# Task-parallel global optimization with application to protein folding

C. Voglis, P. E. Hadjidoukas,
V. V. Dimakopoulos, I. E. Lagaris
*Department of Computer Science,*
*University of Ioannina, Ioannina, Greece,*
{*voglis,phadjido,dimako,lagaris*}*@cs.uoi.gr*

D. G. Papageorgiou
*Department of Materials Science and Engineering,*
*University of Ioannina, Ioannina, Greece,*
*dpapageo@cc.uoi.gr*

## ABSTRACT

*This paper presents a software framework for high performance numerical global optimization. At the core, a runtime library implements a programming environment for irregular and adaptive task-based parallelism. Building on this, we extract and exploit the multilevel parallelism of a global optimization application that is based on numerical differentiation and Newton-based local optimizations. Our framework is used in the efficient parallelization of a real application case that concerns the protein folding problem. The experimental evaluation presents performance results of our software system on a multicore cluster.*

**KEYWORDS:** task parallelism, cluster programming, numerical differentiation, global optimization, protein folding.

## 1. INTRODUCTION

Many scientific and engineering problems can be modeled and solved as numerical optimization tasks. This is possible by developing appropriate objective functions that accurately model the problem at hand. Usually, the *global optimizers*[1] of the objective functions correspond to optimal solutions of the original problem. There are numerous applications that involve the solution of global optimization problems. Among others, we can mention signal processing, telecommunications, finance and operations research, networks and transportation, engineering design and control, molecular biology, hardware and software design, as well as biomedical engineering. Algorithms that tackle global optimization problems usually have high computational demands due to the substantial execution time of the objective function. Also, in most of the cases, the multitude of local optima require thorough investigation of the search space resulting more computational time. Exploitation of parallelism at several levels such as function evaluations, numerical computations and the optimization algorithms themselves can drastically reduce the time required to find a solution.

The *Multistart* method is a standard and widely used stochastic scheme for dealing with global optimization problems. According to this method, a *local optimization* procedure is applied to a number of randomly selected points. Although simple in principle, Multistart lies in the heart of many sophisticated global optimization algorithms. Due to its implementation simplicity, it serves as a great candidate for studying parallelization issues. Local optimization methods are sequential procedures that, beginning from a starting point generate a sequence of iterates that terminates when a solution point is approximated with a prescribed accuracy. These methods provide no guaranty for locating the global optimum value but they are extremely efficient in finding a local optimizer. A large category of local optimization algorithms originate from the well known *Newton method*, a general and powerful method for multidimensional non-linear optimization that makes use of first and second derivatives of the objective function. The most successful local optimization algorithms can be thought as approximations of the Newton method. The need for second order derivative information, introduces further computational complexity because derivative estimation is a time consuming task. A common way to calculate derivatives is via finite differentiation algorithms where derivatives are approximated by function values at suitably chosen points. Clearly, the Multistart algorithm with Newton method as local optimizer is a valuable tool for solving global optimization problems that can benefit from a parallel implementation.

[1] Minimizers or maximizers

Task-based parallelism, as expressed by the master-worker programming model, can be an effective approach for a cluster-aware implementation of global optimization methods such as Multistart. Function evaluations are mapped to tasks and assigned to the workers. The dynamic load balancing of the model further enhances its suitability. A naive implementation of the model, however, cannot meet all the requirements that Multistart imposes. First, the large expected number of spawned tasks (typically on the order of $10^6$) affect the scalability as the single master becomes a bottleneck. Secondly, the exploitation of nested parallelism requires advanced runtime techniques, able to provide efficient management of processing elements. Additionally, it is important to have a hardware-independent solution that transparently uses multi-threading to fully exploit the shared memory of multicore systems.

In this paper, we present a high-performance numerical optimization framework for clusters of multicore systems that deals with all the above limitation issues. At the core, a novel runtime library, TORC, provides support for irregular and adaptive master-worker parallelism on multi-core SMPs and clusters of such machines. Building on TORC, we design a standalone numerical differentiation software package (PNDL) that provides routines for gradient and Hessian computations. Using both TORC and PNDL, we present the implementation of a Newton-based Multistart method that performs multiple local optimizations and gradient/Hessian calculations in parallel. By integrating a molecular modeling software package with our system, we are able to apply our framework to a real application case that deals with the protein folding problem, that is the problem of determining the three dimensional structure of a protein. It is a fundamental problem in biophysical sciences with application in drug design and in decoding genetic information. The experimental evaluation on a dedicated multicore cluster demonstrates the efficiency of our system.

The rest of this paper is organized as follows: Section 2 gives a brief introduction to the global optimization problem and the Multistart method. Section 3 discusses the parallelism issues of Multistart while Section 4 presents the software infrastructure. Section 5 presents the protein folding problem. Experimental evaluation and related work are reported in Sections 6 and 7 respectively. Finally, we conclude in Section 8.

## 2. GLOBAL OPTIMIZATION

The task of numerical optimization is to locate (approximate) the best minimizer of a generally multidimensional objective function. The mathematical formulation is

$$\min_{x \in \mathbb{R}^n} f(x) \tag{1}$$

where $x \in \mathbb{R}^n$ is a real vector and $f : \mathbb{R}^n \to \mathbb{R}$ the objective function. There exist a plethora of applications in physics, chemistry, engineering, and economics that can be formulated as optimization problems.

One of the simplest algorithm for global optimization, is the *Multistart* procedure. According to this method, a local optimization procedure is executed independently for each point in a sample generated from a uniform distribution over the search space. The strong theoretical properties of Multistart render it a widely used method and led to computationally successful adaptations. The most important variants, that share the same principles with Multistart, are the *clustering methods*[1, 2]. The main computational cost of Multistart is the application of the local optimization algorithm[2]. A brief sketch of Multistart is presented in Algorithm 1 below. Notice that local optimization steps (step 4) are independent and can be performed in parallel. We must also mention that all aforementioned variations of Multistart can benefit from a parallel implementation of independent optimization procedures.

**Algorithm 1**: Multistart($f$, $S$, $x^*$)

**Input** : Objective function, $f : S \subset \mathbb{R}^n \to \mathbb{R}$; search region: $S \subseteq \mathbb{R}^n$
**Output**: Approximation of the global minimizer: $x^*$

1 Set $Y = \emptyset$
2 **for** $i \leftarrow 1, 2, \ldots$ **do**
`// Sample a starting point`
3 Sample a random point $x^{(i)}$ uniformly in $S$
`// Local optimization`
4 Apply local optimization procedure from $x^{(i)}$. Let $y^{(i)}$ be the resulting local optimizer.
`// Check if the local minimum is already found`
5 **if** $y^{(i)} \notin Y$ **then**
6 Set $Y = Y \cup y^{(i)}$
7 **end**
`// Stopping rule`
8 Check an appropriate stopping rule
9 **end**
10 Set $x^*$ be the element of $Y$ with minimum function value

A local optimization algorithm is a sequential procedure that, beginning from a starting point $x_0 \in S$, generates a sequence of iterates $\{x_k\}_{k=0}^{\infty}$ that terminates when the solution point is approximated with a prescribed accuracy. In deciding how to move from one iterate $x_k$ to the next the

[2]Uniform sampling is considered a simple task.

optimization algorithm uses information about the function at $x_k$ (function value, first or second order derivatives). A general class of optimization algorithms use second order derivative information of the objective function and use it to build and minimize a quadratic model around the current iteration. The main representative of this class is the *Newton* method. At each iterate, the Newton method makes use of first and second order derivative information to proceed to the next point. This can be achieved using a *line search* algorithm which searches along a descent direction $p_k \in \mathbb{R}^n$ for an iterate with lower function value. The distance to move along $p_k$ can be found by solving the following one-dimensional minimization problem that is to find a step length $\alpha$ that minimizes $f(x_k + \alpha p_k)$. The main computational cost of a single Newton iteration is determined by the objective function and the derivatives calculation that are used to compute the search direction. The search direction is Newton method is calculated by solving a linear system

$$B_k p_k = -\nabla f(x_k)$$

where $B_k$ is a positive definite modification of the matrix of second order derivatives ($\nabla^2 f(x_k)$). A description of Newton's method is presented in Algorithm 2.

**Algorithm 2**: Newton($f$, $x_0$, $x^*$)

**Input** : Objective function, $f : S \subset \mathbb{R}^n \to \mathbb{R}$; starting point: $x_0$

**Output**: Approximation of the minimizer: $x^*$

1 **for** $k \leftarrow 1, 2, \ldots$ **do**

`// Derivative evaluation`

2 Calculate $g_k = \nabla f(x_k)$ and $G_k = \nabla^2 f(x_k)$

`// Modification`

3 Factorize the matrix $B_k$, where $B_k = G_k$ if $G_k$ is positive definite; otherwise, $B_k = G_k + E_k$

`// Linear system solution`

4 Solve $B_k p_k = -g_k$

`// Line search`

5 Set $x_{k+1} = x_k + \alpha_k p_k$, where $\alpha_k$ satisfies sufficient descent conditions

6 Stop if convergence criterion is met

7 **end**

8 Set $x^* = x_{k+1}$

In many cases derivatives cannot be expressed analytically because the underlying functions are represented by large and complicated computer codes. In these cases finite differencing is an approach for calculating the first and second order derivatives of a $n-$dimensional objective function at a point $x$ by examining the objective function behavior on small finite perturbations around $x$. The number of function evaluations depends on the order of the derivative (first or second) and on the requested accuracy (the larger accuracy the more function evaluations). For the gradient vector at least $n + 1$ function evaluations are required and for the Hessian at least $n(n + 1)/2$. Two of the most popular formulas for approximating gradient and Hessian, using central differences are summarized below:

$$\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x + \epsilon e_i) - f(x - \epsilon e_i)}{2\epsilon} \quad (2)$$

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \approx \frac{f(x + \epsilon e_i + \epsilon e_j) - f(x - \epsilon e_i + \epsilon e_j)}{4\epsilon^2} - \frac{f(x + \epsilon e_i - \epsilon e_j) + f(x - \epsilon e_i - \epsilon e_j)}{4\epsilon^2}$$

where $e_i$ is the $i-$th unit vector and $\epsilon$ a small positive scalar. Finite differencing is a perfect candidate for parallel execution. All function evaluations in Eq.(2), $f(x + \epsilon e_i)$ and $f(x + \epsilon e_i + \epsilon e_j)$, can be performed independently and in parallel.

## 3. PARALLELISM ISSUES

Multistart includes several levels of parallelism that can be exploited in order to accelerate the method. According to Multistart, the application initially spawns first-level ($L1$) tasks (Alg. 1, steps 2-9). These tasks perform the Newton *local search* method to multiple independent initial points ($x_i$) and execute iterations until the convergence criterion is met (Alg. 2, steps 1-7). In each iteration, the tasks first proceed with the derivative calculation, spawning two second-level ($L2$) tasks that compute the gradient and Hessian respectively (Alg. 2, step 2). The gradient computation includes a number of function evaluation ($L3$) tasks. The Hessian computation, however, exploits an additional level of parallelism by assigning the numerical calculation of each partial derivative to a ($L3$) task that can spawn a number of function evaluation ($L4$) tasks, depending on the desired accuracy and the bounds. Local search continues with a sequential task that performs the required matrix modification and the solution of the linear system (Alg. 2, steps 3-4). The iterative line search method follows, exploiting each time a single level of parallelism for the gradient computation (Alg. 2, step 5). For a large number of initial points, a gradual execution of Multistart is performed by applying the Newton method to bunches of points. In such case and given that the stopping rule of Multistart is not satisfied, the method is repeated until the desired number of points has been processed.

Multistart is a highly irregular parallel application: first, the local search method is applied concurrently to multiple points, the number of which may not be exactly divided

by the number of available processors. Secondly, the execution time of local search exhibits significant variation as the number of iterations required for convergence depends on the randomly selected initial point. Similarly, the line search method is performed for an unknown number of iterations. Irregularity is found even at the innermost level of parallelism (Hessian calculation), as the number of function evaluations for the derivative computation at a specific point also depends on the imposed bounds on the variables. According to the above, the execution times for finding a minimum for each initial point are neither balanced nor known beforehand. Derivative estimation via finite differencing is computationally expensive for several applications where the time for a single function call is substantial. Therefore, the highly irregular nested parallelism of Multistart must be exploited at all possible levels, without making any assumption about the number of available processors.

## 4. SOFTWARE INFRASTRUCTURE

The architecture of our parallel optimization framework includes the following components:

- Global optimization application: It implements the parallel Multistart method taking advantage of TORC in order to spawn tasks that execute the Newton method. In addition, it issues concurrent calls to PNDL, for parallel gradient and Hessian computations.
- PNDL: The Parallel Numerical Differentiation Library (PNDL) is a standalone software module that exports subroutines for calculating gradients, Hessians and Jacobians by finite differencing, supports multivariate functions, respects variable bounds and offers several prescribed accuracy levels. The parallel implementation of PNDL for multicore clusters has been based on the tasking model that TORC provides. In PNDL, task parallelism is expressed with independent function evaluations assigned by a master to the workers. For each function evaluation, a task is created, with main input argument a vector $x$ and result the computed function value $f(x)$.
- TORC: It is a novel runtime environment for programming and executing irregular and adaptive master-worker applications on multi-core SMPs and clusters of such machines. TORC supports a task-style programming taking advantage of and extending the MPI programming model. TORC's architecture is based on a two-level threading model, uses exclusively POSIX and MPI calls for portability and performance, and integrates seamlessly hardware shared-memory and message passing.

The core PNDL routines for gradient and Hessian computations are the following:

```
subroutine pndlg(f, x, n, iord, grad)
subroutine pndlhf(f, x, n, iord, hes)
subroutine pndlhg(g, x, n, iord, hes)
```

where $f$ is the function to be differentiated, $x$ the vector containing the point of calculation, $n$ the dimensionality of the function, $iord$ the requested order of accuracy, $grad$ and $hes$ the resulting gradient vector and Hessian matrix. The `pndlhg` routine can be used if the analytical gradient routine $g$ of the objective function is available. A preliminary report on the intended PNDL API is available in [3]. Additional information on TORC and the implementation of PNDL on top of it can be found in [4].

## 5. PROTEIN FOLDING

The protein folding problem is defined in [5] as the problem of determining the three dimensional structure of a protein, called its *tertiary structure*, just from the sequence of amino acids that it is composed of (its *primary structure*). Under the assumption that in the native state the potential energy of a protein is globally minimized, the protein folding problem can be regarded as equivalent to solving the problem

$$\min_{x \in \mathbf{R}^{3n}} E(x) \tag{3}$$

where $E(x)$ is the value of a potential energy function for a $n$ atom protein described by a $3n$ dimensional coordinate vector. This optimization problem has a large number of variables, depending on the size of the amino acid sequence, and many local minimizers, a number exponential to the number of atoms. It is believed that in the native state of a protein, the potential energy function of the protein is in its global energy minimum. In Fig. 1 we present an illustration of a protein in a unfolded and in a folded (minimum energy) state.

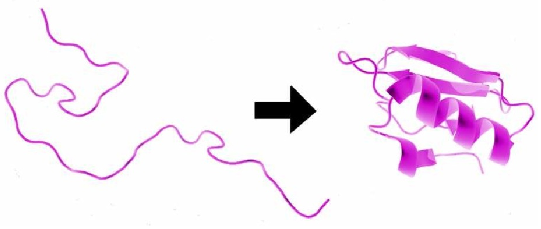

**Figure 1. Example of how protein folds reaching minimum energy**

There are two different but equivalent coordinate systems used to describe the conformation of a protein: internal and

Cartesian coordinates. In Cartesian coordinates each atom is represented by its $x$, $y$ and $z$ coordinates. In the internal coordinates system, which is more closely related to the structure of a protein, we use the *bond length*, *bond angle* and *dihedral angle* notions to specify the coordinates of the atom. The bond length is defined as the Euclidean distance between to consecutive atoms. The bond angle is the angle between three consecutive atoms. Finally, for every sequence of four consecutive atoms the dihedral angle is the angle defined by the plain of the first three atoms and the last three atoms of the sequence. To describe a conformation of a protein we need $3n$ numbers in Cartesian and $3n - 6$ in internal coordinates.

Protein folding is considered one of the most challenging global optimization problems due to the vast number of local optimal conformations and the large objective function computation time. Until recently, only small protein structures were examined thoroughly and their global minimum conformations were revealed. It has been showed [6] that in the native state of the protein there isn't a lot of variation in the values of the bond angles and the bond lengths. In addition, the bond between two amino acids is very rigid and may be kept fixed. It is easier to keep bond lengths and bond angles fixed in internal coordinates than in Cartesian coordinates. So for proteins, the use of internal coordinates is computationally more efficient than the Cartesian coordinate system. Fixing the bond lengths and bond angles to the common values, leads to three free variables (dihedral angles) for an amino acid. Hence, a large dimensionality reduction can be achieved by using internal coordinates, without limiting the folded states a protein can attain.

## 6. PERFORMANCE EXPERIMENTS

In this section we present experimental results from benchmark and application executions on a dedicated 16-node Sun Fire x4100 cluster with Gigabit Ethernet, each node with 2 dual-core AMD Opteron 275 CPUs. The software setup includes Linux 2.6, GCC 4.3 and MPICH2.

**Parallel Multistart** In order to evaluate parallel Multistart we use a test function with 10 variables and artificial delays that range from 1ms to 1000ms. Fig. 2 depicts the speedup for a single starting point, which represents a worst-case but unlikely to occur scenario in global optimization problems. We observe that the Newton method fails to scale as the number of workers increases, regardless of the function evaluation time. This is mostly attributed to the small sequential task ($\simeq$2%) of the Newton method. The speedup can be further affected by the communication overheads, especially when the computational cost of the objective function is low. For function evaluation time

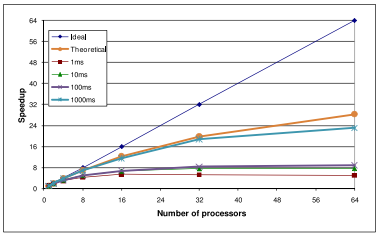


**Figure 2. Speedup for 1 optimization**

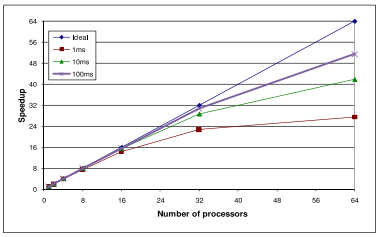


**Figure 3. Speedup for 16 optimizations**

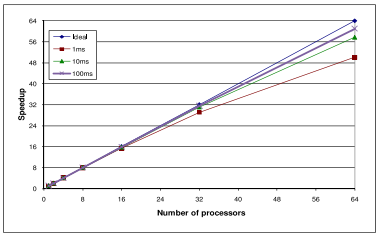


**Figure 4. Speedup for 64 optimizations**

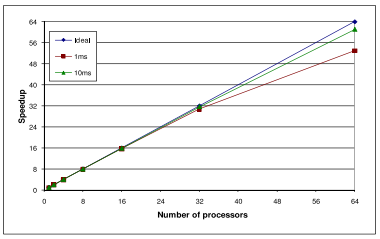


**Figure 5. Speedup for 1024 optimizations**

equal to 1000ms, however, the measured speedup is very close to the maximum theoretical speedup as defined by Amdahl's law. Figs. 3 to 5 show the speedup of Multistart for 16, 64 and 1024 optimizations. The attained speedup increases with the number of optimizations, especially if this exceeds the number of available processing cores. For 1024 optimizations, the speedup almost coincides with the ideal for both 10ms and 100ms evaluation time.

The next experiment studies the performance behavior of Multistart with respect to the number of variables. Fig. 6

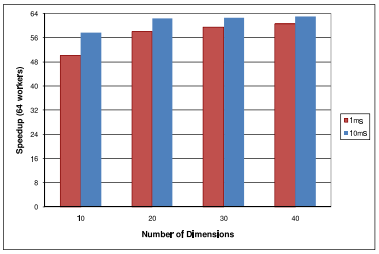


**Figure 6. Speedup for 64 optimizations, 1ms and 10ms function evaluation time and 10-40 variables**

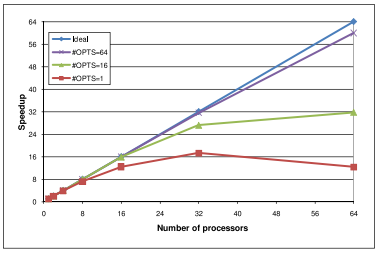


**Figure 7. Speedup for 1, 16 and 64 local searches for Polyalaline-8**

depicts the speedup on 64 workers for 64 initial points and function evaluation time equal to 1ms and 10ms. We observe that the obtained speedup increases with the number of function variables. For 10ms delay, the obtained speedup is significantly higher for 10 variables and approximates the ideal if more function variables are used.

**Protein folding** To model the potential energy of Eq. 3 we used the Tinker [7] software, a flexible system of programs and routines for molecular mechanics and dynamics. The potential energy function used is the AMBER96 [8] potential implemented in Tinker. The AMBER96 potential is minimized with Multistart, using the reduced scheme of dihedral angles. That is, the free variables of minimization are the dihedral angles between consecutive triads of atoms. The protein tested consists of 8 Alaline amino acids (Polyalaline-8) in the primary structure chain, plus one ACE molecule at the beginning and a NME molecule at the end of the chain. This results in a total of 92 atoms and 276 Cartesian/internal coordinates. By allowing only the dihedral angles to vary while bond length and bond angle are kept fixed the final number of optimization parameters is reduced to 35. The specific protein is described from the sequence of amino acids that it is composed of in a special formatted file. The goal is to reproduce the already known global minimum conformation of Polyalaline-8 that was first explored in [9].

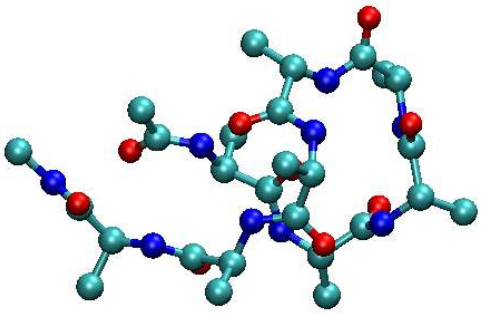

**Figure 8. Polyalaline-8 minimum energy conformation, $E_{AMBER96} = -44.45$**

Fig. 7 shows the speedup of Multistart for 1, 16 and 64 initial points. Every energy function performed by Tinker takes approximately 4 milliseconds for Polyalaline-8. We observe that the speedup increases with the number of independent optimizations and that the results accord with those presented previously using the test function. As the number of initial points increases, the speedup of Multistart approaches the ideal as better load balancing is achieved. In all the experiments we compute the Hessian matrix with function evaluations; an alternative approach is to use the analytic gradient routine of the objective function, if this is available. This reduces the number of function evaluations of Multistart and equivalently the number of spawned tasks. For instance, the computational cost of the gradient function for Polyalaline-8 is approximately 1 ms. When gradient calls are used, the performance scalability of Multistart remains the same, with only difference the considerably smaller execution time of each local optimization. In Fig. 8 we present the final conformation of Polyalaline-8 as computed by the parallel Multistart. The result matches the global best conformation presented in [9].

## 7. RELATED WORK

Although many parallel local and global optimization algorithms were proposed in the last decades, only a handful of actual systems exist. One of the most widely used scientific software programs, MATLAB, presented its first parallel optimization solution in 2009 [10]. In the pioneer work of [11] an interval global optimization method is implemented using dynamic load balancing. PGO [12] is a general parallel computing based on the Genetic Algorithm. In PGO, the parallel (and heterogeneous) computing framework is organized as a global master-slave system using a central database management system for stor-

ing all the data during optimization progress. Oriented in interoperability, the MHGrid platform [13] exploits metaheuristics based search methods and Grid computing to enable the transparent sharing of heterogeneous and dynamic resources offering a versatile Global optimization framework. MANGO [14] is a middleware that involves the development of an extensible and flexible multiagent platform, in which autonomous agents can solve global optimization problems in cooperation. Finally, PaGMO [15] is a recently released open source multi-threaded software that offers a plethora of local and global optimization codes exploiting modern multi-core architectures. In contrast to our infrastructure, none of the above supports hierarchical and multi-level task parallelism. In addition, our system is platform-agnostic supporting transparently both shared and distributed memory architectures. Some work on parallel global optimization methods for protein folding problems is reported in [16, 17, 18]. The proposed algorithms focus mainly on the independent execution of multiple local optimization algorithms.

## 8. CONCLUSIONS

We presented a system for efficient exploitation of nested and irregular parallelism in non-linear optimization problems. At the core of our system is TORC, a runtime library that supports adaptive task-based parallelism on clusters of multicores/SMPs. Using TORC, we manage to extract and execute the multiple levels of parallelism inherent in the Multistart optimization method, performing thus Newton-based local searches, gradient and Hessian calculations and function evaluations in parallel. The scalability of our system was demonstrated on a multicore cluster with synthetic benchmarks and a real application case that deals with the protein folding problem. Our ongoing work includes the integration of additional numerical optimization techniques into our infrastructure. Furthermore, we plan to extend the applicability of our system to computational grids and GPGPU environments.

## REFERENCES


[1] A.H.G Rinnooy Kan and G.T. Timmer, "Stochastic global optimization methods. Part I: Clustering methods," *Math. Progr.*, 39, 27-56, 1987.

[2] C. Voglis and I.E. Lagaris, Towards ideal multistart, "A stochastic approach for locating the minima of a continuous function inside a bounded domain," *Applied Math. and Comput.*, 213, pp. 216-229, 2009.

[3] C. Voglis, P.E. Hadjidoukas, I.E. Lagaris and D.G. Papageorgiou, "A numerical differentiation library exploiting parallel architectures," *Comput. Phys. Commun.*, 180(8), pp. 1404-1415, 2009.

[4] P.E. Hadjidoukas, V.V. Dimakopoulos, C. Voglis, I.E. Lagaris and D.G. Papageorgiou, "High-performance numerical optimization on multicore clusters," *17th Intl. Euro-Par Conf.*, 2011.

[5] C.B. Anfinsen, "Principles that Govern the Folding of Protein Chains," *Science*, 181, pp. 223-230, 1973.

[6] G.N. Ramachadran et al., *Biochim Biophysics*, 74, pp. 359:298-302, 1974.

[7] J.W. Ponder, "TINKER-software tools for molecular design," *St. Louis:Washington University*, Version 3.7, 1999.

[8] S.J. Weiner et al., "An All Atom Force Field for Simulations of Proteins and Nucleic Acids," *J. Comp. Chem.*, 7(2), pp. 230-252, 1986.

[9] P.N. Mortenson and D.J. Wales, "Energy landscapes, global optimization and dynamics of the polyalanine Ac (ala) NHMe," *The Journal of Chemical Physics*, 114, pp. 6443-6454, 2001.

[10] S. Kozola, "Improving Optimization Performance with Parallel Computing," *MATLAB Digest*, 2009.

[11] J. Eriksson and P. Lindstrom, "A parallel interval method implementation for global optimization using dynamic load balancing," *Rel. Comput.*, 1/2, pp. 77-91, 1995.

[12] K. He et al., "PGO: a Parallel Computing Platform for Global Optimization Based on Genetic Algorithm," *Computers & Geosciences*, 2006.

[13] M. Wahib et al., "Mhgrid: Towards an ideal optimization environment for global optimization problems using grid computing," *8th Int'l Conf. on Parallel and Distr. Comput., Applic. and Technologies*, pp. 167-168, 2007.

[14] A. Günay et al., "Solving global optimization problems using MANGO," *Agent and Multi-Agent Systems: Technologies and Applic.*, pp. 783-792, 2009.

[15] F. Biscani, D. Izzo and C. Yam, "A global optimisation toolbox for massively parallel engineering optimisation," *Int'l Conf. on Astrodynamics Tools and Techniques*, 2010.

[16] R. H. Byrd et al., "A Parallel Global Optimization Method for Solving Molecular Cluster and Polymer Conformation Problems," *7th Siam Conf. on Parallel Processing for Scientific Comput.*, SIAM Philadelphia, pp. 72-77, 1995.

[17] T.F. Coleman and Z. Wu, "Parallel continuation-based global optimization for molecular conformation and protein folding," *J. Global Optimization*, 8:1, 49-65, 1996.

[18] S. Crivelli, T. Head-Gordon, R. Byrd, E. Eskow and R. Schnabel, "A hierarchical approach for parallelization of a global optimization method for protein structure prediction," *Euro-Par99 Parallel Processing*, pp. 579-585, 1999.