PROJECT REPORT.

Introduction

About the Project

This report is based on a project done in cooperation with the Norwegian High Performance Computing Consortium (NOTUR) [1]. More specifically, I work with the Emerging Technologies (ET) project at NOTUR [2].

The task at hand is to aid them in determining how to best evaluate which machines best meet their needs.

The NOTUR site in Trondheim has different users with different needs for high performance computing, including:

- Statoil – An oil company that does geological and seismic modelling in the search for petrochemical resources
- DNMI - The Norwegian Meteorological Institute. Does climate modelling to create weather forecasts.
- NTNU – The Norwegian University of Science and Technology. Different user groups, e.g. the chemical institute, which does a lot of molecular simulations.

They are currently running a timeshared 220-processor SGI Origin 3800 SMP (shared memory multiprocessor) system [3], but are investigating if different architectures could meet their needs. Price vs. performance is an important factor. For a brief description of the Origin 3800, see *Appendix A – SGI Origin 3800, p. 21*.

"The mission"

> **BENCHMARKING TECHNIQUES OF PARALLEL AND DISTRIBUTED SUPERCOMPUTING SYSTEMS**
>
> Develop benchmarking techniques in conjunction with NOTUR staff and associates to help evaluate which HPC systems NTNU and NOTUR should acquire in the future.
>
> The project will include analyzing standard user applications from up to 3 different areas (e.g. meteorology, petroleum engineering and chemistry).
>
> - What kind of computations do the applications involve?
> - What kind of resources/platforms/communication patterns are important for the respective application profiles?
>
> The profiling of the applications may be done with the use of public domain and/or commercial profilers, depending on the applications and platforms evaluated. It may also involve developing in-house profiling tools as well as modules.

Architectures

Supercomputer – SMP

The Shared Memory Multiprocessor (SMP) architecture has over the last years been a "standard" architecture for high-performance computers.

High-end SMPs are often expensive machines with up to several thousand processors working in parallel on a shared memory system. The typical rationale for choosing these types of systems has been that they have given the best available performance. The drawback of these high-end systems has often been the price.

The scalability of SMPs vary, some SMPs are scalable in the sense that you can add modules of more processors, more RAM, etc. up to a certain point. They are however not as scalable as for instance cluster solutions, where adding a computing node is a rather straightforward task of adding yet another machine to the cluster.

Another type of SMPs becoming more popular in the recent years is the multiprocessor workstation. These type machines can be a good choice as computing nodes in computing networks.

Figure 1 - The Apple G4 Dual Processor Workstation

Clusters

The recent advances in high speed networks and improved microprocessor performance are making clusters or networks of workstations an appealing vehicle for cost effective parallel computing.

Clusters built using commodity hardware and software components are playing a major role in redefining the concept of supercomputing. [4]

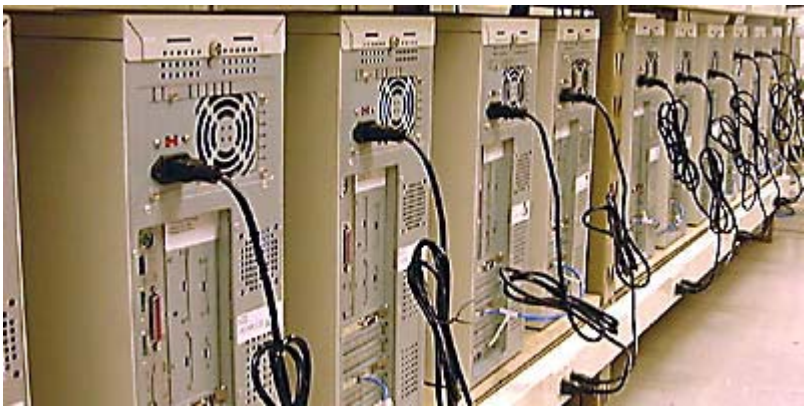


Figure 2 - A Beowulf cluster of workstations

The trend in parallel computing is to move away from specialized platforms to cheaper, general purpose systems consisting of loosely coupled components built up from single or multi-processor workstations or PCs. This approach claims to have a number of advantages including that of being able to build a platform for a given budget, which is suitable for a large class of applications and workloads.

There is a definition of a cluster of computers as a collection of (basically workstation type) computers working together to solve a tightly coupled problem. This means that in a cluster, there is a lot of communication and synchronization between the computing nodes. If on the other hand the same set of computers were working on a loosely coupled problem or if they were each working on different problems altogether, you'd rather call it a compute farm. [30]

NOTUR has projects using dual Intel Itanium workstations as nodes in a cluster setting (typically processors of 2*800MHz), and gigabit-networks connecting them. The nodes can be set up in different ways, depending on the needs and characteristics of the application. One example is having one of the processors in the node being dedicated to running the operating system, while the other processor does the actual computing.

The application signature is the characteristics of the communication between the nodes, and the characteristics of the actual computing. This signature will determine what parts of the cluster system that will be stressed, and will differ from application to application. Knowing the signature of your application is therefore important when evaluating how well suited it is for a cluster system.

(TODO: Insert more)

Grid

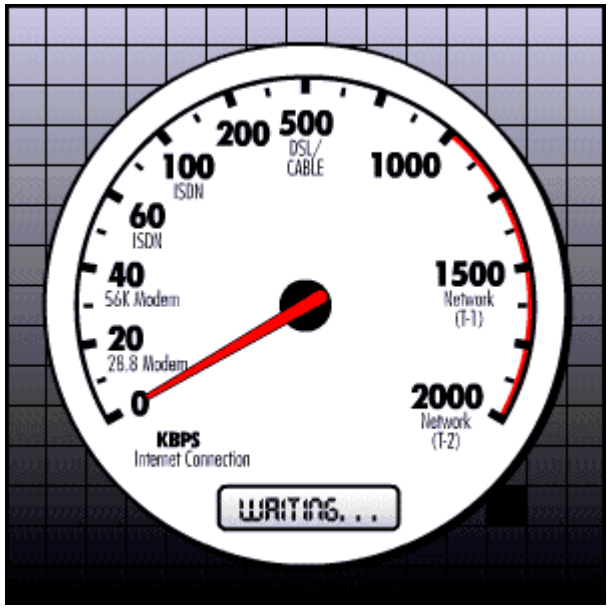
The computational power grid is analogous to the electric power grid. Grid computing allows to couple geographically distributed resources and offers consistent and inexpensive access to resources irrespective of their physical location or access point. It enables sharing, selection, and aggregation of a wide variety of geographically distributed computational resources (such as supercomputers, compute clusters, storage systems, data sources, instruments, people). Thus allowing them to be used a single, unified resource for solving large-scale compute and data intensive computing applications. [5]

The Globus project and others are currently working on enabling the Grid as a computational framework [6, 7, 8]. There is still a lot to do before it is commonly available, but they claim that "the grid" has the potential to be for computing what the Internet was for information sharing.

I believe that the Grid is still too immature a technology to be considered for Notur in this project, but that said Notur could possibly build a "quasi grid" internally in their organization, but that wouldn't constitute a true grid, which by definition has to be an open and dynamic structure. [6]

Benchmarking machines

What is a benchmark?



Benchmarking is a way of testing and measuring the performance of a computer. It is used to compare the performance of different computers both internally in a range of computers, and across platforms and architectures.

It generally considered important that the benchmarks are independent of the vendors of the machines they are testing, so they can be trusted as a neutral measurement of the performance.

Another issue conserning benchmarks, is the portability. It is important that a benchmark can be used across machines, and that it isn't tailored for a certain platform. Some would claim that the Photoshop-performance metrics [9] often used to promote Apple computers isn't fair, since Apple's are especially tailored to do well in this benchmark, and that they aren't as as good on general tasks as the Photoshop-metric could suggest.

But on the other hand, it is important that the benchmarks test the parts of the systems that are relevant to the performance experienced by the user, so if the user is planning to work mostly in Photoshop, then the Photoshop-metric migth be an excellent benchmark for him. The more general a benchmark is, the less useful it is for a *particular* application or domain, and conversely the more specific and narrow a benchmark is, the less useful it is for applications and domains outside that scope. [10]

A benchmark run typically generates a lot of data about the performance of the uncerlying machine. But it is important for the users of the benchmark that the results of a are easily understood, so therefore several benchmarks are ”boiling down” their results to a single metric summarizing the performance. This metric is easily compared across platforms and architectures.

The only totally accurate way to measure the performance of your system, however, is to test the software applications you use on your computer system. Benchmark results are measured on specific systems or components using specific hardware and software configurations, and any differences between those configurations (including software) and the production configuration may very well make those results inapplicable to the production component or system. [11]

Benchmarks are, at most, only one kind of information that you may use during the purchasing process. To get a true picture of the performance of a component or system you are considering purchasing, you must consult other sources of information (such as performance information on the exact system you are considering purchasing).

Different types of benchmarks

Benchmark and performance tests measure different aspects of processor and/or system performance. While no single numerical measurement can completely describe the performance of a complex device like a microprocessor, a PC or a cluster system, benchmarks can be useful tools for comparing different components and systems.

Benchmarks can be divided into two kinds, component and system. Component benchmarks measure the performance of specific parts of a computer system, such as a microprocessor or hard disk drive, while system benchmarks typically measure the performance of the entire computer system. In either case, the performance you see in day to day use will almost certainly vary from benchmark performance, for a number of reasons. First, individual components must usually be tested in a complete computer system, and it is not always possible to eliminate the considerable effects that differences in system design and configuration will have on benchmark results. For instance, system vendors sell systems with a wide variety of disk capabilities and speeds, system memory, system bus features and video and graphics capabilities, all of which influence how the system components (such as the microprocessor) and the computer system perform in actual use and can dramatically affect benchmark results. Also, differences in software, including operating systems and compilers, will affect component and system performance. Finally, benchmark tests are typically written to be exemplary of only a certain type of computer application, which may or may not be similar to your applications. [11]

Component Bencmarks

Component benchmarks set out to test different components in your system. But it is hard to directly compare two different runs unless you know more about the system. For instance, a graphics systems benchmark will most probably run faster if you upgrade your CPU.

It's impossible to totally isolate the components in your system. Every piece of your PC interacts with the others to some extent. Accordingly, you cannot simply compare a component benchmark of two different machines unless you know more about the rest of the systems.

Having said that, component benchmarks can still be useful when comparing different systems, just be careful to examine more than just the single-number metric most benchmarks boil down to.

There really are no benchmarks that let you really know how fast one particular piece is. What component benchmarks are highly useful for is comparing things. You can compare your system to itself after you've changed something, and you can compare your system with ones similar to it to see how it rates.

System Benchmarks

While component benchmarks try to measure the performance of specific subcomponents of a computer system, system level benchmarks try to measure the performance of a computer system as a whole.

System level benchmarks try to mimic *full applications* or application suites, and try to exercise the whole system in the same way as a realistic application would. For the personal computer market, we have benchmarks emulating standard applications such as

Microsoft Office, Adobe Photoshop, Quake, etc. [15] The SPEC CPU suite is another example of a system benchmark. [13, 14]

The SPEC CPU 2000 suite is made up from several different real-life applications, from different application domains such as chemistry, meteorology, etc. The applications are modified and ported to several platforms in order to be suitable for the benchmark.

By using industrial applications as a basis for the benchmark suite, SPEC is trying to ensure that the application signatures of the benchmarking programs are similar to the signature of real world applications. That is, that they exercise the same parts of the system as a typical application in that application domain is likely to exercise. This is an important issue when testing on the system level. Even though the system level benchmarks are designed to test the system as a whole, there always be differences from application to application when it comes to how much focus there should be on the different parts of the system. (For example, the requirements for a computer system that mainly is to a word processor are quite different from the requirements of computer system that is to run 3D games).

Therefore, a user who knows that he is going to use the computer system for computations on fluid dynamics, might test how different computers perform on the SPEC-benchmark that mimics this application domain, wheras another user might be more interested in climate modelling, and should pay more attention to the performance on that part of the benchmark suite.

Specific characteristics of benchmarks for multiprocessor machines

The difference between a single computer (or a single processor machine) and a machine where several processing units work together is an obvious one – we need communication between the computing nodes.

The benchmarking schemes of these systems, be they a multiprocessor computer such as a SMP (p. 3) or a collection of computers (e.g. a cluster, p. 3) need to simulate the *communication patterns* in a realistic way in addition to placing a realistic load on the other parts of the system.

A component level benchmark of a multiprocessor system might of course set out to test only the communication part of the system. The SKaMPI benchmark [17] is an example of such a benchmark, which is created to test the communication subsystem of a MPI-based cluster system.

A system level benchmark must test the communication part in addition to the rest of the system. System level benchmarks for parallel or cluster systems are created in much the same way as system level benchmarks for single node systems – the benchmark suites are often modified versions of real applications. The rationale is the same as for the single node case, to have as realistic a load on the system as possible for a given application domain.

Benchmark examples

LINPACK/LAPACK

The LINPACK (Linear Algebra Package) Benchmark [18] is one of the more famous floating point benchmarks of recent years, created by Jack Dongarra, which get its name

from a linear algebra package. The benchmark solves a dense system of linear equations. Over the years the characteristics of the benchmark has changed a bit. In fact, there are three benchmarks included in the LINPACK Benchmark report.

The LINPACK Benchmark is something that grew out of the LINPACK software project. It was originally intended to give users of the package a feeling for how long it would take to solve certain matrix problems. The benchmark stated as an appendix to the LINPACK Users' Guide and has grown since the LINPACK User's Guide was published in 1979. [18]

LINPACK is a very common benchmark for the pure "number crunching" capabilities of a computer, and there is a even a popular "Top 500" list [20] containing the LINPACK benchmark results. There are also different parallelized versions of the LINPACK benchmark.

The original goal of the LAPACK project [19] was to make the widely used EISPACK and LINPACK libraries run efficiently on shared-memory vector and parallel processors. On these machines, LINPACK and EISPACK are inefficient because their memory access patterns disregard the multi-layered memory hierarchies of the machines, thereby spending too much time moving data instead of doing useful floating-point operations.

LAPACK claims to be "transportable" instead of "portable" because, for fastest possible performance, LAPACK requires that highly optimized block matrix operations be already implemented on each machine.

LAPACK routines are written so that as much as possible of the computation is performed by calls to the Basic Linear Algebra Subprograms (BLAS). Highly efficient machine-specific implementations of the BLAS are available for many modern high-performance computers. The BLAS enable LAPACK routines to achieve high performance with transportable software.

It is possible to use LINPACK/LAPACK in a cluster environment to test the number crunching capabilities of the system, there is even a separate "Top 500" list for this: see [21].

SPEComp

Over the past decade several computer benchmarks have taken aim at parallel machines.

- SPLASH. Used by research community but have not been updated to current computer applications.
- Perfect. Included serial programs which the benchmarker had to turn into parallel versions.
- Parkbench. An effort to create an extensive parallel benchmark suite at system level. No longer an ongoing effort.
- SPEChpc suite. A currently maintained benchmark for high-performance computer systems. Includes large-scale computational applications.

Addressing the fact that parallelism no longer is just an issue for the High Performance Computing society, SPEC has created SPEComp [23, 24], a benchmarking suite that aims at mid-range parallel computers.

The SPEComp benchmarks are adapted from the SPEC CPU 2000 [25] suite of benchmarks. In SPEComp 2001, the benchmark suite is partitioned into a Medium and a Large Data set.

- Medium. For moderate sized SMP (shared memory multiprocessors) systems of about 10 CPUs. About 1.6 GB memory required per CPU.

- Large. Oriented to systems with 30 CPUs or more. Up to 6 GB memory required per CPU.

Runtimes can easily exceed 10 wallclock hrs on a single state-of-the-art processor.

The suite includes large, complex modelling and simulation programs of the type used in many engineering and research organizations. Application areas include:

- Chemistry
- Mechanical engineering
- Climate modelling
- Physics
- Image processing
- Decision optimisation

SPEC has also developed a methodology on how to run the benchmark suite, for instance rules on what optimizations are allowed on the source code, etc.

SPEC HPC

SPEC HPC (currently: SPEChpc96) is a benchmark suite that measures the performance of high-end computing systems running industrial-style applications. The SPEChpc line of benchmarks is especially suited for evaluating the performance of parallel and distributed computer architectures.

As the SPEComp suite, SPEChpc96 also represents a commitment to providing benchmarks that measure sustained performance, instead of the peak performance numbers still used widely today. The benchmarks within the SPEChpc96 suite represent real-world industrial applications that run on today's high-performance systems.

HINT

HINT or Hierarchical INTegration is a computer benchmarking tool developed at the Scalable Computing Laboratory (SCL) of Ames Laboratory, and is funded by the Office of Scientific Computing, U.S. Department of Energy (DOE). Unlike traditional benchmarks, HINT neither fixes the size of the problem nor the calculation time and instead uses a measure called QUIPS (QUality Improvement Per Second).

This enables HINT to display the speed for a given machine specification and problem size. Computers typically start up fast and slow down as they run out of fast memory and start using the main memory, or slow down even more if they have to access the disk. Such changes are easily visible with HINT generated data.

HINT is scalable and easily portable for a variety of architectures. It can be run on anything from a programmable calculator to a supercomputer.

The result from a HINT run is a graph of QUIPS vs. time, and is often very revealing about the system. You can easily see the performance impact when the systems runs out of cache (or L2 cache, memory, etc.) as a result of the memory requirements of the computation ever increasing.

Error! Reference source not found. [27] shows a graph of a HINT run on a workstation. You can easily see the performance drops as the computer in turn runs out of L1 cache (primary memory) L2 cache (secondary memory), RAM (main memory).

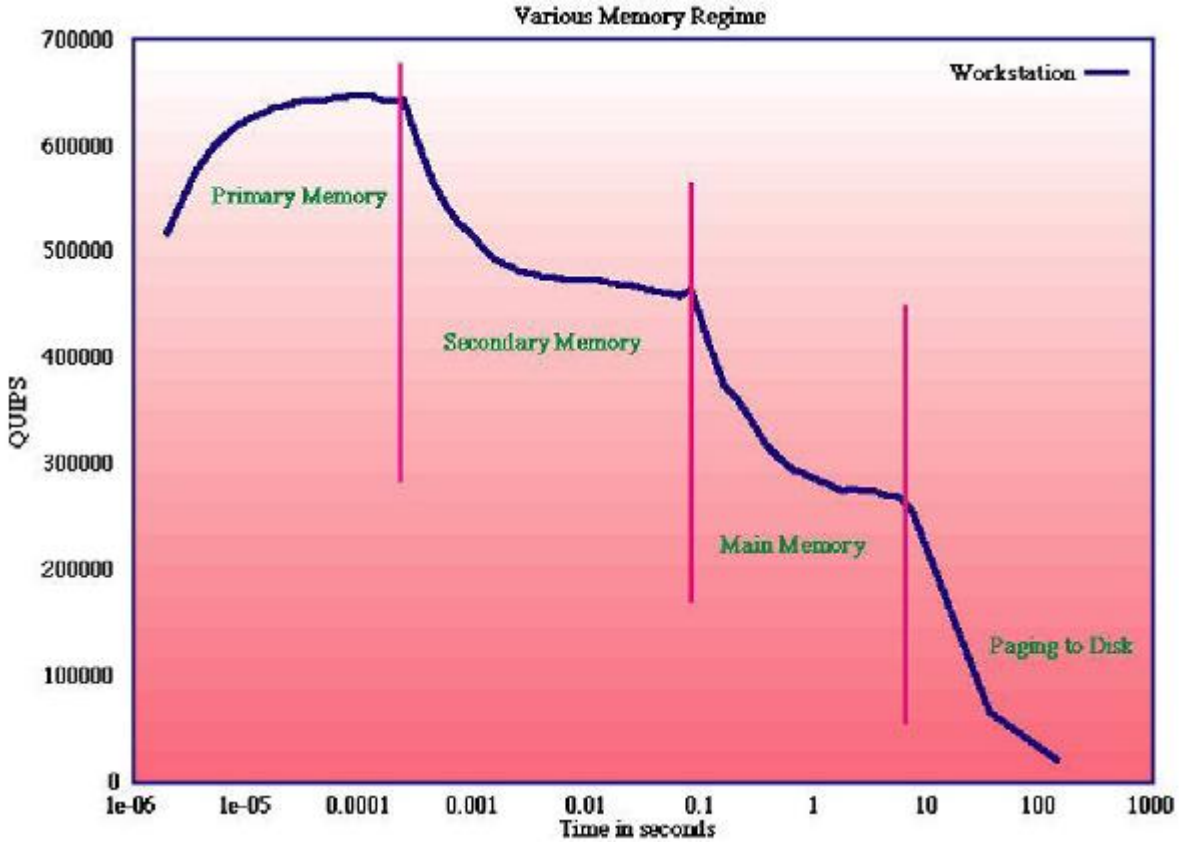


Figure 3 - HINT memory revealing graph

AHINT (Analytical HINT) [27] is a model that allows you to predict which impact a change in the parameters of a system will have on the HINT performance of the system. For instance, what impact replacing the processor with a faster one will have.

AHINT promises performance prediction based on a small set of design statistics. It presents the computer architect with a tool to build a balanced computer by varying design parameters based on cost and performance. The model also benefits users of existing computers. Consider a computer user trying to determine whether or not to upgrade from 128 K secondary cache to 512 K secondary cache. Advertisements indicate 'huge' performance increases. One could run HINT on the existing configuration, match the QUIPS curve using the analytical model to determine the input statistics, and then increase the secondary cache size input to the analytical model which would then predict the performance increase. At this point, an educated decision could be made based on cost increase versus performance increase. By fitting the model to existing HINT graphs, a user can find out the true computer performance parameters and compare them against manufacturer claims. For example, in one case we discovered a so-called "secondary data cache" was nothing of the kind, and served only as an extra memory for instruction storage. [27]

Recommendations for use

(TODO: Insert analysis for the Notur cluster setup, and recommendation for the benchmarks to use).

Profiling applications

Introduction to Application Profiling

The purpose of application profiling is to determine the behavior of an application, in the context of performance, so that analysis may be done. The goal of the analysis is to spot performance problems in the application so that algorithm or coding improvements may be made to the application by the developer.

For instance, on an IBM version of Linux that they use for their servers [], there are currently three application profilers avaiable; gprof, vprof, and cprof. I use these tools here as an example on how typical application profiling tools are used.

- gprof is the GNU profiler that is part of the binutils component of Linux. It supports profiling of single-threaded C applications
- vprof is a visual profiler that is shipped with the SuSE disrtribution and supports profiling of single-threaded C and C++ applications.
- cprof is an open source profiler, distributed by Corel, which supports multi-threaded C and C++ applications.

The 3 tools described here are similar in their implementation and use. The typical use of such application profilers, is to re-build your application with appropriate hooks compiled into your code. This is generally done by adding compiler and/or link options when the program is built. These hooks generate calls to routines that get invoked at entry and exit to the functions you want to get profile information about. Each of the tools use specific compiler and link options to accomplish this. The tool itself may need to be built and some routines link-edited into the application.

Data collection. Examples of the types of data that may be collected are:

- Call graph information
- Time spent in each function
- Statistical sampling

This data is collected when the application is run. Be sure to run the application with the parameters or options that will drive the component of the application you're looking to profile. An output file is generated during the run, which contains information about the execution in it's own internal format.

Post-processing of the output file to generate data in a format that can be analyzed. Once this step is complete, the application execution characteristics can be determined and adjustments to the application algorithms used and/or code can be made.

Different types of profiling tools

(TODO: General info on the different profiling approaches)

Specific characteristics of profiling tools for parallel programs

(TODO: What separates and distinguishes performance analysis tools for parallel environments from their sequential counterparts)

A review of performance analysis tools for MPI parallel programs

This section of the paper is based on an ongoing review process [29] of the performance analysis tools available for MPI [31] parallel programs.

In order to produce MPI applications that perform well on todays architectures, programmers need effective tools for collecting and analyzing performance data. Because programmers typically work on more than one platform, cross-platform tools are highly desirable. A variety of such tools are becoming available.

The reasons for poor performance of parallel, message-passing codes can be varied and complex, and programmers and users need to understand the problems in order to identify bottlenecks and troublespots in the program. Performance tools can help by monitoring a program's execution and producing performance data that can be analyzed to locate and understand these areas of poor performance.

The prevalent approach taken by the tools available today is to collect data during execution of the program, and then provide a post-mortem analysis and display of the performance information. Some tools are specialized to provide just one of these two phases, and some tools can also do a run-time analysis of the performance.

The reviewers focus on tools that are commonly available, and also on tools that work across different platforms, as opposed to vendor tools that work only on one platform.

The set of evaluation criteria are as follows:

1. Robustness. The program should be stable and should produce correct results.
2. Usability. Easy to learn, easy to use. Batch-capabilities are also treasured due to the nature of the applications and calculations commonly found in High-Performance Computing.
3. Scalability. The tools must be able to function on a range of architectures and applications.
4. Portability. Users are reluctant to having to learn a new performance analysis tool for each platform they work on. Having the same tool on different platforms makes the user more confident in the program, and also makes it easier to compare the results across platforms.
5. Versatility. The tools should be able to analyze the data in different ways and also to display the results in different ways.

In addition to these general criteria, there is also included a set of more specific criteria:

1. Support for hybrid environments (a combination of shared and distributed memory). The trend in HPC is towards such systems where SMP computers of up to 32 or more CPUs are interconnected by dedicated high performance networks.
2. Support for distributed heterogeneous environments. (The architecture of the computational grid).
3. Support for analysis of MPI-2 I/O, which is a feature included in the MPI 2 standard in order to reduce the bottleneck effect I/O is tending to have.

Tools reviewed

DEEP/MPI

URL	http://www.psrv.com/deep.html
Version	
Supported languages	Fortran 77/90/95, mixed Fortran and C
Supported platforms	Linux x86, SGI IRIX, Sun Solaris Sparc, IBM RS/6000 AIX, Windows NT x86

DEEP and DEEP/MPI are commercial parallel analysis tools from Veridan/Pacific-Sierra Research [32]. DEEP provides an integrated graphical interface for performance analysis of shared and distributed memory parallel programs.

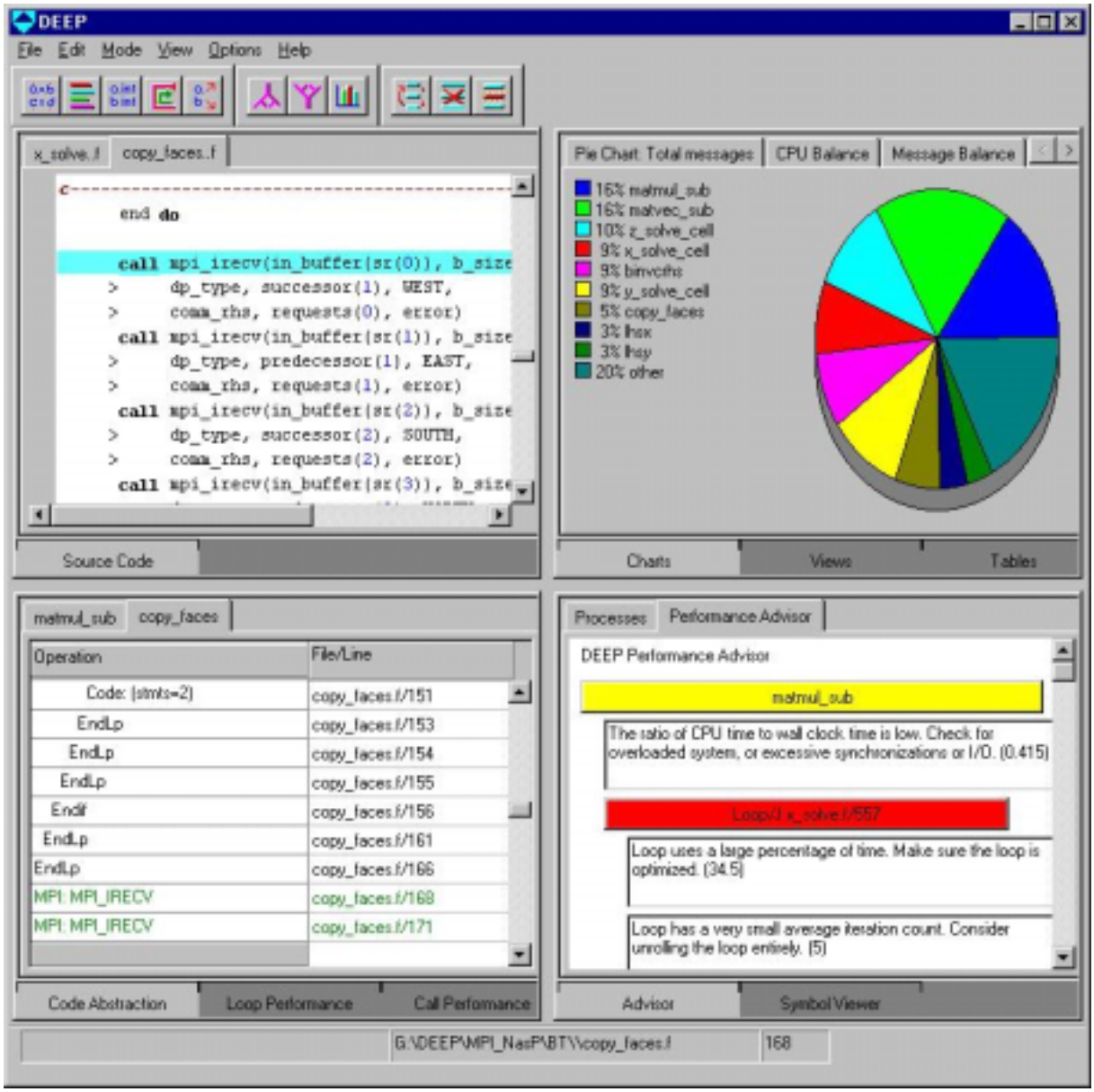


Figure 4 - DEEP/MPI screen dump

To use DEEP/MPI you must compile your MPI program with the DEEP profiling driver **mpiprof**. This step collects compile-time information which instruments the code. After executing your program, you can view the performance information using the DEEP/MPI interface.

The DEEP/MPI interface includes a call tree viewer for program structure browsing and tools for examining profiling data at various levels:

- Whole program: Data such as the wallclock time used by different procedures.
- Sub-program parts: For example loop performance tables.

The DEEP Performance Advisor suggest which parts of the program the user should examine first.

Other information provided is for example the CPU and message balance which shows the distribution of work and the number of messages, respectively, among the processes. DEEP supports the PAPI interface to hardware counters, and can do profiling based on any of the PAPI metrics. [33]

MPE Logging and Jumpshot

URL	http://www-unix.mcs.anl.gov/mpi/mpich
Version	MPICH 1.2.1, Jumpshot-3
Supported languages	Fortran, C, C++
Supported platforms	AIX, Compaq Tru64, UNIX, HP-UX, IRIX, Linux, Solaris, Windows NT/2000

The MPE (Multi-Processing Environment) library is distributed with the freely available MPICH implementation [34] and provides a number of useful facilities, including debugging, logging, graphics and some common utility routines. MPE was developed for use with MPICH, but can also be used with other MPI implementations.

MPE provides several ways to create logfiles, which in turn can be viewed and analysed with graphical tools also provided with MPE. The easiest way to generate logfiles is to link with an MPE library that uses the MPI profiling interface. The user can also manually insert calls to the MPE logging routines in his/her code.

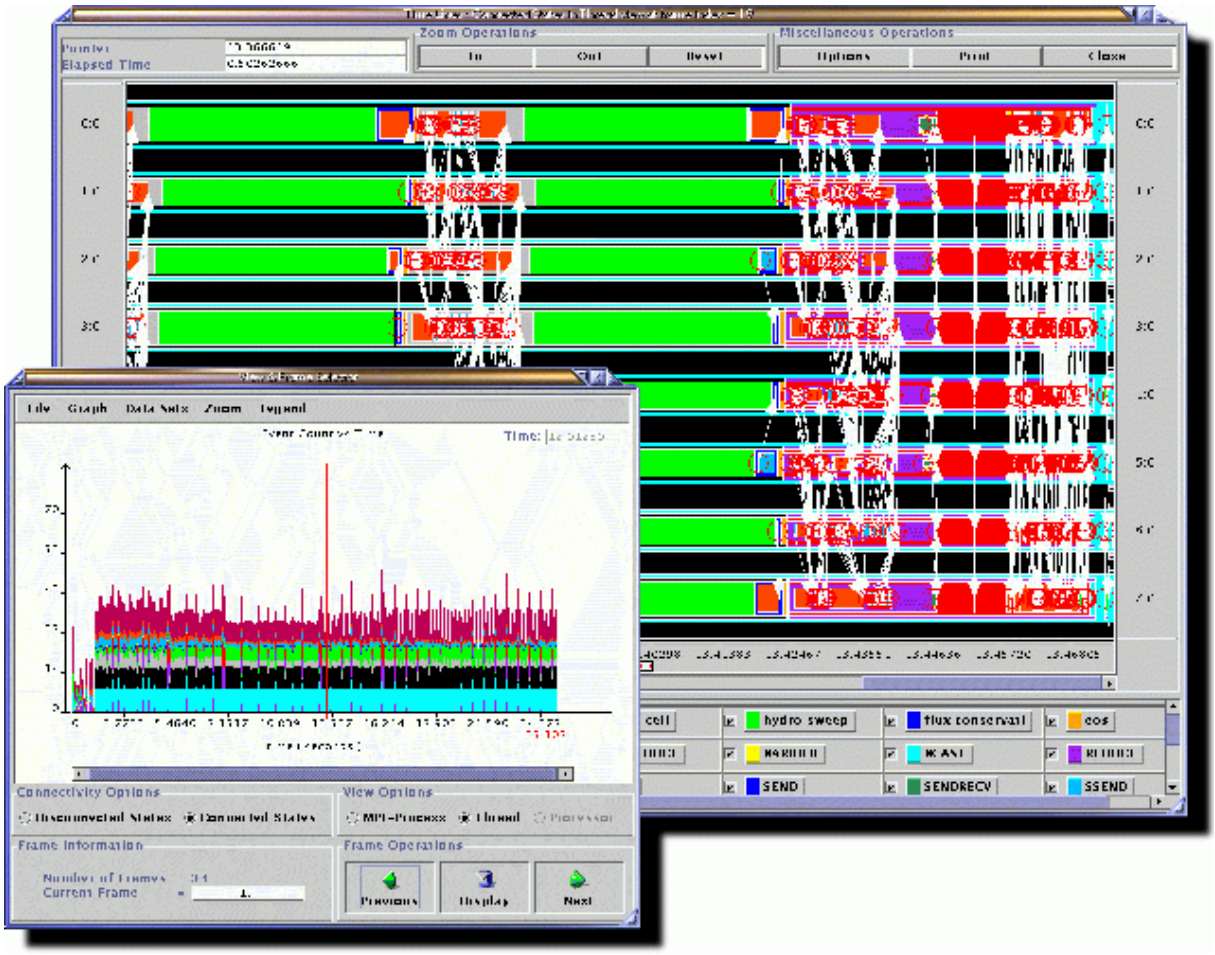


Figure 5 - Jumpshot 3 screen dump

The visualization and presentation tools here are also hierachical in the sense that the user first is presented with a statistical preview of the data, from which the user can select the parts of the information that seems interesting for a more detailed presentation. The user may select and deselect states so that only the interesting ones are displayed. In the case of a multithreaded environment, such as in a SMP node, the logfile may contain thread information, from which the presentation tools will show how threads are dispatched and used in the MPI program. There is click-for-more-info functionality built into the presentation tools.

Pablo Performance and Analysis Tools

URL	
Version	Trace Library 5.1.3, Pablo Performance Capture Facility (PCF), March 2001 release, SvPablo 4.1
Supported languages	Fortran 77/90, C, HPF (Pablo)
Supported platforms	PCF: Sun Solaris, SGI IRIX, Linux x86 SvPablo: Sun Solaris, SGI IRIX, IBM AIX, Linux x86

(TODO: Find URL – univ.of Illinois)

The Pablo Trace Library includes a library for recording timestamped event records and extensions for recording performance about MPI calls, MPI I/O calls and I/O requests. The performance records created are in the Pablo SDDF (Self Defining Data Format) format.

The MPI-extension is in form of a profiling library. Trace records are written that capture timing and parameter information for MPI calls in SDDF format. A separate SDDF file is created for each MPI process. Utility programs are provided for merging the per-process trace files, but they must be invoked manually.

The MPI I/O extension provides additional wrapper routines for

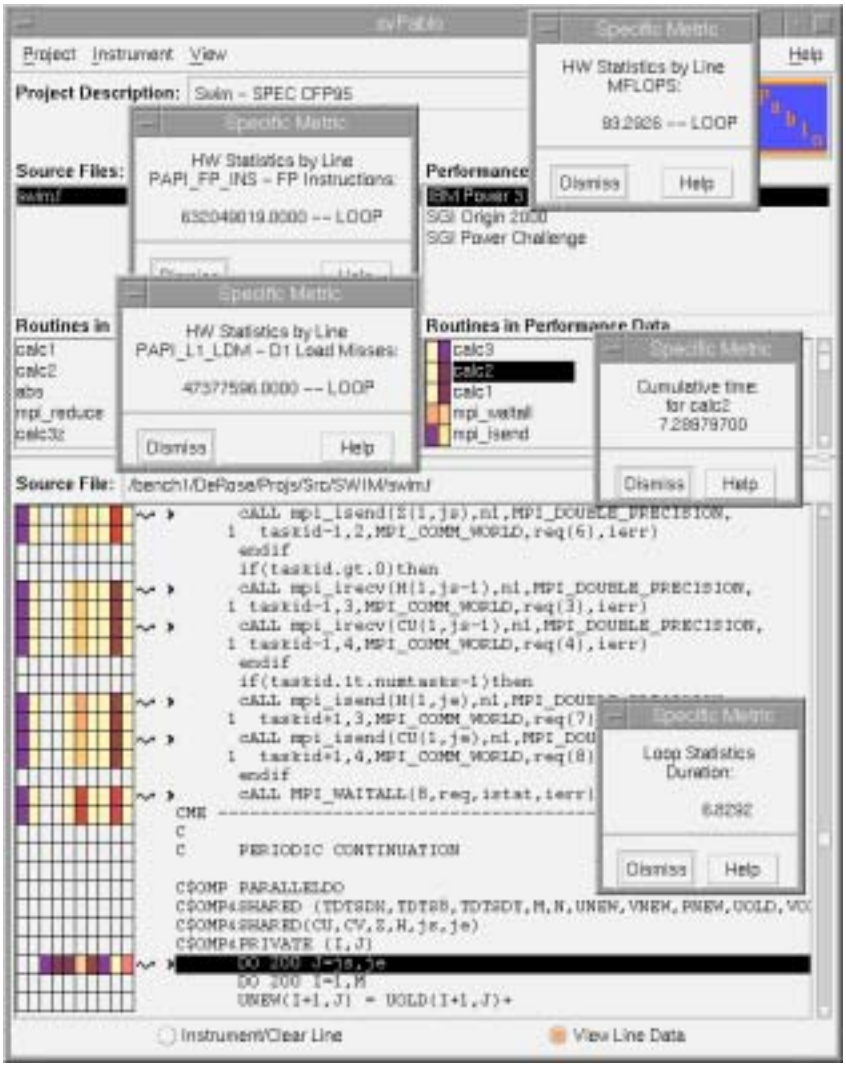


Figure 6 - svPablo screen dump

recording information about MPI I/O calls. The user may choose between a detailed tracing or tracing that provides more summary-mode that summarizes information for each type of MPI I/O call. Utilities are provided to analyze and produce reports from the MPI I/O trace files.

The Pablo Performance Capture Facility (PCF) supports MPI and provides several options for recording performance information for Unix I/O, Hierarchical Data Format (HPF) and MPI I/O operations. PCF can either write its output to SDDF-files, or it can display it run-time through the Pablo Autopilot facility. Unlike the Pablo Trace Utility, PCF is thread-safe, and can be used with mixed MPI and threaded programs.

SvPablo is a graphical interface for instrumenting soruce code and browsing runtime performance data. Applications can either be instrumented interactively or automatically. After running the application, SvPablo compares the results to the source code, and performs statistical analysis. It also displays a graphical representation that visually links performance information to the original source code. Plans are to use the PAPS portable hardware counter interface [33] in future versions of SvPablo.

Paradyn

URL	http://www.cs.wisc.edu/paradyn/
Version	3.2
Supported languages	Fortran, C, C++, Java
Supported platforms	Solaris (SPARC and x86), IRIX (MIPS), Linux (x86), AIX (RS6000), Tru64 Unix (Alpha), heterogeneous combinations

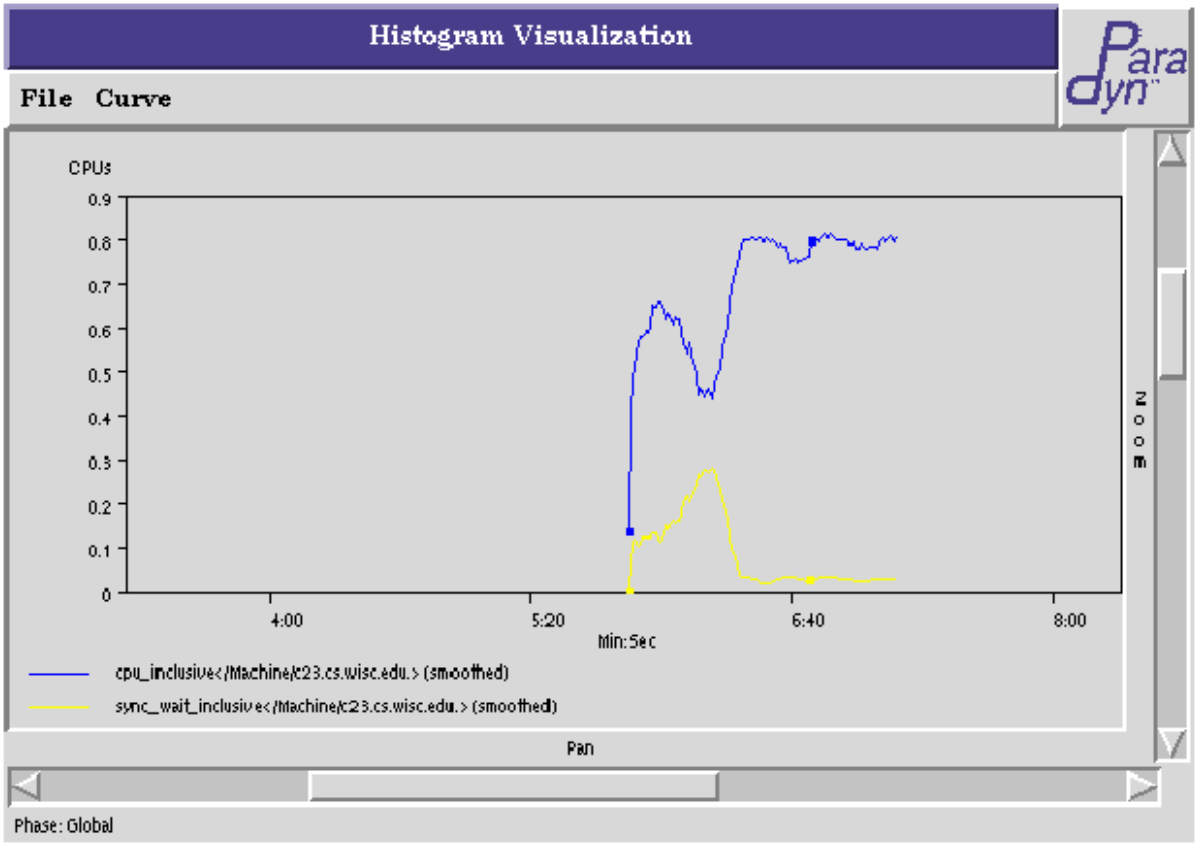


Figure 7 - Paradyn example screen shot

Paradyn is a tool for measuring the performance of parallel and distributed programs. It dynamically inserts instrumentation into a running application and analyzes and displays performance in real-time. It decides which information to collect while the program is running.

Paradyne distinguishes itself from the majority of the performance analysis tools in that the user doesn't have to modify the source code or use special compilers. The program instruments the binary image of the running program using the DyninstAPI dynamic instrumentation library [35].

To use Paradyn, MPI programs can only be run under the POE environment on the IBM SP2, under IRIX on the SGI Origin, and under MPICH on Linux and Solaris platforms.

Paradyn has two main components, the Paradyn front-end and user-interface, and the Paradyn daemons. The daemons run on each remote host where an application is running. The user interface allows the user to display performance visualizations, use the Performance Consultant to find bottlenecks, etc. The daemons operate under the control of the front-end to monitor and instrument the application processes.

The user specifies what performance data Paradyn is to collect in two parts: the type of performance data and the parts of the program from which to collect this data. The performance data can then be displayed using various visualizations. As an alternative to manually selecting which data to collect and analyze, the user can invoke the Paradyn Performance Consultant, which tries to determine the types of performance problems a program is having by testing various hypotheses such as whether the CPU is bound or if the program is experiencing excessive I/O or synchronization waiting time. The user can change the behaviour of the consultant by adjusting the thresholds used to evalute the hypotheses.

TAU

URL	
Version	2.9.11 (Beta)
Supported languages	Fortran, C, C++, Java
Supported platforms	SGI IRIX 6.x, Linux x86, Sun Solaris, IBM AIX, HP HP-UX, Compaq Alpha Tru64 UNIX, Compaq Alpha Linux, Cray T3E, Microsoft Windows

TAU (Tuning and Analysis Utilities) is a portable profiling and tracing toolkit that includes a visualization tool, Racy. In addition, TAU can generate event traces that can be displayed with the Vampir trace visualization tool.

TAU instrumentation must be added to the source code. This can be done automatically for C++ programs, manually through the TAU Instrumentation API or using a TAU run-time instrumentor which is based on the DyninstAPI dynamic instrumentation package. (An automatic instrumentor for Fortran is under development.)

Vampir

URL	
Version	Vampitrace 2.0, Vampir 2.5

Supported languages	Fortran 77/90, C, C++
Supported platforms	All major workstation and parallel platforms

Vampir is a commercially available MPI analysis tool from Pallas GmbH. Vampitrace, also from Pallas, is an MPI profiling library that produces trace files that can be analysed with Vampir. Vampitrace records all MPI and MPI I/O calls, but a runtime filtering mechanism can be used to limit the amount of trace data generated. Vampitrace also automatically corrects clock offset and skew.

Vampir provides several graphical displays for visualizing application runtime behaviour. Source code click-back functionality is available on platforms with the required compiler support. Message passing overview, statistical analysis of program execution, communicatoin operations, and a dynamical calling tree display are among the visualizations that can be provided.

Although the current version of Vampir can display information of up to 512 processes, this information can be overwhelming to the user, and the simultaneous display of thousands of processes would clearly be impractical. A new version of Vampir is under development, and promises a hierarchical way of presenting the processes.

Evaluation summary

Robustness	It is too early to report on the robustness of the tools.
Support	With the exception of Jumpshot, all the tools have fairly complete user guides and other supporting materials (tutorials/examples).
Usability	Paradyn has the drawback that it is difficult or impossible to use in batch queueing environments due to the interactive nature of the tool. Manual instrumentation for MPE/Pablo and TAU (Fortran) can be tedious to the point of being impractical. The source code click-back functionality of DEEP/MPI, SvPablo and Vampir is very helpful for relating performance data to program constructs. Scalable log formats such as MPE's SLOG and the new Vampitrace format are essential for reducing the time needed to load and display the results.
Portability	Of all the tools tested, Vampir is the only one that has been tested extensively on all major platforms and with most MPI implementations. The MPE and Pablo trace libraries are designed to work with any MPI implementation. However, on untested platforms we might expect glitches. Because of the platform dependencies in the dynamic instrumentation technology used by Paradyn, the implementation of some Paradyn features tends to lag behind on all except their main development platforms.

	The SDDF file format is intended to promote interoperability by providing a common performance data meta-format, but not many tools have adopted the format. Several tools use the PAPI cross-platform interface to hardware counters, and make use of hardware performance data in their performance analysis.
Hybrid	DEEP/MPI, MPE/Jumpshot and TAU all support mixed MPI and OpenMP programming, as will the next version of Vampir. The MPE and Pablo trace libraries, when used with MPICH, can generate trace files for heterogeneous MPI programs. Paradyn also supports heterogeneous MPI programs with MPICH. Although profiling of MPI I/O operations is possible with several of the tools, the only tools that explicitly adress MPI I/O performance analysis are the Pablo I/O analysis tools.

Recommendations for use:

(TODO: Insert analysis for Notur setup and recommendation for which tools to use).

Hardware

For Clusters:

- Brief history of parallel processors.
- IA32 (x86)
- IA64 – (Intel Itanium, 64-bit)

- GigaNet
- Other popular network interconnects

Appendix A – SGI Origin 3800

SGI Origin-3800L

- Rev MIPS R12000
- 220 CPUs
- 500 MHz
- 218 GB mem
- 8.0 MB L2
- 220 Gflop/s
- IRIX 6.5

(Source: notur.org)

From SGI's 3800-site [3]:

SGI® Origin® 3800

With the revolutionary SGI® NUMAflexTM computing model in the underlying system structure, you decide how much CPU, I/O, memory, and disk infrastructure to add to SGI Origin 3800. Every system component can be upgraded, maintained, or redeployed independently, so the SGI Origin 3800 system can evolve as quickly as your computing needs.

With the industry's most advanced NUMA architecture from SGI, you can configure your SGI Origin 3800 system up to a single 512-processor shared memory system, or use partitioning to divide it into as many as 32 partitions and run them as a tightly coupled cluster. Many application environments can improve availability by implementing a cluster of smaller partitions that can contain failures and leaving other partitions unaffected. Utilizing the ultralow- latency and ultrahigh-bandwidth NUMAlinkTM interconnect fabric as a communication vehicle, partitioning is an option that can deliver both high availability and high performance.

Processors :	16-512 using 4-processor C-bricks
System bandwidth:	Up to 716 GB/sec
Maximum memory:	1 TB
Router type:	8-port
Base I/O:	I-brick
	I-brick, P-brick, X-brick, D-brick

Built on the reliable SGI® NUMA architecture and IRIX® 6.5 operating system, SGI Origin 3800 servers work with your existing application software and are fully compatible with other IRIX OS- based workstations and servers. The applications you use every day transition effortlessly and perform better than ever. With the same familiar tools and operating system, you can integrate the series with no retraining. The SGI Origin 3800 server protects your investments thoroughly and ensures the availability of a wide range of open systems software into the future.

SARA Supercomputer Facility Installation
SGI, in conjunction with the Netherlands Organization for Scientific Research and the Netherlands Computing Facilities Foundation, has implemented an SGI® Origin® 3800 supercomputer at SARA Computing and Networking Services in Amsterdam. The grand opening was held on November 22nd and was attended by His Royal Highness Prince Wilem-Alexander. The facility will help the Dutch academic community to understand and resolve the world's most complex scientific, technical and medical issues.

REFERENCES

1. Norwegian High Performance Computing Consortium - webpage http://www.notur.org
2. Emerging Technologies (ET) project at NOTUR - http://www.notur.org/et/
3. Silicon Graphics presentation of the SGI Origin 3800 - http://www.sgi.com/origin/3000/3800.html
4. IEEE Computer Society Task Force on Cluster Computing – webpage http://www.ieeetfcc.org/
5. Grid Computing Info Centre (GRID Infoware) – webpage http://www.gridcomputing.com/
6. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* I. Foster, C. Kesselman, S. Tuecke. International J. Supercomputer Applications, 15(3), 2001.
7. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. I. Foster, C. Kesselman, J. Nick, S. Tuecke; January, 2002.
8. The Globus Project – webpage http://www.globus.org
9. *Photoshop Performance and Productivity Benchmarks,* Apple – webpage http://www.apple.com/creative/resources/photoshop/
10. *Standard Benchmarks for Database Systems*, Sigmod 97 Industrial Session 5, Charles Levine, Microsoft - http://www.tpc.org/information/sessions/sigmod/index.htm
11. *Intel Desktop Processor Performance*, Intel Corp., webpage - http://www.intel.com/procs/perf/
12. *Benchmarking Business and Consumer System Performance: Benefits of the IntelTM PentiumTM 4 and XeonTM Processors*, Dell – webpage http://www.dell.com/us/en/gen/topics/vectors_2001-pentium4performance.htm
13. Standard Performance Evaluation Corporation (SPEC) – webpage http://www.specbench.org
14. The SPEC CPU 2000 Benchmark suite – webpage http://www.specbench.org/osg/cpu2000/
15. *How we test desktop systems*, ZDnet.com – article on webpage http://www.zdnet.com/products/stories/reviews/0,4161,2711665,00.html
16. *The Benchmark Handbook,* Morgan Kaufmann Publishers, webpage http://www.benchmarkresources.com/handbook/
17. Special Karlsruher MPI (SKaMPI) benchmark – web page http://wwwipd.ira.uka.de/~skampi/
18. LINPACK (Linear Algebra Package) - http://www.netlib.org/linpack/
19. LAPACK (Linear Algebra Package, modern version of Linpack) - http://www.netlib.org/lapack/
20. LINPACK TOP 500 - http://www.top500.org/
21. Clusters @ TOP 500 - http://clusters.top500.org
22. PARKBENCH (PARallel Kernels and BENCHmarks) – web page http://www.netlib.org/parkbench/
23. *SPEComp: A new benchmark suite for measuring parallel computer performance*, Vishal Aslot, Max Domeika, Rudolf Eigenmann, Greg Gaertner, Wesley B. Jones, and Bodo Parady. In Proc. of WOMPAT 2001, Workshop on OpenMP Applications and Tools, Lecture Notes in Computer Science, 2104, pages 1-10, July 2001, http://www.ece.purdue.edu/~eigenman/reports/wompat01spec.pdf
24. SPEComp 2001 – web page http://www.spec.org/hpg/omp2001/
25. SPEC CPU 2000 Benchmark Suite – web page http://www.spec.org/osg/cpu2000/
26. HINT – Hierarchical Integration benchmark, web page http://www.scl.ameslab.gov/Projects/HINT/
27. *An analytical model of the HINT performance metric*, Quinn O. Snell, John L. Gustavson – web page http://www.scl.ameslab.gov/ahint/
28. IBM.com – Introduction to Application Profiling, Article / web page - http://www-1.ibm.com/servers/eserver/zseries/os/linux/ldt/profs.html

29. *Review of Performance Analysis Tools for MPI Parallel Programs*, Shirley Browne, Jack Dongarra, Kevin London (Computer Science Departement, University of Tennessee), web page http://www.cs.utk.edu/~browne/perftools-review/
30. Apple High Performance Computing web page http://www.apple.com/scitech/research/hiperformance/
31. MPI Forum – The official site of the MPI standard - http://www.mpi-forum.org/
32. DEEP (Development Environment for Parallel Programs) – web page http://www.psrv.com/deep.html
33. *A Portable Programming Interface for Performance Evaluation on Modern Processors*, S. Browne, J. J. Dongarra, N. Garner, G. Ho and P. Mucci, *International Journal of High Performance Computing Applications*, 14:3 (Fall 2000), pp. 189-204.
34. *MPICH – A portable implementation of MPI*, Mathematics and Computer Science Division, Argonne National Laboratory, http://www-unix.mcs.anl.gov/mpi/mpich
35. *An API for Runtime Code Patching*, B. Buck and J. K. Hollingsworth, *Journal of High Performance Computing Applications*, 14:4 (2000), pp. 317-329.