Computer Physics Communications 185 (2014) 2217–2219

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

NDL-v2.0: A new version of the numerical differentiation library for parallel architectures

P.E. Hadjidoukas [a,*], P. Angelikopoulos [a], C. Voglis [b], D.G. Papageorgiou [c], I.E. Lagaris [b]

[a] Computational Science, ETH Zurich, Zurich CH-8092, Switzerland

[b] Department of Computer Science and Engineering, University of Ioannina, P.O. BOX 1186, 45110 Ioannina, Greece

[c] Department of Materials Science and Engineering, University of Ioannina, P.O. BOX 1186, 45110 Ioannina, Greece

A R T I C L E   I N F O

Article history:
Received 19 February 2014
Received in revised form
31 March 2014
Accepted 2 April 2014
Available online 13 April 2014

Keywords:
Numerical differentiation
Finite differences
Optimization
Nonlinear equations
OpenMP
MPI
Python
Parallel processing

A B S T R A C T

We present a new version of the numerical differentiation library (NDL) used for the numerical estimation of first and second order partial derivatives of a function by finite differencing. In this version we have restructured the serial implementation of the code so as to achieve optimal task-based parallelization. The pure shared-memory parallelization of the library has been based on the lightweight OpenMP tasking model allowing for the full extraction of the available parallelism and efficient scheduling of multiple concurrent library calls. On multicore clusters, parallelism is exploited by means of TORC, an MPI-based multi-threaded tasking library. The new MPI implementation of NDL provides optimal performance in terms of function calls and, furthermore, supports asynchronous execution of multiple library calls within legacy MPI programs. In addition, a Python interface has been implemented for all cases, exporting the functionality of our library to sequential Python codes.

New version program summary

Program title: NDL-v2.0
Catalog identifier: AEDG_v2_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEDG_v2_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: Standard CPC licence, http://cpc.cs.qub.ac.uk/licence/licence.html
No. of lines in distributed program, including test data, etc.: 63036
No. of bytes in distributed program, including test data, etc.: 801872
Distribution format: tar.gz
Programming language: ANSI Fortran-77, ANSI C, Python.
Computer: Distributed systems (clusters), shared memory systems.
Operating system: Linux, Unix.
Has the code been vectorized or parallelized?: Yes.
RAM: The library uses $O(N)$ internal storage, N being the dimension of the problem. It can use up to $O(N^2)$ internal storage for Hessian calculations, if a task throttling factor has not been set by the user.
Classification: 4.9, 4.14, 6.5.
Catalog identifier of previous version: AEDG_v1_0
Journal reference of previous version: Comput. Phys. Comm. 180(2009)1404
Does the new version supersede the previous version?: Yes
Nature of problem:
The numerical estimation of derivatives at several accuracy levels is a common requirement in many computational tasks, such as optimization, solution of nonlinear systems, and sensitivity analysis. For a large number of scientific and engineering applications, the underlying functions correspond to simulation codes for which analytical estimation of derivatives is difficult or almost impossible. A parallel implementation that exploits systems with multiple CPUs is very important for large scale and computationally expensive problems.

* Corresponding author.
E-mail addresses: phadjido@mavt.ethz.ch, phadjido@gmail.com (P.E. Hadjidoukas).

http://dx.doi.org/10.1016/j.cpc.2014.04.002

Solution method:

Finite differencing is used with a carefully chosen step that minimizes the sum of the truncation and round-off errors. The parallel versions employ both OpenMP and MPI libraries.

Reasons for new version:

The updated version was motivated by our endeavors to extend a parallel Bayesian uncertainty quantification framework [1], by incorporating higher order derivative information as in most state-of-the-art stochastic simulation methods such as Stochastic Newton MCMC [2] and Riemannian Manifold Hamiltonian MC [3]. The function evaluations are simulations with significant time-to-solution, which also varies with the input parameters such as in [1, 4]. The runtime of the N-body-type of problem changes considerably with the introduction of a longer cut-off between the bodies. In the first version of the library, the OpenMP-parallel subroutines spawn a new team of threads and distribute the function evaluations with a PARALLEL DO directive. This limits the functionality of the library as multiple concurrent calls require nested parallelism support from the OpenMP environment. Therefore, either their function evaluations will be serialized or processor oversubscription is likely to occur due to the increased number of OpenMP threads. In addition, the Hessian calculations include two explicit parallel regions that compute first the diagonal and then the off-diagonal elements of the array. Due to the barrier between the two regions, the parallelism of the calculations is not fully exploited. These issues have been addressed in the new version by first restructuring the serial code and then running the function evaluations in parallel using OpenMP tasks. Although the MPI-parallel implementation of the first version is capable of fully exploiting the task parallelism of the PNDL routines, it does not utilize the caching mechanism of the serial code and, therefore, performs some redundant function evaluations in the Hessian and Jacobian calculations. This can lead to: (a) higher execution times if the number of available processors is lower than the total number of tasks, and (b) significant energy consumption due to wasted processor cycles. Overcoming these drawbacks, which become critical as the time of a single function evaluation increases, was the primary goal of this new version. Due to the code restructure, the MPI-parallel implementation (and the OpenMP-parallel in accordance) avoids redundant calls, providing optimal performance in terms of the number of function evaluations. Another limitation of the library was that the library subroutines were collective and synchronous calls. In the new version, each MPI process can issue any number of subroutines for asynchronous execution. We introduce two library calls that provide global and local task synchronizations, similarly to the BARRIER and TASKWAIT directives of OpenMP. The new MPI-implementation is based on TORC, a new tasking library for multicore clusters [5–7]. TORC improves the portability of the software, as it relies exclusively on the POSIX-Threads and MPI programming interfaces. It allows MPI processes to utilize multiple worker threads, offering a hybrid programming and execution environment similar to MPI+OpenMP, in a completely transparent way. Finally, to further improve the usability of our software, a Python interface has been implemented on top of both the OpenMP and MPI versions of the library. This allows sequential Python codes to exploit shared and distributed memory systems.

Summary of revisions:

The revised code improves the performance of both parallel (OpenMP and MPI) implementations. The functionality and the user-interface of the MPI-parallel version have been extended to support the asynchronous execution of multiple PNDL calls, issued by one or multiple MPI processes. A new underlying tasking library increases portability and allows MPI processes to have multiple worker threads. For both implementations, an interface to the Python programming language has been added.

Restrictions:

The library uses only double precision arithmetic. The MPI implementation assumes the homogeneity of the execution environment provided by the operating system. Specifically, the processes of a single MPI application must have identical address space and a user function resides at the same virtual address. In addition, address space layout randomization should not be used for the application.

Unusual features:

The software takes into account bound constraints, in the sense that only feasible points are used to evaluate the derivatives, and given the level of the desired accuracy, the proper formula is automatically employed.

Running time:

Running time depends on the function's complexity. The test run took 23 ms for the serial distribution, 25 ms for the OpenMP with 2 threads, 53 ms and 1.01 s for the MPI parallel distribution using 2 threads and 2 processes respectively and yield-time for idle workers equal to 10 ms.

References:

[1] P. Angelikopoulos, C. Paradimitriou, P. Koumoutsakos, Bayesian uncertainty quantification and propagation in molecular dynamics simulations: a high performance computing framework, J. Chem. Phys 137 (14).

[2] H.P. Flath, L.C. Wilcox, V. Akcelik, J. Hill, B. van Bloemen Waanders, O. Ghattas, Fast algorithms for Bayesian uncertainty quantification in large-scale linear inverse problems based on low-rank partial Hessian approximations, SIAM J. Sci. Comput. 33 (1) (2011) 407–432.

[3] M. Girolami, B. Calderhead, Riemann manifold Langevin and Hamiltonian Monte Carlo methods, J. R. Stat. Soc. Ser. B (Stat. Methodol.) 73 (2) (2011) 123–214.

[4] P. Angelikopoulos, C. Paradimitriou, P. Koumoutsakos, Data driven, predictive molecular dynamics for nanoscale flow simulations under uncertainty, J. Phys. Chem. B 117 (47) (2013) 14808–14816.

[5] P.E. Hadjidoukas, E. Lappas, V.V. Dimakopoulos, A runtime library for platform-independent task parallelism, in: PDP, IEEE, 2012, pp. 229–236.

[6] C. Voglis, P.E. Hadjidoukas, D.G. Papageorgiou, I. Lagaris, A parallel hybrid optimization algorithm for fitting interatomic potentials, Appl. Soft Comput. 13 (12) (2013) 4481–4492.
[7] P.E. Hadjidoukas, C. Voglis, V.V. Dimakopoulos, I. Lagaris, D.G. Papageorgiou, Supporting adaptive and irregular parallelism for non-linear numerical optimization, Appl. Math. Comput. 231 (2014) 544–559.