INFLUENCE OF INPUT/OUTPUT OPERATIONS ON PROCESSOR PERFORMANCE

JOSE MARIA RODRÍGUEZ CORRAL
Lenguajes y Sistemas Informáticos, Universidad de Cádiz,
Escuela Superior de Ingeniería, C/Chile 1, 11003 Cádiz, Spain

ANTON CIVIT BALCELLS, GABRIEL JIMENEZ MORENO
and JOSE LUIS SEVILLANO RAMOS
Arquitectura y Tecnología de Computadores, Universidad de Sevilla,
Escuela Superior de Ingeniería Informática,
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain

ARTURO MORGADO ESTEVEZ
Ingeniería de Sistemas y Automática, Tecnología Electrónica y Electrónica, Universidad de Cádiz, Escuela Superior de Ingeniería,
C/Chile 1, 11003 Cádiz, Spain

Nowadays, computers are frequently equipped with peripherals that transfer great amounts of data between them and the system memory using direct memory access techniques (i.e., digital cameras, high speed networks, . . .). Those peripherals prevent the processor from accessing system memory for significant periods of time (i.e., while they are communicating with system memory in order to send or receive data blocks). In this paper we study the negative effects that I/O operations from computer peripherals have on processor performance. With the help of a set of routines (SMPL) used to make discrete event simulators, we have developed a configurable software that simulates a computer processor and main memory as well as the I/O scenarios where the periph-erals operate. This software has been used to analyze the performance of four different processors in four I/O scenarios: video capture, video capture and playback, high speed network, and serial transmission.

Keywords: Direct memory access; input/output; performance; real time; superscalar processors.

1. Introduction

In a few years processor architecture has experimented a great advance. After sequential processors pipeline processors[1] were designed and then superscalar, superpipeline[2,3] and VLIW[4,5] processors. Furthermore, instruction sets have evolved with the development of new processor architectures and these facts have lead to the Reduced Instruction Set Computers (RISC) processors.[6,7]

On the other hand, I/O devices have evolved in order to get faster so as not to penalize computer performance. Furthermore, new system buses[8–10] have been developed, with a higher bandwidth in order to improve data transfer speeds among computer devices (processor, memory, and I/O peripherals). Memories[11] and caches[2,4,12] have evolved too. Finally, technological advances will allow to increment the complexity of the chip and its working frequency. Thus, there exist several research lines dedicated to the design of complex uniprocessor architectures,[13] as trace, multiscalar and datascalar processors.

However, the influence of I/O device operations on processor performance is not so studied, although there is some research work done about it.[14,15] Thus, as peripherals become faster and are able to transfer large blocks of data directly to system memory, the influence of I/O operations on processor performance gets more importance and thus, it must be seriously analyzed.

Thus, in order to study this negative effect, we will develop two simulation programs.[16] The first one (processor and system memory simulator) will be configured with a set of typical parameters for each type of processor. The second one generates a set of I/O scenarios for the different processor and system memory models selected. Then, once the sample code fragment have been chosen, the simulation result will depend on two variables: the processor type and the I/O scenario used.

2. Simulator Design

At present, there exist many simulators and simulation languages. The election of the most suitable,[17] as its properties fit the nature of the system to model, is an important decision that can ease in a great manner the latter work, including the obtaining of results. Whereas a simulation or a general purpose language allows to develop any necessary tool (though it takes a long time and requires some experience), a simulator looks like a box with different tools, some of them more flexible than others. However, the flexibility of a simulator is not as great as that of a simulation language, though the first option allows to save a great amount of development time.

We have chosen the simulation language SMPL,[18] which is a set of C functions used to build discrete event simulators. First, this type of simulation seems to be the most suitable than continuous simulation due to the nature of the system to model, based on synchronous sequential circuits (main processor, DMA controllers, and peripherals), where the time unit is the clock cycle. There also exist processor simulators that are more recent and have many interesting features, as SimpleScalar,[19] HASE,[20] RSIM,[21] POWER RISC,[22] or CASTLE.[23] However, many of them do not allow to include external elements to the processor, and the simulators that allow it (as SimpleScalar, HASE, and RSIM) do not include peripheral modeling, whose behavior must be simulated in order to analyze the influence of their I/O operations on the system processor performance, which is the objective of our study. Finally, SMPL complies with an important number of requirements that are relevant in

order to choose a simulator or a simulation language[17] and recommend its use, as flexibility and ease of use, portability, simulation tracing, optional interactive mode, random number generator, statistical facilities, independent replication generator, report generator, and an acceptable documentation.

Once the simulation tool has been chosen, we must define the processor model that will be implemented by the simulator to develop by using SMPL. Figure 1 shows the basic processor structure: it is a sliding window superscalar 32-bit RISC processor with an instruction cache, a nonblocking data cache and a branch target buffer. The only restriction is static scheduling. We have chosen this option as a starting point, as present work is only a first step in the study of the influence of I/O operations on processor performance. Furthermore, there exists a lot of embedded systems based on processors with static scheduling.[24,25]

Figure 2 shows a hierarchical diagram with the most important modules of the processor and system memory simulator.[16] This diagram reflects the modeled processor internal architecture. `inidata` ( ) function initializes all the simulator data structures, `readfiles` ( ) function reads the files with the sample program assembler code, the memory block trace (corresponding to a concrete I/O scenario and generated by the I/O subsystem simulator) and the instruction issue patterns, respectively. `pipeline` ( ) function request system processor, main memory, and cache parameters, initializes SMPL facilities and starts the execution of the simulation loop. Finally, `main` ( ) function shows the simulation results: the number of instructions executed, the total simulation time, the instruction execution mean time and, if requested by user, the timetable corresponding to the instruction stream execution, generated by `chrono` ( ) function.

ID_stage ( ) issues n instructions in a cycle from the instruction queue to the respective functional units according to the instruction issue patterns, the absence

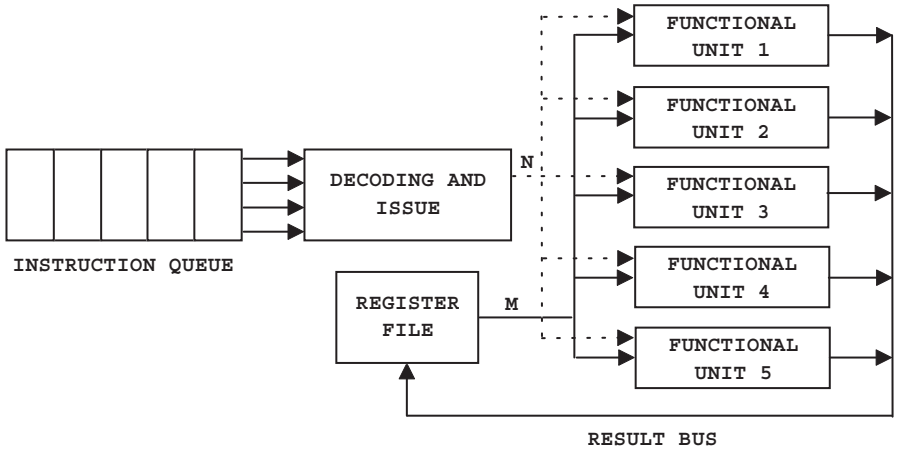

Fig. 1. Processor structure.

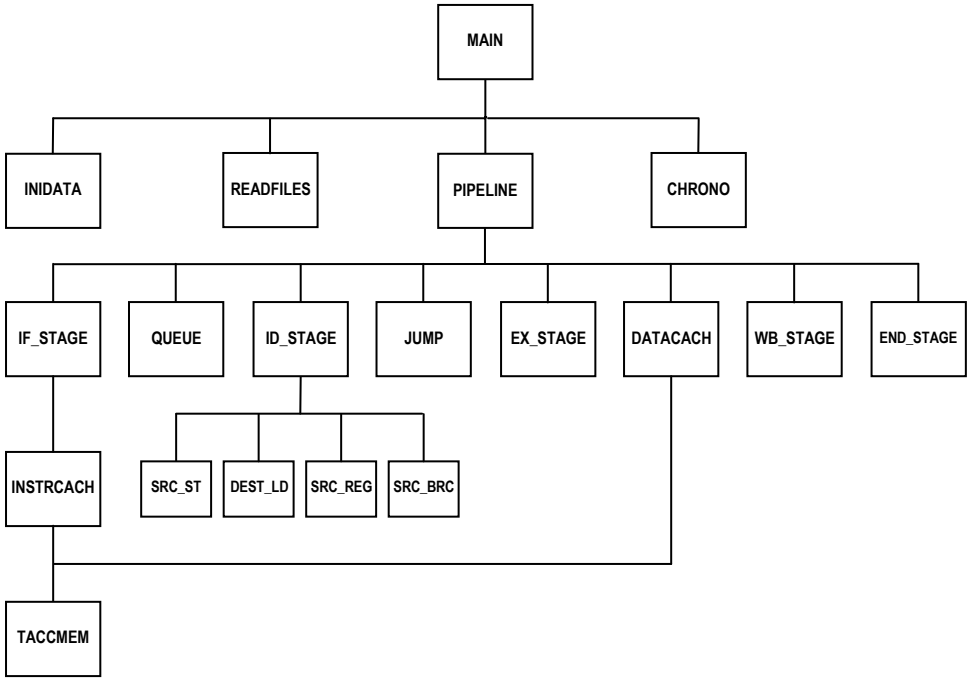

Fig. 2. Processor and system memory simulator diagram.

of data dependences for every instruction to be issued on the ones that are already being executed and the absence of a structural stall, which prevents an instruction from being issued. This function calls another four (src_st( ), dest_ld( ), src_reg( ), and src_brc( )) in order to check that the current instruction to be issued has no Read-After-Write (RAW) or Write-After-Write (WAW) data dependences on any issued instruction that is being executed.

An essential aspect to consider when modeling a processor architecture consists of exception handling.[4,26] In one of the proposed I/O scenarios (serial transmission), the I/O peripheral (serial port) must interrupt the processor at certain moments. When our processor model detects that a peripheral is requesting an interrupt (IRQ), it stops issuing sample program instructions until the interrupt has been handled. Meanwhile, since the sample program point of view, the instruction queue output remains blocked, whereas those of its instructions that were already issued are finishing their execution. Thus, from the IRQ detection to the completion of the interrupt treatment, we assume that the instructions of the peripheral Interrupt Service Routine (ISR) are being executed and so they are passing through the different processor pipeline stages.

The simulator software source code[16] mainly consists of the SMPL functions file (`smpl.c`)[18] and two C files named `simpipe.c` and `auxpipe.c`. The second of this two files is a set of auxiliary functions used by the main file.

In relation to the sample program for the experiments, we have chosen the code fragment corresponding to the key step in gaussian elimination.[4] As this code uses double precision variables, it is commonly named DAXPY because of the arithmetic operations that it performs.

```
  i = 0;
  do {
  y[i] = a * x[i] + y[i];
  i++;
} while (i < 1000);
```

For our purposes, we use a generic RISC assembler which is loosely based on the DLX assembler.[4] In the assembler version of DAXPY loop, the END pseudoinstruction finishes program execution once it has left the processor ID phase and the previous instructions have completed their execution. The STP pseudoinstruction blocks the arrival of more instructions to the processor ID phase and it is used in order to prevent nonexisting instructions from being executed, as they are not part of the sample program.

```
 0  SUB R0, R0, R0     ; i = 0
 1  LD R1, 0(R0)       ; R1 = #08
 2  LD R2, 50(R0)      ; R2 = #8000
 3  LDF F0, 100(R0)    ; F0 = a
 4  LDF F1, 8000(R0)   ; F1 = x[i]
 5  LDF F2, 16000(R0)  ; F2 = y[i]
 6  MULF F1, F1, F0    ; F1 = a * x[i]
 7  ADDF F2, F1, F2    ; F2 = F1 + y[i]
 8  STF 16000(R0), F2  ; y[i] = F2
 9  ADD R0, R0, R1     ; i++
10  SUB R2, R2, R0     ; R2 = #8000 - i
11  BRC R2, 4          ; jump if i < #8000
12  END                ; psinstr: end of program
13  STP                ; psinstr: stops ID phase
14  NOP                ; no operation
```

As our simulator models a processor with static scheduling, we have unrolled and reordered the loop instructions in order to eliminate data dependences and increase the amount of parallelism between instructions in each iteration.[4] These dependences decrease processor performance in such a way that the results obtained are not valid in order to extract conclusions about them.

In order to choose the different input parameters which will define the processor model for the simulations, we have selected a set of typical values after reviewing

the features of various well-known processors.[27–29] These parameters are as follows.

- *Frequency of operation*: 1 GHz.
- *System memory access time*: 5 ns for a SDRAM at 200 MHz.[11]
- *Probabilities for a hit on instruction and data caches*: 0.9 for the instruction cache and 0.85 for the data cache.[4,30]
- *Line size for the instruction cache*: 32 bytes (eight 32-bit instructions).[28,29]
- *Parameter for the normal distribution that generates data cache line numbers*: 500 for the mean and 250 for the variance. We consider that a data cache line is 32 bytes long[28,29] and that the two vectors of the sample program have both a size of 1000 double precision (64 bits) real elements.
- *Number of consecutive misses for the data cache without blocking*: four is an acceptable value.[4,27]
- *Instruction queue size*: 32 bytes (capacity for eight 32-bit instructions).[28]
- *Latency of floating point units*.[4,27,29] Addition (ADDF): 3'o clocks. Product (MULF): 6'o clocks. Division (DIVF): 21'o clocks.
- *Latency of memory access units*: One'o clock for calculating the access effective address and another clock for accessing the data cache.[4,29]

Finally, we have chosen four types of processors for the performance of the experiments. Thus, for each I/O scenario four results will be obtained, one for each processor type.

- *4-issue superscalar processor with infinite resources*: 12 FP addition units (12 ADDF), 24 FP product units (24 MULF), eight memory access units (8 MEM) and four integer units (4 EX).
- *4-issue superscalar processor with limited resources*: six FP addition units (6 ADDF), 12 FP product units (12 MULF), four memory access units (4 MEM) and two integer units (2 EX).
- *2-issue superscalar processor with infinite resources*: six floating point (FP) addition units (6 ADDF), 12 FP product units (12 MULF), four memory access units (4 MEM) and two integer units (2 EX).
- *2-issue superscalar processor with limited resources*: three FP addition units (3 ADDF), six FP product units (6 MULF), two memory access units (2 MEM) and one integer unit (1 EX).

3. Scenario Design

In this section, we will treat all the questions related to the I/O scenarios where our processor model will execute the sample program instructions (of course, we mean a simulated execution). These scenarios will complete the system to simulate, as processor access to main memory will be blocked during the data transfers among the I/O scenario peripherals and the memory. The I/O scenario generator software

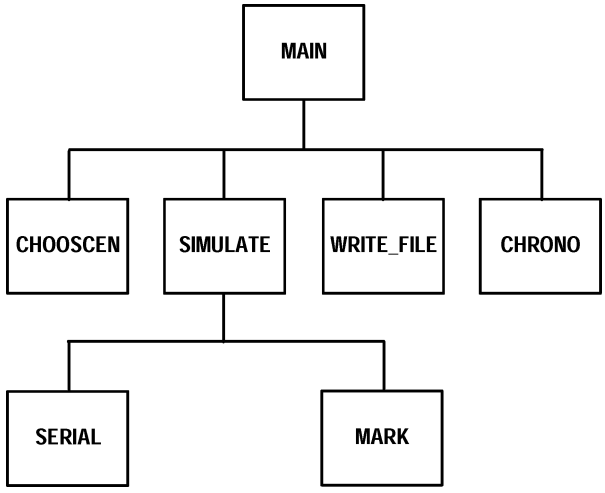

Fig. 3. I/O subsystem simulator diagram.

is provided with the selected scenario parameters, so it generates a memory block trace and, optionally, an interrupt trace if there exist peripherals that must warn the processor about events related to their operation using IRQs.

Figure 3 shows the I/O subsystem simulator diagram. `main` ( ) function initializes all the program data structures (simulation timetable, memory block, and interrupt traces). Then, it requests the trace file name and the simulation time. Next, `chooscen` ( ) function requests the user to choose the specific scenario to simulate as well as its parameters. The simulation loop resides in `simulate` ( ) function, which generates the different events related to peripheral operations. These events make the simulation to advance.

Finally, `mark`( ) function is called by simulate ( ) function in order to register the initial and the final instants of each memory block and each interrupt in the data structure that supports the traces. `serial` ( ) auxiliary function only supplies the time taken by a byte to arrive to the serial port. The writing of the memory block and the interrupt traces is performed by `write_file` ( ) function and the presentation on the monitor of the simulation timetable is carried out by `chrono` ( ) function. The execution of the latter two functions is optional, depending on whether the user wants to verify an I/O scenario simulation or to generate sample traces to use in the experiments.

The I/O scenarios in which the processor and system memory simulator will "execute" the sample program are: video capture, video capture and playback, high speed network, and serial transmission.

- *Video capture*: A digital video camera sends frames in high definition format (HDTV)[31] to a computer through a channel with an adequate bandwidth (IEEE

bus 1394).[32] As frame data arrives to main memory they are stored into the system hard disk. General scenario attributes are as follows.

— Frame resolution is 1920×1080 pixels and it has a 24 bits color.
— Vertical frequency is 60 frames/s.
— Hard disk read/write rate is 20 Mbytes/s.[33]
— The camera adapter and the hard disk controller are both connected to the system through a 66 MHz. PCI bus with a 64-bit data bus size.

- *Video capture and playback*: It is similar to previous scenario with the only difference that there is an additional peripheral. When a data block arrives to main memory from the camera adapter, it is sent to the hard disk buffer and also to the video adapter buffer in order to achieve real time playback. We assume that the video adapter is also connected to the system through a 66 MHz. PCI bus with a 64-bit data bus size.
- *High speed network*: A remote computer sends information to our system through an *Asynchronous Transfer Mode* (ATM) network.[34] As previous scenarios, the system uses the IEEE 1394 bus to communicate with other devices (i.e., video player/recorder, digital camera, ...), since it is a cheaper connection than the ATM network. Thus, there exists an ATM/IEEE 1394 bridge[32] that connects the network to the bus. As data blocks arrive to main memory, they are stored into the system hard disk. General scenario attributes are as follows.

  — ATM network transfer rate is 155 Mbits/s.
  — Hard disk read/write rate is 20 Mbytes/s.
  — The IEEE 1394 adapter and the hard disk controller are both connected to the system through a 66 MHz. PCI bus with a 64-bit data bus size.

- *Serial transmission*: The system continuously receives bytes through a serial port. Every time a data block of a certain size has completely arrived to the main memory, it is written into the hard disk. General scenario attributes are as follows.

  — The serial port is an ISA device which has a small FIFO buffer and receives data at 56 Kbits/s.[35]
  — When a byte arrives to the serial port, this one requests an interrupt. In the ISR the processor requests the data and the Host/PCI bridge also requests it to the PCI/ISA bridge, which effectively accesses the serial port. Once the PCI/ISA bridge has provided the data to the Host/PCI bridge by means of a delayed transaction,[8] then the latter bridge supplies it to the processor using a delayed response.[28]
  — Hard disk read/write rate is 20 Mbytes/s.
  — The hard disk controller is connected to the system through a 66 MHz. PCI bus with a 64-bit data bus size.

4. Analysis of Results

Let $X_1, X_2, \ldots, X_N$ be the results obtained from various simulation experiments. The parameters that are normally interesting are the mean: (i) and the variance (ii). If N is high enough, the results of Eqs. (1) and (2) are approximately equal to the really important parameters: the expected value of X (3) and its second moment, respectively.

$$\bar{X} = \frac{1}{N} \sum_{i=1}^{N} X_i \,, \tag{1}$$

$$S^2 = \frac{1}{N-1} \sum_{i=1}^{N} (X_i - \bar{X})^2 \,, \tag{2}$$

$$\mu = \lim_{n \to \infty} E(X_n) \,. \tag{3}$$

As X_i values are approximately normally distributed when N is high enough, a confidence interval for the estimated value by $\bar{X}$ is given by Eq. (4), where $t_{\alpha/2,n-1}$ is the value that leaves the $(\alpha/2 * 100)$ percent of the t-Student's distribution area on the left. Thus, $P[\bar{X} - H \leq \mu \leq \bar{X} - H]$ is equal to the confidence level $(1 - \alpha)$ for the interval.

$$H = t_{\alpha/2,N-1} \frac{S}{N^{1/2}} \,. \tag{4}$$

However, the samples (X_i) obtained from the experiments must be independent and identically distributed for the confidence interval calculated in Eq. (4) to be valid. We may use one of the following two methods,[36] which are also easy to apply.

The batch means method divides an execution in various blocks so the means obtained for each block are approximately independent. However, the means calculated in this way are not strictly independent and furthermore, estimating the necessary duration for each block is difficult.[37] On the other hand, the replication method is the simplest one and it is correct as replications are independent whenever the seed of the random number generator functions is different in each replication.

We have chosen the second method for our experiments as it seems the most suitable, resource availability is enough and execution times are not too high. Each experiment will consist of a temporal simulation of a 500 000 000 instruction execution from the sample program. As the processor model frequency is 1 GHz, we will consider that the selected amount of instructions is statistically significant. For each experiment we will make ten executions (replications) and we will state an accuracy of 10% with a confidence interval of 95%. If the desired accuracy were not achieved for an experiment, the corresponding execution lengths would be increased.

Experiment results[16] (instruction execution mean times) are shown in Table 1. For each processor, the video capture and playback scenario is the most aggressive for processor performance whereas the serial transmission scenario is the least, as it was predictable. Furthermore, the processor that gives the best result in

Table 1. Instruction execution mean times (processor clock cycles).

Processor	Ideal	Capt. and play	Capture	Network	Serial
4-issue	0.6032	0.8736	0.8123	0.6944	0.6146
4-issue and hazards	0.6311	0.8848	0.8237	0.7061	0.6370
2-issue	0.8631	1.3994	1.3047	1.1203	0.9556
2-issue and hazards	0.9861	1.4433	1.3540	1.1851	1.0317

Table 2. Slow downs (%).

Processor	Capt. and play	Capture	Network	Serial
4-issue	144.8	134.7	115.1	101.9
4-issue and hazards	140.2	130.5	111.9	100.9
2-issue	162.1	151.2	129.8	110.7
2-issue and hazards	146.4	137.3	120.2	104.6

each column is the 4-issue superscalar one with infinite resources (without structural hazards), whereas the 2-issue processor with limited resources gives the worst performance.

Table 2 shows the slow downs (i.e., the quotients between the instruction execution mean time for the sample program in each I/O scenario and the instruction execution mean time without system memory hazards due to I/O device operations) for each processor and I/O scenario. For processors with infinite resources (odd rows of Table 2), the analytical expression of the slow downs correspond to Eq. (5), where T_{ID} is the instruction execution mean time for each processor in the ideal case (see the corresponding column in Table 1) and T_{ES} is a different time for each processor and I/O scenario, which stands for the penalization due to system memory hazards because of peripheral I/O operations and must be added to the ideal time (T_{ID}).

$$S_1 = \frac{T_{\mathrm{ID}} + T_{\mathrm{ES}}}{T_{\mathrm{ID}}} \times 100\,. \tag{5}$$

In the case of the two processors with limited resources, we must consider the additional time taken, in an average term, by each instruction in executing due to the structural hazards (T_{BE}). The slow downs experimented by these processors (even rows in Table 2) respect to the ideal case (absence of system memory hazards due to I/O operations) should be expressed by Eq. (6), as the penalization due to the structural hazards (T_{BE}) must be added to both terms of the quotient stated in the equation.

$$S_2 = \frac{T_{\mathrm{ID}} + T_{\mathrm{BE}} + T_{\mathrm{ES}}}{T_{\mathrm{ID}} + T_{\mathrm{BE}}} \times 100\,. \tag{6}$$

However, the calculated results from applying Eq. (6) for the two processors with limited resources are only approximations to the slow downs in the even rows

Table 3. Errors (processor clock cycles).

Processor	Capt. and play	Capture	Network	Serial
4-issue and hazards	0.0167	0.0165	0.0162	0.0055
2-issue and hazards	0.0791	0.0737	0.0582	0.0469

```
                B B B B B B B B B B B B
                  B B B B B B B B B B D D D D D

                  C

16 (LDF)   F D M M W
17 (LDF)   F D M M M M M M M M M M M M M M W
18 (LDF)     F D D M M W
19 (LDF)     F D D D D M M W
20 (MULF)      F F D D E E E E E E W
21 (MULF)      F F F F D E E E E E W
22 (MULF)          F F D E E E E E W
23 (MULF)            F D E E E E E W
24 (MULF)              F D E E E E W
25 (MULF)              F D E E E E W
26 (MULF)                F D D D E E E E E W
27 (MULF)                F D D D D E E E E W
28 (ADD)                   F F F D E W
29 (ADDF)                  F F F F D E E E W
30 (ADDF)                        F D E E E W
31 (ADDF)                          F D E E E W
32 (ADDF)                            F D D E E E W
33 (ADDF)                            F D D E E E W
34 (ADDF)                              F F D D E E E
35 (ADDF)                              F F D D D E E
36 (ADDF)                                  F F D E E
37 (SUB)                                   F F F D E
38 (STF)                                       F D M
39 (STF)                                         F D
40 (STF)                                         F F
41 (STF)                                           F
```

Fig. 4. Example instruction execution timetable.

of Table 2. Table 3 shows the error in each scenario between the expected value for the instruction execution mean time ($T_{\mathrm{ID}} + T_{\mathrm{BE}} + T_{\mathrm{ES}}$) and the corresponding value obtained by simulation, for the two processors with limited resources (even rows in Table 1). The reason of these errors is due to the overlapping of structural hazards with system memory blocks due to I/O operations, as the instruction stream keeps on executing until it stalls if a memory block remains so much time. Thus, penalization due to structural hazards (T_{BE}) results reduced.

Figure 4 shows an example instruction execution timetable where some structural stalls (in bold type) overlap with a memory block (first row of "B"). It is a simplified fragment (the instruction queue is not considered) of the sample program execution in the 2-issue superscalar processor with limited resources. A cache miss ("C") is generated when instruction 17 tries to access the data cache and therefore the system memory must be accessed. However, the access to the memory is blocked since a device is using it in a data transfer. Once the device has finished its transfer, the data cache controller can access the system memory (row of five

"D") and instruction 17 completes its execution. Instructions 23 and 24 cannot be fetched simultaneously since they are in different lines in the instruction cache and the same occurs with instructions (31 and 32) and (39 and 40). Finally, a data stall (ID phase of instruction 34 typed in italics) exists as instruction 34 cannot be issued until instruction 17 has completed its memory access.

Thus, Eq. (7) shows the analytical expression for the errors consigned in Table 3, calculated as the difference between the expected value for the instruction execution mean time ($T_{\mathrm{ID}} + T_{\mathrm{BE}} + T_{\mathrm{ES}}$) for each processor with limited resources and each scenario, and the corresponding values obtained by simulation in the even rows of Table 1 ($T_{\mathrm{ID}} + R * T_{\mathrm{BE}} + T_{\mathrm{ES}}$). Thus, a reduction factor R is used to denote the moderation of the penalization due to structural hazards, as the resolution of part of them overlaps with system memory blocks due to peripheral I/O operations, which give place to the penalization denoted T_{ES}.

$$E = (T_{\mathrm{ID}} + T_{\mathrm{BE}} + T_{\mathrm{ES}}) - (T_{\mathrm{ID}} + R \times T_{\mathrm{BE}} + T_{\mathrm{ES}}) = (1 - R) \times T_{\mathrm{BE}}\,. \tag{7}$$

Finally, Eq. (8) states the corrected analytical expression for the slow downs in Table 2 for the 2-issue and 4-issue superscalar processors with limited resources (even rows). The mean reduction factor for the four scenarios is similar in both cases: for the 2-issue processor it is equal to 0.51 and for the 4-issue processor it is equal to 0.47. Thus, the penalization due to structural hazards (T_{BE}) in both processors is approximately reduced to the half, as the resolution of part of these blocks overlaps in time with the system memory blocks due to peripheral I/O operations.

$$S_3 = \frac{T_{\mathrm{ID}} + R \times T_{\mathrm{BE}} + T_{\mathrm{ES}}}{T_{\mathrm{ID}} + T_{\mathrm{BE}}} \times 100\,. \tag{8}$$

5. Conclusions

(1) A simulation software has been designed in order to study the influence of peripheral I/O operations on processor performance.
(2) This software has been used to analyze the performance of four different processors in four I/O scenarios (video capture, video capture and playback, high speed network, and serial transmission).

6. Future Research Lines

(1) Enhance the simulator functionality in order to study the influence of I/O operations on processors with dynamic scheduling.
(2) Develop a methodology for designing code optimizers that not only consider the processor architecture but also system I/O devices and the influence of their operations on code execution.

References

1. H. S. Stone, *High-Performance Computer Architecture*, 2nd edn. (Addison-Wesley, New York, 1990).
2. J. L. Hennessy and N. P. Jouppi, Computer technology and architecture. An evolving interaction, *Computer* **24** (1991) 18–29.
3. H. S. Stone and J. Cocke, Computer architecture in the 1990s, *Computer* **24** (1991) 30–38.
4. J. L. Hennessy and D. A. Patterson, *Computer Architecture. A Quantitative Approach*, 3rd edn. (Morgan Kaufmann Publishers (Elsevier Science), San Francisco, 2003).
5. J. H. Moreno *et al.*, Simulation/evaluation environment for a VLIW processor architecture, *IBM J. Res. Dev.* **41** (1997) 287–302.
6. D. Patterson, Reduced instruction set computers, *Commun. ACM* **28** (1985) 8–21.
7. J. C. Heudin and C. Panetto, *RISC Architectures* (Chapman & Hall, London, 1992).
8. T. Shanley and D. Anderson, *PCI System Architecture*, 3rd edn. (Mindshare Inc., 1995).
9. PCI Special Interest Group, *PCI-X 2.0 Protocol Specification* (2002).
10. PCI Special Interest Group, *PCI Express Base Specification 1.0a* (2003).
11. Y. Katayama, Trends in semiconductor memories, *IEEE Micro.* **17** (1997) 10–17.
12. J. K. Peir, W. W. Hsu and A. J. Smith, Functional implementation techniques for CPU cache memories, *IEEE Trans. Comput.* **48** (1999) 100–110.
13. J. Silc, T. Ungerer and B. Robic, A survey of new research directions in microprocessors, *Microprocessors and Microsystems* **24** (2000) 175–190.
14. L. Diaconescua and S. Majumdar, The effect of average parallelism and CPU-I/O overlap on application speedup, *Proc. 7th Int. Conf. Parallel and Distributed Systems Workshops*, Iwate, Japan (2000), pp. 370–376.
15. J. M. del Rosario and A. N. Choudhary, High performance I/O for massively parallel computers. Problems and prospects, *Computer* **27** (1994) 59–68.
16. J. M. Rodríguez Corral, Una aportación al estudio de la emulación de buses (in Spanish), Ph.D. thesis, Universidad de Sevilla (2002).
17. J. Nikoukaran and R. J. Paul, Software selection for simulation in manufacturing: A review, *Simulation Practice and Theory* **7** (1999) 1–14.
18. M. H. MacDougall, *Simulating Computer Systems*: *Techniques and Tools* (The MIT Press, Cambridge, MA, 1987).
19. D. Burger and T. M. Austin, The simpleScalar tool set, version 2.0, Technical Report 1342, Department of Computer Sciences, Wisconsin-Madison University (1997).
20. R. N. Ibbet, Computer architecture visualisation techniques, *Microprocessors and Microsystems* **23** (1999) 291–300.
21. V. Pai, P. Ranganathan and S. Adve, RSIM: An execution-driven simulation for ILP-based shared memory multiprocessors and uniprocessors, *Proc. IEEE 3rd Annual Workshop on Computer Architecture Education at HPCA-3*, Texas, USA (1997).
22. O. Starostenko, A. Sánchez and S. Lobato, Simulation facilities for risc processors data flow and performance optimizations, *Proc. 21st Int. Conf. Computers and Industrial Engineering* (1997), pp. 109–112.
23. Y. Zhang and C. B. Adams, An interactive visual simulator for the DLX pipeline, *Proc. IEEE 3rd Annual Workshop on Computer Architecture Education at HPCA-3*, Texas, USA (1997).
24. E. Kappos and D. J. Kinniment, Application-specific processor architectures for embedded control: Case studies, *Microprocessors and Microsystems* **20** (1996) 225–232.
25. M. O. Tokhi and M. A. Hossain, CISC, RISC and DSP processors in real-time signal processing and control, *Microprocessors and Microsystems* **19** (1995) 291–300.

26. J. E. Smith and A. R. Pleszkun, Implementing precise interrupts in pipelined processors, *IEEE Trans. Comput.* **37** (1988) 562–573.
27. T. A. Diep, C. Nelson and J. P. Shen, Performance evaluation of the PowerPC 620 microarchitecture, *Proc. 22th Symp. Computer Architecture*, Santa Margherita, Italy (1995).
28. T. Shanley, *Pentium Pro and Pentium II Processor System Architecture*, 2nd edn. (Mindshare Inc., 1998).
29. K. C. Yeager, The Mips R10000 superscalar microprocessor, *IEEE Micro.* **16** (1996) 28–40.
30. J. D. Gee, M. D. Hill, D. N. Pnevmatikatos and A. J. Smith, Cache performance of the SPEC92 benchmark suite, *IEEE Micro.* **13** (1993) 17–27.
31. K. Nahrstedt and R. Steinmetz, Resource management in networked multimedia systems, *Computer* **28** (1995) 52–63.
32. D. Moore, *IEEE-1394*. The cable connection to complete the digital revolution (1996), http://www.vxm.com/21R.49.html.
33. Y. Hu and Q. Yang, A new hierarchical disk architecture, *IEEE Micro.* **18** (1998) 64–76.
34. C. A. Thekkath and H. M. Levy, Limits to low-latency communication on high-speed networks, *ACM Trans. Comput. Syst.* **11** (1993) 179–203.
35. B. M. Cook, IEEE 1355 data-strobe links: ATM speed at RS232 cost, *Microprocessors and Microsystems* **21** (1998) 421–428.
36. A. M. Law, Statistical analysis of simulation output data, *Oper. Res.* **31** (1983) 983–1029.
37. P. Heidelberger and S. S. Lavenberg, Computer performance evaluation methodology, *IEEE Trans. Comput.* **C-33** (1984).