https://doi.org/10.26552/com.C.2003.4.83-88

komunikácie
COMMUNICATIONS

Juraj Hanuliak *

TO A PERFORMANCE EVALUATION OF PARALLEL ALGORITHMS IN NOW

A recent trend in high performance computing (HPC) is to use networks of workstations (NOW) as a cheaper alternative to massively parallel multiprocessors or supercomputers. In such parallel systems (NOW's) individual workstations are connected through widely used communication standard networks and co-operate to solve one large problem. Every workstation is treated similarly as a processing element in a conventional multiprocessor system. To make the whole system appear to the applications as a single parallel computing engine (a virtual parallel system), run-time environments such as PVM (Parallel virtual machine), MPI (Message passing interfaces) are often used to provide an extra layer of abstraction. In this paper, we discuss a new performance evaluation method on the example of multidimensional DFFT (Discrete Fast Fourier Transform) in a NOW's based on Intel's personal computers.

1. Introduction

There has been an increasing interest in the use of networks of workstations (cluster) connected together by high – speed networks for solving large computation-intensive problems [1, 6, 14, 15, 19]. This trend is mainly driven by the cost effectiveness of such systems as compared to large multiprocessor systems with tightly coupled processors and memories. Parallel computing on a cluster of workstations connected together by high – speed networks has given rise to a range of hardware and network related issues on any given platform. Performance prediction and evaluation, load balancing, inter-processor communication, and transport protocol for such machines are being widely studied. With the availability of cheap personal computers, workstations and networking devises, the recent trend is to connect a number of such workstations to solve computation-intensive tasks in parallel on such clusters.

The workstations can be connected using different network technologies such as off the shelf devices like Ethernet to specialised networks. Such networks and the associated software and protocols introduce latency and throughput limitations thereby increasing the execution time of cluster – based computation. Researchers are engaged in designing algorithms and protocols to minimise the effect of this latency [16, 18].

Network of workstations (Fig. 1.) has become a widely accepted form of high-performance parallel computing [1, 6, 14, 15, 19]. As in conventional multicomputers, parallel programs running on such a platform are often written in an SPMD form (Single – program – multiple data) to exploit data parallelism. Each workstation in a NOW is treated similarly to a processing element in a multicomputer system. However, workstations are far more powerful and flexible than the processing elements in conventional multicomputers. We can also use the advantages of the new SIMD (Single instruction Multiple data) instructions in the modern personal computers.

2. Effective parallel algorithms

The duty of a programmer is to develop an effective parallel algorithm for the given parallel system and for the given application problem. This task is more complicated in those cases, in which we have to create conditions for a parallel activity, that is through dividing the sequential algorithm to many mutual independent parts, which are named processes (decomposition strategy). Principally the development of the parallel algorithms includes the following activities [7, 16]:

- Decomposition – the division of the application into a set of parallel processes and data
- Mapping – the way in which processes and data are distributed among the computer elements of a used parallel system
- Inter-process communication – the way in which individual processes are cooperated and synchronized
- Tuning – performance optimisation of a developed parallel algorithm.

The most important step is to choose the best decomposition method for a given application [7, 16]. To do this it is necessary to understand the concrete application problem, the data domain, the used algorithm and the functional flow of activities in a given application.

3. Decomposition strategies

When designing a parallel program the description of the high-level algorithm must include, in addition to design, a sequential program the method you intend to use to break the application into processes and distribute data to different nodes – the decomposition strategy. The chosen decomposition method drives the rest of program development. This is true in case of developing a new

* **Juraj Hanuliak**
University of Žilina, Faculty of Management and Informatics, Veľký Diel, Unimo 2, 010 26, Žilina, Slovakia

application as porting serial code. The decomposition method tells you how to structure the code and data and defines the communication topology.

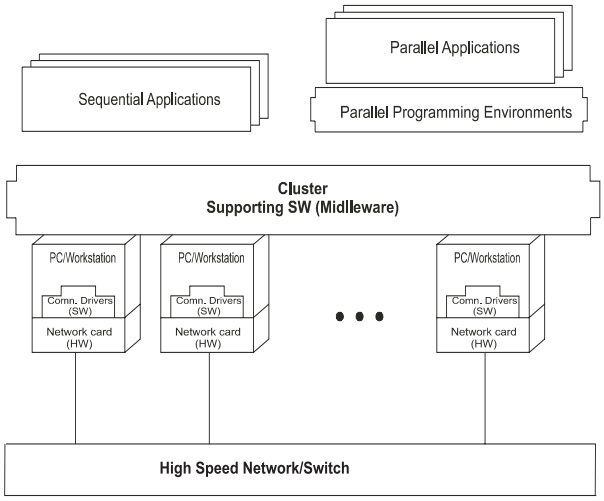


Fig. 1 Network of workstations- a principal structure.

To choose the best decomposition method for these applications, it is necessary to understand the concrete application problem, the data domain, the used algorithm and the flow of control in given application. According to a concrete application we can use the following decomposition models:

- Perfectly parallel decomposition
- Domain decomposition
- Control decomposition
- Object-oriented programming OOP (the latest modern programming technology)

3.1. Perfect parallel decomposition

Certain applications fall naturally into the category perfectly parallel. Perfectly parallel applications can be divided into a set of processes that require little or no communication with one another. Applications of this kind are usually the easiest to decompose. To this class belong, for example, all numerical integration algorithms. Obvious way to implement perfect parallelism is simply to run equivalent sequential programs on the various nodes, each with different data set. On a single processor system, this would require running each case sequentially.

3.2. Domain decomposition

Another decomposition technique is called domain decomposition. Problem subjects to domain decomposition are usually characterized by a large, discrete, static data structure. Decomposition "the domain" of computation, the fundamental data structures, provides the road map for writing the program.

3.3. Control decomposition

Another major decomposition strategy is called control decomposition. When there is no static structure or fixed determination of the numbers of objects or calculations to be performed, domain decomposition is not appropriate. Instead you can focus on the flow of control in the application. As the development progresses you will also distribute the data structures but the guideline to development remains the flow of control.

3.4. Object-oriented programming

Object-oriented programmers view applications as a set of abstract data structures or objects. Associated with these objects are tasks so that they are in no confusion about the parts of the code and data that affect other parts.

4. The theoretical part – The discrete Fourier Transform

The discrete Fourier transform (DFT) has played an important role in the evolution of digital signal processing techniques. It has opened new signal processing techniques in the frequency domain, which are not easily realisable in the analogue domain. The discrete Fourier transform (DFT) is defined as [3, 8, 17]:

$$X_n = \sum_{m=0}^{N-1} x_m \cdot w^{m,n}, \quad n = 0, 1, ..., N-1$$

and the inverse discrete Fourier transform (IDFT) as:

$$X_m = \frac{1}{N} \sum_{m=0}^{N-1} x_n \cdot w^{-m,n}, \quad m = 0, 1, ..., N-1,$$

in which w is N – root of unity that is $w = e^{-i(2p/N)}$ for generally complex numbers. In principle the mentioned equations are the linear transforms. Direct computations of the DFT or the IDFT, according to definitions require N^2 complex arithmetic operations. In such a way we could take into account only the calculation times and not also the overhead times caused through a parallel way of an algorithm implementation.

Cooley and Tukey [3, 7] developed a fast DFT algorithm which requires only $O(N. \log 2(N))$ operations. The difference in execution time between a direct computation of the DFT and the new DFFT (Discrete Fast Fourier Transform) algorithm is very large for large N. Direct computations of the DFT or the IDFT, according to the following program, requires N^2 complex arithmetic operations.

```
Program Direct_DFT;
var
   x, Y: array[0..Nminus1] of complex;
begin
   for k:=0 to N-1 do
      begin
         Y[k] :=x[0];
         for n:=1 to N-1 do
```

```
            Y[k] := Y[k] + Wnk * x[n];
        end;
end.
```

For example the time required for just the complex multiplication in a 1024-point FFT is $T_{mult} = 0.5.N \log_2(N).4.T_{real} = = 0.5.1024. \log_2(1024).4. T_{real}$, where the complex multiplication corresponds approximately to four real multiplication. The basic idea of Cooley and Tukey algorithm, which uses a divide-and-conquer strategy, is illustrated in Fig. 2. Several variations of the Cooley-Tukey algorithm have since been derived. These algorithms are collectively referred to as the DFFT algorithms.

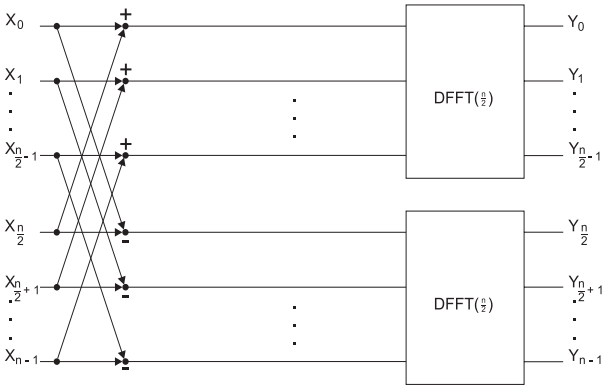


Fig. 2 Divide - and - conquer strategy for DFFT.

The basic form of parallel DFFT is the one-dimensional (1D), unordered, radix-2 (a use of divide and conquer strategy according to the principle in Fig. 2). The effective parallel computing of DFFT tends to computing one - dimensional FFT's with radix equals and greater than two and computing multidimensional FFT's by using the polynomial transfer methods. In practical part of this article we computed 2DFFT (two-dimensional DFFT). In general a radix-q DFFT is computed by splitting the input sequence of size s into a q sequences of size n/q each, computing faster the q smaller DFFT's, and then combining the result. For example, in a radix-4 FFT's, each step computes four outputs from four inputs, and the total number of iterations is $\log_4$ s rather than $\log_2 s$. The input length should, of course, be a power of four. Parallel formulations of higher -radix strategies (e. g. radix-3 and 5) 1-D or multidimensional DFFT's are similar to the basic form because the underlying ideas behind all sequential DFFT are the same. An ordered DFFT is obtained by performing bit reversal (permutation) on the output sequence of an unordered DFFT. Bit reversal does not affect the overall complexity of a parallel implementation.

5. Performance evaluation

To the performance evaluation of parallel algorithms we can use analytical approaches to get under given constraints some relations such as the known theorem of Munro - Paterson [7], Amdahl's law [14, 15], Gustafson's law [14, 15] etc. But all these relations have been derived in an idealised way without considering architecture and communication complexity. That means a complexity C_p is a function only of parallel algorithm calculation. Such assumption could be real in some centralised multiprocessor systems but not in NOW's (network of workstations based on personal computers).

In such a parallel system we have to take into account all complexity elements according to the relation $C_p = f$ (architecture, communication, calculation). In such a case we can use the following solution methods to get a complexity:

- Direct measurement - real experimental measure of P_p and its components for a concrete developed parallel algorithm on the concrete parallel system [7, 8]
- Analytical (to find C_p on basis of some closed analytical expressions or statistical distributions for overheads) [9, 10, 11, 12, 13]
- Simulation [2, 4, 5] (real experimental measure of C_p and its components for a concrete developed parallel algorithm) on NOW's.

6. The results

For direct measuring of complex performance evaluation in a NOW we used the structure according to Fig. 3.

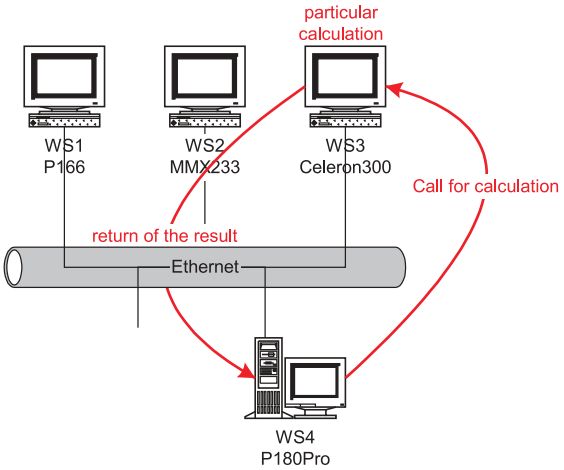


Fig.3 Illustration of the measure in NOW (Ethernet network)

The developed parallel algorithms were divided into the two logical parts - manager and servers. All programs were written on the WNT (Windows New Technology) platform. The manager controls the computer with starting services, makes connections and starts remote functions in a parallel way. At the end it sums the particular results. Every server waits for calculation starting and then calculates the particular results. At the end of calculation it returns to the manager the calculated results and the calculation time. The results are not only the computed results, but also the computation, communication and synchronization times (all overhead components for a given parallel algorithm). To measure these times we used the function "Query Performance Counter", which measures calculation times in ms.

Calibration power results in our experiments for 2DFFT are in Fig. 4. The achieved results for 2DFFT algorithm document increasing of both computation and communication parts in a geometricall way with the quotient value nearly four (increasing matrix dimension mean to do twice more computation on columns and twice more on rows). Therefore for a better illustration we used dependencies on relative input load. At these experiments we used computers according Table 1.

Parameters of the used personal computers Table 1.

Label	Processor	RAM [MB]	Operation system
WS1	Pentium I 233 MHz	64	Windows 2000
WS2	Pentium II 450 MHz	128	Windows 2000
WS3	Pentium III 933 MHz	256	Windows 2000
WS4	Pentium IV 1Ghz	512	Windows 2000
WS5	Pentium IV 1.4 Ghz	512	Windows 2000
WS6	Pentium IV 2.26 Ghz	1000	Windows 2000
WS7	Pentium IV Xeon – 2 proc., 2.2 GHz	1000	Windows 2000 Server

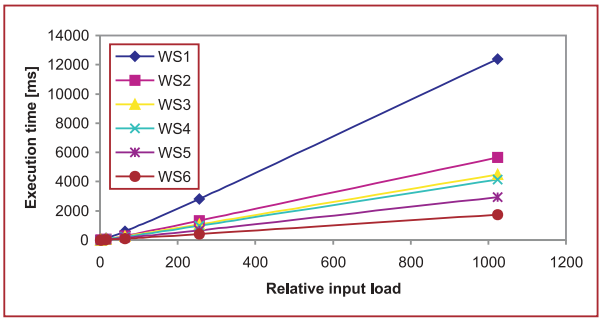


Fig. 4 Calibration power results for 2DFFT.

The results in Ethernet NOW are graphically illustrated for 2DFFT in Fig. 5. For a better graphical illustration we limited the measured values for WS1 network node. The achieved results for 2DFFT-algorithm document increasing of both computation and communication parts in a geometrical way with the quotient value nearly four. The influence of matrix dimension to the network load illustrates the Fig. 6.

Percentile amounts of the individual parts (computation, overheads – network load, initialisation) at 2DFFT execution time for the matrix 1024 × 1024 illustrate Fig. 7. The high network loads are involved through the needed matrix transpositions during 2DFFT computation.

7. Conclusions

Distributed computing was reborn as a kind of "lazy parallelism". A network of computers could team up to solve many problems at once, rather than one problem in higher speed. To get the most out of a distributed parallel system, designers and soft-

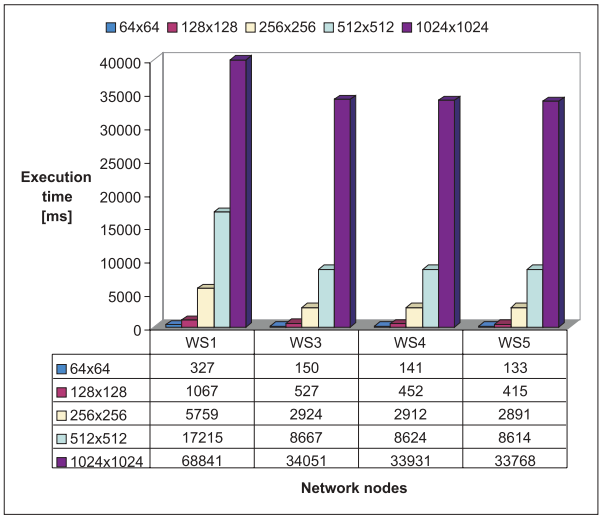


Fig. 5 Results in NOW for 2DFFT calculation (Ethernet network).

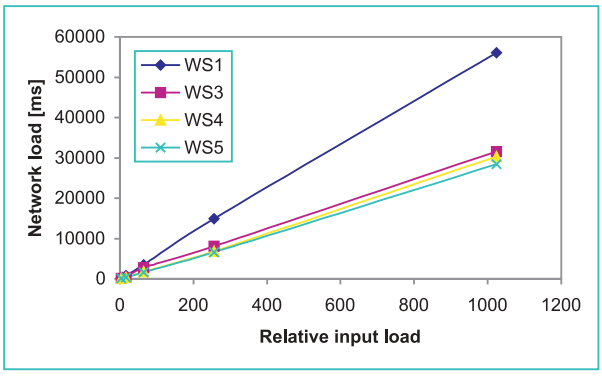
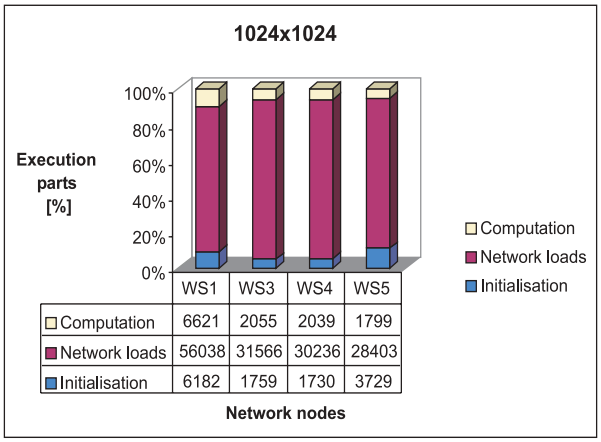


Fig. 6 The influence of matrix dimension to network load

ware developers must understand the interaction between hardware and software parts of the system. It is obvious that the use of a computer network based on personal computers would be principally less effective than the used typical massively parallel architectures in the world, because of higher communication overheads, but a network of workstations based on powerful personal computers, belongs to very cheap, flexible and perspective asynchronous parallel systems. This trend we can see in recent dynamic growth in the parallel architectures based on the networks of workstations as a cheaper and more flexible architecture in comparison to conventional multiprocessors and supercomputers. Second the used principles in world – realised multiprocessors are implemented in recent time in new symmetric multiprocessor systems (SMP), which are implemented on a single motherboard. Unifying of both mentioned approaches open the new possibilities in HPC computing in our country.

The next steps in the evolution of distributed parallel computing will take place on both fronts: inside and outside the box. Inside, parallelism will continue to be used by hardware designers to increase performance. Intel's new SIMD (Single instruction Multiple data) or MMX (Multimedia extensions) instructions tech-

nology, which implements a small/scale form of data parallelism, is one example. Out-of-order instruction execution by super-scalar processors in all latest powerful processors is another example of internal parallelism.



	WS1	WS3	WS4	WS5
Computation	6621	2055	2039	1799
Network loads	56038	31566	30236	28403
Initialisation	6182	1759	1730	3729

Fig. 7 The individual parts of the 2DFFT execution time (1024 × 1024)

In relation to achieved results at our faculty (Faculty of Control and Informatics, Zilina) we are able to do better load balancing among network nodes (performance optimisation of parallel algorithm). For these purposes we can use calibration results of network nodes in order to apply the input load according the measured performance power of used network nodes. Second we can do load balancing among network nodes based on modern SMP parallel systems and on network nodes or with only single processors. Generally we can say that the parallel algorithms with more communication overheads (similar to analyzed 2DFFT algorithm) will have better speed-up values for modern SMP parallel system as in its parallel implementation in NOW. For the algorithms with small or constant communication overheads (similar to π computation [7, 14, 15]) we can prefer to use the other network nodes based on single processors.

Queueing networks and Petri nets models, simulation, experimental measurements, and hybrid modelling have been successfully used for the evaluation of system components. Via the form of experimental measurement we illustrated the use of this technique for the complex performance evaluation of parallel algorithms. In this context I presented the first part of achieved results. We would like to continue in these experiments in order to derive more precise and general formulae (generalisation of the used Amdahl's and Gustafson's laws) and to develop suitable synthetic parallel tests (SMP, NOW) to predict performance in NOW for some typical parallel algorithms from linear algebra and other application oriented algorithms. In the future we will report about these results.

References

[1] ANDREWS, G. R., F*oundations of Multithreaded, Parallel,* and Distributed Programming, Addison Wesley Longman, Inc., 664 pp., 2000, USA
[2] BANKS, J., DAI, J. G.: *Simulation studies of multiclass queueing networks,* IEEE Transactions, Volume 29, 1997, pp. 213-219
[3] BASOGLU, CH., LEE, W., KIM, Y.: *An efficient FFT algorithm for Super-scalar and VLIW Processor Architectures,* Real Time Imaging 3, pp. 441-453, 1997, USA
[4] FODOR, G., BLAABJERG, S., ANDERSEN, *A.: Modelling and simulation of mixed queueing and loss systems,* Wireless Personal Communication, N. 8, 1998, pp. 253-276
[5] GREENBERG, D. S., PARK, J. K., SCHVABE, E. J.: *The cost of complex communication on simple networks,* Journal of Parallel and Distributed Computing 35, pp. 133-141, 1996
[6] HANULIAK, I.: *Parallel architectures - multiprocessors, computer networks* (in Slovak), 187 pages, July 1997, Book centre, Žilina, Slovakia
[7] HANULIAK, I.: *Parallel computers and algorithms* (in Slovak), Košice, Slovakia, ELFA Press 1999, 327 pp.
[8] HANULIAK, J.: *To a complexity of parallel algorithms,* In Proceedings: TRANSCOM 2001 (4-th European Conference in Transport and Telecommunications), 25-27 June 2001, pp. 51-54, Žilina, Slovakia
[9] HANULIAK, I.: *On the analysis and modelling of computer communication systems*, Kybernetes, The International Journal of Systems & Cybernetics, West Yorkshire, United Kingdom, Vol. 31, No. 5, pp., 715-730, 2002
[10] HANULIAK, I.: *Buffer management control in data transport network node,* The International Journal of Systems Architecture, Volume 47, Elsevier Science, Netherlands, pp. 529-541, 2001
[11] HANULIAK, M.: *To the behaviour analysis of mobile data networks,* in Proceedings of 7th Scientific conference, November 9-10, pp. 170-175, 2001, TÂRGU – JIU, Romania
[12] HARRISON, P. G., PATEL N.: *Performance modelling of communication networks and computer architectures,* Addison – Wesley Publishers 1993, 480 pp.
[13] HSU, W. T., PEN-CHUNG, Y.: *Performance Evaluation of Wire-Limited Hierarchical Networks,* Parallel and Distributed Computing 41, 1997, pp. 156 - 172
[14] HESHAM, E. L., REWINI, TED. G. LEWIS: *Distributed and parallel computing,* 467 pp., Manning Publications Co., 1997, USA

[15] HWANG, K. X., SCALABLE, Z.: *Parallel Computing:* Technology, Architecture, Programming, Mc Graw-Hill Companies, 802 pp., 1998, USA
[16] KUMAR, V., GRAMA, A., GUPTA, A., KARYPIS. G.: *Introduction to Parallel Computing* (Second Edition), Addison Wesley, 856 pp., 2001
[17] MARINESCU, D. C., RICE, J. R.: *On the scalability of asynchronous parallel computations,* Parallel and Distributed Computing 31, pp. 88-97, 1995
[18] NANCY, A. L.: *Distributed Algorithms*, 872 pp., 1996, Morgan Kaufmann Publishers, Inc., USA
[19] WILLIAMS, R.: C*omputer Systems Architecture - A networking approach,* Addison Wesley, 660 pp., 2001, England.