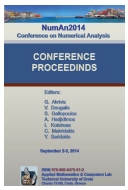
OpenAccess

Proceedings of the 6th International Conference on Numerical Analysis, pp 1-6

Contents lists available at AMCL's Digital Library

NumAn2014 Conference Proceedings

Digital Library Triton : http://lib.amcl.tuc.gr



Application of a hybrid parallel Monte Carlo PDE Solver on rectangular multi-domains

Panayiotis Alefragis[a], **Alexandros Spyrou**[b], **Spiridon Likothanassis**[b]
[a]Computer and Informatics Engineering Dept.
Technologigal Educational Institute of Western Greece
Antirio, Nafpaktos, Greece
[b]Computer Engineering and Informatics Department
University of Patras
Rio, Patras, Greece
alefrag@teimes.gr,kspyrou,likothan@ceid.upatras.gr

Abstract. The paper presents a hybrid parallelization approach that uses MPI and POSIX threads to harness the performance of clustered multicore servers in solving Poisson Elliptic PDEs on 2D and 3D rectangular multi-domains using the Monte Carlo method. A sequential performance analysis and the parallel performance achieved on a virtual cluster are presented. Extensive results using different parameter configurations are discussed and comparison between the performance of sequential and the hybrid implementations are presented. The hybrid implementation manages to demonstrate significant speedup while maintaining the quality of the solution on sample input datasets. Finally, insight for further extensions to the proposed algorithm are presented.

Key words: Hybrid Parallel Programming, MPI, POSIX Threads, Monte Carlo PDE solver.

Introduction

The Monte Carlo method is known for many years but only after the advent of the first electronic computers, the method has received an increasing interest as a tool for solving important physical problems. Monte Carlo algorithm generates a random walk using a proposed density and includes a method for rejecting / accepting the proposed moves. Nowadays it is used in nearly every aspect of scientific inquiry. Monte Carlo methods always attracted people working in the field of numerical solutions of PDE. This attraction has been evolved with the passing of the years together with the rapid changes in technology like parallel computing , clusters etc. [1] [2] [3].
The paper is organized as follows. In section 2 we present the sequential analysis of the algorithm. The distributed memory parallelization approach is presented in section 3. Parallel platform details and experimental results can be found in section 4. Finally, conclusions and future extensions are given in section 5.

ISBN: 978-960-8475-22-9 *http://lib.amcl.tuc.gr/handle/triton/20*

Base Algorithm Analysis

Base Algorithm Description

The initial step in harnessing the performance of a multicore cluster was to perform a profiling of the sequential implementation to identify hotspots in the code that could lead to reduced execution times, if they were efficiently parallelized. The performance analysis was based on an implementation of the algorithms proposed in Vavalis et al. [4]. The aforementioned implementation is able to solve Poisson's equation with Dirichlet boundary conditions on 2D and 3D hyper rectangles domains, but can easily be extended to use other methods and domains. The algorithm is a hybrid algorithm in the sense that both not deterministic and deterministic methods are used to solve the domains. Initially, a probabilistic domain decomposition is applied to compute local solutions' estimates along the boundaries of the subdomain using the Monte Carlo method [5] and subsequently an interpolation of the estimated local solutions is applied. In particular, for 2D domains the Burkardt's splines library is used [7], while the Multilevel B-spline (MBA) [8] library is used for the 3D domains to form boundaries of the new subdomains. Finally, as the boundaries of the subdomains are known a deterministic Laplace Solver that utilizes Deal.II [9] is used to find the solution of the problem for each subdomain. The base implementation can be described by pseudo code in Figure 1. An interesting implementation detail was that shared memory parallelization was already supported using the Pthreads library.

```
Determine the node coordinates of all subdomains
For each node
        Generate one job
        Assign the job to a thread
        Perform Monte Carlo to estimate node values
Use (2D|3D) interpolation function to find new boundaries
For each subdomain
        Generate one job
        Assign the job to a thread
        Solve using Laplace solver
Output the results
```

Fig. 1 Pseudo Code of base algorithm

Base Code Profiling

Before parallelizing legacy code in a distributed memory architecture a profiling step is important in order to pin point where the code spends execution time and to determine execution sequence. In this way, the code can be segmented to kernels that can be analyzed to determine if it is possible to harness computational resources in parallel and if the parallelization of the kernel will have significant impact on the overall execution of the profiled program. We have performed an extensive performance analysis using a number of different configurations of parameters for the used problem set, see Table 1, and we have identified 3 kernels that could potentially improve the overall performance if a distributed implementation existed. The three kernels are the Monte Carlo algorithm, the Interpolation Algorithm and the Laplace Solver. As for most input problems the Monte Carlo Algorithm was taking from 50% to 94% of the execution time,

we focused our efforts in efficiently parallelizing the Monte Carlo algorithm using MPI.

Table 1 presents the main characteristics of the input problem set. *dim* represents if the problem is 2D or 3D, *dlen* the length of the domain along each dimension, *subd* the number of subdomains along each dimension, *dect* the decomposition type for each dimension with *u* representing uniform and *n* non-uniform, *decc* the coordinates of the interfaces corresponding to each dimension along which the domain is decomposed non-uniformly, *nppp* the number of nodes on an interface along a dimension. For 2D it takes 2 arguments corresponding to the decomposing lines along dimension Y and X respectively, while for 3D it takes 6 arguments, with each pair corresponding to the decomposing planes YZ, XZ and XY respectively. Finally, *btol* represents the boundary tolerance. Table 2 displays the percentage of the sequential execution time for different

Table 1. Problem set characteristic.

Problem	dim	dlen	subd	dect	decc	nppp	btol
1	2D	1. 1.	2 4	n u	.3	12 12	1e-15
2	2D	1. 1.	2 4	n n	.3 .25 .5 .7	12 12	1e-15
3	2D	1. 1.	2 4	n n	.3 .25 .5 .7	12 12	1e-15
4	3D	1. 1. 1.5	3 3 2	u u u	5 5 5 5	12 12	1e-15
5	3D	1. 1. 1.5	3 3 2	u n u	.5 .55 5 5 5 5	5 5 5 5 7 7	1e-15
6	3D	1. 1. 1.5	3 3 2	n n n	.5 .4 .5 .55 1.2 5 5 5 5	5 5 5 5 7 7	1e-15
7	3D	1. 1. 1.5	3 3 2	n n n	.5 .4 .5 .55 1.2 5 5 5 5	5 5 5 5 7 7	1e-15

walk sizes of the problem set for the aforementioned kernels, with MC standing for Monte Carlo, I for Interpolation and LS for the Laplace Solver.

Table 2. Performance analysis

Walks	5K			20K			80K		
Problem	MC	I	LS	MC	I	LS	MC	I	LS
1	60%	-	5%	79%	-	4%	95%	-	5%
2	50%	-	10	76%	-	3%	90%	-	5%
3	74%	-	9%	88%	-	4%	92%	-	4%
4	80%	13%	7%	85%	11%	4%	88%	5%	7%
5	70%	20%	10%	83%	9%	8%	90%	6%	4%
6	78%	12%	10%	84%	10%	6%	94%	3%	1%
7	74%	10%	6%	81%	11%	8%	90%	6%	4%

Hybrid Parallelization Approach

As the base implementation already supports shared memory parallelization of the Monte Carlo and Laplace Solver using Pthreads, we focused on directly implement-

ing a hybrid parallel approach using OpenMPI [6] for inter process communication. Based on the experimental results of the performance analysis, we concluded that:

1. a simple master/worker scheme was sufficient to distribute the computation load of the Monte Carlo kernel between processes. This was due to the fact that each Monte Carlo thread work is mainly determined by the number of walks so a fair distribution of nodes between the worker process and subsequently the local distribution of nodes between threads will not create a major load imbalance.
2. the master process could efficiently perform the interpolation and the Laplace Solver Kernel for each subdomain locally using threads, after it has received the required estimation of the relevant nodes from the other processes.

The base code logic remains mainly unchanged. A layer that distributes the computational load between and gathers the estimation results from each worker process was added.

Experimental Results

We have performed extensive experimental tests on a 8 machines virtual cluster running over an OpenStack Cloud infrastructure of the computing center of the Computer and

Table 3. Hybrid execution times and achieved speed up

Problem	1							2						
np	1	2	4	8	16	32	64	1	2	4	8	16	32	64
nt. Walks	1	2	4	6	6	8	8	1	2	4	6	6	8	8
40K	26	8	6	3	3	4	3	26	9	7	4	3	3	4
80K	49	14	11	5	4	6	5	50	14	10	6	4		5
160K	98	24	19	9	6	6	8	99	25	19	9	6	7	9
Problem	3							4						
40K	25	10	5	3	3	2	3	538	183	120	70	62	68	77
80K	40	26	10	4	3	4	4	931	298	170	88	80	86	90
160K	99	38	18	8	5	6	6	1777	511	295	124	109	122	129
Problem	5							6						
40K	534	183	120	70	62	68	74	494	176	423	78	72	75	77
80K	946	298	170	88	180	86	90	847	273	164	97	82	92	94
160K	1757	511	295	124	109	122	129	1550	464	292	135	106	121	130
Problem	7							Problem Set						
40K	365	176	109	78	72	75	77	2008	745	610	306	277	295	312
80K	713	273	164	97	82	92	94	3576	1196	699	385	335	371	382
160K	1416	464	292	135	105	121	130	6796	2037	1230	544	446	505	541
Problem Set Speedup								Min	2.81	2.03	6.33	6.86	6.59	6.42
								Avg	3.08	4.17	8.85	10.60	9.53	9.12
								Max	3.34	5.13	11.48	14.62	12.81	11.92

Informatics Department of the TEI of Western Greece. The cloud infrastructure uses a

Dell Blade Enclosure with 16 dual processor Xeon servers, with each processor having 6 hyper-threading cores and 48Gbytes of main memory. The blades were interconnected with 1 GBit Ethernet. Each virtual machine had 4 virtual CPUs and 8 GBytes RAM. For all the available problem in the input set, we created all run combinations for the maximum number of threads *nt*={1,2,4,6,8} per process *np*={1,2,4,8,16,32,64} and the number of random walks to be performed per node by the Monte Carlo algorithm, *walks* = {20K, 40K, 80K, 160K} in order to be able to determine the speedup and the scaling of the proposed approach. For the profiling of the hybrid implementation we have used mpiP profiler [10] which is a lightweight profiling library for MPI applications. Because it only collects statistical information about MPI functions, mpiP generates considerably less overhead and much less data than tracing tools. All the information captured by mpiP is task-local.

Table 3 presents the execution time and the speed up achieved for all the input set. Due to space restriction only part of the results are presented. Detailed executions analysis show that the hybrid algorithm scales almost linearly to the number of active processing threads, ie the number of processes multiplied by the number of threads, until the execution time of the Monte Carlo kernel is significantly reduced due to parallelization and the other kernels now dominate the execution time. Figure 2 presents the speedup and the execution time for some of the input examples.

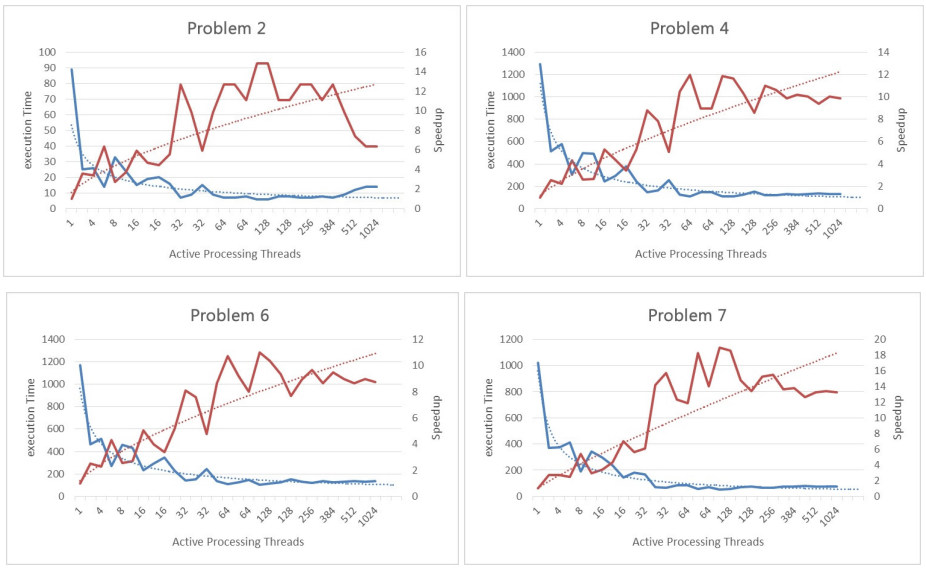


Fig. 2 Execution time and speedup for 160K walks

Conclusions and Future Work

This paper analyzes the sequential performance of a PDE solver that is applied on Elliptic PDEs (and Poisson in particular) on rectangular multi-domains in both two and three

dimensions. The paper presents implementation details and extensions to the sequential algorithm in order to allow efficient implementation in a distributed memory environment. The experimental results presented show that an efficient implementation of the base algorithm can be achieved on distributed memory architectures. The hybrid implementation manages to demonstrate significant speedup while maintaining the quality of the solution on sample input datasets. Our future plans include the parallelization of the Laplace Solver on distributed memory machines and and the integration of the hybrid algorithm to the Fenics platform [11].

References

1. K. P. Esler, Jeongnim Kim, L. Shulenburger, and D.M. Ceperley *"Fully accelerating quantum Monte Carlo simulations of real materials on GPU clusters"*
2. John Strikwerda, *"Finite Difference Schemes and Partial Differential Equations"*, SIAM, 2nd edition, 2004.
3. Norma Alias , *"Some Parallel Numerical Methods in Solving Partial Differential Equations "*, submitted 2nd International Conference on Computer Engineering and Technology,2010.
4. G. Sarailidis and M. Vavalis, *"Hybrid Solvers for Elliptic PDEs on rectangular multi-domains in 2 and 3 dimensions"*, submitted to Scientific Computing, 2013.
5. J.M Delaurentis, L.A Romero, *"A Monte Carlo method for poisson's equation", Journal of Computational Physics, Volume 90, Issue 1"*, September 1990, Pages 123-140, ISSN 0021-9991, http://dx.doi.org/10.1016/0021-9991(90)90199-B.
6. Edgar Gabriel et al. *"Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation"*, In Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, September 2004.
7. SPLINE, a C++ library which constructs and evaluates spline functions. http://people.sc.fsu.edu/j̃burkardt/cpp_src/spline/spline.html
8. MBA - Multilevel B-Spline Approximation Library : http://www.sintef.no/Projectweb/Geometry-Toolkits/MBA/
9. deal.II - an open source finite element library http://www.dealii.org/
10. mpiP Profiler - a lightweight profiling library for MPI applications. http://mpip.sourceforge.net/
11. Fenics - The FEniCS Project is a collection of free software with an extensive list of features for automated, efficient solution of differential equations. http://fenicsproject.org/

Acknowledgments

The present research work has been co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program Education and Lifelong Learning of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALIS . Investing in knownledge society through the European Social Fund (MIS 379416).