the superscalar dependency checking overhead may increase to a level to cause serious reconsideration of VLIW.

There is clearly some limit to ILP, and at that point (which may soon arrive), the best use of silicon is to implement on-chip multiprocessors [11]. I would expect relatively low levels of on-chip multiprocessing, perhaps 2 to 4 processors sharing some type of data cache to ensure data consistency. The state of the software task partitioning and scheduling act seems well able to provide efficient use of at least this level of multiprocessing.

How well can any of these organizations manage access to large memories? This seems to be the crucial question, especially in light of an increasing memory latency problem. Technologists use advancing technology to provide increasing DRAM density for memory chips, leaving the access time relatively constant. As processor speeds increase, the ratio of memory access time to process cycle time increases. This is compounded by the execution of multiple instructions in a single cycle. Suppose we define the *logical memory latency* as the maximum number of instructions that are executable in a memory access time.

The logical memory latency has significantly increased as processor cycle time improved and processors use enhanced instruction level parallelism. There is a maximum logical memory latency beyond which an application sees no performance advantage. In one form or another, this notion of a memory bound to processor performance has been referred to as the *memory wall* [16]. It has been described as a latency or a bandwidth limitation of memory.

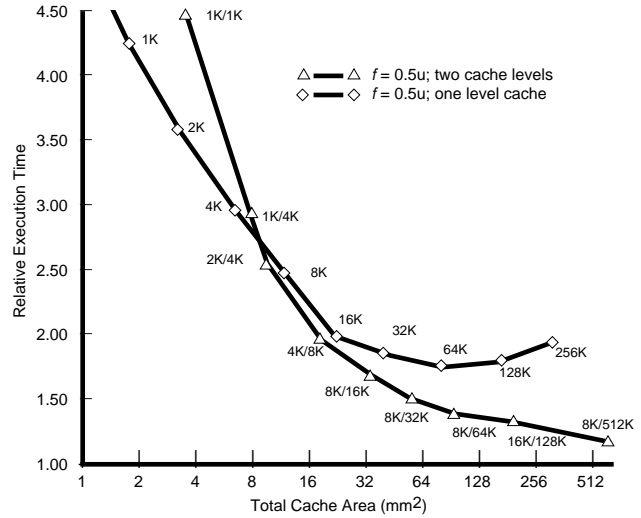
There seems to be at least some small confusion in choosing latency or bandwidth, in discussing the memory problem. To some degree, bandwidth can be traded for latency, but it is an imperfect tradeoff. Suppose we use a synchronous DRAM to access an entire column of data and read out this column at a high rate. We have significantly improved memory bandwidth at least at the chip level, but have left access to the first word unaffected. This can be helpful in reducing the cache miss penalty. But this reduction is usually by less than a factor of two. Moreover, for large caches at least, in order to reduce the hit rate, one is naturally inclined towards using large line sizes. Large line sizes increase the bandwidth required of memory. Burger [1] shows that there is a significant opportunity to increase effective pin (and memory) bandwidth between one and two orders of magnitude by making better use of on-chip memory. The basis for this improvement is adapting the cache organization (line size, associativity and write policy) to the application. The most important lesson is that large caches blindly implemented (without reference to the application) are not a solution to processor performance.

Small feature sizes coupled with rather larger die size enable large (> 1 MB) on-chip caches, but these caches are limited in two ways:

1. Large caches cannot be accessed at the fast processor cycle times ( $\Delta t \approx 1$  nsec).
2. Applications limit the effectiveness of large caches.

McFarland [9] has proposed an access time model for caches with feature size  $f$ , size  $C$  in Kbytes, and associativity  $A$  as (approximately)

Access time (ns) =  $(0.35 + 3.8f + (.006 + .025f)C) * (1 + .3(1 - 1/A^3))$ .



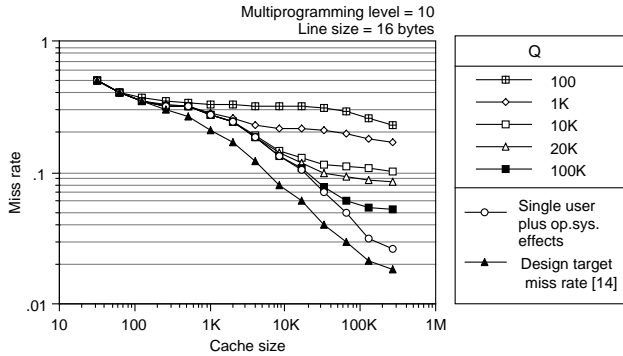
**Figure 1. Relative execution time per instruction for SPEC92. Time is relative to a processor with a perfect cache.**

At  $f = 0.1\mu$ , we expect cycle time ( $\Delta t$ ) to be 1 nsec (Table 1). The cache model [9] suggests that at  $f = 0.1\mu$ , we could have a maximum cache size of

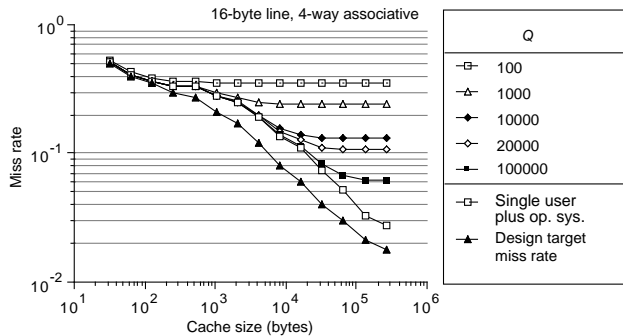
- $32^{KB}$  with access time 1 ns.
- $128^{KB}$  with access time 2 ns.

The cache access in a processor cycle time problem can be managed with multiple levels of on-chip cache. Figure 1 illustrates the need for multiple levels of cache for a simple pipelined processor even at  $f = 0.5\mu$  [7]. At  $f = 0.1\mu$ , one might argue for even *three* levels of on-chip cache. The designer can have the area (for large caches) or the access time (to match fast cycle times), but not both.

Ultimately, the effectiveness of cache is heavily dependent on the application. In an application consisting of a large number of small transactions, a system cannot effectively use a large cache, as before the cache fills, the transaction is complete and the application moves on to another process. The number of instructions between a task switch ( $Q$ ) has been used as a measure of stability or latency tolerance of an application [14]. Figures 2 and 3 [6], show two multitasked environments, one where control is never returned to a task (a cold or transaction environment), and another where, in high degrees of multiprogramming applications, control does return to an application (this is called a warm cache environment). If applications are indeed latency tolerant, then high degrees of instruction issue and large caches make a great deal of sense. If applications are latency intolerant, such as a transaction or a highly multi-



**Figure 2. Miss rate for a warm cache with ten tasks (MP = 10).**



**Figure 3. Miss rate for a cold-start cache including system activities.**

tasked environment, then the systems become completely memory limited, and the exact processor organizational details are not significant.

Some have suggested the use of an integrated processor memory chip to avoid the memory wall problem [12]. While this may have a great deal of appeal in client (embedded) processors, it has a number of obvious shortcomings.

1. It significantly limits memory size.
2. Access to the first DRAM word remains unaffected. It only decreases the logical access latency by a factor of 2 or 3, it does not eliminate it.
3. DRAM technology (power supplies and other process variables) are not compatible with the best state of the art in processor design. Therefore, either the DRAM or the processor technology must be compromised.

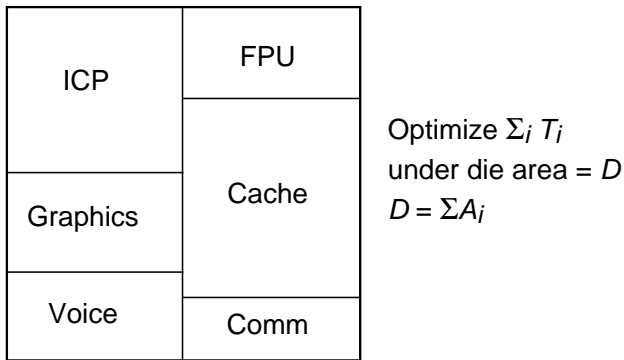
It is interesting to note that the density of an SRAM technology (the same as processor technology) is about the same as the DRAM density of a generation earlier (see Table 1).

## 2.2. The client processor: area–time optimization

Optimizing the client processor is a different problem from the server processor, as one’s goal is to optimize the overall cost–performance of the system, not performance alone. There is more flexibility in dealing with a number of pieces of the system than with a processor in isolation. The modern client system could increase functionality to provide services such as multimedia compression and decompression, encryption and decryption, and network access. A “system on a die” could be an attractive alternative to simply another big processor providing a relatively marginal improvement in performance. Instead of the compromise of including “main memory” on the processor chip, one could explicitly include frame buffer (SRAM) as well as network buffers, etc., on chip and couple this to a simpler processor that has a lower logical memory access ratio.

The system processor (Figure 4) includes the integer core processor (ICP), the floating point unit (FPU), and multiple levels of cache/specialized memory, as well as the various multimedia functional units required at the client processor. Since there is a broad area–execution time tradeoff for each unit, the system die designer must minimize the total application execution time,  $T$ , which is the sum of the time spent in each functional unit,  $T_i$ . This must be done under constraint of a fixed die size  $D$  which encompasses the area of the various systems components,  $A_i$ .

Since different tradeoffs are made for different applications, the primary determinant in the effectiveness of the above is the cost of the initial design, as the result becomes a type of ASIC system. We look ahead to a day when the role



**Figure 4. "System" chip.**

of CAD is to provide an assortment of state-of-the-art application designs that are already fully articulated and validated, which can rapidly be integrated into an ASIC system.

The various functions that might be incorporated include:

- integer core
- floating point unit
- one, two, or three levels of cache
- data compress/decompress (e.g., MPEG II or extensions)
- data encryption/deencryption
- voice synthesis
- voice recognition
- 3D graphics
- support for video
- network communications
- portion of main memory

The system chip could include all functions with a compatible technology base. Function optimized to another technology base (e.g., main memory, analog signal processing, etc.) could reside on different die on the same multichip module.

### 3. The basis for design

With the forward push of technology, significant new possibilities are arising for processor architects in the area of purely high speed processor designs. These include increasing levels of instruction level parallelism, coupled with

multiple processors on a chip, but a problem in achieving high performance is the latency tolerance of the application. For those applications which exhibit predictable and expected behavior, it is straightforward to achieve high performance. Note that this has been a true statement for several decades of computer engineering, going back at least to the era of the vector processors. The real challenge is to create organizations that are robust in the face of applications that are less predictable. Providing this robustness requires a combination of technology management and careful architecture selection. In the client area, in order to make best use of the technology, the architect must consider optimum combinations of functions which compose a system. The recent systems emphasis on networking, multimedia, and the increasing use of a variety of signal processors for embedded applications ensures that there is a significant opportunity for system on a chip development. Whether this can be totally effective or not depends not so much on basic technological parameters, but on the availability of robust CAD tools and robust designs which are supported by these tools to enable the systems designer to make optimum tradeoffs across a broad front of space-time possibilities.

### References

- [1] D. Burger, J. R. Goodman, and A. Kägi. Memory bandwidth limitations of future microprocessors. *Proceedings of ISCA'96* pp. 78–89.
- [2] M. Butler, et al. Single instruction stream parallelism is greater than two. In *Proceedings of ISCA'18*, pp. 276, May 1991.
- [3] M. Butler and Y. Patt. An investigation of the performance of various dynamic scheduling techniques. In *Proceedings of MICRO-25*, December 1992.
- [4] D. Chang, S. Mahlke, W. Chen, N. Warter and W. Hwu. IMPACT: An architectural framework for multiple-instruction-issue processors. In *Proceedings of ISCA'18*, pp. 266, May 1991.
- [5] T. A. Diep, C. Nelson and J. Shen. Performance evaluation of the Power PC 620 microarchitecture. In *Proceedings of ISCA'92*, pp. 163 June 1995.
- [6] M. J. Flynn. *Computer architecture: Pipelined and parallel processor design*. Jones and Bartlett, Boston, 1995.
- [7] S. Fu and M. Flynn. Optimal on-chip cache hierarchy synthesis with scaling of technology. In *Proceedings of IPCCC'96*, Phoenix, March 1996.

- [8] S. Jourdan, P. Seurat, and D. Litaige. Exploring configurations of functional units in an out-of-order superscalar processor. In *Proceedings of ISCA'92*, pp. 117. June 1995.
- [9] G. McFarland. *CMOS technology scaling and its impact on cache delay*. PhD thesis, Stanford University, 1997.
- [10] Kevin Nowka. *High performance CMOS system design using wave pipelining*. PhD thesis, Stanford University, August 1995.
- [11] K. Olukotun, B. Nayfeh, L. Hammond, K. Wilson, and K.-Y. Chang. The case for a single chip multiprocessor. In *Proceedings of ASPLOS VII*, pp. 2–11, October 1996.
- [12] A. Saulsbury, F. Fong, A. Nowatzky. Missing the memory wall: The case for processor/memory integration. *Proceedings of ISCA'96* pp. 90–100.
- [13] Semiconductor Industry Association. The National Technology Roadmap for Semiconductors. San Jose, CA, 1994.
- [14] A. J. Smith. Cache memories. *ACM Computing Surveys*, 14(3):473–530, September 1982.
- [15] D. C. Wong. *Techniques for designing high-performance digital circuits using wave pipelining*. PhD thesis, Stanford University. August, 1991.
- [16] W. Wulf and S. McKee. Hitting the memory wall: Implications and the obvious. *ACM Computer Architecture News* vol. 13 no. 1, March 1995.