

# NIMROD/O: A TOOL FOR AUTOMATIC DESIGN OPTIMISATION USING PARALLEL AND DISTRIBUTED SYSTEMS

DAVID ABRAMSON †, ANDREW LEWIS §, TOM PEACHEY †

† *School of Computer Science and Software Engineering, Monash University,  
CLAYTON, VIC 3168, Australia  
david@csse.monash.edu.au*

§ *High Performance Computing Facility, Griffith University, Nathan, Qld, 4111, Australia*

This paper describes a novel tool called Nimrod/O that allows a user to run an arbitrary computational model as the core of a non-linear optimization process. Nimrod/O allows a user to specify the domain and type of parameters to the model, and also a specification of which output variable is to be minimized or maximized. Accordingly, a user can formulate a question like: “what parameter settings will minimize the model output?”. Nimrod/O currently employs a number of built-in optimization algorithms, namely BFGS, Simplex, Divide and Conquer and Simulated Annealing. Jobs can be executed on a variety of platforms, including distributed clusters and Computational Grid resources. The paper demonstrates the utility of the system with a number of case studies.

## 1 Introduction

Computational science and engineering (CS&E) has become the preferred method of design for a number of disciplines, including aircraft and automotive manufacture, electronics and RF design, environmental engineering, chemical engineering and rational drug design. The driving technologies common to these different areas are the availability of affordable super-computing hardware and robust computational models. The most significant advantage of CS&E is that it allows a designer to experiment with a number of different scenarios before committing to an actual implementation, so the design cycle is reduced and the product (or outcome) quality can be improved.

To date, there are very few generic computational frameworks that support the routine application of CS&E in the product design and development cycle. Thus, in many cases software developers must build an appropriate computational model using conventional programming languages, and then run the model as though it were just another executable program. Whilst this is sufficient, it can be error prone when a large number of alternatives are being considered, because the user must track the path taken as variables are changed and designs are evaluated. Further, if any form of *automatic* optimization is required (like minimizing or maximizing some objective function), then the optimization code has to be integrated into the

computational model itself. The major disadvantage of this approach is that the optimization technology must be *re-engineered* each time a new model is applied.

For a number of years we have been experimenting with a generic tool, called Nimrod, that allows a user to execute a CS&E model over a range of design parameters using high performance computers. Nimrod makes it very easy for a user to specify a range of parameters over which a model must be executed. It then distributes each different execution across a range of computational resources, such as individual workstations, parallel machines and clusters. A version of Nimrod called Nimrod/G supports execution on a computational Grid, in particular, using the Globus toolkit [1][2][3]. A new version called Nimrod/L is currently being built using the Legion environment [4], and a commercial version called enFuzion is also available for workstation clusters [5][6]. A wide range of case studies has been developed [7][8][9][10].

Nimrod can be used to support CS&E based design because it allows a user to explore many different scenarios selecting the ones that minimize or maximize the objective function. However, its biggest disadvantage is that Nimrod explores *all of the search space exhaustively*. Whilst this is feasible for small experiments it rapidly becomes impractical for large, high dimensioned problems.

A number of projects have addressed parts of the design optimization problem over the years. The DAKOTA project at Sandia laboratories [11] investigated the combination of optimization with computational models. DAKOTA is a C++ based tool kit that allows a user to choose different optimization algorithms to guide the search for optimal parameter settings. DAKOTA has been successfully demonstrated on a number of structural mechanics applications. However, DAKOTA does not support rapid prototyping and still requires the construction of new programs for each different type of experiment which is performed. NEOS is a distributed web based optimization engine that allows a user to solve optimization problems using special remote optimization servers. However, NEOS relies on the objective function being specified in an algebraic form and does not support objective functions implemented by CS&E models. Likewise, NetSolve [12] provides a web based engine for solving linear algebra problems, but does not explicitly address optimization using CS&E objective functions. SciRun [13] is an interactive tool which allows a user to build a CS&E model very rapidly using a graphical programming interface. However, it does not support optimization.

In this paper we describe Nimrod/O, a tool which augments the distributed parameter sweep functionality of Nimrod with automatic optimization. We believe that Nimrod/O is the first system to combine rapid application development, parallel and distributed supercomputing and optimization in the one tool. Nimrod/O is modular, and incorporates a number of standard non-linear optimization methods. It uses a declarative style developed for Nimrod, which allows a user to specify the types of parameters and commands necessary for running the underlying CS&E model. The user can then specify which optimization algorithm (or combination of

algorithms) to apply to the problem. Thus, if a user already has a robust CS&E model, it can be converted into a parallel optimizing design tool very quickly.

The paper describes the Nimrod system as background, and then shows the architecture of Nimrod/O. It then discusses the types of optimization algorithms that are supported, followed by a set of case studies. The case studies implement problems in environmental engineering, mechanical engineering and RF electronic design.

## 2 Nimrod and Nimrod/G

Nimrod is a tool that manages the execution of parametric studies across distributed computers. It takes responsibility for the overall management of an experiment, as well as the low-level issues of distributing files to remote systems, performing the remote computation and gathering the results. EnFuzion [5] is a commercial version of the research system Nimrod.

When a user describes an experiment to Nimrod, they develop a *declarative* “plan” file which describes the parameters, their default values, and the commands necessary for performing the work. The system then uses this information to transport the necessary files and schedule the work on the first available machine.

When the user invokes Nimrod on a workstation, the machine becomes known as the “root” machine because it controls the experiment. When the dispatcher executes code on remote platforms, each of these is known as a computational “node”. Thus a given experiment can be conducted with one root and multiple nodes, each of a different architecture if required.

Nimrod/G is a “Grid aware” version of Nimrod built on the Globus toolkit [1]. It exploits an understanding of its problem domain as well as the nature of the computational Grid to provide a high level interface to the user. Specifically, it provides transparent access to the computational resources, and implements user level scheduling. Nimrod/G is described in more detail in [26] and [27].

Nimrod/O makes use of Nimrod to perform its remote execution, as discussed in the next section. This isolates the optimisation algorithms from the functions required to access high performance computers of varying architecture.

## 3 Nimrod/O architecture

The architecture of Nimrod/O is summarized in Figure 1. Nimrod/O accepts a declarative plan file as previously developed for Nimrod. This file indicates the name and domains of the various parameters and provides descriptions of how to run the CS&E model.

Nimrod/O is built to allow the integration of a number of different optimization engines. These are linked together into the one executable, and are chosen by command line arguments. The CS&E model is actually executed on a distributed

supercomputer using Nimrod, so Nimrod/O is not concerned with the details of how jobs are run. Since Nimrod, Nimrod/G and enFuzion all share a common API, it is possible to substitute different back end engines depending on the type of supercomputing facilities that are available. For example, the Nimrod/G backend allows jobs to be run on a globally distributed computation grid like GUSTO [2][16], whereas the enFuzion backend supports department level clusters efficiently.

The system supports concurrency in a number of places. First, if the algorithm itself is parallel (like P-BFGS discussed in the next section) then the system supports concurrent execution within the algorithm. For example, in P-BFGS a number of points are evaluated concurrently, and Nimrod/O does this by requesting Nimrod to run the model multiple times with different parameters. Second, the system allows different searches (possible using different optimization algorithms) to be performed in parallel using different starting locations. Because many of the algorithms we have considered depend on the choice of an initial point, it is important to run the optimization more than once, starting it at different places. Nimrod/O supports this by multi-threading its internal control and using the Nimrod backend to run multiple jobs. When we evaluate the performance of the system later in the paper, we use this feature to demonstrate the variability in final objective function values that are obtained.

Currently, the value of the objective function is returned from the application via a special file. However, in future we expect to add other methods of extracting the function value such as using environment variables to convey the information. A special cache server (called NimCache) is placed between Nimrod/O and Nimrod. This process caches all function requests, and if it detects a request for a point that has already been calculated, it returns the previous result.

## 4 Optimization algorithms

The current implementation of Nimrod/O supports 4 non-linear search heuristics, namely P-BFGS, Simplex, Divide and Conquer and Simulated Annealing.

### 4.1 P-BFGS

In general, gradient descent methods like the BFGS method, use the derivative of the cost function, as well as its value, to select a search direction, essentially reducing the multivariate optimization problem to uni-variate minimization along a search vector; a line search. They thus consist of two main operations that are executed repeatedly, namely gradient calculations and line searching. The P-BFGS algorithm is based on the quasi-Newton BFGS (Broyden, Fletcher, Goldfarb, Shanno) method, widely regarded as one of the most efficient and robust gradient descent methods for use with continuous functions [17][18][19].

The BFGS method maintains an approximation to the Hessian,  $H$ , of  $f$ , where:  
New approximations to the solution vector,  $x$ , are derived by:

1. Compute a search direction,  $d = -H^{-1}\nabla f(x)$
2. Find a new  $x$ , using a line search, i.e.  $x_+ = x + \mathbf{I}d$
3. Update  $H$  using the current approximation to  $H$ ,  $x$  and  $x_+$   $H_{ij} = \partial^2 f / \partial x_i \partial x_j$   
The P-BFGS algorithm performs step 1 by computing the finite difference approximations for the gradient in parallel. It also uses a modified parallel line search algorithm for step 2. There is insufficient space in this paper to discuss the algorithm in detail, however, more information can be found in [20].

#### 4.2 Simplex

The Simplex algorithm used is based on the method of Nelder and Mead [21]. In common with other such methods, at each iteration it examines a simplex of points and then changes the simplex in response. A simplex is a geometrical figure consisting, in  $N$  dimensions, of  $N+1$  points, or vertices, and all their interconnecting line segments. For example, in two dimensions, a simplex is a triangle.

The algorithm is started with  $N+1$  points, defining a simplex, and their associated cost function evaluations. It then takes a series of steps, at each step moving the point of the simplex where the cost function is highest through the opposite face of the simplex to a lower point.

If the cost function value at a reflection trial point is found to be particularly advantageous, the simplex is expanded in the given direction; if it is less advantageous, the simplex is contracted. Should no progress be possible, the simplex is simultaneously contracted in all dimensions. These steps are repeated until the cost function values at all points fall within a desired tolerance of each other.

The basic method has been modified in the current study in two ways.

- Simple bounds have been applied to all trial points, so that the search is contained within a given space.
- The evaluation of trial points has been parallelised. At each step, in a high proportion of cases at least two function evaluations will be made: the initial reflection, and either an expansion of it, or a contraction. The modified algorithm evaluates all three of these possibilities concurrently, yielding a speedup observed to approach a factor of two, for the use of three processors.

### 4.3 Divide and Conquer

The Divide and Conquer (DC) method is a simple heuristic which performs a repeated subdivision of the search space. The domain for each of the  $N$  parameters is subdivided into three parts and the objective function is computed at the  $(3+1)^N$  grid points that result. The grid point which produces the minimum objective and its neighbours give the domain for the next iteration. There are two variants of this procedure, depending on a parameter called “drift”. If “drift” is allowed then the next search is centred on the end value, so the next search domain will be partly outside the previous space. Alternately with drift disabled the new domain will be a proper subset of the previous space. The search terminates when the absolute value of the difference in the objective values at the best and worst grid points, as a proportion of the mean of those values, is less than the tolerance.

### 4.4 Simulated Annealing

Simulated annealing (SA) is a *Monte Carlo* method, based on the analogy between the aggregation of many particles in a physical system as it is cooled and the optimization of large combinatorial problem. In the physical system, annealing is the process of heating a solid until it melts, followed by a cooling operation until the substance crystallizes. Fast cooling leads to a process called *quenching* in which the crystal that is formed has a metastable amorphous structure (or imperfect lattice), with a very large internal energy. Slow cooling allows the system to reach its equilibrium (lowest energy level) at each decremental temperature. Hence it is possible to reach the zero energy level or ground state. In the optimisation perspective, search space can be viewed as a physical system in which elements or variables correspond to particles. The atomic bonding force is equivalent to the set of constraints imposed on the problem. The energy of the system is modelled as the penalty cost, so that when the system reaches the ground state, the process arrives at the global optimum configuration. Simulated Annealing has been applied to a wide range of problems from continuous systems to combinatorial optimization. In this work, we have experimented with both a public domain package [22] and one that we have written ourselves.

## 5 Case Studies

In this section we examine the performance of Nimrod/O on 5 real world case studies, shown in Table 1. Table 2 shows which algorithms were applied to the case studies, since we were not able to apply all algorithms to all problems. The exact details of the case studies are not discussed here. All of them required the optimisation of a particular objective function value. The number of parameters is shown in Table 1, and varies from 1 to 7.

Problem	Number of Dimensions	Description
Smog	2	Photo Chemical Pollution Model [23]
S-Plate	1	Static Strength of a Biaxially Loaded Plate – Simple cutout
C-Plate	3	Static Strength in a Biaxially Loaded Plate – Complex cutout
Ceramic	3	Radio frequency properties of a ceramic bead [24]
Antenna	7	Multi frequency antenna

Table 1 – Problem definitions

Problem	Algorithm				
	Exhaust	P-BFGS	Simplex	DC	SA
Smog	X	X			
S-Plate	X	X			
C-Plate	X	X	X	X	
Ceramic	X	X	X		X
Antenna			X		

Table 2 – Problem-Algorithm assignment

### 5.1 Results

Table 3 shows the results obtained when an exhaustive search is performed on 4 of the case studies. In these cases, the domain of the parameters is either integer, or converted into a number of discrete values. We were only able to run an exhaustive search on 4 of the problems, because the search space for the Antenna problem is too large to enumerate, even for a coarse decomposition.

The “# Func. Evals” column indicates how many model executions were required. The “Optimal Cost” column shows the value of the best objective function value. The “Time for 1 Func. Eval” column shows the average time to run the model once. The “Number of CPUs” column shows how many processors were available on the cluster we used. The “Wall Clock Time” column shows how long it took to run the entire experiment with 64 CPUs.

From these results, it can be seen that for many of the problems, an exhaustive search is expensive even when a reasonably sized cluster is available. For example, the Smog case study required 60 days of continuous use of a 64 way cluster!

Table 4 shows the results of applying the various algorithms to the problems. The “#Starts” column indicates how many independent runs were performed for the problem-algorithm pair. The “Best Cost”, “Worst Cost” and “Mean Cost” columns indicate the best, worst and mean costs across the independent starts. If only one start was performed then the worst and mean are omitted. If a ‘\*’ is present in the

“Opt” column, then the best cost was optimal. The “# Iterations” indicates how many iterations of the algorithm were required to achieve the best reported solution. For Simplex, DC and SA this involves sending out a number of function values for concurrent evaluation before computing a new position. In P-BFGS, each iteration involves both a gradient computation and a line search. The “#Func Evals” column reports the total number of times the model was run in order to find the best solution. The “Wall Clock Time” column indicates how long it took to find the best solution. The last column shows the ratio of the wall clock time for each problem with the time for an exhaustive search (from Table 3). The “Ave # of Processors Required” indicates how many processors were needed on average to achieve the best result in the time shown. This is calculated on the assumption that all independent starts are run concurrently and that each algorithm requires some number of processors to operate in the time that is reported.

One of the most striking features of the data in Table 4 is the variability of the results across different problems and search algorithms. P-BFGS performs well on all problems for which the optimal cost was known, achieving optimal in all cases. However, Simplex does not manage to achieve optimal on any of the problems attempted, although the comparison is a little unfair since the running times are quite different from P-BFGS. In addition, the cost achieved on C-Plate using Simplex, whilst not optimal, is also quite acceptable. DC performs well on the one problem on which it was attempted, and our particular version of SA does not perform well on Ceramic. However, this SA code is still under development and further enhancements may substantially improve its performance. Further, we obtained good results using the ASA package [22] on the Ceramic problem, but did not have all of the statistics to allow a fair comparison at this stage. Accordingly, we have not reported the ASA results in Table 4. Our results do indicate that one of the algorithms managed to achieve an optimal cost for each of the problems, validating the idea of making multiple algorithms available from the one optimization package.

The timing performance of the various algorithms indicates that whilst some of the searches might still take a large amount of time, in most cases the automatic optimization was faster than an enumerative search. For the Smog problem, P-BFGS achieves optimal 17.4 times faster than a 64 processor cluster performing exhaustive search. Moreover, it does this with only 5.1 processors on average. Similarly, P-BFGS achieves optimal on Ceramic 11.2 times faster than exhaustive search, using only 27.1 processors on average. At the other extreme, P-BFGS is about the same speed as an exhaustive search on C-Plate using 48 processors. However, DC solves this problem 5 times faster than exhaustive search with 53.8 processors. The variation in the number of processors used is a reflection of the parallelism available within the optimisation algorithm itself, and the number of parallel starts that were performed. For example, running Simplex on the Antenna problem did not use many processors because we only performed one start, and the algorithm has little internal parallelism. The DC algorithm on the C-Plate problem,

on the other hand, achieves high parallelism using only one start because the algorithm itself is naturally parallel.

Problem	# Func Evals	Optimal Cost	Time for 1 Func. Eval. (Secs)	Number of CPUs	Wall Clock Time (Hrs:Mins:Secs)
Smog	10000	68	31680	64	1375: 0: 0
S-Plate	201	15.0	204	64	0: 10: 40
C-Plate	2541	14.7	326	64	3: 35: 35
Ceramic	12896	-39.8	3000	64	167: 55: 0
Antenna	-	-	7800	-	Not Possible

Table 3 – Exhaustive Search of case studies

It may be noted, since the exhaustive search data was available for 4 of the problems, it was not necessary for search algorithms to actually run the model again in order to provide the timing results in Table 4. In particular, for the Ceramic problem, we allowed the algorithms to perform a table lookup using the data from the exhaustive search, and then we predicted the total run time using the time required to execute the model once. Thus, when SA was performed on the Ceramic problem, we did not actually have to wait 370 hours for the result.

## 6 Conclusions

In this paper we have shown the design and implementation of a novel tool called Nimrod/O, which allows a user to produce an optimizing decision support system using existing computational models. The tool provides a range of optimization algorithms, and using a simple declarative specification technique, these algorithms can be applied to an arbitrary computational model even if source code for the model is not available. A range of backend tools means that we can target platforms ranging from department level clusters to a global computational grid.

The paper reports work which is in progress, so we do not regard Nimrod/O as a completed system. We plan to perform more work on the core optimization algorithms to improve their performance. We also plan to improve the engineering of Nimrod/O to allow arbitrary external optimization libraries to be integrated easily. In the coming months we plan to add a restart and recovery feature to allow long running experiments to be terminated and restarted at a later time. One of the most significant algorithmic enhancements is to improve the parallelism of the optimization methods themselves, with the hope of delivering shorter run times overall.

## Acknowledgements

The Nimrod/O project has been supported by the Australian Research Council, as part of an ARC Collaborative Grant. We wish to acknowledge our partners in this grant, namely the Environment Protection Authorities of Queensland, Victoria and Western Australia. We wish to thank Turbo Linux Inc. for access to the enFuzion software, and the Distributed Systems Technology CRC, for access to Nimrod/G. We also acknowledge our research partners in the Globus group, in particular Ian Foster from Argonne National Laboratories.

The case studies discussed in this paper involved the work of a number of colleagues. We wish to acknowledge

- Dr Martin Cope, Dr Steven Spencer and Dr Maciej Skierski for their assistance with the Smog study;
- Professor Rhys Jones, Dr John Sawyer and Mr Paul Chaperon for their assistance with the Plate study;
- Mr Seppo Saario and Dr Jun Lu for their assistance with the Ceramic and Antenna studies.

## References

1. Foster, I., Kesselman, C., *Globus: A Metacomputing Infrastructure Toolkit*, International Journal of Supercomputer Applications, 11(2):115-128, 1997.
2. <http://www.globus.org/>
3. Foster, I. and Kesselman, C. *Globus: A Toolkit-Based Grid Architecture*. In *The Grid: Blueprint for a Future Computing Infrastructure*, pages 259-278. Morgan Kaufmann Publishers, 1999.
4. Grimshaw, A. and Wulf, W., et al, "The Legion vision of a woroldwide virtual computer", *Communication of the ACM*, 40(1):39-45, 1997.
5. Turbo Linux Inc, [www.turbolinux.com](http://www.turbolinux.com)
6. Abramson, D. "Parametric Modelling: Killer Apps for Linux Clusters", *Linux Journal*, May 2000.
7. <http://hathor.csse.monash.edu.au/RJK/>
8. Abramson, D., Power, K., and Sobic, R., "Simulating Computer Networks using Clusters of PCs", HPC-TelePar'99 at the 1999 Advanced Simulation Technologies Conference (ASTC'99), April 11-15, San Diego, California, USA
9. D. Abramson, I. Foster, J. Giddy, A. Lewis, R. Sobic, R. Sutherst, N. White, "The Nimrod Computational Workbench: A Case Study in Desktop Metacomputing", *Australian Computer Science Conference (ACSC 97)*, Macquarie University, Sydney, Feb 1997, pp 17 - 26.

10. Lewis, A., Abramson D., Susic R., Giddy J., "Tool-based Parameterisation : An Application Perspective", Computational Techniques and Applications Conference, Melbourne, July 1995.
11. Eldred, M., Outka, D., Fulcher, C. and Bohnhoff, W. "Optimization of complex mechanics simulations with object-oriented software design.", Proceedings of the 36th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, pp 2406 - 2415, New Orleans, LA, April 1995.
12. Casanova, H. and Dongarra, J. "NetSolve: A Network Server for Solving Computational Science Problems", The International Journal of Supercomputing Applications and High Performance Computing, Vol 11, Number 3, pp 212-223, 1997.
13. Parker, S. and Johnson, C. "SCIRun: A Scientific Programming Environment for Computational Steering", Proceedings of IEEE Supercomputing 1995, San Diego, December 95.
14. Abramson D., Susic R., Giddy J. and Hall B., "Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations", The 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995.
15. Foster, I., and Kesselman, C. (editors), *Computational Grids. The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.
16. <http://www-fp.globus.org/overview/testbeds.html>
17. Gill, P.E., Murray, W. and Wright. M, "Practical Optimization". Academic Press, London, 1981.
18. Fletcher. R. "Practical Methods of Optimization." 2nd ed., Wiley, New York, 1987.
19. Press. W. H. , "Numerical Recipes: the Art of Scientific Computing". Cambridge University Press, 1986.
20. Lewis, A and Abramson, D. "A Parallel BFGS algorithm and its application to Automatic Design Optimization", Griffith University High Performance Computing Facility Technical Report, 2000.
21. Nelder, J.A., and Mead, R., "A simplex method for function minimization", *Comput. J.*, 7, pp. 308-313, 1965.
22. Ingber, L, "Very fast simulated re-annealing", *Journal of Mathematical Computer Modeling*, Vol 12, #8, pp 967-973, 1989. See <http://www.ingber.com>.
23. McRae G.J., Russell A.G. and Harley R.A., 1992. CIT photochemical airshed model -users manuals, Carnegie Mellon University, Pittsburgh, PA and California Institute of Technology, Pasadena, CA.
24. Lewis, A., Saario, S., Abramson, D. and Lu, J., "An Application of Optimisation for Passive RF Component Design", to appear, Conference on Electromagnetic Field Computation, Milwaukee, June 4-7<sup>th</sup>, 2000.

25. Lewis, A, Abramson, D and Simpson, R., "Parallel non-linear optimization : Towards the design of a decision support system for air quality management", IEEE Supercomputing 97, San Jose, 1997.
26. Abramson, D., Giddy, J. and Kotler, L. "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?", International Parallel and Distributed Processing Symposium (IPDPS), pp 520-528, Cancun, Mexico, May 2000.
27. Buyya, R., Abramson, D. and Giddy, J. "Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid", HPC Asia 2000, May 14-17, 2000, pp 283 – 289, Beijing, China.

Problem	Alg'm	Starts	Best Cost	Opt	Worst Cost	Mean Cost	# Iter'ns for Best	# Func Evals for Best	Wall Clock for Best (Hrs:Min:s:Secs)	W'll Clk Exh' st W'll Clk Best	Ave # of Procs req'd
Smog	P-BFGS		68	*	-	-	2	46	79: 12: 0	17.4	5.1
S-Plate	P-BFGS		15	*	15	15	2	46	0: 34: 0	0.3	36.8
C-Plate	P-BFGS		14.7	*	15.7	15.1	7	240	3: 37: 11	1.0	48.0
C-Plate	Simplex		15		16.7	15.4	7	21	0: 38: 0	5.7	24.0
C-Plate	DC		14.7	*	-	-	8	430	0: 43: 26	5.0	53.8
Ceramic	P-BFGS		-39.8	*	26.3	-2.1	5	61	15: 0: 0	11.2	27.1
Ceramic	Simplex		-9.4		37.4	16.9	10	24	9: 10: 0	18.3	17.5
Ceramic	SA		-18.5		-14.2	-17.5	22	472	370: 0: 0	0.5	8.5
Antenna	Simplex		35.7	-	-	-	23	84	49: 50: 0	-	3.7

Table 4 – Results of Case Studies with Nimrod/O

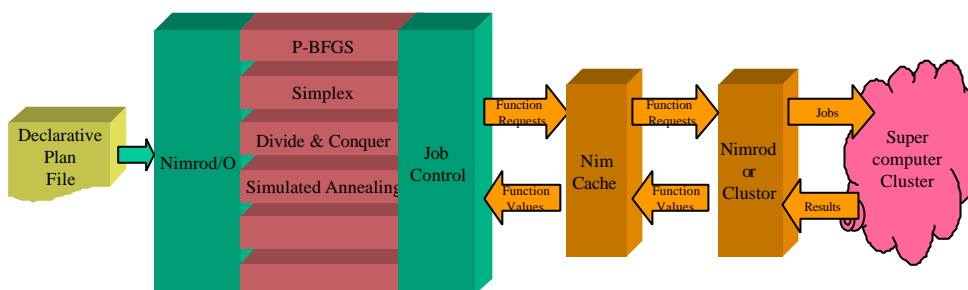


Figure 1 - Architecture of Nimrod/O

ica3pp World Scientific Form submitted to World Scientific : 24/08/2000 : 4:39 PM  
 12/12  
 ica3pp World Scientific Forms submitted to World Scientific :  
 24/08/2000 : 4:39 PM  
 12/12