Parallelization of the DBW Render Package on a Transputer Farm *

M.Gajęcki, K.Jędrzejek, K.Zieliński
Institute of Computer Science
University of Mining & Metallurgy

November, 1992

1 Introduction

This paper investigates topics of sequential ray tracing software package parallelization using a transputer farm. A ray tracing technique [5] as a well-known computer method of the realistic graphic images generation typically leads to very intensive computational programs. A process of the image generation for a complicated scene may take even a few days. Never the less this method is technically sound for a quite large area of computer graphic, visualization and animation applications. To speed up a computer implementation of the ray tracing algorithm two basic approaches may be pointed out:

- Hardware support of typical steps of the realistic images generation, by more specialized and more powerful processors development [4]
- Parallelization of the graphic image generation process using a general purpose multiprocessor system [2, 1, 5].

The former case concerns a state-of art graphic processors design and is not a subject of this report oriented rather on general purpose processor applications and multiprocessor architectures. The ray tracing parallelization may concern the algorithm or data processing. In this paper the second aspect, which creates an opportunity for the application of a large number of public domain rendering software, is investigated. From many available

*The research was supported under grant no. :8.0077.91.01.

programs the package DBW-Render from the SIMTEL database has been chosen. That program has been first very carefully tested on single processor architectures and, next , parallized on a transtuper farm. The presentation of applied parallelization methodology and a detailed study of the DBW-Render parallel version are the main goal of this paper.

The scope of the paper is as follows. In Section 2 the paper reports results of computational complexity tests of the DBW-Render package running on various single processor computers. The obtained results may be treated as a reference point for a further study of the parallel version of the program. Next, in Section 3, the applied technique of the program parallelization on a transputer farm has been described. The comments about implementation details are presented in Section 4. The results of the experimental study of the program parallel version together with the factors influence obtained speed-up are discussed in Section 5.

The suggested methodology of a sequential program parallelization and obtained results seems to be valid not only for the ray tracing case study but for a larger class of problems with geometric parallelism so common in computer graphics.

2 Ray Tracing as a Computational Intensive Task

The fact that ray tracing is a very intensive computational problem is well-known and documented elsewhere [5, 2]. However to justify an effort for parallelization it seems to be very interesting to compare a relative performance of commonly available computers to a running of the typical ray tracing program.

For the experiment the DBW-Render package by Wiliam T.Baldridge has been used. In all, the DBW-Render package represents about 4000 lines of C code. It is largely based on original algorithms for the VAX by Tom Dahl and Tone Engel at Digital Equipment Corporation. That package has been chosen mostly because its C source version is available free what is sound for further work on its parallelization.

The DBW-Render package uses an algorithm called Distributed Ray Tracing. In addition to neatly solving the hidden surface problem, ray tracing (in general) affords the opportunity for doing excellent (that is, highly realistic) shading and illumination calculations.

The RAY program knows how to create images composed of four primitive geometric objects: spheres, parallelograms, triangles, and flat circular rings (disks with holes in them). Some of the features of the program are:

- Determination of correct shadows cast by arbitrarily shaped objects, onto arbitrarily shaped objects, due to arbitrarily placed light sources.
- Diffuse and specular reflections (with arbitrary levels of gloss or polish). Rudimentary modeling of object-to-object diffusely reflected light is also implemented, that among other things accurately simulates color bleed effects from adjacent contrasting colored objects.
- Mirror reflections, including varying levels of mirror smoothness or perfection.
- Refraction and translucency (which is akin to variable microscopic smoothness, like the surface of etched glass).
- Two types of light sources: purely directional (parallel rays from infinity) of constant intensity, and spherical sources where intensity is determined by the inverse square law.
- Photographic depth-of-field. That is, the virtual camera may be focused on a particular object in the scene, and the virtual camera's aperture can be manipulated to affect the sharpness of foreground and background objects.
- Solid texturing. Normally, a particular object is considered to have constant properties over the entire surface of the object, often resulting in fake looking objects. Solid texturing is a way to algorithmically change the surface properties of an object to try and model some real world material. Currently the program has built in rules for modelling wood, marble, bricks, snow covered scenes, water (with arbitrary wave sources), plus more abstract things like color blend functions.
- Fractals. The program implements what's known as recursive triangle subdivision, which creates all manners of natural looking surface shapes.

The precision of an image generation, in terms of reality, influences the computation time and may be set on a certain level using the following parameters:

1. The maximum computation time may be set in the range 1 - 30 hours. When the given period of time is not long enough, then the colours of some subsets of pixels are calculated using approximation.

2. A number of rays passing through each pixel on a screen. A light intensity and a colour is calculated for each ray. The final value for a pixel is an average of calculated values.

3. A threshold value of the variation calculated for each value of the ray light intensity of the given pixel. If the obtained value of the variation is greater than the chosen value of the threshold then the computation for that pixel is repeated for the doubled value of the rays number.

The DBW-Render is a professional software package. Its modularity and clarity helps during implementation of a parallel version of this program.

The experiment under study in that section concerns not a very complicated scene (shown in Section 5.) with a number of rays r = 2 and variation threshold v = 0.2. The calculation took about 30 minutes on a single transputer T800/20.

The same case has been tested on other processors. The obtained results are summarized in Table 1 and in Fig. 1. The time of computation varies from 1800[s] on T800/20 to 212[s] on HP 9000/720 and the speed-up index defined as follows:

(time on T800)/(time on a given architecture)

changes from 1 to 8.43.

It is evident that despite of architecture, the problem of a realistic image generation even for a very simple case takes enough time to justify an effort of its parallelization.

3 Parallel model of the DBW-Render

The ray tracing computational task is characterized by data-oriented parallelism [6]. It means that the simultaneous performance of an operation on multiple data elements e.g. pixels is possible. Whenever the size of the data warrants parallelization, activities are replicated for application across data subsets. In effect, the program execution is viewed as a single thread of control which temporarily diverges into the parallel action sequence that converges later on. The methods used to distribute data and synchronize convergence become critical factors.

In the case of the ray tracing the master/slave model, shown in Fig. 2, is suitable for the processing organization. This model is anthropomorphic: one logical processor, identified as the "master", is clearly superior to the others, his "slaves". In effect the master owns all logical processors and

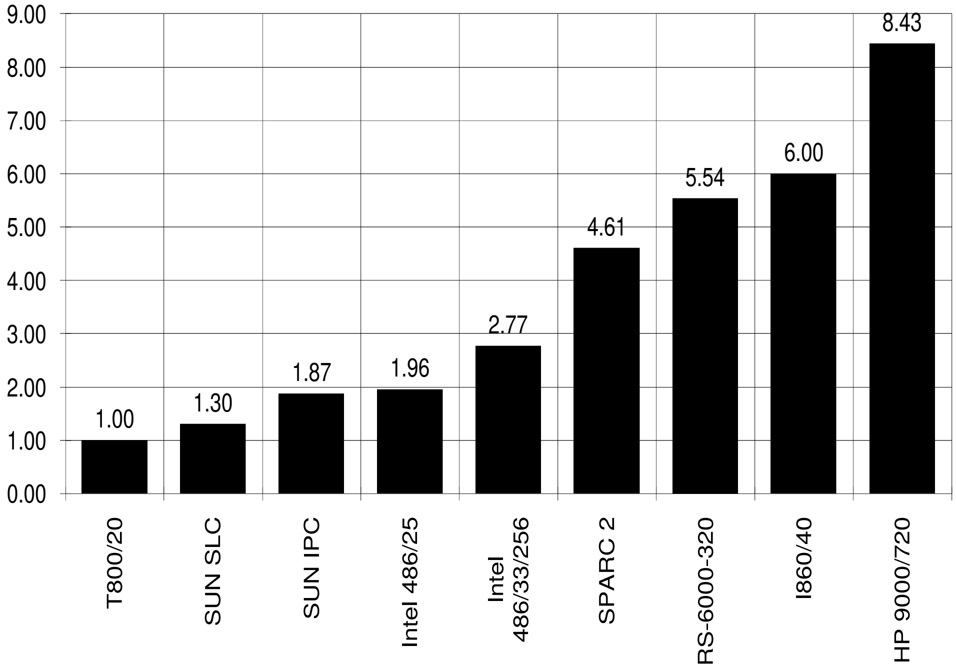


Figure 1: Speed-up index of the DBW-Render computations on various single processor computers

decides when and how to use each of them. When parallelization is appropriate, the master activates the desired number of slaves, then waits for them to complete their work before continuing. Typically, the master is responsible for balancing the load among available processors as well as managing all communication. A variant of the master-slave mode, called sometimes the processor-farm [6], uses a queue manager to keep the track of activities to be performed. As each logical processor, called the worker, finishes a task, it reports the results to the farmer and requests a new task; since tasks are distributed round-robin style as processors become available, load balancing is implicit.

The described model has been used for the parallelization of the DBW-Render. The original structure of the sequential code is shown in Fig. 3. The body of the most inner loop contains the function **do_raytrace** that

Processor	Compiler	Time[s]	Relative performance
T800/20	3L Parallel C v2.1	1802	1.00
SUN SLC	C Sun OS 4.1.3	1372	1.31
SUN IPC	C Sun OS 4.1.3	958	1.88
Intel 486/25	C SCO System V/386	914	1.97
Intel 486/33/256	C SCO System V/386	646	2.79
SPARC 2	C Sun OS 4.1.1	388	3.54
RS-6000-320	XLC AIX 3.1	323	5.58
I860/40	GNU C ver. 1.37 DOS 5.0	298	6.05
HP 9000/720	C HP UX ver 8.5	212	8.43

Table 1: Computation time of the same realistic image on different processors

calculates of a pixel colour.

The worker process contains this function extended by the communication harness. The communication harness complexity depends on a hardware connections topology of processors. The choice of a topology should be a solution of trade-off the between potential parallelism of the wiring connection and complexity of the routing algorithm. For the reasonably small number (15) of processors used to illustrate the parallel version of the DBW-Render in this report very simple structures like a pipe or ring are preferred.

For transputers connected in the pipe, the logical structure of the processor farm model is shown in Fig. 4. The worker process main part is a calculator program – **do_raytrace**, that is in our case a collection of the DBW–Render routines, hence a majority of work has been saved.

For the investigated case , the communication harness is not very complicated and its task is to handle a routing of data and results between the worker and the farmer process. Library routines which create a framework for the processing farm model already exist for many parallel oriented languages [8, 7]. Unfortunately these routines are very often not optimized enough to gain the best possible efficiency provided by a hardware. The most crucial features are:

- Double buffering mechanism of tasks allocation for the worker pro-

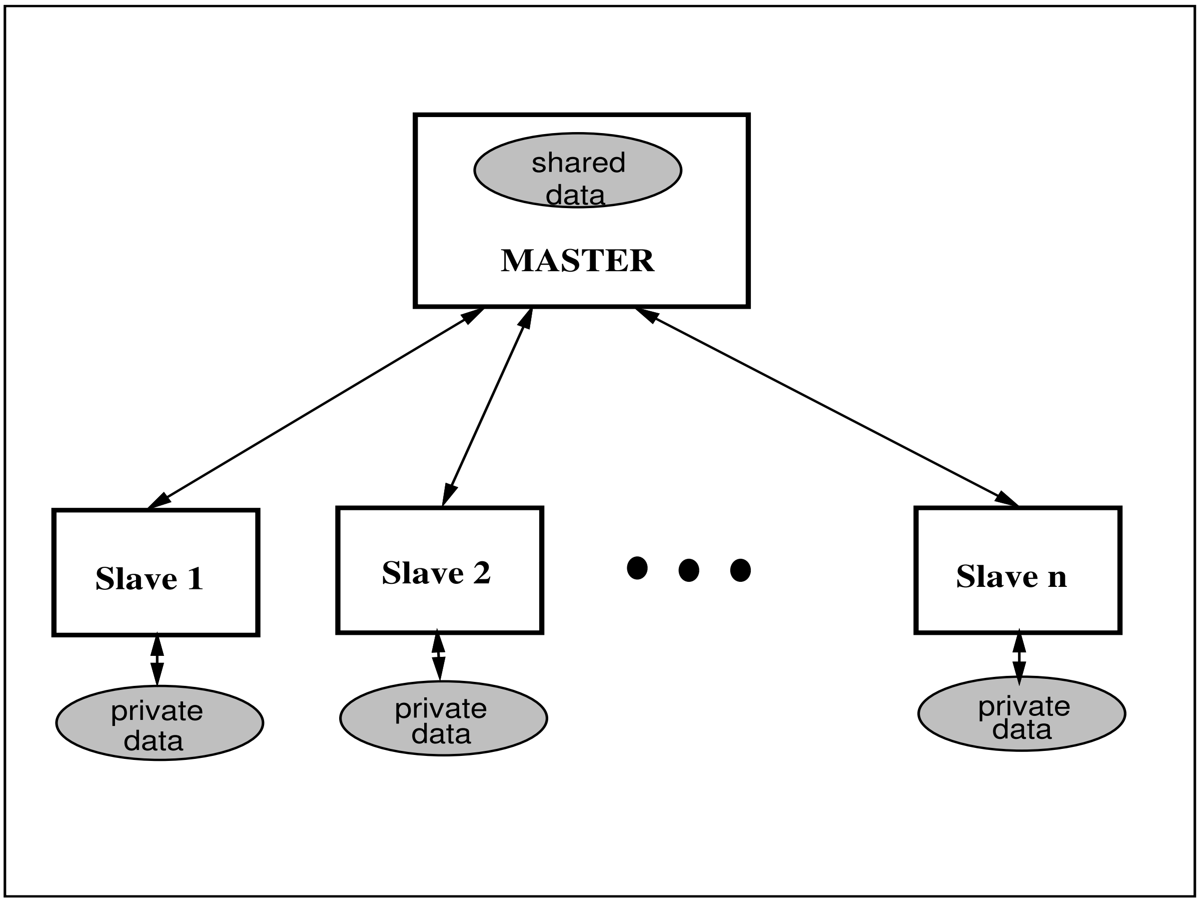


Figure 2: Master/slave model

cessor should be implemented. It makes communication delays less critical by an elimination of the idle period of time of worker processes. It is a time between reporting of the results to the farmer that is also request a new task, and its reception.

- Input and output operations should be done as two independent parallel processes. It makes possible to route task and data in parallel using a full-duplex capability of communication channels if they are provided by the hardware.

- Direct addressing of a worker process or processor where a task is

```
read_image_file ;
{ It reads a sort of command file, that
contains a geometric description of the
scene to be rendered, plus desired global
lighting conditions, camera parameters, etc. }
   for  i := 1 to SCREEN_WIDTH do
      for  j := 1 to SCREEN_HEIGTH do
         do_raytrace(i,j);
         { This functions calculates the coulor of
         each pixel on a screen separately }
   write_image_file ;
   { Results of computations are wirittem to
   disk for further processing if necessary}
```

Figure 3: The structure of the DBW-Render code

allocated should be possible.

The logical structure of the farmer process is shown in Fig. 4. The interface of the host computer is controlled by the sendPatches process. It makes decomposition of the scene into fragments, and sends them to the loadBalance process. A load balancing strategy implemented by that process defines a way of data and results handling, and has a crucial influence on the whole system performance.

4 Implementation Issue

The parallel version of the DBW-Render has been implemented using the 3L Parallel C language and two TMB008 boards with 15 transputers T800 with the 20 MHz clock hosted the IBM PC. Transputer T800 has an efficient floating point unit and seems to be very well fitted for such a computation as a ray tracing. On the other hand, it was the only multiprocessor configuration available for the authors in due time. Unfortunately, the number of transputers was limited to 15, what make impossible to study in full some communication problems typical for that family of transputers.

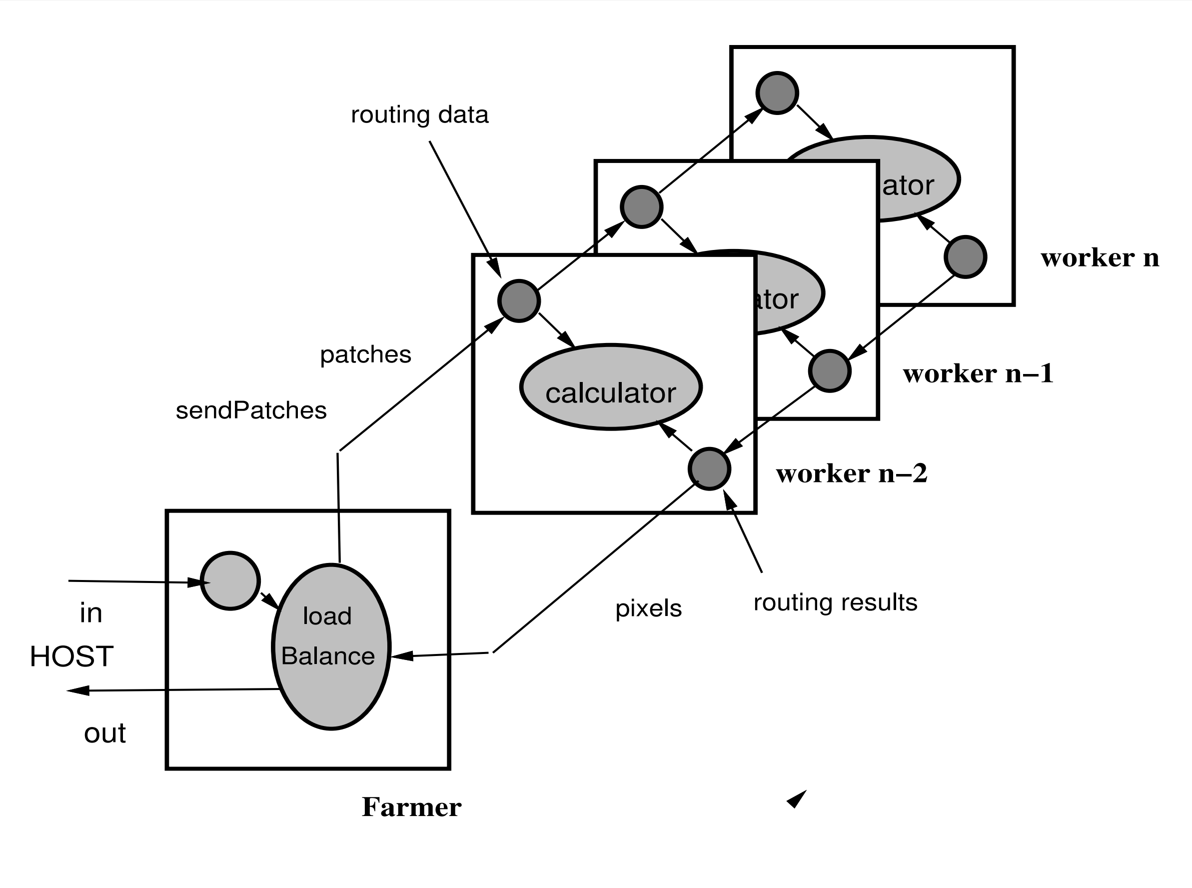


Figure 4: Logical structure of the processor farm for pipe topology

The 3L Parallel C is a dialect of the standard C language extended by some constructions to make a parallel processing possible. A program written in 3L Parallel C may be run into two modes : a processor farm or a basic one. In the processor farm mode the system organizes the farming and a programmer has a limited control over the tasks distribution and the load balancing strategy. In the basic mode in contrast a programmer has a full control over the computation and has to configure the system on the hardware level and organize the whole communication. This mode gives a chance to gain better performance by a tuning of the system hardware and software structure to an application.

For our experiments the basic mode has been chosen. Simple routing modules for the communication between farmer and workers handling have been written in 3L Parallel C. Although these module are very simple, they must be very carefully designed because of their substantial influence on the whole system performance. The standard farm procedures library provided by the 3L parallel C has been redesigned. A new communication harness has been implemented. It makes possible:

1. Double buffering during the allocation of data to workers.

2. Truly parallel activity of routing data and results processes.

3. Direct control over tasks allocation to worker processors.

It is important to point out that the investigated improvements are available in the processor farm software written in the OCCAM [3] and they are crucial for the good performance to be reached.

The 15 available transputers have been wired up in the pipe structure. More complicated topologies are, of course, possible but with 15 nodes an additional complexity to handle a routing of data and results exceeds the possible gains. In fact, a single routing path through the 15 transputer nodes for the ray tracing computation does not to be a communication bottleneck. The experiments reported in the next section and performed for dual pipe, tree and ring topologies give almost identical or a little worse results. It is also justified by the double buffering task allocation mechanisms used in our experiments. That makes the communication delays far less critical form the final speed-up point of view.

More complicated topologies are, of course, possible but with 15 nodes the choice of topology seems to be not very substantial. A more detailed investigation of this problem will be a subject of another paper.

5 Experimental Study

The study concerns two related topics. First the influence of the parameters r - the number of rays passing through a pixel and v - the variation threshold on the computational complexity were measured. Next, the parallel version was studied.

The results reported in this section concern the pipe topology of the transputer system connection and the task allocation with double buffering. The double buffering improves the obtained speed-up by approximately 5%.

The parameters r and v were introduced to control a quality of a generated image and their setting significantly influences the amount of computation as shown in Table 2. The influnce of the value r and v on the quality of the generated images are shown in Fig. 5. These results concerns the same scene as in Section 2.

The efficiency of the parallel processing depends on the ratio of the computation time to communication. Hence the parameters r and v make it possible to change this ratio.

There are many ways to measure the performance of a parallel algorithm running on a multiprocessor system [7, 8, 9, 10]. The most commonly used measurements are elapsed time, speed-up and efficiency. Although the definitions of these performance indexes are well known we remind them once again to avoid misunderstanding.

- *elapsed time* - The time to run a particular job.

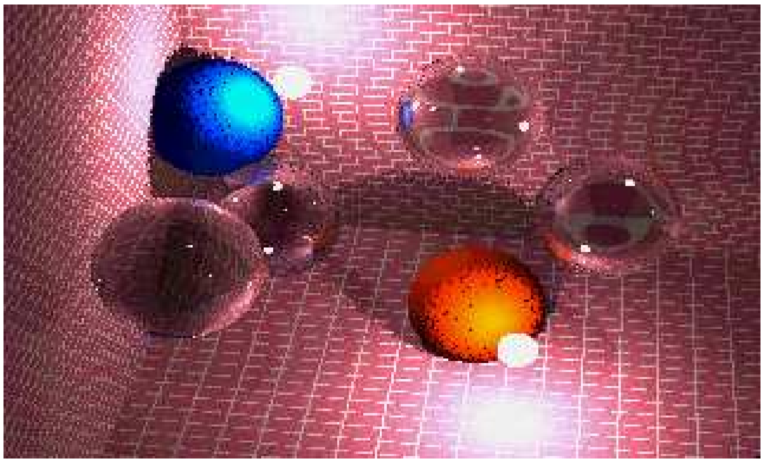
- *speed-up* - More precisely the absloute speed-up has been used defined as the elapsed time needed by 1 processor divided by the time needed by p processors:

  $$s = \frac{T(1)}{T(p)}$$

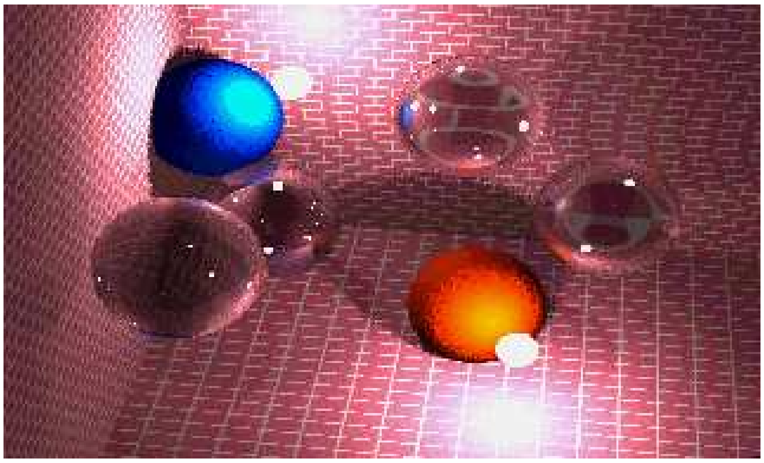
  The T(1) is called a sequential time of the parallel algorithm. This definition is a good measurement of parallel processing gain versus sequential processing [10].

- *efficiency* - The speed-up per processor:

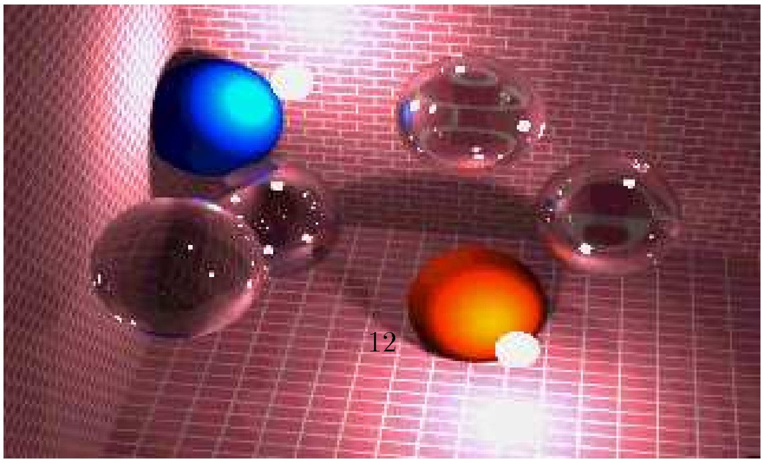
  $$e = \frac{T(1)}{pT(p)} = \frac{s}{p}$$



r = 1, v = 1



r = 2, v = 0.2



r = 32, v = 0.001

Figure 5: Secene under study genarated for different values r and v.

v	r = 1	r = 2	r = 4	r = 8	r = 16	r = 32
1	30.79	124.53	202.77	322.29	568.88	1054.65
0.5	30.93	127.77	199.48	328.95	572.05	1049.83
0.2	30.82	133.93	205.80	341.50	590.93	1081.71
0.1	30.93	135.93	238.91	387.23	643.58	1128.85
0.05	30.82	164.89	294.67	459.47	730.76	1202.73
0.02	30.91	212.72	411.96	605.41	886.00	1343.18
0.01	30.93	276.09	583.88	838.62	1166.12	1609.14
0.005	30.90	443.19	945.62	1256.45	1619.66	2029.78
0.002	30.92	745.93	1471.81	1871.99	2233.46	2552.40
0.001	30.86	1038.30	1906.88	2363.13	2736.30	2978.72

Table 2: Computation time as a function of v – variation threshold and r – number of rays r

No. of processors	Time[s]	Speed-up	Efficiency
1	450.4	1.00	1.00
2	225.4	2.00	1.00
3	150.7	2.99	1.00
4	112.8	3.99	1.00
5	90.2	4.99	1.00
6	75.9	5.93	0.99
7	64.9	6.93	0.99
8	56.7	7.95	0.99
9	51.0	8.83	0.98
10	45.6	9.88	0.99
11	41.5	10.84	0.99
12	38.0	11.83	0.99
13	35.4	12.74	0.98
14	32.9	13.68	0.98
15	30.8	14.60	0.97

Table 3: Time, speed-up, efficiency for r = 1 and v = 1

The study concerns the two chosen pairs of the values r and v. One of

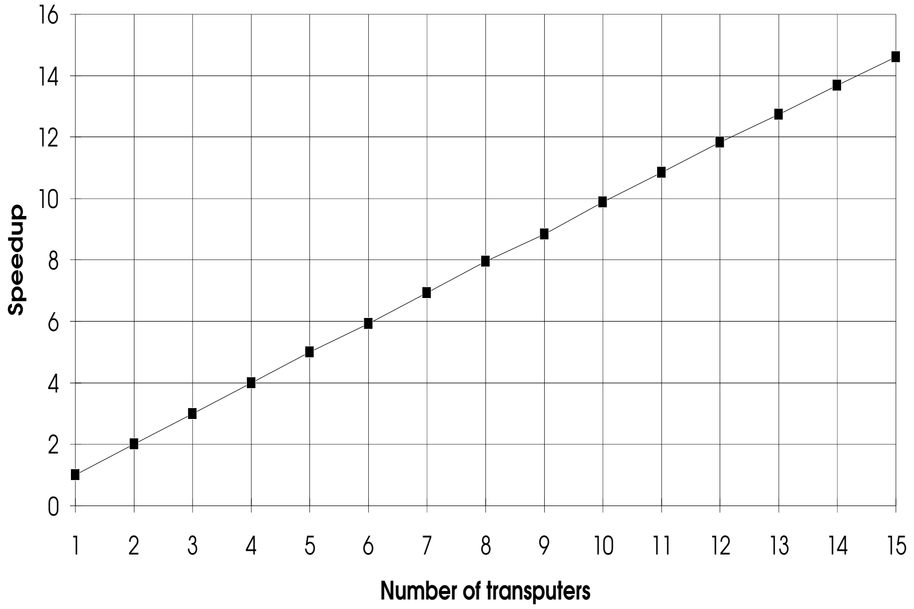


Figure 6: Speed-up vs. number of processors

them corresponds to the same parameters setting as for the first experiment described in Section 2. The unit of the task allocated to the worker processes by the farmer was one line on a screen.

The results of the study have been summarized in Table 3 and in Fig. 6

The efficiency is pretty close to 1 and decreases only sightly as the number of transputers increases. It means that the parallelization has been done correctly and the applied methodology is very effective for the ray tracing computations. Using a few transputers it is possible to make the elapsed time shorter than the fastest processor currently available. The important fact is that then the amount of computations increases the obtained performance of the parallel processing is even better.

6 Conclusions

The obtained results fully justify the conclusion that parallelization is one of very efficient ways to speed up the ray tracing computation. The problem is scaled very well on the array of transputers and the proposed method of parallelization has been proved to be very effective.

The complicated C code of the sequential rendering algorithm has been used without any changes to implement the parallel program. It saves a lot of effort and time. It is quite obvious that many graphical problems with geometric parallelism may be speed up this way.

In this paper the influence of the data domain decomposition in term of amount of computation assigned to the worker in one task on the efficiency was not studied. Also the related problem of load balancing was not investigated. These important issues will be a subject of another paper.

The open question is also behavior of the parallized version of the DBW-Render on the different parallel or distributed hardware platform. The experiment on a network of SUN workstations is currently under study.

References

[1] A Transputer Farm Accelerator for Network Computing Facilities, INMOS technical note 54

[2] High-Performance Graphics with the IMS T800, INMOS technical note 37

[3] The Transputer Implementation of Occam, INMOS technical note 21

[4] Turner Whitted, An Improved Illumination Model for Shaded Display, Communication of the ACM, pp. 343-349, June 1980, 23(6)

[5] J.D. Foley, A van Dan , Fundamentals of Interactive Computer Graphics, The Systems Programming Series, 1989.

[6] C.M. Pancake, D. Bergmark Parallel Programming Languages: Do They Meet the Needs of Scientists and Engineers?, Cornell Theory Center, CTC89TR18.

[7] A.H.Karp, H.P. Flatt, Measuring Parallel Processor Performance, Communication of the ACM, May 1990 Vol.33 Number 5.

[8] 3L Parallel C User Guide 1989 3L Ltd.

[9] X. Sun, J.L. Gustafson, Toward a Better Parallel Performance Metric, Parallel Computing, Vol.17, Dec. 1991.

[10] X. Sun, L.Ni, Another View on Parallel Speedup, Proc. of Suprcomputing'90.