BERICHT NR. 1996/12

PERFORMANCE IN MULTIPROCESSOR SYSTEMS*

Walter H. Burkhardt
Institut für Informatik, Universität Stuttgart
Breitwiesenstr. 20., D-70565 Stuttgart
e-mail: burkhardt@informatik.uni-stuttgart.de
Fax: +49-711-781-6220

Abstract.

The performance degradation in multiprocessor systems by input/output has been investigated again theoretically for a linear systemsconfiguration and by experiment with programs in a functional language on a Transputer multiprocessor systems of up to 16 modules. Processing the results from our experiments shows excellent correlation with the theory and aperformance degradation of up to an order of magnitude, depending on systemsconfiguration and problem type.

Keywords: Multiprocessors, parallel systems, degradation, Transputers

1. Introduction.

This study aims at identifying the influence of input/output and other communication overhead on systems performance in multiprocessor systems analytically and experimentally. The problem of input/output and its influence on systems performance has been neglected in most previous investigations of parallel processor systems [1]. Raw estimates given in literature diverge extremely from a zero influence by supposed overlapping with processing [2] to several orders of magnitude of performance degradation [3].

2. Theoretical Investigation.

A polynomial approach had been applied in a previous investigation [4]. It offers very good correlation with the experimental results, but it has the disadvantage of not correctly explaining the behaviour of the computational for a large number of processor modules. The polynomial approach misses all the cases where some processors still do some useful work, despite the total loading of others for input/output or communications. Here, we use another approach, leading to a solution with an exponential function. Only one model is reported here, due to space limitations, a simple linear array of processor modules.

2.1. Linear Model.

This is the simplest possiblility for a multiprocessor system, consisting of a linear array of n equal processors, each with its own memory and the input and output for processing entering and leaving at one end, see Fig. 1. This type of architecture cannot, of course, be employed for applications with a large degree of communication requirements, because the single bus will saturate with a small number (less than two) of processors with average applications, already. Of help is the use of cash memories for extending this range to about six processors [5]. Any average consists of a wide disparity of cases, so we wanted to check different applications for their communication requirements, especially with transputer systems.

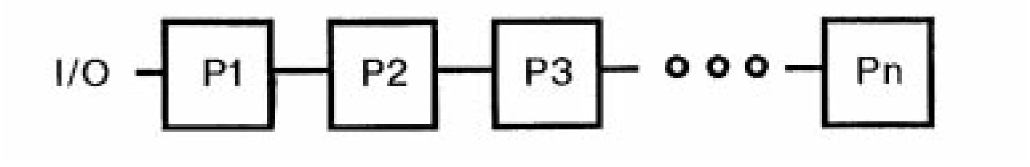

Fig. 1: Linear Processor Array

Let p be the computing power of one processor module, and all processors executing the same, or similar, tasks, e.g. the same routines on different sets of data of the same length. Also, let k be the communications requirements factor, as a fraction of p, for transferring data between input and output to one processor in the row. Then let's assume an arbitrary point in the linear array for consideration. At this point the computational power to be p. This computational power is then at a distance of d.x to the right, if equally distributed, the value

$$p + d.x * dp/dx \qquad (1)$$

according to Taylor's theorem. During the transition by d.x into the processor array we suffer the input/output or communications overhead d.k. This could be overlappedd by a factor r with computation. Thus we obtain for the decrease of the computational power, or the communications requirement during a step of d.x the expression

$$d.k = r * k * d.p \qquad (2)$$

which increases proportionally with the step size d.x in the processor array. The computational power at this point is p + d.p, with d.p proportional to the remaining power to the maximal value of po. Therefore, with the current increment in power $d.p = p_o - p$ (3)

we obtain alltogether:

$$p = p + d.x * dp/dx - r * k * (p_o - p) * d.x \qquad (4)$$

From this we derive the differential equation for the change of systems power in the direction of x for stated problem to:

$$dp/dx = r * k * (p_o - p) \qquad (5)$$

as the solution for the computational power p in the system by integration as:

$$p = p_o * (1 - \exp(-r*k*x)) \qquad (6)$$

as a function of x, where x is the running depth of the linear processor array and p_o the maximally achievable power of the parallel system.

We can have some estimate for the product of overlap factor and communications factor $k* = r*k$ from equation 6 from the highest obtained computational power p. We assume that for the value of x = 0 the computational power is p = 1. This gives the equation:

$$k * = - \ln (1 - 1/po) \qquad (7)$$

for a determination of the overlap and communications factors from the known maximal systems power, or vice versa.

2.1.1. Total Overlap of Input and Output or Communication.

This case reduces to the usual assumption of the computational power increasing linearly with he number of processors. This option is, however, not available in real world systems, as our and the experience from literature shows, despite some claims to the contrary.

2.1.2. No Overlap of Input and Output or Communication

Here, the bare eponential function covers the increase in computational power with the further addition of processors. The maximally reachable power rests at po and shows no additonal gains for adding any more processors. We combine the factor r to k for simplicity reasons. If r is known, then the true value of k can be obtained easily.

3. Experimental Results.

Several experiments have been run with input/output on a Transputer system of selectable dimension D of 1 to 4 with up to 16 processors, running a variety of applications programs in a functional programming language [6]. All results show saturation of systems performance at different optimal processor numbers N. These results are compared with the ones reported in literature [7] and the theory above. We obtain communications factors between .8% and 4% without external input/output and between .2 and .9 with input/output, depending on experiment settings. The values for N between 25 and 120 without and between 3 and 10 with external input/output of our own and the results from literature correspond closely to the amount predicted by the theory above. This indicates that input/output and other communications tasks can degradate performance of multiprocessor systems by up to an order of magnitude.

3.1. Transputer System.

The configuration of the Transputer system consists of eight processor modules, connected as in Fig. 1. One module has the 32-bit Transputer processor, 1 Mbyte local memory, the clock and 5 simple integrated circuits. The Transputer chip contains a 2 Kbyte internal memory, refresh control for the dynamic memory, programmable wait-state generators, 4 fast serial link connectors with a 20 Kbit/s data transfer rate, and the boot logic. The link connections can communicate, after initiation of a transfer by the CPU, without interference by the processor.

3.2. Programs Performed.

Previous investigations have shown little exploitable parallelism, aside from vector- or matrix-loops, in programs in procedural languages [9]. The assumption arose that the serial pattern of thinking in such languages could be the reason. The limitations should then disappear with programs in a language of the next higher level of the declarative languages [10]. Functional Programming languages are one group among them. Programs on this level are free from side-effects and extremely terse. This means that modifications and additions can be applied without problems and the productivity of the programmers is increased by nearly an order of magnitude. Execution of such programs needs, however, immense computational resources.

The application programs, that were run on the Transputer system, are written in a language derived from KRC [10] with the most important additional features of global and local functions for reasons of modularity. They were: The matrix inversion (Mat20D1), quicksort program (Qs600D1), Hartley transform (Hart50D1), knapsack problem(Knap13D1), differential quotient (Dif1500D1), Fibonacci-Series (Fib20D1), Towers of Hanoi (Han10D1), and the Queens problem (Queen7D1). The number after the shorthand description of the problem indicates the size of the data set, D1 refers to the dimension of the Transputer array, as a linear string. The programs are listed in [4]. They are translated by a compiler into an intermediate language with the detection of exploitable parallel paths as the most interesting feature [12]. Several different strategies (FIFO, LIFO; queue length; degree of parallelization; etc.) have been investigated for the distribution of the parallel portions to the modules during execution of the programs. The results reflect the best options thereof [6].

3.3. Measurements and Results.

Measurements performed were the run-time of the various programs and the distribution of the computational load on the different modules. Three data sets of differing size have been used for excluding the influence of this size. The results of the required time for computing the problems have been transformed to the speedup in reference to the number of modules, relative to the time for one module, as a function of the number of modules working on the problem, see Table 1.

Fib20D1	Han10D1	Har50D1	Kna13D1	Qs600D1	Mat20D1	Di1500D1
1.000	1.000	1.000	1.000	1.000	1.000	1.000
1.926	1.700	1.899	1.970	1.709	1.830	1.873
2.780	1.832	2.642	2.672	2.016	2.480	2.617
3.398	1.905	3.233	3.343	2.205	2.89	3.001
4.163	1.983	3.810	3.789	2.267	3.04	3.399
4.294	1.861	3.959	3.875	2.267	3.09	3.626
4.557	1.796	4.039	3.804	2.269	3.11	3.663
4.560	1.751	4.286	3.875	2.272	3.10	3.656

Table 1. Speedup for the problems.

As can be seen, the Fibonacci program reaches the best performance with a measured speedup of 4.5; the Towers of Hanoi program the worst with a speedup of only 1.75, running on the system with eight processing modules. The other problems range in between these values. .4. Analysis of the results.

The results, as they appear in Table 1, have been analyzed by a least-squares regression program for the function y=a*(1-exp(-bx)), similar to the approach in [14], see Appendix B. We obtain correlation factors of over 99%, except in one case, when reduced to the performance equation from the theoretical section, see Table 2 for the data and Appendix A for the diagrams. The variable a in the equation is our maximal achievable systems power po and b represents the communications factor k.

Problem	k	po	R
Fib20D1	.226	5.70	99.59
Hart50D1	.251	5.04	99.78
Knap13D1	.323	4.39	99.31
Dif1500D1	.324	4.11	99.73
Queen7D1	.419	3.57	99.31
Mat20D1	.429	3.33	99.56
Qs600D1	.673	2.33	99.59
Han10D1	.927	1.89	98.80

Table 2: Results of data analysis.

The values for k are the communications factors, po the values for the highest possible system powers at the optimal module number, and R the correlation coefficients for the problems. The communications factors for all of these problems range from about .2 to about .9, giving at best an optimal module number of about six to seven, considering the law of diminishing returns, and maximal systems power of between two to six. see Figs. 2-9. The correlation ctors are all well above 99%, except in one case. This exception for the wers of Hanoi problem indicates a lack of feasibility for parallelization, which translates also into low systems power.

Most interesting is the penalty to be paid for a given amount of communication or input/output in a multiprocessor system. The data in Table 2 has been analysed, therefore, by a linear, an exponetial, a power, and a logarithmic regression program. The power function gives the best results with a correlation factor of over 99.7%. The performance p decreases then with the communication factor k according to the steep hyperbolic function:

$$p = 1.75 * k \wedge (-.782) \qquad (8)$$

see Fig 10. Even a small amount of communcation decreases systems power, according to a power function.

This picture changes, however, if we add the obvious data points for the extrem values of k = 1 then po = 1, and for k = 0 then po = 8 to the measured ones. Then we obtain different approximations, as in Table 3. The wanted function is to be assumed in the form

y = a + f(b*x).	Old Values	New Values
Linear Fktn.	6.03 -5.02 .929	6.69 -6.02 .942
Exponential	7.11 -1.53 .975	7.96 -1.85 .977
Power Fktn.	1.75 -0.78 .997	2.79 -0.11 .617
Logarithm	1.38 -2.64 .982	3.04 -0.47 .801

Table 3: Communications Penalty.

Now the exponential approximates the data points much better than the other functions with a regression factor of 97.7%, see Fig. 11. All other functions fail considerably. The exponential also gives the boundary values best with an error at the upper limit of less than 4%, but at the lower end with 25%. Thus, we obtain for the computational power with a communications penalty the function

$$p = 7.97 * \exp (-1.85*k) \qquad (9)$$

These results indicate that even a small amount of communication or input/output decreases computational power considerably. Because we obtain very similar results for the maximally obtainable systems power for similar communication factors at completely different problems, e.g. for the problems Knap13D1 and Dif1500D1, we can assume with high probability that equations (8 and 9) hold independently of the type of problem for any other cases.

Bibliography.

1. Burkhardt, W. H.: Aspects of multiprocessors systems - A brief survey and new concepts. Proc. COMPEURO `87, pp. 99-105.

2. Rettberg, R. and R. Thomas: Contention is no problem to shared memory multiprocessors. Communications ACM Vol. 29, pp. 1202-1212 (Dec. 1986).

3. Hack, J.J.: Peak vs. Sustained Performance in Highly Concurrent Vector Machines. IEEE Computer Sept. 1986, pp. 11-19.

4. Burkhardt, W. H.: Performance degradation by input/output in multiprocessors systems. Proceedings 32nd Midwest Symposium on Circuits and Systems, 1989.

5. Burkhardt, W. H.: Locality aspects and cache memory utility in microcomputers. Microprocessing and Microprogramming, Vol. 26 (1989), 51-62.

6. Bodenschatz, W.: Multi-Tranputer-Maschine zur parallelen Reduktion von Funktionalsprachenprogrammen. Doctoral Dissertation Universität Stuttgart (June 1989).

7. Büdenbender, H.-W.: Aufbau und Leistungsverhalten eines modularen Multiprozessor-Systems mit dezentral gesteuertem Kreuzschienenschalter. Doctoral Diss. Universität Stuttgart (Febr. 1982).

8. Fuller, S. H. et. al.: Multi-microprocessors: An overview and working example. Proc. IEEE, Vol 66, No. 2, pp. 216-228 (Febr. 1978).

9. Makram, E. N.: Automatic analysis for the detection of inherent concurrency in programs for parallel computing. Diplomarbeit, Universität Stuttgart (May 1984).

10. Burkhardt, W. H.: Universal programming languages and processors. Proc. FJCC 1965, Vol. I, pp. 1-21 and Vol. II, pp. 163-164.

language. In Functional programming and its applications, pp. 1-22, Cambridge University Press 1982.

12. Burkhardt, W. H.: Automation of program-speedup on parallel-processor computers. Computing, Vol. 3, pp. 297-310 (1968).

13. Bhuyan, L. N. et. al.: Performance of multiprocessor interconnection networks. IEEE Computer (Febr. 1989), 25-37.

14. Hartung, J. et. al.: Statistik. Oldenbourgh, Munich 1991, p.648, ff.

Appendix A: Diagrams.

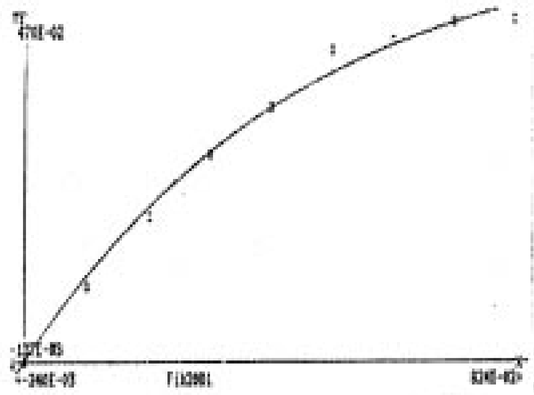

Figure 2: Systems power for Fib20D1.

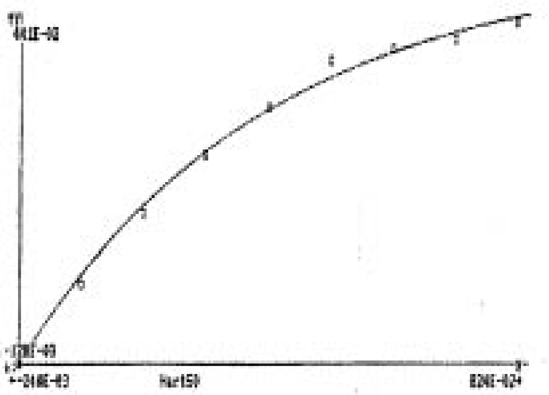

Figure 3: Systems power for Hart50D1

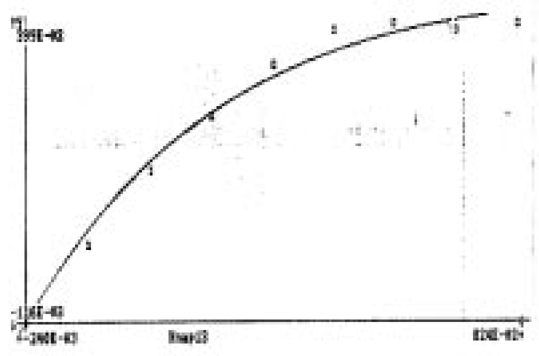

Figure 4: Systems power f. Knap13D1

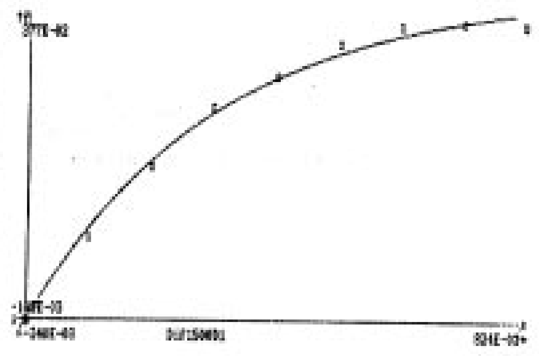

Figure 5: Systems power Dif1500D1

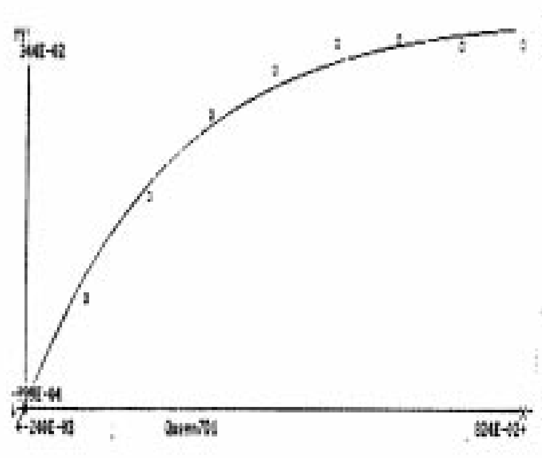

Figure 6: Systems power f. Queen7D1

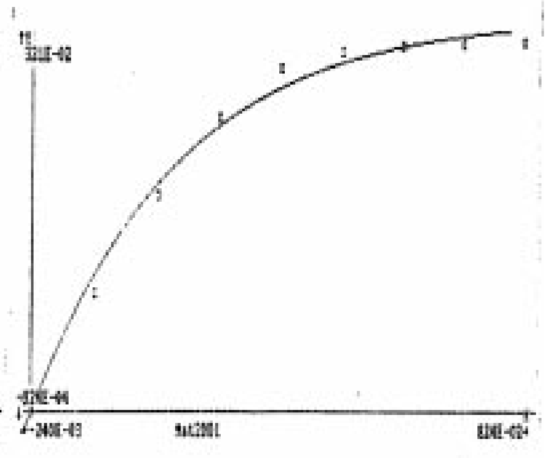

Figure 7: Systems power f. Mat20D1

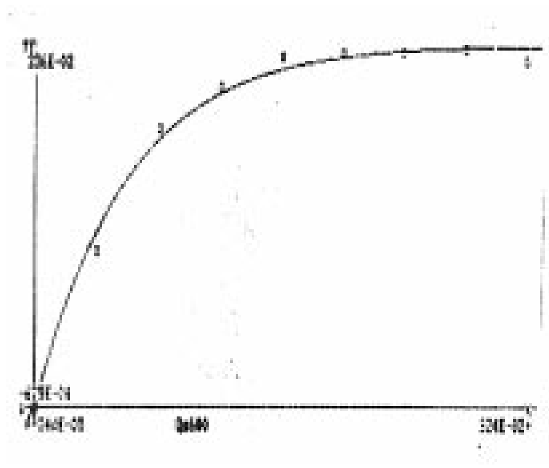

Figure 8: Systems power f. Qs600D1

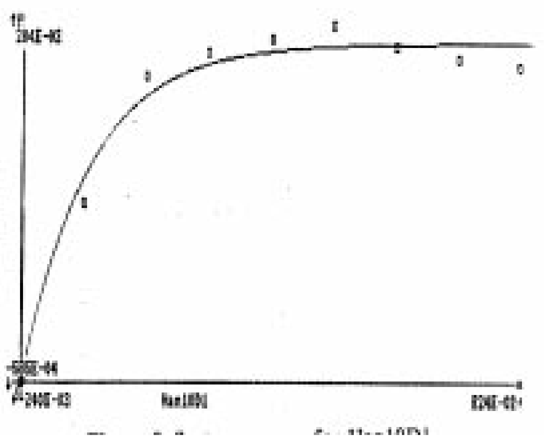

Figure 9: Systems power f. Han10D1

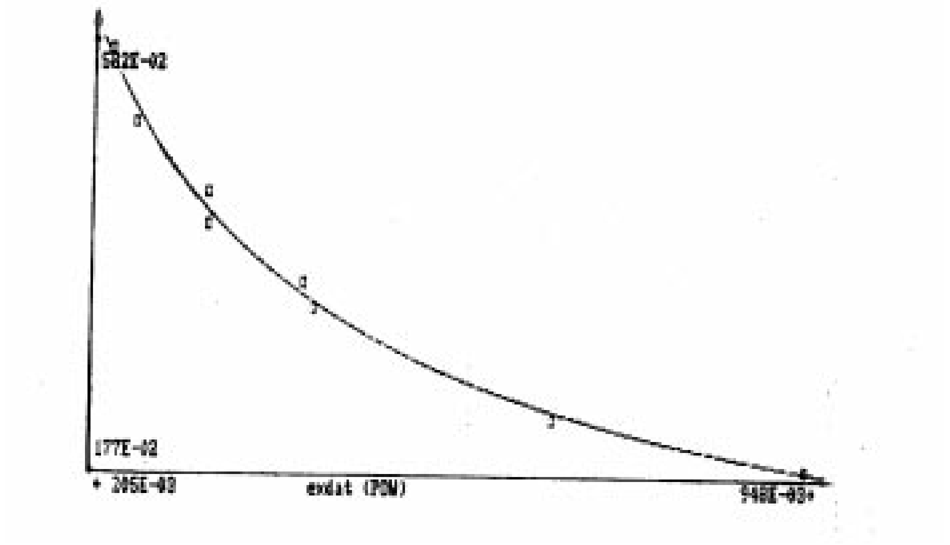

Figure 10: Decrease of systems power

Appendix B: Logistic Function Least-Squares Approximation.

We follow [14] for the approximation of our measured results to the function

$$y = a * (1 - \exp(-b*x)).$$

For a least-squares method we have to minimize the function:

$$\delta [y_i - a*(1-\exp(-b*x_i))]^2.$$

This will be achieved, if the partial derivations for the coefficients a and b are set to zero. Thus, we obtain two equations for the unknowns a and b from:

$$\eth[y_i x_i \exp(-b*x_i - a* \eth[(1-\exp(-b*x_i*x_i \exp(-b*x_i)] = 0$$

and $$\eth[y_i(1-\exp(-b*x_i)] - a* \eth[(1-\exp(-b*x_i(1-\exp(-b*x_i] = 0$$

for all the values of xi and yi of a given set of data.

The unknown a can be expressed and eliminated for the equation to determine b:

$$ð[y_i*x_i*exp(-b*x_i)]*ð[(1-exp(-b*x_i))] - ð[y_i*(1-exp(b*x_i))]* ð[x_i*exp(-b*x_i)*(1-exp(-b*x_i))] = 0$$

This is, unfortunately, a transcendential equation that can be solved only by approximation methods. We obtain with such a program, e.g. using the regula falsi, the required value for b from an appropriately chosen starting value. Then the value for a arrives with the equation:

$$a = ð[y_i]/ð[(1-exp(-b*x_i))]$$

for the pair of coefficients a and b.