An Evaluation of the Design of the Gamma 60

T.J. Tumlin, Mark Smotherman
Department of Computer Science
Clemson University
Clemson, SC 29634-1906 USA

Abstract

The Bull Gamma 60 remains a major innovation in computer design. Its use of explicit FORK-JOIN parallelism is shown by a simulation model to wisely exploit a large difference in speeds between logic components and memory elements, as found on some machines of the 1950's. Recently the reappearance of a large speed ratio makes the same type of explicit FORK-JOIN parallelism attractive in advanced designs and validates the latency-tolerant design philosophy of the Gamma 60. The major difficulty of the design is the programming effort required to fully express the parallelism available in programs.

1 Introduction

Compagnie des Machines Bull of France announced the Gamma 60 in 1958, and a total of nineteen systems were built [KUHN90]. The logical designers were Pierre Chenus, Jean Bosset, and J.P. Cottet, and the project manager was Bruno Leclerc [DAT58]. The machine was a unique design incorporating explicit FORK-JOIN parallelism in its instruction set. This made it one of the first multiprocessors and the first design to be composed of multiple functional units and multiple I/O processing units. The design was the first to hardware time-share a central processor control unit for asynchronous parallel processes.

In the Gamma 60 design, memory is the fastest unit so parallelism is at the instruction level. Each complete instruction, rather than each logical collection of work (i.e., program or subroutine), can be defined to be a separate thread of execution that the machine initiates under program control and then coordinates. A central memory and control unit is used to rapidly supply the slow processing units with data and commands as each thread finishes an operation and its processing unit becomes available. Duverger describes this design:

> Le Gamma 60 a été conçu à une époque où bien des ensembles à traiter l'information considéraient l'organ de calcul comme le centre moteur: tout transit l'utilisait. La grande révolution qui s'est produit ces dernières années consiste à considérer cet organ de calcul comme un élément périphérique, au même titre que les entrées et sorties, et à donner le rôle de dispatching a la mémoire centrale. Il s'ensuit que le Gamma 60 est divisé en deux parties: la mémoire centrale et les éléments périphériques. Chaque élément peut entre en conversation avec la mémoire centrale, deux éléments ne peuvent converser ensemble sans recoiurir à la mémoire central. [DUV61]

There is also in this paper a recognition that best performance can only be obtained from a carefully tailored program [DUV61]):

> Dire que les éléments du Gamma 60 peuvent travailler simultanement signifie qu'à certains niveaux du programme celui-ci cesse d'être <<filiforme>> pour acquérir une structure <<maillée>>.

Better performance was also recognized to result from multiple concurrently executing programs [DAV60]

> Non seulement on obtient ainsi une optimimisation du temps de deroulément d'un programme, mais on peut traiter sur la même machine plusieurs problèmes indépendants, utilisant des problemes différents, les programmes se développant simultanément: l'utilisation maximale des circuits électroniques est assurée, par l'obtention du rendement maximal de la Mémoire Centrale, nœud de toute la machine.

The speed difference between logic and memory that allowed computation to be treated as a "peripherial unit" did not remain static but rapidly changed in the late 1950's and 1960's. Thus the Gamma 60's design was not as efficient or as easy to program as register-based sequential machines for logic and memory of similar speeds, and the register-based machines became the standard design up through today. However, the speed difference has continued to change and now memory is slow in comparison to arithmetic and logic operations. Serving as an example of current work in multi-threaded architectures, the P-RISC, for "Parallel RISC," architecture from Massachusetts Institute of Technology has many processing elements connected to each other and to memory across an interconnection network. For this design each non-local memory access is defined to be a separate thread of execution that the machine has to initiate under program control and then coordinate.

These two machines, the Gamma 60 and the P-RISC, have an inverse relationship in their ratio of logic speed to memory speed. However, both use explicit parallelism to tolerate long latency operations. We present a simulation of both machines to identify their bottlenecks and to explore the effects of the overhead involved in using parallel operation of multiple processing/memory units at the instruction level. We believe that these simulations show that the Gamma 60 is a well-designed machine for the given technological ratio, as is the P-RISC for the current-day technological ratio.

2 The Gamma 60

The Gamma 60 can be described as a multiple computer system since it contains a collection of autonomous processors connected to a central control unit. It allows for up to nine clusters of five processors each. Thus one can imagine a fully laden Gamma 60 with five clusters of five ALU's each for a total of twenty-five ALU's. The other four clusters would be used by 1) the logical calculator and control desk, 2) magnetic memories other than the central memory, 3) punched or printed card readers, and 4) tape readers and perforators.

2.1 Types of Processing Units

The Arithmetic Calculator executes the four basic arithmetic functions on numbers in BCD and floating-point format while the Logical Processor (ALU) performs arithmetic operations in binary. The Arithmetic Calculator also performs all logical operations such as logical AND. The Control Desk is the computer console. The Comparator compares variable length alphanumeric operands and the Translator converts data between the internal Gamma 60 format and the input/output devices. The Magnetic Drum externally stores up to 25K of words.

2.2 The Central Unit

Each of the processing units has an identical instruction request interface to a single supervisory control unit, which is composed of two subunits: the Transfer Distributor and the Program Distributor. The control unit is connected to the clusters of processors by the Data Collecting and Data Distributing buses. The Transfer Distributor transfers data to or from the high-speed memory while the Program Distributor distributes instructions to the processing units. Both units work similarly. They receive requests for data/instructions from the processing units, select a request to honor by internal priority circuits while placing the other requests on a queue to handle, and then perform the transfer. For data, the Program Distributor must compete as do all other units for the Transfer Distributor's time. Priority is based on the speed of the requesting processing unit. The Program Distributor has the lowest priority so that it doesn't interfere with the execution of any of the threads. A magnetic drum would have its request honored before an Arithmetic Calculator since the latency of the drum is much longer, so if its request is not honored first the mean time for fulfilling both requests would be longer.

It should be noted that the processing units request data and instructions from the distributors. They are autonomous and several parts of the same program or several parts from different programs can execute concurrently.

2.3 High-Speed Memory

The central memory contained 32K 24-bit words (called "catena") for a total of 786,432 bits. It is connected to the Transfer Distributor and both the Data Collection and Data Distribution busses. The average random access time is 10 microseconds. The central high-speed memory acts as a crossroads between the various functional units. No unit can communicate with another without going through the central memory. In each cycle, the memory transfers one catena to or from a processor.

2.4 Data

Data is stored in groups of 24 bits (catena) homogeneously as six BCD numerals or four alphanumeric symbols. Each cantena is also accompanied by a control key of three bits for data integrity. The choice for word size was based upon memory transfer rate.

2.5 Instructions

The Gamma 60 contains five types of instruction words: A) address, B) branch, C) cut, D) directive, and E) blank. A set of these words represents a complete instruction, which must always end in a directive.

The cut word is like an interrupt in that it designates a processing unit and initiates its operation. It indicates that a different processing unit should be used while a branch word changes the flow of control of the program but continues with the same unit. The cut word can also be considered as both a FORK and a JOIN. It designates a new processor and how many calls to that new processor must occur before continuing execution with the following instruction. If it has only one call it acts as a FORK, and in this mode the cut word is called by the special name of SIMU. In this case, the cut word does not mobilize a processing unit; rather, it transfers control to another instruction so that two instruction can execute simultaneously. A cut word consists of two additional bits so that up to four threads of execution can be joined by the cut (a one (1) is automatically added to the number so that a three (3) would actually mean wait for four branches to the cut before continuing execution). Joining more threads requires additional cuts. Address words represent the addresses of data in the high-speed memory and a blank word has no other effect than to provide a location for queuing.

2.6 Shared Memory and Virtual Units

Subroutines can be shared between processing units. To provide mutual exclusion, shared subroutines are considered virtual units and a cut word is used to access the routine. Other threads of execution requesting the shared routine are placed on a queue for execution as soon as the routine is freed.

2.7 Example of Execution

For an example of execution consider the following high-level code segment:

$a = b * c$
$d = a + d$

In the Gamma 60 execution of the code would take place as following:

```
[C] cut to multiplication unit:
    [A] address (load) b
    [A] address (load) c
    [A] address (store) a
    [D] directive (mult)

[C] cut to arithmetic unit:
    [A] address (load) a
    [A] address (load) d
```

```
        [A] address (store) d
        [D] directive (add)
```

Both instructions would be in a single thread and execute sequentially. Since in the Gamma 60 memory speed is assumed to be much greater than logic speed, a separate thread could begin execution.

An example of concurrent execution in the same thread on the Gamma 60 would be:

$a = b * c$
$d = d + e$
$f = a * d$

which would be executed as:

```
    [C] cut (fork or SIMU) to I2
    I1: a = b * c
    [B] branch unconditionally to I3
    I2: d = d + e I3:
    [C] cut (join) two calls
        f = a * d
```

Executing instructions from separate threads is handled just as easily. The Program Distributor sends another processor an instruction to execute upon the request of the idle processor. Synchronization of data is obtained by the fact that each instruction is its own thread executing on a single processor. Thus data is never corrupted between threads as the Transfer Distributor knows the originator of the data request. Since the Program Distributor handles all synchronization, the number of threads allowed to execute concurrently is not limited by anything other than memory space.

2.8 Reception of the Design in the USA

The Gamma 60 generated significant interest in the United States among machine designers. After announcing the system at the 1958 Western Joint Computer Conference in Los Angeles, California (and the Eastern Joint Computer Conference in 1959), Phillippe Dreyfus presented the design in summer courses at the University of North Carolina at Chapel Hill in 1959 and then at the University of Michigan at Ann Arbor in 1961.

A.J. Critchlow of the IBM Corporation presented a status report on multiprocessing and multiprogramming at the 1963 AFIPS Fall Joint Computer Conference and featured the Gamma 60 as one of four advanced computer systems [CRI63]. His assessment:

> Problems of loading and scheduling are critical in the success of such a system. A very large number of problems is required to produce a good statistical mix so units can be efficiently used.

In the 1970's, Caxton Foster of the University of Massachusetts at Amherst also featured the Gamma 60 as an example of advanced computer design in his 1970 text on Computer Architecture [FOS70]. M. Bataille also wrote an impressive article on the design of the Gamma 60 for the Honeywell Computer Journal in 1971 and includes the quote from Professor Saul Gorn [BAT71]:

> If we can extrapolate the trends in the development of programming languages, I would say that the following is our goal... It should be capable of expression in parallel, in order to match the truly parallel machines we can expect as the next generation, and of which the Gamma 60 is a prototype.

2.9 Design Critique

The definition of single instructions as independent threads of execution in the Gamma 60 results in tremendous supervisory overhead in the machine and a large investment in man-hours to determine the most efficient program structure.

This definition naturally results from the memory-oriented design, as opposed to the now-traditional CPU-oriented design. The designers assumed that memory was much faster (10 microsecond cycle time) than any of the processing units (200 microsecond additions and 350 microsecond multiplies), and everything possible is done to use available memory cycles and overlap all the slow logic operations.

By attempting to meet these goals by using parallel operation of multiple processing units at such a low level, many memory cycles are necessarily devoted to the supervisory overhead of threads switching back and forth between units and often many cycles go idle due to congestion at critical, bottleneck units (c.f., Critchlow's recognition of the need for a proper workload mix which would releave the programmer of the burden of finding the most parallelism within a given program and substitutes instead parallelism between multiple programs). In summary, as logic speeds became more commensurate with memory, there was no longer the ability to perform 10-50 memory cycles per logic operation. Thus the single, central control unit became an unavoidable bottleneck, and even the Gamma 60's advanced design (called "une machine <<ouverte>>" in the conclusion of [DUV61]) could not easily adapt to the closing of its technological window.

3 Multithreaded Architectures

3.1 A Recent Design: The P-RISC

The P-RISC, for "Parallel RISC", was developed at the Massachusetts Institute of Technology. It has many processing elements connected across an interconnection network to each other and to memory elements. For it, memory is slow in comparison to arithmetic and logic operations; therefore, each non-local memory access is defined to be a separate thread of execution that the machine has to initiate under program control and then to coordinate. Thread initiation, as in the Gamma 60, is done using a FORK mechanism.

The P-RISC developed as an extension to RISC with fine-grained dataflow capability. Although the processing and memory elements are assumed to be distributed, a global address space for all memories is used. Each processing element runs a separate thread of execution with all arithmetic operations local to the unit. Data is moved only via load and store instructions. [NIK89]

For an example of execution in P-RISC, consider the same code segment used previously:

$a = b * c$
$d = d + e$
$f = a * d$

The first multiply instruction is sent to one unit while the second instruction is sent to a separate functional unit for execution. Each unit requests the loading of its operands and when each finishes their execution of its respective operations the third instruction starts.

The effectiveness of the P-RISC is best seen when it is concurrently executing several threads of execution and when the latency for memory accesses is great in comparison to the time required to perform logical operations. Thus if the memory latency is sufficiently long, the processor can start another thread while waiting for the return of data from a load. So, in the previous example, the processor makes requests for both of the operands for the first instruction then while waiting on the return starts the execution of the second instruction by requesting its operands, or it switchs to another thread.

Threads in P-RISC are initiated with a START instruction which automatically follows every load. As soon as the load is finished, the "loading" thread is allowed to continue execution. Data synchronization is provided with a structure similar to "I-Structures" in Monsoon in that each memory access is accompanied by the information of which thread produced it. As with the Gamma 60, the number of threads executing concurrently is not limited.

3.2 Other Examples of Multithreaded Architectures

A number of multithreaded architectures have appeared during the years. All share a common goal of using parallel processing to tolerate long latency operations.

The Honeywell H-800 [FOS70] is a late 1950's machine that implements a hardware controlled multiprogramming scheme for eight different programs. These programs are in strict priority order, and the highest priority ready program is allowed to completely execute one instruction. This design is an attempt to make I/O-bound programs the highest priority programs and cause the hardware to automatically tolerate long I/O operations.

The Control Data CDC-6600 supercomputer designed in the early 1960's uses hardware-time-sharing for its peripheral processing units [FOS70]. The memory for these units requires 100 microseconds while a CPU operation requires 10 microseconds. Thus ten PPUs are rotated in a "barrel" processor, with each virtual PPU allowed one CPU operation ("minor cycle") per 100 microseconds ("major cycle"). Note that one of these CPU operations can be a memory operation initiation; this memory operation will complete by the time the requesting PPU gets its next turn.

The Denelcor HEP, designed by Burton Smith in the 1970s, can handle up to 64 threads [SMITH78]. Multithreading is accomplished by switching threads on each clock pulse. Once a load is encountered, the thread is removed from the queue of executing threads and placed on another queue of threads awaiting memory access. In this way, the long latency loads do not hinder overall execution. The Tera machine is a continuation of this design style [ALV90].

A dataflow machine using these concepts is the Monsoon, which, like P-RISC, is a collection of pipelined processing elements. Up to eight threads can be simultaneously processed per processing element, as in the H-800. Data synchronization is accomplished by use of an Explicit Token Store Architecture, which stems from the Tagged-Token Dataflow Architecture. Its design goal was to simplify the matching of operations with operands. [PAP90]

4 A Comprehensive Machine Simulator

The simulator used is parametized to handle different latencies for memory access, arithmetic operations, synchronization operations, and communication overhead. With it one writes a program based upon the machine used, in this case either a register-based von Neumann, the Gamma 60, or the P-RISC. To handle multiple threads, a queue of virtual memories is maintained and used for each new thread. The output of the simulator is the order of execution and the cycles required.

The simulator was designed to model machine behavior and thus the input programs do not match the proper programming of the respective machines, but rather gives us a model of the behavior of programs on the respective machines.

4.1 Example inner loop benchmark:

$C[i] = A[i] * B[i] + k$

Note:

l = load	s = store
m = multiply	a = add
c = communication overhead	f = fork
j = join	

register based		Gamma 60		P-RISC	
Logic	Representation	Logic	Representation	Logic	Representation
mult l r	l	load	l	fork	f
add l r	l	load	l	load	c
store	m	mult l r	m	join	l
	a	store	s	mult l r	f
	s	load	l	add l r	c
		load	l	fork	l
		add l r	a	store	j
		store	s	join	m
					a
					f
					c
					s
					j

This Gamma 60 representation does not show the FORK and JOINS inherent in the programming of the Gamma 60. When programming multiple loops, as in our inner loop benchmark of 300 loops, we can FORK two multiplies at the same time (or as many multiples as we have Logical Calculator units available) and then proceed with the adds since all multiplies are independent. We show the P-RISC as having two loads and corresponding communications overhead for each multiply. This represents having operands located in different distributed memory modules. To represent operands located in the same memory module one would just use an average communications overhead instead of the full cost.

4.2 Limitations of Simulation

One limitation of our simulator is that it does not consider local caches (which could be exploited by register machines and the P-RISC). But with local caches comes the need for data coherency which becomes problematic combined with multithreading. Furthermore, one can consider local caches as an average lower memory to logic ratio. With this in mind, we feel that our simulator does model machine behavior and performance.

We also did not allow the machines to multithread between programs as this would cloud the results. To multithread, all that would need to be done is to assign a context-switch penalty for the register based machines as both the Gamma 60 and P-RISC multithread naturally as part of their machine designs.

5 Comparisons

We simulated the execution of 300 of the inner loops used above. The memory to logic ratio was changed for each test starting with a ratio of 1:300 and ending with 100:1 for a total of 13 tests. Memory access is considered as the time needed before a memory request can be fulfilled. It includes both memory access time and communications time. For the Gamma 60 and register based machines, the memory access time was changed, while for the P-RISC, the communications overhead was changed after 1:1 ratio was reached. This method illustrates the model of execution better. With varying the memory/logic ratios, we can see when an architecture is a good design choice for a given technological time period.

Figure 1 represents the performance of the three machine designs with varying memory to logic ratios. At the far left we start with the lowest ratio of 1:300 as seen in the RCA 501 and Burroughs 200 and end with a 100:1 ratio which could be possible with the increasing speeds of logic and distributed nature of memory being seen now. As can be seen the Gamma 60 performs as well as the P-RISC until memory speeds reach the 1.5:1 ratio. The machines are constrained to single-unit execution. In Figure 2 we let the Gamma 60 and P-RISC have multiple functional units.

When allowing the machines to have multiple functional units, performance is seen to greatly improve for slow memory, as could be predicted. As the ratio between memory and logic increases to 2:1 the P-RISC starts to show its

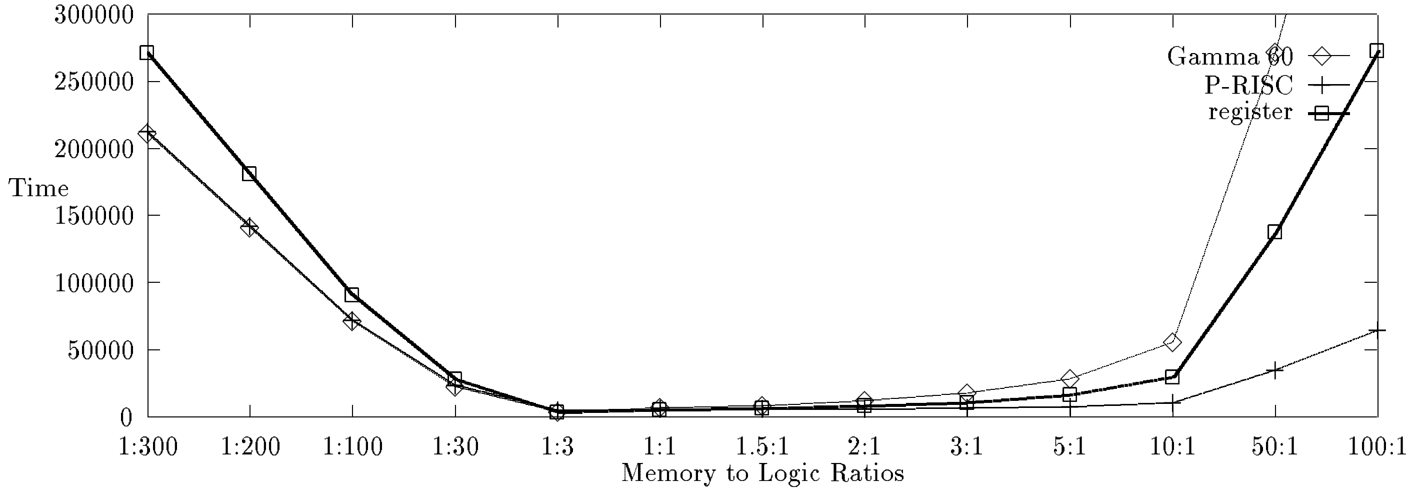

Figure 1: Single Unit Execution

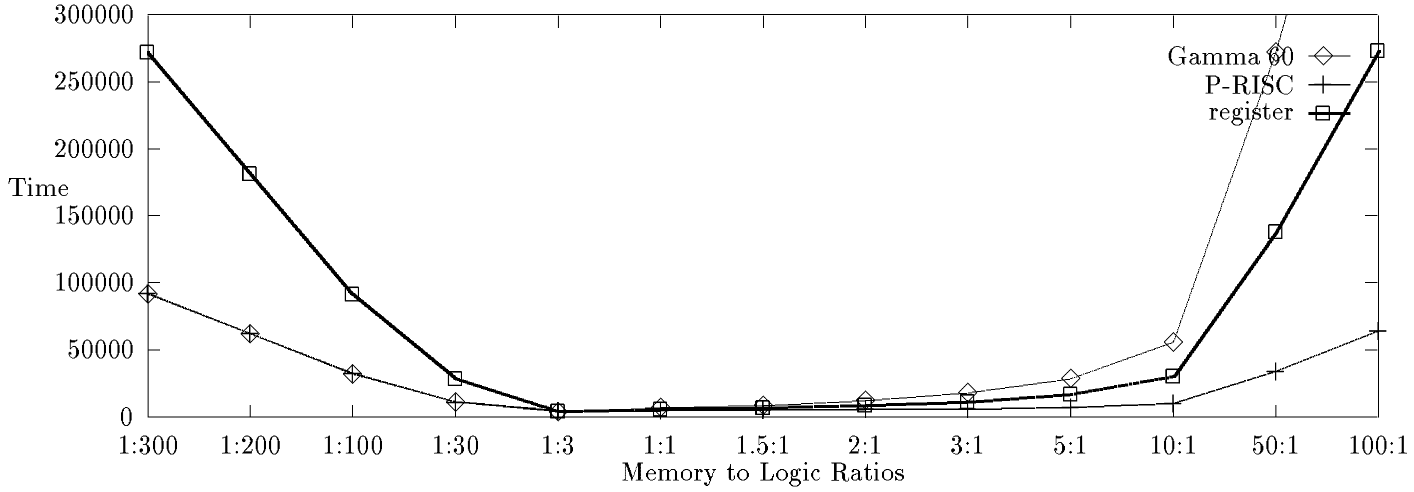

Figure 2: Multiple Unit Execution

performance. Once memory becomes so slow, the Gamma 60 doesn't share in the improvement associated with having multiple functional units since the machine was designed with fast memory in mind. Therefore, a bottleneck develops and only one functional unit is useful. Even the P-RISC shows only a 2 percent increase in performance. This is a key to understanding the importance of multithreading as told by Critchlow. We should not forget that a register-based machine can also have multiple functional units as in recent superscalar architectures.

Looking at the performance of past machines in relation to their memory/logic ratios, we see that the Gamma 60 is quite the norm. The performance measurements are from Phister and Mounier-Kuhn [PHI79], [KUHN90]. Speeds are in microseconds (μs).

Year	Machine	Add	Mult	Memory	A/Mem	M/Mem	Avg Ratios	Price	Price(Avg Ratio)
1951	Univac I	525	2150	220	2.39	9.77	6.08	750	123.36
1954	B 205	1100	9300	8500	0.13	1.09	0.61	135	220.67
1954	IBM 650	5200	11600	2400	2.17	4.83	3.50	157	44.86
1955	IBM 704	24	240	12	2.00	20.00	11.00	450	40.91
1955	IBM 705	85	749	17	5.00	44.06	24.53	562	22.91
1957	Univac II	120	1800	40	3.00	45.00	24.00	970	40.42
1958	B 220	200	2070	10	20.00	207.00	113.50	370	3.26
1958	Univac SS80	1300	1275	1700	0.76	0.75	0.76	110	145.24
1958	IBM 709	24	132	12	2.00	11.00	6.50	500	76.92
1959	RCA 501	330	5700	15	22.00	380.00	201.00	320	1.59
1959	IBM 7090	4.32	17.4	2.18	1.98	7.98	4.98	818	164.20
1960	CDC 1604	7.2	29.2	4.8	1.50	6.08	3.79	700	184.62
1960	**Gamma 60**	**200**	**350**	**10**	**18.18**	**31.82**	**25.00**	**720**	**28.80**
1960	H 800	18	200	6	3.00	33.33	18.17	410	22.57
1960	IBM 1401	230	2085	12	19.17	173.75	96.46	71	0.74
1961	B 200	414	3348	6	69.00	558.00	313.50	225	0.72
1962	IBM 7094	4	10	2	2.00	5.00	3.50	368	105.14
1962	Univac III	8	80	4	2.00	20.00	11.00	390	35.45
1963	CDC 3600	2.1	4.3	1.5	1.40	2.87	2.13	911	427.03
1964	B 5500	1	32	4	0.25	8.00	4.13	488	118.30
1964	CDC 6600	0.333	1	1	0.33	1.00	0.67	3450	5176.29
1965	IBM S/360-5	4	28.8	2	2.00	14.40	8.20	409	49.88
1965	IBM S/360-6	1.4	4.8	0.75	1.87	6.40	4.13	960	232.26

6 Conclusions

The Gamma 60 and the recent P-RISC both exploit parallelism at the instruction level to tolerate long-latency operations. In the Gamma 60, each instruction is its own thread in order to tolerate long-latency logic operations, and in the P-RISC each memory operation is its own thread in order to tolerate long-latency memory. These threads can be done simultaneously with other units from other threads of computation.

A simulator has been used to investigate performance of different design styles over various technological ratios between memory and logical operations. We feel that the results show that the Gamma 60 is a well-designed machine for the assumed technological ratio. However, a careful assessment of the machine indicates that the design was not flexible enough to adapt to changes in technological ratios and that the choice of explicit parallelism required the computer manager to carefully balance the workload and the programmer to carefully craft the individual programs to gain the best performance.

7 Acknowledgment

We would like to thank Pierre-E. Mounier-Kuhn for his significant contributions in time, advice, and literature towards this paper. Without his constant help, we would not have been able to fully envision this work.

8 Bibliography

[ALV90]
R. Alverson, et al., "The Tera Computer System," International Conference on Supercomputing, June 1990, pp. 1-6.

[ARC]
"The Gamma 60," Archives Historiques Bull, vol 60, pp.105-150.

[BAT71]
M. Bataille, "The Gamma 60: The Computer that was Ahead of its Time," Honeywell Computer Journal, vol. 5, No. 3, 1971, pp. 99-105.

[CRI63]
A.J. Critchlow, "Generalized Multiprocesing and Multiprogramming Systems: Status Report," Proceedings of AFIPS FJCC, vol. 24, Las Vegas, November 1963, pp. 107-126.

[CUR64]
W. A. Curtin, "Multiple Computer Systems," Advances In Computers, vol 4, New York: Academic Press, 1964, pp. 245-303.

[DAT58]
"France's Gamma 60: A step forward in data processing?" Datamation, vol. 4, No. 3, May June 1958, pp. 34-35.

[DAV60]
Davous, Bataille, Harrand, "Le Gamma 60," L'Onde Electrique 40, No 405, December 1960, pp. 889-919.

[DRE58]
P. Dreyfus, "System Design of the Gamma 60," Proceedings of the Western Joint Computer Conference, Los Angelas, CA, 1958, pp.130-183.

[DRE59]
P. Dreyfus, "Programming on a Concurrent Digital Computer," Frontier Research on Digital Computers, vol. 1, University of North Carolina Summer Institute, section V, 1959, pp. 1-49.

[DREY59]
P. Dreyfus, "Programming Design Features of the Gamma 60 Computer," Proceedings of the Eastern Joint Computer Conference, 1959, pp.174-181.

[DUV58]
L. Duverger, "Le Gamma 60 Bull," Automatisme, vol. 3, No. 12, December 1958.

[DUV61]
L. Duverger, "Logique et Fonctionnement du Gamma 60," Revue Generale de Chemin de Fer, March-April 1961, pp. 25-44.

[FOS70]
C.C. Foster, Computer Architecture. New York: Van Nostrand Reinhold, 1970.

[KUHN90]
P.-E. Mounier-Kuhn, "Specifications of Twelve Early Computers Made in France," Annals of the History of Computing, vol. 12, No. 1, 1990.

[KUHN92]
P.-E. Mounier-Kuhn, "Product Policies in Two French Computer Firms: SEA and Bull (1948-1964)," Global Perspectives on Business Information Conference, 24-25 April 1992.

[LEC88]

B. Leclerc, “Le Gamma 60: Une Aventure Humaine et Technologique,” Colloque sur L’Histoire de L’Informatique en France, 3-5 May 1988.

[NIK89]
R. S. Nikhil, Arvind, “Can Dataflow Subsume von Neumann Computing?” The 16th Annual International Symposium on Computer Architecture, Jerusalem, Isreal, May 1989, pp. 262-272.

[PAP90]
G. M. Papadopoulos, David E. Culler, “Monsoon: an Explicit Token-Store Architecture,” The 17th Annual International Symposium on Computer Architecture, Seattle, Washington, 28-31 May 1990, pp. 82-91.

[PHI79]
M. Phister, Jr., “Data Processing: Technology and Economics,” 2nd Ed., Santa Monica Publishing and Digital Press, 1979.

[SALLE63]
F. Salle, J. Newey, “Specification d’un Systeme de Programmation et d’Utilisation pour un Calculatuer Scientifique de Grande ou Moyenne Puissance,” 3rd Congress de Calcul et de Traitement (AFCALTI),” Dunod, Paris, 1963, pp. 14-21.

[SMITH78]
B.J. Smith, “A Pipelined Shared Resource MIMD Computer,” Proceedings of International Conference on Parallel Processing, 1978, pp. 6-8.