Workstation Cluster Computation from the Perspective of the User

Clemens H. CAP*

Institut für Informatik der Universität Zürich
Winterthurerstrasse 190, CH–8057 Zürich, Schweiz
Tel. +41–01–257/4326, 4331, Fax +41–01–363 0035
e-mail: cap@ifi.unizh.ch

Abstract. The use of workstation clusters for solving demanding application problems has widely been adopted. This paper analyses structural aspects which influence the performance of algorithms on such "hypercomputers". Two structurally very different examples help estimating the reasonable range of quantitative measures. An overview on organizational and infrastructural requirements answers the question who and where clusters presently may be used successfully. The paper concludes with recent trends and future developments.

1 Introduction

The *typical RISC workstation* has 20–200 MIPS and 1–20 MFLOPS processor performance, 16–128 MB main memory, 0.5–5 GB disc capacity and 1–5 MB/s bandwidth for disc I/O. It runs a UNIX flavored operating system and costs from 5.000$ to 100.000$. Presently employed communication technology is the ethernet, with faster media just being introduced to the market. Today's RISC workstations therefore are able to cope with problems which eight years ago could be solved only on mainframes. With their capabilities workstations can cope with a large number of today's computational problems.

In order to put the combined performance of several workstations at the easy disposal of a single application, software platforms were developed, which use the performance of the individual workstations to form a *(virtual) parallel computer* or *hypercomputer*, allowing to program a network of workstations just like a distributed memory parallel computer [9], [10]. With this technique a performance or more than 1 Gflops [16] and 20 Gips [8] can be reached, exploiting the computing power of otherwise idling workstations. [30] estimates that 90% of Lawrence Livermore National Laboratory's workload could be solved with this technology.

There is already a large number of *research prototypes* like PVM [15], Parform [10], Picl [14], P4 [6], Parmacs [17], Lips [25] or POSYBL [27], and there are even commercialised solutions like LINDA [1] or Express [23]. Examples demonstrate, that for special applications the hypercomputer is a cheaper [22] and faster [8] choice than a traditional supercomputer, and international prizes have been awarded for the performance of workstation cluster solutions (Gordon Bell prize 1992 [22], SuParCup 1993 [12]). The hypercomputer is beginning to establish itself for certain classes of applications as an interesting alternative to the supercomputer. However, it will never be and never was intended to be an equivalent substitute.

* Research partially supported by *Firma Siemens AG, München* and *Schweizer Bundesamt für Konjunktur und Wirtschaftsförderung*, grants 2255.1 and 2554.1.

This paper evaluates such workstation clusters from the perspective of the user, with a special emphasis on performance. It is organised as follows: In Section 2 we introduce and discuss *performance measures* for workstation clusters. Section 3 gives *two examples* of cluster applications and analyses the obtained performance. In Section 4 we develop more general *criteria for evaluating* which applications are suited for cluster solutions. Section 5 addresses *organizational aspects* of cluster computing. The paper concludes with a survey on present *trends and developments* in hypercomputing.

2 Performance Measures

In order to evaluate the performance of a hypercomputer, several measures are available. They may depend on the number and the performance of the individual workstations, on the capabilities of the communication media, on the parallel algorithms and on the size of the problem.

The *ideal parallel run time*, which would be obtained by an optimal parallelization on p processors in case of a fully parallelizable algorithm, is $T(1)/p$. The *actual parallel run time* or *elapsed time* of a specific parallel algorithm usually is smaller and will be denoted by $T(p)$.

The *communication free run time* $T_{\mathrm{CF}}(p)$ would be obtained by the parallel algorithm under the assumption of immediate communication without delay and with infinte bandwidth. It can be derived from a complexity analysis of the algorithm.

Speedup $S(p) = T(1)/T(p)$ is the factor, by which the parallel algorithm with p processors is faster than the sequential run with one processor. *Efficiency* $E(p) = S(p)/p$ denotes that fraction or percentage of the ideal linear speedup, $S(p) = p$, which actually is obtained. It compares the ideal parallel run with the actual parallel run. *Scaleup* $Sc(p)$ is the factor by which the size of the application may be increased, in order to obtain the same run time with p processors as with one processor. The size is measured in application dependent units: When solving linear equations we could for example use the number of equations as size.

A user, who wants to solve an application of a given size, is interested in speedup. If the goal is to solve the largest possible instance of a problem in a given amount of time, scaleup is the more important measure. Examples for such a situation are problems, which in their full size are too large and hence one is interested in the largest subproblem, solvable in a reasonable amount of time. In computational chemistry and quantum theory many problems are of this kind. Scaleup and speedup may differ: Depending on the structure of the application, larger instances of a problem may be easier or more difficult to parallelise than smaller ones. This effect is better described by scaleup than by speedup. On the other hand, scaleup rarely is measured, since it requires a very large number of test runs for determining the size of an application, producing a given run time.

Limit speedup $S_l(p) = S(p+1) - S(p)$ denotes the increase in speedup when adding one processor to a configuration of p processors. *Efficacy* $W(p)$ is some kind of *profit to price ratio*. When using a configuration of p processors for a computation time of $T(p)$ seconds, the price to pay is proportional to $p \cdot T(p)$. On the other hand we may interprete speedup as the profit of our configuration. Neglecting the constant $T(1)$ we obtain for the profit to price ratio $W(p) = 1/(p \cdot T(p)^2)$. The number of processors for which the computation is the fastest, is determined by maximum speedup. The number of processors for which the computation is the cheapest, at least with respect to above price profit model, is determined by maximum efficacy.

The *sequential component* is that time of the computation, which has to be performed sequentially. This measure is based on the approximative assumption, that the entire load consists of a sequential component seq_{t} and a parallel component par_{t}, running on all p processors in parallel with an optimal efficiency of 100%. Under this model, the (theoretical) run times $T_{\mathrm{t}}(p)$ with p processors and the sequential and parallel components satisfy equation (1)

$$T_{\mathrm{t}}(1) = seq_{\mathrm{t}} + par_{\mathrm{t}} \tag{1}$$

and for all p we have

$$T_{\mathrm{t}}(p) = seq_{\mathrm{t}} + \frac{par_{\mathrm{t}}}{p} \tag{2}$$

We obtain equation (3), which holds for all values of p. Since seq_{t} denotes a constant, the right side of (3) takes identical values, independent of the choice of p.

$$seq_{\mathrm{t}} = \frac{p \cdot T_{\mathrm{t}}(p) - T_{\mathrm{t}}(1)}{p-1} \tag{3}$$

This model is only an approximation of reality. Replacing the theoretical run times $T_{\mathrm{t}}(p)$ by measured run times $T(p)$, the right side of equation (3) no longer is independent of p. However, we may use equation (3) to define a (measured) sequential component $seq_{\mathrm{m}}(p)$, which now depends on p, for estimating the behaviour of an application, when further increasing the number of processors:

$$seq_{\mathrm{m}}(p) = (p \cdot T(p) - T(1))/(p-1) \tag{4}$$

The sequential component may be given in absolute units as seconds run time and in relative units as percentage of the total run time $T(p)$.

The difference between actual parallel runtime and ideal parallel runtime is an estimate for the communication overhead and the time processors have to wait for data to be communicated by other processors. Algebraically it is $T(p) - T(1)/p = (p \cdot T(p) - T(1))/p$. For larger numbers of processors ($p \geq 5$) this difference approximates $seq_m(p)$ and therefore allows a further interpretation of the sequential component.

Communication time $T_{\mathrm{c}}(p)$ is the time spent by a processor for communication, averaged over all processors. *Halftime throughput* $H(p)$ is the throughput required from the communication medium, so that the communication time equals the time spent in computation, and hence is half of the total run time. Both measures can only be derived theoretically.

As a consequence of AMDAHL law, speedup must be smaller than the number of processors used. However, in certain situations we may experience *superlinear speedup*:

"speeddown"	$S(p) < 1$	
sublinear speedup	$S(p) < p$	$E(p) < 100\%$
linear speedup	$S(p) = p$	$E(p) = 100\%$
superlinear speedup	$S(p) > p$	$E(p) > 100\%$

Partitioning a problem into p parts may further reduce the amount of work to be done in these parts. For example, an application might not entirely fit into the main memory of a workstation, requiring additional swapping, paging and garbage collection. Splitting the problem reduces its size and possibly allows to store all data in main memory. Furthermore it influences the cache hit ratio. Applications which use sorted data may profit from partitioning this data base. Sorting or otherwise structuring data often scales with $n \log n$, n being the size of the data base. Splitting of these data sets also may lead to superlinear speedup.

In case of a hypercomputer consisting of workstations with *different computing speeds*, there are several choices for $T(1)$: We can either derive the performance measures with respect to a specific machine, for example the slowest or the fastest one in our configuration, or with respect to a virtual average machine. In this case $T(1)$ is calculated as weighted average $\sum_i T_i(1) \cdot \gamma_i / \sum_i \gamma_i$ of the run time $T_i(1)$ of machine i and a performance value γ_i derived from a benchmark program.

3 Examples

In this section we present two structurally very different examples of hypercomputer applications. Both problems are well known and better algorithms than those presented here are available. However, these examples and their algorithms were chosen since from their structure they are representative for two large classes of problems, local physical laws and combinatorial optimisation. Furthermore, they provide suitable test applications for studying the behaviour of clusters under high communication load and with respect to scalability in numbers of processing elements.

3.1 Temperature Distribution

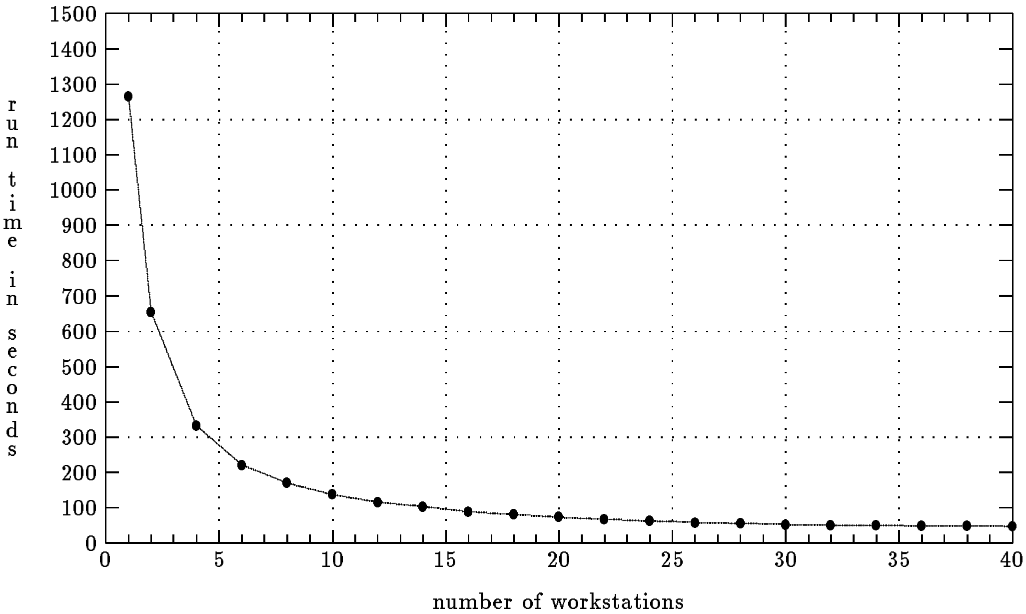

Fig. 1. Run time for temperature distribution.

In the first application [10] we calculated the temperature distribution of a rectangular plate. We started with an initial temperature distribution and imposed boundary conditions, like heating or cooling along the borders of the plate. We then calculated the temperature distribution of the plate as a function of time using a simple finite difference algorithm. Parallelisation is straightforward by decomposing the rectangular plate into stripes and scheduling the work on such stripes to the individual workstations. For each time step the temperature on the stripe must be calculated and the temperature at the boundaries of the stripe must be communicated to the workstations responsible for the neighboring stripes. For the test runs we chose a 3800×100 grid and 500 time steps, using double precision arithmetic.

Figure 1 shows the run time in seconds wall clock time for the parallel algorithm in a network of Sparc workstations using the cluster software Parform, which we developed at Zurich. [10] describes further details of the measurements. We will now apply the performance measures from section 2 to these data.

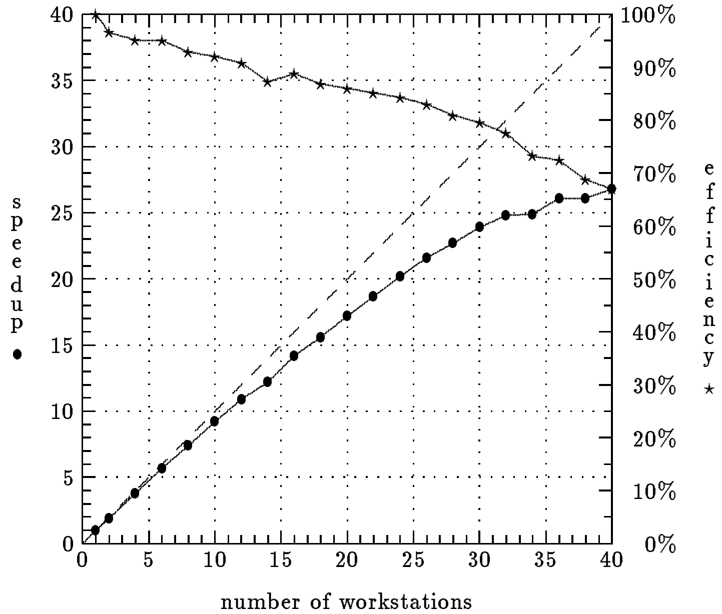

Fig. 2. Speedup and efficiency for temperature distribution.

Figure 2 shows the efficiency and the speedup curves. Whereas the performance of the parallel algorithm is satisfactory for upto 32 workstations, we notice a decline in the slope of the speedup curve for 34 and more processors. This is due to the *communication bottleneck* of the hypercomputer, as we will show in the following.

To get a better understanding of this limiting factor, we will look at two characteristic indices associated with a parallel algorithm: The *granularity* or *grain size* is the average amount of calculation performed before a communication takes place. It can be measured in number of operations or in seconds. The *locality* is determined by the average amount of data which must be communicated during each communication phase. Note the indirect proportionality: Small locality means, that large amounts of data must be transferred. Both indices are easily calculated for our algorithm but may be difficult to obtain for more complicated programs. However, as a general guideline they may provide more insight into the performance an algorithm will deliver on a hypercomputer.

In our example, each of the p processors calculates the temperature of its $100 * 3800/p$ grid points before communicating the stripe boundary values. Updating a single grid point on a Sparcstation1 takes a measured time of $7.21 \mu s$. It consists of 7 floating point operations and several memory and register accesses, which roughly is the work equivalent to 10 floating point operations. This results in $2.74/p$ seconds or $3.8 \cdot 10^6/p$ floating point operations before a communication takes place. With p processors we have to communicate $(2p-2)$ stripe boundaries. Each boundary contains 100 double precision numbers, hence 800 bytes. Under the approximating assumption, that the communication is distributed over the entire time $T(p)$ of the parallel computation, we may calculate an average communication load.

procs	grain in sec.	grain in flop	communicated Mbytes	average load Mbit / sec
10	0.27	380.000	7.2	0.419
20	0.14	190.000	15.2	1.654
30	0.09	126.666	23.2	3.501
40	0.07	95.000	31.2	5.288

Note, that with 40 processors the average communication load is 5.288 Mbit/sec, which is more than half of the physical bandwidth of the ethernet. Since the peak load still is higher and an ethernet already at a load of 20 % of its physical bandwidth is considered highly loaded, it is obvious that the break down in speed up for 32 and more processors is due to the communication bottleneck.

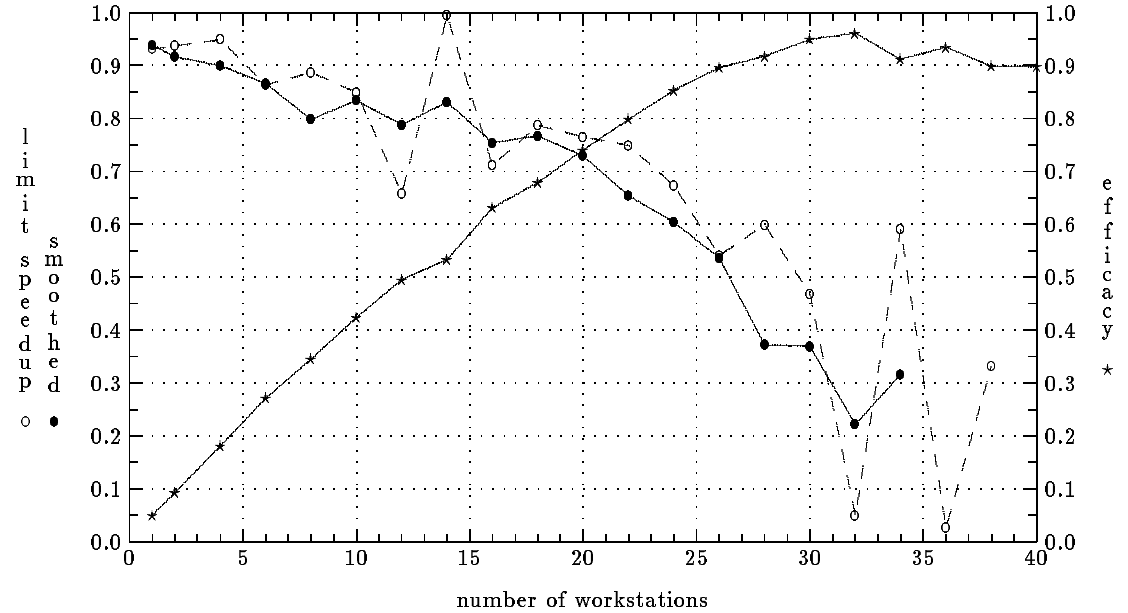

Fig. 3. Limit speedup and efficacy for temperature distribution.

Figure 3 shows the limit speedup for this experiment. Although the measurements were made in a dedicated network, averaging over three distinct runs, the nondeterminism of the ethernet technology produces some noise in the run time curve, which leads to drastic variations of the limit speedup as a differential curve. We therefore included in figure 3 a smoothed limit speedup curve, obtained by forming the average of each three consecutive data points. We see that adding a further processor to a parallel run with some 30 processors roughly increases speedup by 0.5 – the additional processor may only be used half.

The efficacy curve rises fairly linear with a marked decline in slope at $p = 32$. Then it decreases, with some nondeterministic effects. Hence with 32 processors our parallel algorithm is cost optimal in the sense of our model.

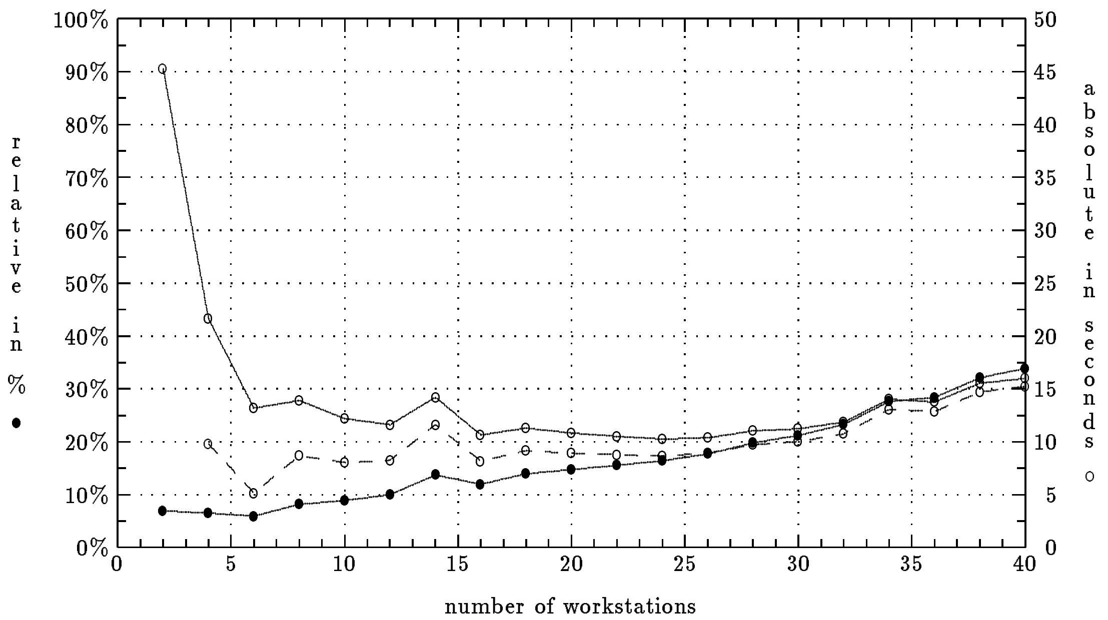

Fig. 4. Sequential component for temperature distribution, see text.

Figure 4 shows the sequential component for this experiment in the absolute and relative scale. The solid lines show the sequential component calculated from equation (4), whereas the dashed line shows it with respect to equation (5):

$$seq'_{\mathrm{m}}(p) = \frac{p \cdot T(p) - 2 \cdot T(2)}{p - 2} \tag{5}$$

The absolute sequential component is more or less constant in the region $10 \leq p \leq 32$, supporting the model assumption, that the sequential component (and consequently the overhead) for this fully parallelisable application is independent of the number of processors. This assumption is violated for large number of processors, since here the bottleneck of communication becomes critical. Its violation for a smaller number of processors ($p \leq 14$) is astonishing at first sight. However, it only points out that the sequential computation, despite an algorithm as close as possible to the parallel one was used, does not fit into our model. Instead of equation (4) which determines $seq_m(p)$ from $T(1)$ and $T(p)$ we could have used equation (5) determining $seq'_{\mathrm{m}}(p)$ from $T(2)$ and $T(p)$. Since our model only is an approximation, these two variants $seq_{\mathrm{m}}(p)$ and $seq'_{\mathrm{m}}(p)$ do not coincide exactly (see figure 4). However, the dashed variant of equation (5) uses the value $T(2)$, which as a parallel run suits our model assumption better than the value $T(1)$ of the sequential run. The violation of our assumption for smaller p hence is due to $T(1)$, an effect which becomes less visible for larger p.

3.2 Gene and Protein Sequence Analysis

Our second example is the homology analysis of genetic and protein sequences [8]. Genes and proteins are linear macromolecules, chemically described by their sequences of nucleic bases or amino acids. For the computer scientist this molecule is a string of characters. When a new gene or protein is found in the laboratory, data bases are searched for similar genes and proteins with an already known biologic function. The results of these similarity searches may give a clue to the rôle of the new macromolecule [18]. This similarity search is performed by a dynamic programming algorithm, which determines those strings in a data base which have a short *string edit distance* [31] to a given string. This algorithm can be parallelized by decomposing the data base of known molecules. Since there are more than 100.000 sequenced genes, the degree of possible parallelism is extremely high and even surpasses the number of processors usually found in a massively parallel system. The data base is decomposed and distributed to the individual workstations as part of the software installation process. At the beginning of the computation the sequence of the new gene is broadcast in a tree like communication pattern down to all participating workstations. The gene with the minimum string edit distance to the given one can be determined, when all workstations have sent their best candidate up the tree. No further communication is necessary.

For our test runs we chose two different configurations. Measurements in a local area network consisting of a variety of machines are reported by [29]. A speedup curve scaled to Sparcstation1 units is reproduced in figure 5.

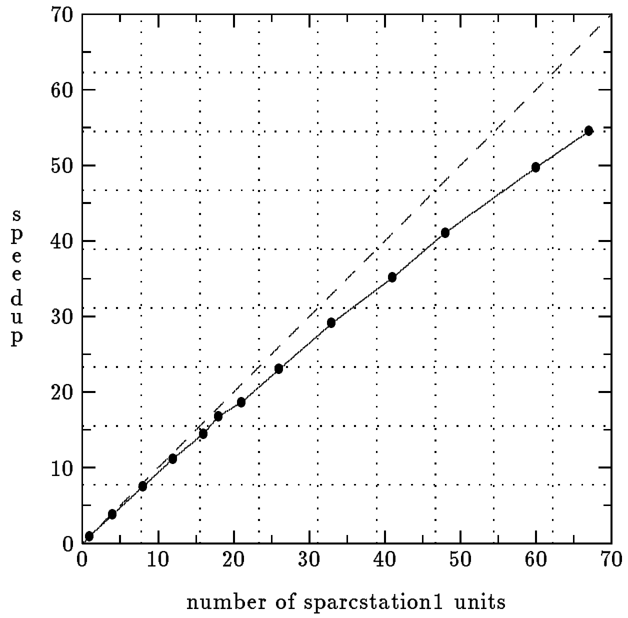

Fig. 5. Speedup of sequence analysis in local area network.

[8] describes a massively parallel approach with several hundred workstations on all five continents. Several experiments with varying sizes of the application and different numbers of processors were made [8], [12], [29]. The following table gives the sequential run times of such experiments on a Sparcstation1, the run time of a parallel run with the given number of workstations and the resulting speedup with respect to a Sparcstation1. The machines used in these experiments range in performance from a relatively slow Sparcstation1 to faster RS6000 models. This must be kept in mind when judging the speedup values presented below. However, in a configuration extending over the entire globe it is practically impossible to get a detailed machine type and machine model

information of every employed workstation. Nevertheless the marked break down in speedup noticed at $p \geq 32$ in the first example does not show up in the second example.

sequential	**parallel**	**workstations**	**speedup**[1]
16.4 hours	190 sec	642	310
21.5 hours	5.0 min	803	258
6.6 days	18.4 min	803	516

In this second example there is nearly no communication at all and the computation benefits from the extremely high degree of possible parallelism. Here the limiting factor for higher speedup is of an organizational nature: Coordinating nearly thousand workstations for a single computation, fault tolerance techniques – since it is out of the control of the user in Zurich, when someone in Australia suddenly turns off his workstation – highly portable software which must be installed on all workstations, the administration of userids and passwords for all networks, several thousand processes which must be started remotely and so on. Experiences however indicate, that for this type of applications, massively parallel approaches using workstation clusters in wide area networks are a feasible and interesting option.

4 Applications and Performance

We may roughly classify applications as typical for hypercomputers or supercomputers with the help of the following criteria:

Criterium	**Hypercomputer**	**Supercomputer**
Degree of parallelism	high (10 – 5.000)	low (1 – 10)
Locality	large (only several KB)	nearly arbitrary
Granularity	large (10^6 flop, 0.1 sec)	nearly arbitrary
Dominating operations	integer arithmetic, boolean comparisons	floating point
Control structure	arbitrary	linear, no jumps
Memory access	irregular, indirect	regular, linear structures arrays, vectors
Size of data	medium ($\leq$ 20MB per proc.)	large (upto several GB per proc.)

The degree of parallelism provides an upper bound for the number of processors, which may be used. The main factors for performance are locality and granularity. We already identified communication as the bottleneck of the hypercomputer. It comprises two components, *latency time* and *bandwidth*. In case of small locality, the intermediate results of a computation depend on big amounts of other intermediate results, which are dispersed over all employed workstations. It is therefore necessary to transport large chunks of data over a medium of relatively small bandwidth. In case of small granularity we must communicate very often during the computation over a medium of high latency time.

In the example of the temperature distribution the degree of parallelism could be chosen in a wide range, from one processor upto a parallelisation scheduling every grid point to an individual workstation. The granularity of the parallel algorithm depends on the number of processors and with larger p we obtain a too small grain size. The locality per processor, i. e. the amount of data every single workstation must communicate, remains constant. The overall amount of data, which must be communicated over a network with exclusive access, hence is proportional to the number

[1] With respect to a Sparcstation 1.

of processors, and is adversely affected by larger p.

In the example of the gene sequences the degree of parallelism is very high. Locality is high and independent of the number of processors, granularity can be estimated to be uncritical. This explains the relatively good behaviour of this application even with several hundred processors.

Typical applications for a hypercomputer are most kinds of combinatorial optimisation problems and all "embarassingly parallel" problems, which allow a very high degree of parallelism, but do not make too high demands on communication. Applications which typically should not be run on a hypercomputer are large scale problems involving local physical laws, like three dimensional hydrodynamic and electrodynamic problems or wheather simulation.

5 Organizational Aspects

Using workstation clusters requires special *organizational and infrastructural support*. Although there are already commercial systems available offering customer support, the installation, maintenance and programming of clusters still requires special know how, which is beyond traditional workstation administration and the usual industrial programming style. Due to the heterogeneity of various brands of UNIX systems, the complexity of distributed systems, insufficiently tested and developed hypercomputer software, production runs need special supervision: During the execution of hypercomputer applications, processes are started on remote machines and resources are allocated throughout the entire network. Other users from time to time are bothered by processes or command interpreters which were not correctly terminated, are running amok and are consuming CPU power, possibly to the extent requiring the rebooting of a workstation.

Since the hypercomputing software has to interact with the operating system at a very low level, *heterogeneous clusters* provide special problems. This begins with different or nonexisting system calls for determining the load status of a processor in different implementations of UNIX and ends with the unique challenge of process migration in heterogeneous hardware environments. Smooth operation of a hypercomputer can only be expected in homogeneous configurations.

On the average the *level of security* in a network of workstations is lower than in mainframe installations. Furthermore, hypercomputer systems require network services like remote login or file transfer, which often are disabled in environments, where confidentiality is of a special concern. Therefore, either no sensitive data should be stored on any machine of the cluster or the entire network should be inaccessible from the outside world.

A hypercomputer therefore is suited for *academic and research environments* capable of providing the necessary infrastructure, but it should (not yet) be used in industrial environments which are in need of stable and reliable production runs. In order to minimize interference with other users of the network, there should exist guidelines on those periods of time during which individual respectively hypercomputer activities have preference in the cluster. Good cooperation of all local workstation users and fully established and accepted operating procedures are essential for nondedicated cluster computing.

6 Trends and Developments

In the present arena of hypercomputer applications there are four trends. More and more academic environments provide *small dedicated clusters* of six to thirty high performance workstations for batch operations of parallel applications. They are typically homogeneous configurations, interconnected with high speed networks like FDDI or with additional ethernet connections and are not available for interactive or sequential jobs. They are used as a cost effective and reasonably powerful substitute for supercomputers. In such environments PVM [15] or similar systems are used as hypercomputer software, together with batching software like DQS [24] for scheduling purposes. Organizational and load balancing problems are easily solved in such environments. Clusters of this kind are used for example at the universities of Zurich, Vienna, Graz, Innsbruck, Munich, Utah, at CERN and at the Fermi national accelerator laboratory, to name just a few examples.

Other institutions use workstation clusters in a *nondedicated background mode* with the aim of using the CPU cycles of machines which otherwise would be idle. Despite their high performance personal desktop workstations in an academic research environment are utilised only 10 to 20 percent of their run time. Figure 6 shows the percentage of CPU cycles during which such workstations were idle, averaged over all running workstations in an academic network of 38 Sparc workstations. The values were obtained by a software monitor, reading UNIX kernel variables every 5 minutes. One may observe some activity during the night, mostly dumps, news, automated system administration and operating system activities. People come to work at 9:00 and leave around 18:00. At 16:00 a parallel application was started on all machines which were turned on at that time.

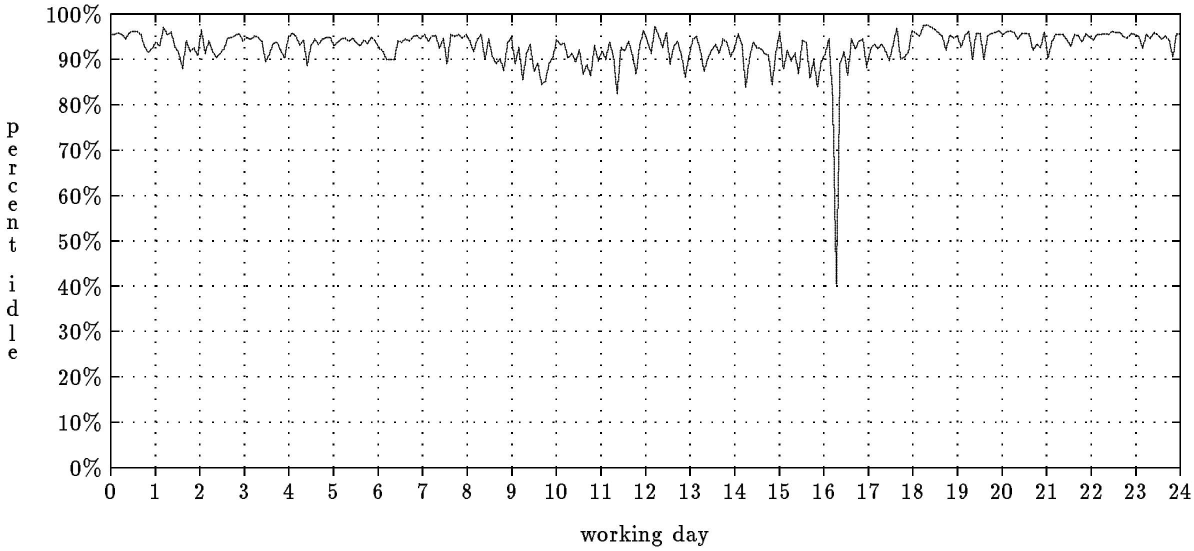

Fig. 6. Average idle percentage of 38 personal workstations.

This *low average utilization* at first sight seems to indicate a waste of resources and, if presented to management, could even lead to a cut in funding. However, we should not interprete figure 6 in this way: Today's graphical user interfaces, desktop publishing, visualisation, multi media and scientific programs are typical applications for workstations and they all need high peak performance for an adequate reaction to interactive input, but producing low average utilization in long term measurements. Sharing the CPU with other processes when such an application is running would quickly increase interactive response time to a level, which is inacceptable for the user. During longer periods of inactivity – to be detected by the hypercomputer – and outside normal working hours this computing power could be made available to other users.

In this approach care must be taken not to interfere with interactive users. This makes it necessary to remove hypercomputer load from a machine, as soon as keyboard activity or load measurements indicate the presence of such processes. Grace to its automatic load balancing mechanism, Linda and its derivates [1] are the preferred software platforms. Piranha [5] is a project exploiting expecially idle workstations, Parform [10] aims at automating load balancing for specific application problems. Such nondedicated clusters may be found at the universities of Zurich, Saarbrücken and Yale.

As the software technology of effectively utilising clusters becomes more and more available, also the hyper*super*computer consisting of serveral *interconnected vector- and supercomputers* is studied for extremely compute intensive problems. [26], [20]. The National Science Foundation NSF presently is integrating their centers for studying cluster computing at the supercomputer level [21].

Workstation clusters also provide excellent solutions for *special applications* where mainly integer operations are used. The combined fast integer units of clustered RISC workstations can easily surpass the traditionally weak scalar integer units of supercomputers. Examples are the above mentioned gene sequencing problem and problems of structural chemistry [22], optimisation problems or cryptologic algorithms.

Further developments in hypercomputing software must try to remedy problems associated with infrastructural problems. Presently there exist tools for the administration of individual UNIX workstations and networks, but there is still insufficient support for the administration of a cluster of workstations from the hypercomputing point of view. Especially aspects like accounting, user authentification, load monitoring, software installation, system configuration and external device access need different support in a hypercomputer than in a traditional network of computers. The problem of how to achieve a reasonable load balancing in a practical and efficient way still remains an open problem.

Although PVM plays the role of a de facto standard for the user and the software developer and even is evolving to an official message passing interface (MPI) standard [13], there is a lack of official *standards* or even agreed concepts at *higher levels of abstraction* for programming.

We already explained above, that hypercomputers presently are unstable systems. Further efforts must be made to solve associated problems and to be able to cope with workstations, which in nondedicated clusters may suddenly be turned off by their users.

In the near future, hypercomputers will solve more and more routine problems, which traditionally were solved on mainframes. The continuing importance of supercomputers and special architectures like massively parallel systems will lie in the solution of grand challenge problems and in applications which are not suited for cluster computers.

References

1. S. Ahuja, N. Carriero, D. Gelernter, Linda and Friends. *IEEE Computer* **19**(8), 1986, 26-34.
2. M. Asawa and S. Misra, Robinhood: Resource Sharing by Time Stealing Between DOS PCs on a LAN, Technical report 91/2, James Cook University North Queensland, cctober 1991.
3. J. Baud et al., SHIFT: The Scalable Heterogeneous Integrated Facility for HEP Computing. In Y. Watase (ed.), *Computing in High Energy Physics 91*, Universal Academy Press, Tokyo.
4. R. D. Bjornson, Linda on Distributed Memory Multiprocessors. PhD Thesis and technical report YALEU/DCS/RR-931, University of Yale, November 1992.

5. R. Bjornson, N. Carriero, D. Gelernter, T. Mattson, D. Kaminsky and A. Sherman. Experience with Linda. Technical report 866, department of computer science, Yale University, august 1991.
6. R. Butler, E. Lusk, User's Guide to the p4 Parallel Programming System. Technical report, Argonne National Laboratory, august 1992.
7. J. Corbin, The Art of Distributed Applications. Sun Technical Reference Library, Springer Verlag, 1991.
8. C. H. Cap, Massively Parallel Computing in Wide Area Networks. Technical report, department of computer science, University of Zurich, 1993.
9. C. H. Cap, Architekturelle Entscheidungen zur Unterstützung parallelen Rechnens in Workstation Clustern und deren Realisierung in existenten Produkten. Proceedings 2. ITG/GI Workshop Workstations, Hagen 1993, VDE Verlag.
10. C. H. Cap and V. Strumpen, Efficient Parallel Computing in Distributed Workstation Environments. *Parallel Computing*, **19**(11), 1993.
11. C. H. Cap and V. Strumpen, The Parform – A High Performance Parallel Platform for Efficient Computation in a Distributed Workstation Environment. Technical report IFI-TR 92.07 Institut für Informatik, Universität Zürich, 1992.
12. C. H. Cap and V. Strumpen, Massively Parallel Computing in the Internet – Entry to the SuParCup'93. Department of computer science, University of Zurich, 1993. This entry won the first prize of the SuParCup93.
13. J. J. Dongarra, R. Hempel, A. J. G. Hey and D. W. Walker, A Proposal for a User-Level, Message-Passing Interface in a Distributed Memory Environment. Technical report CS-93-186, University of Tennessee, Knoxville, january 1993.
14. G. Geist, M. Heath, B. Peyton and P. Worley. PICL a Portable Instrumented Communication Library. Technical report ORNL-TM-11130, Oak Ridge National Laboratory, Juli 1990. netlib@ornl.gov.
15. G. Geist and V. Sunderam. The PVM system: Supercomputer Level Concurrent Computation on a Heterogeneous Network of Workstations. *Proceedings of the Sixth Distributed Memory Computing Conference*, IEEE Press, 1991,258–261.
16. M. Griebel, W. Huber, T. Störtkuhl and C. Zenger, On the Parallel Solution of 3D PDEs on a Network of Workstations and on Vector Computers.
17. R. Hempel, The ANL/GMD Macros (PARMACS) in FORTRAN for Portable Parallel Programming Using the Message Passing Programming Model, User's Guide and Reference Manual. Gesellschaft für Mathematik und Datenverarbeitung mbH (GMD), november 1991.
18. R. J. Lipton, T. G. Marr and J. D. Welsh, Computational Approaches to Discovering Semantics in Molecular Biology. *Proc. IEEE* **77**(7) 1989, pp. 1056–1060.
19. M. Litzkow, M. Livny and M. Mutka, Condor – Hunter of Idle Workstations. Technical report 730, University of Wisconsin, december 1987.
20. H. P. Lüthi, J. Almlöf, Network Supercomputing: A distributed-concurrent direct SCF scheme. *Theoretica Chimica Acta* **84**, 443–455 (1993).
21. C. D. Marsan, NSF pursues computing without walls. *Federal Computer Week* **6**(35), 53, november 30, 1992.
22. H. Nakanishi, V. Rego and V. Sunderam, Superconcurrent Simulation of Polymer Chains on Heterogeneous Networks. *Proceedings Supercomputer 92, Minneapolis*, IEEE Press, 1992, 561-569.
23. Parasoft Corporation, Express Version 1.0: A Communication Environment for Parallel Computers. 1988.
24. L. R. Revor, DQS users guide. Computing and Telecommunications Division, Argonne National Laboratory, september 1992.
25. R. Roth, T. Setz, LiPS: A System for Distributed Processing on Workstations. Technical report, FB-14 Informatik, Universität des Saarlandes, february 1993.
26. W. Scott, P. Arbenz, S. Vogel and H. P. Lüthi, Network Supercomputing. *EPFL Supercomputing Review*, november 1991.
27. G. Shoinas, Issues on the Implementation of Programming Systems for Distributed Applications. Technical report, University of Crete, 1991.
28. C. Sprenger, Netzwerk-Computing. Diploma thesis, Institut für wissenschaftliches Rechnen, ETH Zurich, march 1992.
29. V. Strumpen, Parallel Molecular Sequence Analysis on Workstations in the Internet. Technical report, department of computer science, University of Zurich, 1993.

30. L. H. Turcotte, A Survey of Software Environments for Exploiting Networked Computing Resources. Technical report of the engineering research center for computational field simulation, june 1993.
31. R. A. Wagner and M. J. Fischer, The String to String Correction Problem. *JACM* **21**(2) 1974, pp. 168–173.

Acknowledgements

I want to thank professor Lutz Richter for an environment which is fun to work in and for his interest in my research. The financial support of Firma Siemens AG, München and Schweizer Bundesamt für Konjunktur und Wirtschaftsförderung made this work possible. Volker Strumpen helped with the implementation of the examples and with some measurements. Edgar Lederer provided numerous ideas and suggestions and Silvano Maffeis worked on alternative implementation techniques. The Suparcup organized by professors Werner Meuer and Wolfgang Gentzsch gave the stimulus and a reward for the second example, the measurements for which could be made when many friends and collaborators provided us with accounts on their workstations.