Parallel Processing on a Transputer-based Graphics Board

João Pereira, Francisco Reis, Carlos Vinagre, and Mário R. Gomes

This paper discusses the design of a graphics board with parallel architecture based on Transputers and a resolution of 1024 × 1024 × 8 [VIN88], namely: the processing unit (it plays the role of a display processor), the organization of the frame buffer and the video output hardware which includes the video controller and a RAMDAC (lookup-table + DACs).

Display Processor

Parallel processing can be successfully applied as a solution to the Computer Image Generation (CIG) Problem. In fact, the parallelism may be obtained by decomposing the CIG problem into a number of smaller, simpler tasks which can be executed in parallel [MAY86].

One of these tasks is a process called "transformer" that transforms the objects defined in Real World coordinates into viewing coordinates (Image Space) according to the viewing position and direction. Objects are represented by planar and convex polygons. In this way the viewing transformation of objects is reduced to the transformation of points, namely the vertices of the polygons. The transformer process will perform (4 × 4)-vector-matrix multiplication using homogeneous coordinates, clipping and, then, perspective transformation to obtain the illusion of depth.

Each object transformed is passed to a drawing process (the pixel processing part of the CIG pipeline). Additional transformers may be used to increase the throughput of the system. As a transformer becomes free, the object store can send another object to transform (see Figure 1). N-transformers can process data at up to N-times the rate one transformer can. This speed-up depends on the rate at which the object store is able to supply objects and on the drawing process being able to draw objects fast enough. The later factor can, again, be achieved by introducing parallelism in the drawing process. This

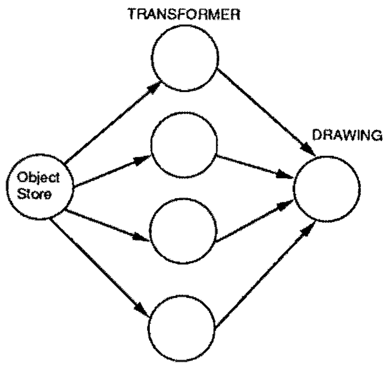

Figure 1: Parallel processes in Computer Image Generation.

parallelism is reached by considering Image Space partitioning or Object Space partitioning [STR86].

Facing these considerations, IMS-T800 Transputers (20 MHz version) from INMOS [INM87] were chosen to implement a concurrent display processor. The concurrency of the display processor has two levels: the software level (existence of concurrent processes in each transputer) and the hardware level (several transputers processing concurrently).

The IMS-T800 transputer is a CMOS microcomputer which integrates a 32-bit 10-MIPS integer CPU (20 MHz version), a full 32/64 bit IEEE 754 floating point processor capable of a sustained 1.5 MFLOPS, 4 Kbytes on-chip fast static RAM which can be accessed at 80 Mbytes/s, a memory interface with integral DRAM controller to address up to 4 Gbytes of external memory at a data rate up to 26.6 Mbytes/s and four full-duplex interprocessor communications links on a single chip. High performance graphics support is provided by microcoded block move instructions which operate at the speed of memory. These kind of instructions can be used in graphics operations such as windowing, zooming, text manipulation and screen updating. The transputer is designed to implement the OCCAM language, but also efficiently supports other languages such as C, Pascal and Fortran.

Its four serial links provide a simple way to connect multiple transputers together, thus allowing concurrent operation. Interprocessor communications is implemented by connecting a link interface on one transputer to a link interface on the other transputer by two uni- directional signal lines. Each of these interprocessor links has two on-chip DMA engines associated with it, one for input and the other for output. Each links runs at to 20 Mbits/s, providing sustained data rates of 2.35 Mbytes/s, independent of the processor and the other links. So, the four links can run simultaneously, providing data transfers at up to 9.4 Mbytes/s. The existence of these links together with an extremely fast hardware scheduler on-chip (switching processes takes on the order of 1μs) allows the power of parallel processing to be exploited. Taking advantage of the transputer capability to form networks easily, a ring network with four transputers was implemented [VIN88] (Figure 2).

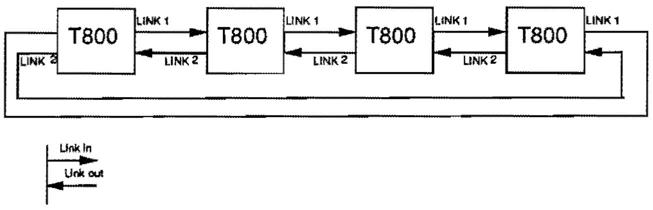

Figure 2: Transputers in a ring network.

All transputers are connected to the system bus, so an arbitration on the access to the bus must be done. Each one of them has its own local memory. Using its local memories and its links the problem of saturation is avoided because the transputers accesses to the system bus decrease (Figure 3).

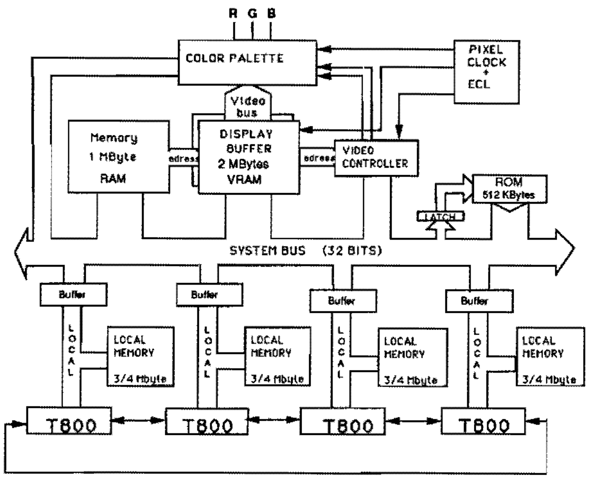

Figure 3: Block diagram of the graphics system.

Let us examine the splitting of the CIG problem into several tasks and how these are distributed among the four transputers.

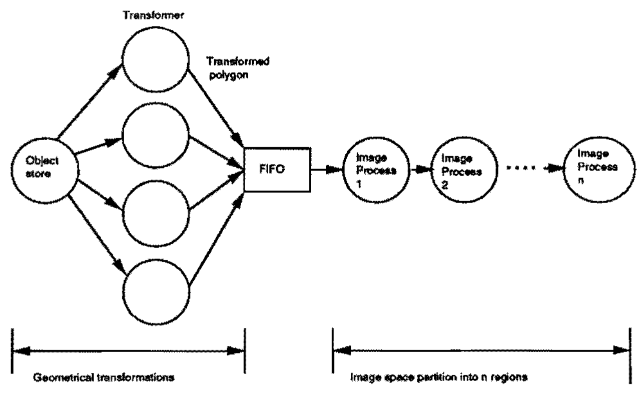

Figure 4: CIG tasks

For the drawing part it was decided to implement image space partitioning which leads to several processes, each one responsible for a region of the screen. This partitioning is suitable to implement the Z-buffer. If object space partitioning was used, there would be a lot of dependency between the processors which must be avoided. To take full advantage of the CIG pipeline, there will be four "transformer" processes whose outputs (transformed polygons) will be delivered, in a FIFO scheme, to a pipeline of image processes (Figure 4). For this we need a scheduler process that, according to the throughputs of the geometrical and drawing parameters, will control the rate at which transformed polygons are fed to the image generating pipeline.

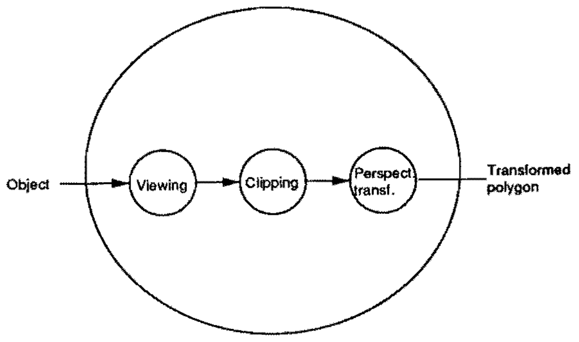

Figure 5: The transformer process.

The picture is defined by a display file, containing 3D polygons definitions. To simplify the implementation, the polygons will be restricted to triangles. The display list is located in local DRAM memory and is loaded by the host. At each vertex of the polygon depth and intensity are stored. The scheduler process traverses the display file and distributes the polygons among the "transformer" processes. Each one of these processes can, in turn, be distributed over a pipeline. Then, the scheduler process determines if the FIFO is ready to accept the transformed polygon. In the positive case, the polygon is passed and the "transformer" process receives another polygon to transform. If not, the "transformer" process must wait.

The display will be broken up into 16 regions. So, in each transputer there will be four image processes (IP) each one with its own local Z-buffer and local frame-buffer (64K × 2) occupying a total of 1/2 Mbyte of local memory. The remaining 256K (see Figure 3) will mainly be reserved for geometrical transformations and for the scan-conversion algorithm.

The operations required to render a scene includes :

- scan-conversion
- hidden surface removal
- shading

Simple Gouraud shading will be incorporated in the scan-conversion algorithm, since it involves only linear interpolation between intensities defined at polygon vertices. The hidden line removal is performed by the Z-buffer algorithm.

Now, it can be seen what operations an image process (IP) will perform. At first, details of the edges of a polygon will be calculated so that for each pixel it can be decided whether it is inside or outside the polygon. Only pixels inside the polygon are eligible for subsequent depth comparison. Each IP has its own local Z-buffer, so it will compute the depth of the polygon at the pixel and will store it if it is less than the previous minimum. In this case, the IP will calculate the intensity at the pixel by interpolation between values stored at the polygon vertices and will store it in its local frame-buffer.

Again, parallelism can be exploited. In fact, each IP can be split into three tasks and distributed over a pipeline (Figure 6).

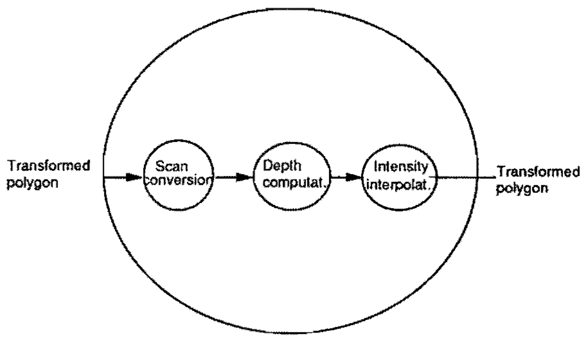

Figure 6: An image process.

After a transformed polygon has been processed by an IP, it is fed into another image process which can run on the same transputer or in one of the following transputers of the ring.

When the last polygon had been processed by the last IP of the image pipeline, the scheduler will spawn a display process that copies the sixteen local frame-buffers to the final frame-buffer. This copy operation involves system bus access by the four transputers, so that arbitration is needed. This is simply solved by a "token" which travels around the ring. The display process which has the token is allowed to send its local frame buffer. At that time, the others cannot access the system bus. After the display process has finished sending, it passes the token to the following transputer which in turn will start to send its local frame buffer.

Frame Buffer

The systems design for real-time graphics performance is expensive and difficult because it implies to generate moving images on a computer screen in real time, i.e. the system must be able to recalculate and redraw the image 30 to 60 times per second to give the illusion of smooth motion. An analysis of the architecture of a graphics system shows that the major performance problem is the organization method used to implement the frame buffer [WHI84]. This memory is accessed by the display processor, which writes data into the memory, and by the video refresh controller, which reads from the memory and routes the pixel data to the video output circuitry and CRT monitor. For the viewing image to be stable, the video controller must supply the data to the monitor according to strict timing requirements. To meet this required data rate, the video controller must have adequate access to the memories. As a result, the display processor and the video controller contend for a finite number of available memory cycles.

So, three problems have to be considered in the design of the frame buffer: the first is the contention for memory cycles and the others two are the adequate bandwidths required for the image memory by the display processor and by the video controller.

The contention of the frame buffer is totally eliminated with the use of double-buffering and almost eliminated by using video Rams (VRAMs). Our implementation of the frame buffer comprises double buffering and VRAMs. One of the goals of this design was the possibility of obtaining real time animation. Only double buffering enables that feature. If double buffering was not used, some images would be composed of parts of the previous frame and parts of the current frame.

The reason to use VRAMS was simply because it was necessary to guarantee the bandwidth which the video controller needs to do the display refresh. In fact, the VRAM internal shift-registers provide a means to get the necessary pixel rate. The alternative —but more expensive— solution would have been to use faster memories.

The frame buffer uses TMS4461 64K × 4 memories from Texas Instruments [TI86]. With a 1024 × 1024 display, 32 chips are required to implement one buffer (so, in total, 64 chips are required for the two buffers) which are organized in 4 rows of 8 memories each, as illustrated in Figure 7. The address lines are the same to every line except the upper two bits which are used to select what row is active.

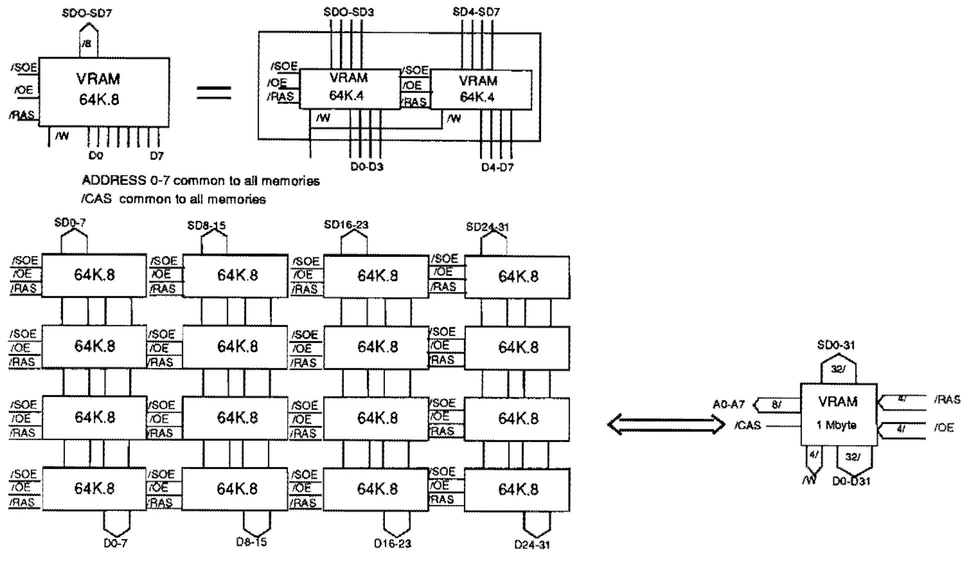

Figure 7: Frame buffer organization.

It is essential to know how often a new pixel must be supplied to the video output hardware in order to determine how fast the frame buffer must be accessed to support display refresh. In this case, a 1024 × 1024 display with a refresh rate of 60 Hz implies a pixel time of 11.42 ns [WHI84]; so that a bandwidth of 87.57 Mbytes/s (8 bits per pixel) is needed.

Looking again at Figure 7, it can be seen that a read operation of the frame buffer by the video controller, taking advantage of its internal shift-registers, occurs only once every four scan lines. So only one memory cycle is sufficient for the video controller to access 4096 pixels (this read operation generates the data transfer from memory to internal shift-registers). It is important to note that between two read operations, generated by the video controller, data are shifted out at a rae controlled by the shift on TMS4461, which can handle a maximum shift frequency of 25 MHz (equivalent with 40 ns). Since with this configuration 16 pixels can be accessed per shift operation, the maximum bandwidth that the frame buffer can guarantee is 400 Mbytes/s, which is more than enough.

The overall architecture of the system is represented in Figure 3.

Video Output Hardware

A TMS-34061 from Texas Instruments [TI85] plays the role of the video refresh controller. This device controls the video display and the dynamic memory of a bit-mapped graphics system. Primarily designed to provide control of VRAMs, it also controls the conventional 64K DRAMs. Beside generating the video timing signals necessary to interface to a raster scan CRT display, it relieves the processing unit of the burden of controlling the system memory, refreshing video memory and reloading VRAM internal shift-registers. It has 18 programmable registers and an arbiter which determines whether the host, the video shift-register reload logic or the DRAM-refresh logic can access the memory.

When the video controller accesses the frame buffer it fetches and stores multiple pixels in a video buffer. In this case the VRAMs internal shift-registers function as a large video buffer with a organization of 1024 × 4 pixels (Figure 7). Then, it is necessary to convert the data stream from parallel pixels to serial pixels. This function is performed by the BT458/125 MHz CMOS RAMDAC from Brooktree [BRO86].

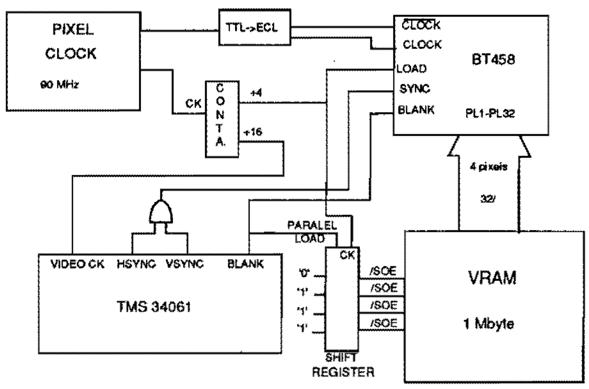

Figure 8: Functional block diagram of video output hardware.

The architecture of this device enables display resolutions up to 1280 × 1024 pixels (up to 8 bits per pixel plus up to 2 bits of overlay information), minimizing the use of costly ECL interfacing, as most of the high speed (pixel clock) is contained on chip. The multiple pixel ports (it may operate either with four pixel ports or five pixel ports) and internal multiplexing (serialization) enables TTL compatible interfacing (up to 32 MHz) to the frame buffer, while maintaining the high video data required for the graphics board. It also has a 256 × 24 colour lookup-table with triple 8-bit video D/A converters, supporting up to 256 simultaneous colours from a 16.8 million colour palette. As illustrated in Figure 8, 4:1 multiplexing is required because four of the pixel inputs of the RAMDAC are used.

Due to the high clock rates at which BT458 can operate (90 MHz in this case), it is designed to accept differential clock signals (CLOCK and CLOCK*). These clock inputs are designed to be generated by ECL logic operating at +5 Volts. They must be differential signals due to noise margins of the CMOS process. On the rising edge of the signal LOAD, four pixels are latched into the device. This signal is derived by externally dividing the signal CLOCK by four (Figure 8).

There were two alternatives to generate the CLOCK and CLOCK* signals: designing a Pixel Clock at TTL levels and translate these to ECL input levels or directly implementing an ECL Pixel Clock. The first method was chosen.

Although, the use of ECL is minimized, some rules must be followed in the design with this high speed logic. The major extra design constraint of ECL systems as compared to any other lower speed logic is that the gate propagation delays and output transition times ($\approx$2 ns) are comparable with the signal "round-trip" time along typical on-card interconnections. In these conditions the interconnections exhibit transmission line properties. This has two results. First, propagation delays along interconnections now become a significant part of the total propagation time of signals between circuits nodes. Circuit layouts must be designed to minimize interconnect distances on critical paths, and most important, clock line lengths must be matched in synchronous systems to ensure that conflicts do not occur.

The second effect is that reflections which always occur at the end of a mismatched line are no longer concealed in the transition of the driving rate. A gate with a rise time of 2 ns driving a one foot long interconnection has almost completed its transition before the voltage at the end of the remote end of the line starts to change. Reflections manifest themselves as overshoot and undershoot at the end of the line. Excessive overshoot must be avoided in ECL, because it can cause the input transistor to saturate, and hence slow the circuit down. Equally important, the undershoot can put the voltage into the threshold region and cause false transitions at the output of a driven gate. The solution to this problem is to use constant impedance and properly terminated transmission lines so as to minimize reflections.

One of the greatest advantages of ECL is the availability of both true and complementary output functions with, essentially, matched delays; these outputs are required to drive the signals CLOCK and CLOCK* of the RAMDAC. ECL circuits normally operate with ground on VCC pins and a negative -5.2 Vdc power supply on VEE pin. While ECL may be used with ground on VEE pin and +5 Vdc on VCC pins, the negative supply operation has noise immunity advantages and is recommended for larger systems. However, according to the BT458 technical specifications ECL must operate on a single +5 Volts supply. ECL works well in this mode if care is taken to isolate the TTL generated noise from the ECL +5 volts supply line. Translators for interfacing TTL and ECL in this mode are built with discrete components since integrated circuit translators do not operate on a single +5 volts supply (except the 10H350 from Motorola). The TTL to

ECL translator shown in Figure 9 consists of three resistors in series to attenuate TTL output to ECL inputs requirements [BLO83].

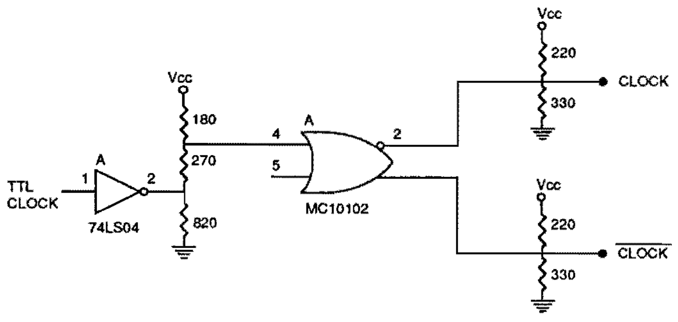

Figure 9: Interfacing the BT458 to the TTL clock.

As said before, to minimize the reflections, both the CLOCK and the CLOCK* lines require standard ECL termination and equal length. A typical ECL parallel termination to a 120Ω transmission would be the Thevenin equivalent resistor pair (220/330) to +3 volts, as shown in Figure 9.

Conclusions

The parallel architecture was designed to reach real time performance. However, this requirement depends on the software used in the implementation of the visualization pipeline. In fact, the performance of the display processor is highly dependent on the degree of parallelism that is possible to get from algorithms (concurrent programming) and how efficiently the tasks are distributed over the four transputers.

In order to estimate the performance of the geometrical part of the visualization pipeline, the figure of a "typical" image with approximately 1000 triangles, each one with an average size of 150 pixels base by 150 pixels height, can be used.

The viewing transformation which performs scaling, translation and rotation involves, in the most general case, 30 FLOPS (16 multiplications, 12 additions and 2 divisions) for each vertex [STR86]. The clipping process will be implemented by the Sutherland-Hodgman algorithm [SUT74]. This algorithm requires one floating-point compare per clipping plane, 6 compares per vertex in a 3D system. In total, each "transformer" process will involve, in average, 36 FLOPS per vertex.

Table 1 summarizes the performance, considering floating-point operations either in single or double length precision and the Typical/Maximum processor cycles involved in each one [INM87]. The processor cycle time are 50 ns.

	Single	Double
	Typical / Maximum	Typical / Maximum
Multiplication cycles	11 / 18	20 / 27
Addition cycles	6 / 9	6 / 9
Division cycles	17 / 28	32 / 43
Compare cycles	3 / 6	3 / 6
Time to transform 1 vertex	(300) 15μs / (488) 24.4μs	(474) 24μs / (662) 33μs
Time to transform 3000 vertices	45ms / 73ms	72ms / 99ms
Achievable images per second	22.2 / 13.6	13.89 / 10.1

Table 1: Performance of the transformation process (all cycles are floating point instruction cycles).

Assuming an optimistic estimate that the FIFO will never fill up (this implies that the first IP of the image pipeline is always ready to accept a transformed polygon whenever a "transformer" process sends its output to the queue), then, the four "transformers" processes (each one in each transputer) can process data at up to 4-times the rate of one. To be more realistic, we can assume that the four processes will perform at 2-times the rate of one. Therefore, considering the worst conditions (FLOPS in double precision and the Maximum processor cycles), it will be possible to reach a number of 20 "typical" images per second.

It's a well-known fact that the rasterization is much more computation demanding than the geometric transformations. To estimate the performance of this stage, let us consider, the same "typical" image for which 11.25 million pixels (= 1000 triangles of $\frac{150 \times 150}{2}$ pixels) have to be drawn.

Pixels per memory cycle	4
Average time to access 4 pixels	150 ns
Average time to rasterize 11.25 million pixels	0.421875 s
Average time to rasterize 30 $\times$ 11.25 million pixels	12.66 s
Achievable images per second	2.37

Table 2: Performance of the drawing process.

Assuming a very optimistic approach that the software can modify four pixels every instruction, the best possible performance is summarized in Table 2. The IMS-T800 memory interface cycle has six timing states, referred to as Tstates [INM87]. The duration of each Tstate is configurable to suit the memories devices used and can range from one to four Tm periods. One Tm period is half the processor cycle time, i.e. 25 ns. In our case each Tstate is configured to one Tm period, so that the memory cycle time is 150 ns.

The drawing process however, is implemented by a pipeline of 16 IPs, each one responsible for 64K of the display. So, it is realistic to estimate that these 16 processes can improve the drawing process, at least, by a factor of 8, which results in 18 images per second.

However, these operations are done in transputers local memories, so it remains to account for the time needed to perform the data transfer to the frame buffer. There are sixteen 256×256 words blocks move. Each block move involves $(2 \times 256 + 23) \times 256$ processor cycles, i.e. 6.85 ms [IMS87]. Thus, the average time to rasterize a "typical" image is:

$$\frac{0.421875}{8} + 16 \times 6.85 \times 10^{-3} = 0.163 \text{ s}$$

which results in 6 images per second.

It can be seen that if there were more IPs, for instance 32, the following number would be obtained:

$$\frac{0.421875}{16} + 32 \times (2 \times 128 + 23) \times 128 \times 50 \times 10^{-9} = 0.084 \text{ s}$$

or 11.9 images per second.

References

[BLO83] Blood, William J.R., "MECL System Design", Motorola INC, 1983

[BRO86] Brooktree Corporation:"Preliminary Information BT 458/451", 1986

[INM87] INMOS Limited, "Preliminary Data: IMS T800 Transputer", April 1987

[MAY86] May, David, and Roger Shepherd, "Communicating Process Computers", Technical Note 22, INMOS, 1986

[STR86] Strasser, W., "Tutorial B1: VLSI-Oriented Graphics System Design", Eurographics 1986

[SUT74] SUTHERLAND, I. E., G. W. Hodgman, "Reentrant Polygon Clipping", *Comm. ACM*, vol. 17, no. 1, Januari 1974.

[TI85] Texas Instruments, "TMS34061: User's guide", 1985

[TI86] Texas Instruments: "MOS Memory DataBook European Edition"

[VIN88] Vinagre, C., F. Reis, J. Pereira, "Placa gr'afica baseada em transputers",3rd *Simpósio da Electrónica das Telecomunicações, May 1988.*

[WHI84] Whitton, M. C. :"Memory Design for Raster Graphics displays", *IEEE Computer Graphics & Applications*, March 1984