Научном већу Института за физику у Београду

Београд, 4. oktobar 2016. године

Предмет: **Молба за покретање поступка за реизбор у звање истраживач сарадник**

М О Л Б А

С обзиром да испуњавам критеријуме прописане од стране Министарства просвете, науке и технолошког развоја за реизбор у звање истраживач сарадник, молим Научно веће Института за физику у Београду да покрене поступак за мој реизбор у наведено звање.

У прилогу достављам:

1. Мишљење руководиоца пројекта
2. Кратку стручну биографију
3. Списак објављених радова и других публикација
4. Потврду о уписаним докторским студијама
5. Кратак преглед научне активности
6. Копије објављених радова и других публикација
7. Решење о претходном избору у звање истраживач сарадник
8. Решење о прихватању теме докторске дисертације
9. Уверење о положеним испитима на докторским студијама
10. Уверење о стеченом високом образовању другог степена мастер академских студија са списком положених испита
11. Уверење о стеченом високом образовању са списком положених испита

Са поштовањем,

Владимир Лончар

Научном већу Института за физику у Београду

Београд, 4. oktobar 2016. године

Предмет: **Мишљење руководиоца пројекта о реизбору Владимира Лончара у звање истраживач сарадник**

Владимир Лончар је запослен у Лабораторији за примену рачунара у науци Института за физику и ангажован је на пројекту основних истраживања Министарства просвете, науке и технолошког развоја Републике Србије ОН171017, под називом "Моделирање и нумеричке симулације сложених вишечестичних физичких система". На поменутом пројекту ради на темама из развоја нумеричких метода за паралелне рачунарске системе под руководством др Антуна Балажа. С обзиром да испуњава све предвиђене услове у складу са Правилником о поступку, начину вредновања и квантитативном исказивању научноистраживачких резултата истраживача МПНТР, сагласан сам са покретањем поступка за реизбор Владимира Лончара у звање истраживач сарадник.

За састав комисије за реизбор Владимира Лончара у звање истраживач сарадник предлажем:

(1) др Антун Балаж, научни саветник, Институт за физику у Београду
(2) др Ненад Вукмировић, научни саветник, Институт за физику у Београду
(3) проф. др Срђан Шкрбић, ванредни професор Природно-математичког факултета Универзитета у Новом Саду

Руководилац пројекта

[illegible]

др Антун Балаж
научни саветник

Биографија Владимира Лончара

Владимир Лончар је рођен 28. октобра 1985. године у Новом Саду. Основне студије на Природно-математичком факултету Универзитета у Новом Саду, смер дипломирани информатичар - пословна информатика, уписао је 2004. године, а завршио 2009. године. Мастер студије на истом факултету, на смеру информациони системи, завршио је 2011. године. Израдом дипломског и мастер рада руководио је проф. др Срђан Шкрбић. Школске 2011/2012 године је уписао докторске студије информатике на Департману за математику и информатику Природно-математичког факултета Универзитета у Новом Саду. Ментори докторских студија Владимира Лончара су проф. др Срђан Шкрбић са Природно-математичког факултета Универзитета у Новом Саду и др Антун Балаж са Института за физику у Београду.

Владимир Лончар је од 2012. до краја 2014. године активно учествовао у развоју информационог система Природно-математичког факултета у Новом Саду, где је био и запослен. Од 2015. је запослен у Лабораторији за примену рачунара у науци Института за физику у Београду, на пројекту основних истраживања ОН171017 "Моделирање и нумеричке симулације сложених вишечестичних система".

Од претходног избора у звање Владимир Лончар је објавио 2 рада категорије М21а, једно саопштење категорије М33 и два саопштења категорије М34. Један од радова категорије М21а је у Web of Science означен као Highly Cited Paper за период од објављивања до јуна 2016. године.

Списак радова Владимира Лончара

Радови у међународним часописима изузетних вредности (M21a) након претходног избора у звање

1. CUDA programs for solving the time-dependent dipolar Gross-Pitaevskii equation in an anisotropic trap
   V. Lončar, A. Balaž, A. Bogojević, S. Škrbić, P. Muruganandam, and S. Adhikari
   Comput. Phys. Commun. **200**, 406 (2016)
2. OpenMP, OpenMP/MPI, and CUDA/MPI C programs for solving the time-dependent dipolar Gross–Pitaevskii equation
   V. Lončar, L. E. Young-S., S. Škrbić, P. Muruganandam, S. Adhikari, and A. Balaž
   Comput. Phys. Commun. **209**, 190 (2016)

Саопштења са међународних скупова штампана у целини (M33) након претходног избора у звање

1. Parallelization of minimum spanning tree algorithms using distributed memory architectures
   V. Lončar, S. Škrbić, and A. Balaž
   Transactions on Engineering Technologies, pp. 543-554, Springer (2014)
   G.-C. Yang, S-I. Ao, L. Gelman (Eds.), Special Volume of the World Congress on Engineering 2013.
   DOI: 10.1007/978-94-017-8832-8_39

Саопштења са међународних скупова штампана у целини (M33) пре претходног избора у звање

1. Parallel implementation of minimum spanning tree algorithms using MPI
   V. Lončar, S. Škrbić
   IEEE 13th International Symposium on Computational Intelligence and Informatics (CINTI), pp. 35-38 (2012).

Саопштења са међународних скупова штампана у изводу (M34) након претходног избора у звање

1. Rosensweig instability due to three-body interaction or quantum fluctuations?
   V. Lončar, D. Vudragović, A. Balaž, A. Pelster
   DPG 2016 conference, Q17.2, Hannover, Germany (2016)
2. Trapped Bose-Einstein Condensates with Strong Disorder
   V. Lončar, A. Balaž, A. Pelster
   Book of abstracts of V International School and Conference on Photonics - Photonica 2015, Belgrade, Serbia, 24-28 August 2015

PRIRODNO-MATEMATIČKI FAKULTET
Univerzitet u Novom Sadu

FACULTY OF SCIENCES
University of Novi Sad

TRG DOSITEJA OBRADOVIĆA 3, 21000 NOVI SAD, SRBIJA (SERBIA)
tel +381.21.455.630 **fax** +381.21.455.662 **e-mail** dekanpmf@uns.ac.rs **web** www.pmf.uns.ac.rs
PIB 101635863 **MB** 08104620

Uverenje br.: 287/2016
Broj dosijea: 72d/11

Na osnovu čl. 161 Zakona o opštem upravnom postupku ("Službeni list SRJ" br. 33/97 i 31/2001) i ("Službeni glasnik RS" br. 30/2010) i po molbi Lončar (Milenko) Vladimir od 13.06.2016. godine izdaje se

U V E R E NJ E

kojim se potvrđuje da je Lončar (Milenko) Vladimir, rođen 28.10.1985. godine u mestu Novi Sad, opština Novi Sad, država Republika Srbija student koji se sam finansira. Upisan je na 3. godinu (3. put) doktorskih studija Prirodno-matematičkog fakulteta Univerziteta u Novom Sadu, na Departmanu za matematiku i informatiku, na studijskom programu Doktorske akademske studije informatike, školske 2015/2016 godine.

Lončar (Milenko) Vladimir upisan je prvi put školske 2011/2012. godine, na Departmanu za matematiku i informatiku, na studijskom programu Doktorske akademske studije informatike.

Uverenje se izdaje na lični zahtev imenovanog.

Republika Srbija Autonomna Pokrajina Vojvodina Univerzitet u Novom Sadu Prirodno-matematički fakultet Novi Sad XIV

Referent

[illegible]

Saša Kočiš

Novi Sad, 13.06.2016.

Кратак преглед научне активности Владимира Лончара

Владимир Лончар је од фебруара 2015. године ангажован у Лабораторији за примену рачунара у науци Института за физику у Београду и његов истраживачки рад се одвија под руководством др Антуна Балажа. Истраживачки рад Владимира Лончара је био фокусиран на примену рачунара у науци, у складу са планираним садржајем његове докторске тезе. У оквиру овог истраживачка правца Владимир је објавио неколико публикација у међународним часописима, односно у зборницима радова са међународних конференција.

У оквиру овог истраживачког правца, Владимир Лончар се бавио рачунарством високих перформанси, посебно имплементацијом и оптимизацијом нумеричких алгоритама за решавање сложених математичких проблема. Један од задатака је била и имплементација паралелних нумеричких алгоритама везаних за проучавање Бозе-Ајнштајн кондензата са контактном и дипол-дипол интеракцијом. Владимир је развио више паралелних алгоритама за различите хардверске архитектуре, од класичних Intel процесора, до Nvidia графичких картица, који се могу извршавати како на једном рачунару, тако и на рачунарском кластеру. Развијао је и алгоритме за хетерогене рачунарске системе, као и хеуристичке методе за оптимизацију алгоритама за хетерогене системе. Користећи развијене алгоритме проучавао је утицај дипол-дипол интеракције на особине Бозе-Ајнштајн кондензата хрома и диспрозијума, као и формирање квантних дроплета приликом нагле промене контактне интеракције у јако диполарним кондензатима.

Владимир се бавио и визуелизацијом података добијених из нумеричких симулација. За ове потребе развио је механизме преко којих се подаци лако визуелизују током извршавања симулације, и преко којих се може управљати симулацијом, тзв. in situ визуелизација.

Од претходног избора у звање Владимир Лончар је објавио два рада категорије M21a, једно саопштење категорије M33 и два саопштења категорије M34. Један од радова категорије M21a је у Web of Science означен као Highly Cited Paper за период од објављивања до јуна 2016. године.

Computer Physics Communications 200 (2016) 406–410

Contents lists available at ScienceDirect

Computer Physics Communications

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

CUDA programs for solving the time-dependent dipolar Gross–Pitaevskii equation in an anisotropic trap

Vladimir Lončar [a,*], Antun Balaž [a], Aleksandar Bogojević [a], Srdjan Škrbić [b], Paulsamy Muruganandam [c], Sadhan K. Adhikari [d]

[a] Scientific Computing Laboratory, Institute of Physics Belgrade, University of Belgrade, Pregrevica 118, 11080 Belgrade, Serbia
[b] Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia
[c] School of Physics, Bharathidasan University, Palkalaiperur Campus, Tiruchirappalli – 620024, Tamil Nadu, India
[d] Instituto de Física Teórica, UNESP – Universidade Estadual Paulista, 01.140-70 São Paulo, São Paulo, Brazil

ARTICLE INFO

Article history:
Received 12 November 2015
Accepted 17 November 2015
Available online 17 December 2015

Keywords:
Bose–Einstein condensate
Dipolar atoms
Gross–Pitaevskii equation
Split-step Crank–Nicolson scheme
Real- and imaginary-time propagation
C program
GPU
CUDA program
Partial differential equation

ABSTRACT

In this paper we present new versions of previously published numerical programs for solving the dipolar Gross–Pitaevskii (GP) equation including the contact interaction in two and three spatial dimensions in imaginary and in real time, yielding both stationary and non-stationary solutions. New versions of programs were developed using CUDA toolkit and can make use of Nvidia GPU devices. The algorithm used is the same split-step semi-implicit Crank–Nicolson method as in the previous version (Kishor Kumar et al., 2015), which is here implemented as a series of CUDA kernels that compute the solution on the GPU. In addition, the Fast Fourier Transform (FFT) library used in the previous version is replaced by cuFFT library, which works on CUDA-enabled GPUs. We present speedup test results obtained using new versions of programs and demonstrate an average speedup of 12–25, depending on the program and input size.

New version program summary

Program title: DBEC-GP-CUDA package, consisting of: (i) imag2dXY-cuda, (ii) imag2dXZ-cuda, (iii) imag3d-cuda, (iv) real2dXY-cuda, (v) real2dXZ-cuda, (vi) real3d-cuda.

Catalogue identifier: AEWL_v2_0

Program Summary URL: http://cpc.cs.qub.ac.uk/summaries/AEWL_v2_0.html

Program obtainable from: CPC Program Library, Queen's University of 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.: 18297.

No. of bytes in distributed program, including test data, etc.: 128586.

Distribution format: tar.gz.

Programming language: CUDA C.

Computer: Any modern computer with Nvidia GPU with Compute Capability 2.0 or higher, with CUDA toolkit (compiler and runtime, with cuFFT library, minimum version 6.0) installed.

Operating system: Linux.

RAM: With provided example inputs, programs should run on a computer with 512 MB GPU RAM. There is no upper limit to amount of memory that can be used, as larger grid sizes require more memory, which scales as NX*NY or NX*NZ (in 2d) or NX*NY*NZ (in 3d). All programs require roughly the same amount of CPU and GPU RAM.

Number of processors used: One CPU core and one Nvidia GPU.

Classification: 2.9, 4.3, 4.12.

External routines/libraries: CUDA toolkit, version 6.0 or higher, with cuFFT library.

Catalogue identifier of previous version: AEWL_v1_0.

* Corresponding author.
E-mail addresses: vladimir.loncar@ipb.ac.rs (V. Lončar), antun.balaz@ipb.ac.rs (A. Balaž), aleksandar.bogojevic@ipb.ac.rs (A. Bogojević), srdjan.skrbic@dmi.uns.ac.rs (S. Škrbić), anand@cnld.bdu.ac.in (P. Muruganandam), adhikari@ift.unesp.br (S.K. Adhikari).

http://dx.doi.org/10.1016/j.cpc.2015.11.014
0010-4655/© 2015 Elsevier B.V. All rights reserved.

Journal reference of previous version: Comput. Phys. Commun. 195 (2015) 117.

Does the new version supersede the previous version?: No.

Nature of problem: These programs are designed to solve the time-dependent nonlinear partial differential Gross–Pitaevskii (GP) equation with contact and dipolar interactions in two or three spatial dimensions in a harmonic anisotropic trap. The GP equation describes the properties of a dilute trapped Bose–Einstein condensate.

Solution method: The time-dependent GP equation is solved by the split-step Crank–Nicolson method by discretizing in space and time. The discretized equation is then solved by propagation, in either imaginary or real time, over small time steps. The contribution of the dipolar interaction is evaluated by a Fourier transformation to momentum space using a convolution theorem. The method yields the solution of stationary and/or non-stationary problems.

Reasons for the new version: Previously published dipolar Fortran and C programs [1], based on earlier programs and algorithms for GP equation with the contact interaction [2], are already used within the ultra-cold atoms community [3]. However, they are sequential, and thus did not allow for use of the maximum computing performance modern computers can offer. For this reason we have explored possible ways to accelerate our programs. Detailed profiling revealed that the calculation of FFTs is the most computationally demanding part of our programs. Since using GPUs to compute FFTs with optimized libraries like the cuFFT can lead to much better performance, we have decided to parallelize our programs using Nvidia CUDA toolkit. Also, the massive parallelism offered by GPUs could be exploited to parallelize the nested loops our programs have. We have focused on 2d and 3d versions of our programs, as they perform enough computation to justify and require the use of massive parallelism.

Summary of revisions: Previous C programs in two or three spatial dimensions are parallelized using CUDA toolkit from Nvidia and named similarly, with "-cuda" suffix appended to their names. The structure of all programs is identical. Computationally most demanding functions performing time evolution (calcpsidd2, calcnu, calclux, calcluy, calcluz), normalization of the wave function (calcnorm), and calculation of physically relevant quantities (calcmuen, calcrms) were implemented as a series of CUDA kernels, which are executed on GPU. All kernels are implemented with grid-stride loops [4], which allow us to use the same kernel block sizes for all of our kernels. These block sizes can be changed in src/utils/cudautils.cuh, containing the optimal values for current Nvidia Tesla GPUs.

As before, CPU performs the initialization of variables and controls the flow of programs, offloading computation to GPU when needed. Because of the initialization, programs still require almost the same amount of CPU RAM as GPU RAM. Before any computation begins, relevant variables are copied to GPU, where they remain during computation, and only wave function array is returned back to CPU when it is required for writing output.

Parallelization with CUDA toolkit required some dynamically allocated arrays (tensors, matrices, or vectors) to become private for each GPU thread. This has caused an increase in the amount of used GPU memory, since the number of running threads on GPU is very large. Coupled with the fact that GPUs usually have smaller amount of RAM than CPU, this meant that our GPU versions of programs could be used for much smaller input in comparison to sequential versions. In order to fix this problem and reduce memory usage, our programs reuse temporary arrays as much as possible. Aside from allocation of complex tensor/matrix (for 3d or 2d case, respectively) in which we store wave function values, we allocate one complex tensor/matrix, and up to two double precision tensors/matrices, and reuse them for different purposes in computations. Allocated complex tensor/matrix is later also used as two double precision tensors/matrices, for other purposes. This required some reorganization of computation in several functions, mainly in calcmuen and calcpsidd2. In calcmuen we have reorganized computation to reuse temporary array and store partial derivatives in it, so instead of using three (in 3d) or two (in 2d) separate tensors/matrices for partial derivatives, we now use a single temporary tensor/matrix, which we also use for different purposes in other places in programs. In calcpsidd2 we have removed the use of additional temporary array that was only used in FFT computation, and also use real-to-complex and complex-to-real FFT transformations in place of complex-to-complex transformations of previous program versions. This change was possible because condensate density (input array for FFT) is purely real, and thus it exhibits Hermitian symmetry. Some FFT libraries, like the cuFFT used in these programs, can exploit this to reduce memory usage and provide better performance by calculating only non-redundant parts of the array. Additionally, programs can further reduce GPU RAM consumption by keeping the tensor/matrix used to store trap potential and dipolar potential in main RAM, configurable through POTMEM parameter in the input file. Setting value of POTMEM to 2 maximizes performance, and means that programs will allocate two separate tensors/matrices for storing trap potential and dipolar potential in GPU memory. This provides the best performance, but at the cost of a larger total memory consumption. If we set the value of POTMEM to 1, only one tensor/matrix will be allocated in GPU memory, to which trap potential and dipolar potential will be asynchronously copied from main memory when they are needed for computation. In this case, tensor/matrix will initially contain trap potential, which will be replaced with dipolar potential during execution of FFT in calcpsidd2, and replaced back with trap potential during inverse FFT. Finally, setting POTMEM to 0 will instruct the programs not to allocate any GPU memory for storing potentials and will instead use main memory, which GPU can access through slower PCI-Express bus. Figure 1 explains how memory is used and the possible values of POTMEM. We suggest using POTMEM value of 2 if memory permits, and using values of 1 or 0 if problem cannot fit into GPU memory. If POTMEM is not specified, programs will check if GPU memory is large enough to fit all variables and set POTMEM accordingly.

Time propagation functions calclux, calcluy, and calcluz have a recursive relation that makes them difficult to parallelize. In principle, recursive relations could be parallelized using a higher-order prefix sum algorithm [5] (also known as scan algorithm), but implementation of this would require multiple CUDA kernels [6]. Since recursive relations are in the innermost loop, launching of all required kernels would create a sizeable overhead. Also, the number of grid points in each dimension is usually not large enough to compensate that overhead. Therefore, we have chosen an approach that, instead of parallelizing the inner loop which has the recursive relation, we parallelize the outer loops, and each GPU thread computes the whole innermost loop. Since each GPU thread now requires its own array for storing Crank–Nicolson coefficients cbeta, we reuse existing temporary tensor/matrix for storing these values. Similar pattern of parallelizing outer loops was also used in calcnorm, calcrms, and calcmuen.

We tested our programs at the PARADOX supercomputing facility at the Scientific Computing Laboratory of the Institute of Physics Belgrade. Nodes used for testing had Intel Xeon E5-2670 CPUs with 32 GB of RAM and Nvidia Tesla M2090 GPU with 6 GB of RAM. Figure 2 shows the speedup obtained for six DBEC-GP-CUDA programs compared to their previous versions [1] executed on a single CPU core. Profiling reveals that the execution time is dominated by execution of FFTs and that the speedup varies significantly with changing of the grid size. This is due to FFT libraries used (FFTW in previous CPU version [1] and cuFFT in this version), which use different algorithms for different input array sizes. We thus conclude that the best performance can be achieved by experimenting with different grid sizes around the desired target.

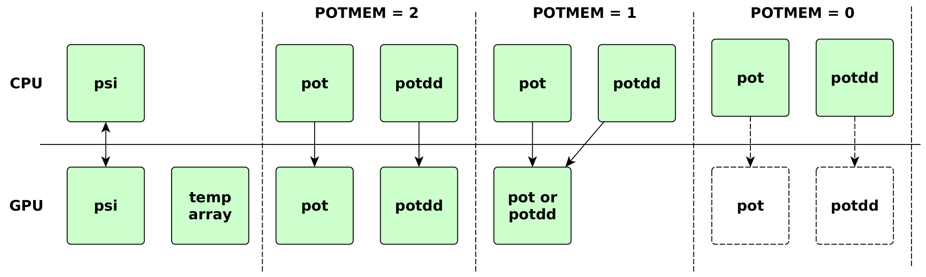

Fig. 1. Illustration of placement of relevant variables in CPU and GPU memory. CPU initializes its own wave function tensor/matrix (*psi*), trap potential (*pot*) and dipolar potential (*potdd*), which is copied to GPU memory. Depending on value of POTMEM variable, GPU will either allocate the same tensors/matrices for trap and dipolar potential (POTMEM = 2), allocate only one tensor/matrix and use it for different purposes (POTMEM = 1), or will map *pot* and *potdd* from CPU and not allocate extra memory on GPU (POTMEM = 0). Additionally, GPU allocates one complex tensor/matrix which is used for temporary data. This tensor/matrix is used either as a single complex tensor/matrix, or is divided into two double tensors/matrices which can then each contain the same number of elements as the complex tensor/matrix.

Restrictions:
Programs will only run on computers with Nvidia GPU card (Tesla or GeForce) with Compute Capability 2.0 or higher (Fermi architecture and newer) and with CUDA toolkit installed (version 6.0 or higher).

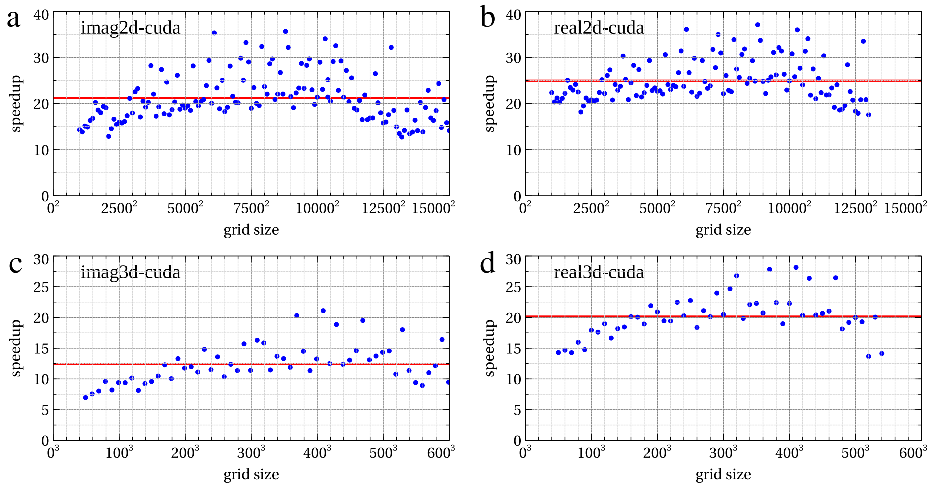

Fig. 2. Speedup in execution time of imag2dXY-cuda and imag2dXZ-cuda (top-left), real2dXY-cuda and real2dXZ-cuda (top-right), imag3d-cuda (bottom-left) and real3d-cuda (bottom-right) compared to the previous versions of programs [1] executed on a single CPU core. Solid red line represents average speedup obtained. We tested linear grid sizes starting from 50^3 in 3d and 1000^2 in 2d, up to the maximum that could fit in GPU memory, which was 600^3 for imag3d-cuda, 540^3 for real3d-cuda, 15000^2 for imag2dXY-cuda and imag2dXZ-cuda, and 13000^2 for real2dXY-cuda and real2dXZ-cuda. Note that the dispersion of data is due to the use of FFTW_ESTIMATE flag in library calls to FFTW in the CPU programs.

Unusual features of all programs:
As part of the memory usage optimizations, programs may slightly increase the number of spatial grid points in each dimension (NX, NY, NZ). This is due to FFT algorithms of cuFFT library that require additional memory to store temporary results. Our programs reuse already allocated memory to provide cuFFT with the temporary memory it requires, however, some problem sizes require much more memory, up to eight times more [7]. For instance, if the number of grid points in any dimension is a large prime number, cuFFT

uses an algorithm that requires eight times more memory than similarly sized power of two number. Adjustments of the number of grid points made in the programs ensure that cuFFT will not require such significantly increased additional memory. In case the programs perform the adjustments to grid size, this is reported in the output.

Additional comments:
This package consists of 6 programs, see Program title above. For the particular purpose of each program, please see descriptions below.

Running time:
Example inputs provided with the programs take less than one minute on Nvidia Tesla M2090 GPU.

Program summary (i)

Program title: imag2dXY-cuda.

Title of electronic files: imag2dXY-cuda.cu and imag2dXY-cuda.cuh.

Maximum RAM memory: No upper bound.

Programming language used: CUDA C.

Typical running time: Minutes on a medium PC.

Nature of physical problem: This program is designed to solve the time-dependent dipolar nonlinear partial differential GP equation in two space dimensions in an anisotropic harmonic trap. The GP equation describes the properties of a dilute trapped Bose–Einstein condensate.

Method of solution: The time-dependent GP equation is solved by the split-step Crank–Nicolson method by discretizing in space and time. The discretized equation is then solved by propagation in imaginary time over small time steps. The method yields the solution of stationary problems.

Program summary (ii)

Program title: imag2dXZ-cuda.

Title of electronic files: imag2dXZ-cuda.cu and imag2dXZ-cuda.cuh.

Maximum RAM memory: No upper bound.

Programming language used: CUDA C.

Typical running time: Minutes on a medium PC.

Nature of physical problem: This program is designed to solve the time-dependent dipolar nonlinear partial differential GP equation in two space dimensions in an anisotropic harmonic trap. The GP equation describes the properties of a dilute trapped Bose–Einstein condensate.

Method of solution: The time-dependent GP equation is solved by the split-step Crank–Nicolson method by discretizing in space and time. The discretized equation is then solved by propagation in imaginary time over small time steps. The method yields the solution of stationary problems.

Program summary (iii)

Program title: imag3d-cuda.

Title of electronic files: imag3d-cuda.cu and imag3d-cuda.cuh.

Maximum RAM memory: No upper bound.

Programming language used: CUDA C.

Typical running time: Tens of minutes on a medium PC.

Nature of physical problem: This program is designed to solve the time-dependent dipolar nonlinear partial differential GP equation in three space dimensions in an anisotropic harmonic trap. The GP equation describes the properties of a dilute trapped Bose–Einstein condensate.

Method of solution: The time-dependent GP equation is solved by the split-step Crank–Nicolson method by discretizing in space and time. The discretized equation is then solved by propagation in imaginary time over small time steps. The method yields the solution of stationary problems.

Program summary (iv)

Program title: real2dXY-cuda.

Title of electronic files: real2dXY-cuda.cu and real2dXY-cuda.cuh.

Maximum RAM memory: No upper bound.

Programming language used: CUDA C.

Typical running time: Tens of minutes on a good workstation.

Unusual feature: If NSTP $= 0$, the program requires and reads the file imag2dXY-den.txt, generated by executing imag2dXY-cuda with the same grid size parameters.

Nature of physical problem: This program is designed to solve the time-dependent dipolar nonlinear partial differential GP equation in two space dimensions in an anisotropic harmonic trap. The GP equation describes the properties of a dilute trapped Bose–Einstein condensate.

Method of solution: The time-dependent GP equation is solved by the split-step Crank–Nicolson method by discretizing in space and time. The discretized equation is then solved by propagation in real time over small time steps. The method yields the solution of dynamical problems.

Program summary (v)

Program title: real2dXZ-cuda.

Title of electronic files: real2dXZ-cuda.cu and real2dXZ-cuda.cuh.

Maximum RAM memory: No upper bound.

Programming language used: CUDA C.

Typical running time: Tens of minutes on a good workstation.

Unusual feature: If NSTP = 0, the program requires and reads the file imag2dXZ-den.txt, generated by executing imag2dXZ-cuda with the same grid size parameters.

Nature of physical problem: This program is designed to solve the time-dependent dipolar nonlinear partial differential GP equation in two space dimensions in an anisotropic harmonic trap. The GP equation describes the properties of a dilute trapped Bose–Einstein condensate.

Method of solution: The time-dependent GP equation is solved by the split-step Crank–Nicolson method by discretizing in space and time. The discretized equation is then solved by propagation in real time over small time steps. The method yields the solution of dynamical problems.

Program summary (vi)

Program title: real3d-cuda.

Title of electronic files: real3d-cuda.cu and real3d-cuda.cuh.

Maximum RAM memory: No upper bound.

Programming language used: CUDA C.

Typical running time: Tens of minutes on a good workstation.

Unusual feature: If NSTP = 0, the program requires and reads the file imag3d-den.txt, generated by executing imag3d-cuda with the same grid size parameters.

Nature of physical problem: This program is designed to solve the time-dependent dipolar nonlinear partial differential GP equation in three space dimensions in an anisotropic harmonic trap. The GP equation describes the properties of a dilute trapped Bose–Einstein condensate.

Method of solution: The time-dependent GP equation is solved by the split-step Crank–Nicolson method by discretizing in space and time. The discretized equation is then solved by propagation in real time over small time steps. The method yields the solution of dynamical problems.

Acknowledgments

V. L., A. B., A. B., and S. Š acknowledge support by the Ministry of Education, Science, and Technological Development of the Republic of Serbia under projects ON171017, III43007, ON174023, and IBEC, and by the DAAD - German Academic and Exchange Service under project IBEC. P.M. acknowledges support by the Science and Engineering Research Board, Department of Science and Technology, Government of India under project No. EMR/2014/000644. S.K.A. acknowledges support by the CNPq of Brazil under project 303280/2014-0, and by the FAPESP of Brazil under project 2012/00451-0. Numerical simulations were run on the PARADOX supercomputing facility at the Scientific Computing Laboratory of the Institute of Physics Belgrade, supported in part by the Ministry of Education, Science, and Technological Development of the Republic of Serbia under project ON171017.

References

[1] R. Kishor Kumar, L. E. Young-S., D. Vudragović, A. Balaž, P. Muruganandam, and S. K. Adhikari, Fortran and C programs for the time-dependent dipolar Gross–Pitaevskii equation in an anisotropic trap, Comput. Phys. Commun. **195** (2015) 117.

[2] P. Muruganandam and S. K. Adhikari, Comput. Phys. Commun. **180** (2009) 1888;
D. Vudragović, I. Vidanović, A. Balaž, P. Muruganandam, and S. K. Adhikari, Comput. Phys. Commun. **183** (2012) 2021;
P. Muruganandam and S. K. Adhikari, J. Phys. B: At. Mol. Opt. Phys. **36** (2003) 2501.

[3] R. Kishor Kumar, P. Muruganandam, and B. A. Malomed, J. Phys. B: At. Mol. Opt. Phys. **46** (2013) 175302;
S. K. Adhikari, Bright dipolar Bose–Einstein-condensate soliton mobile in a direction perpendicular to polarization, Phys. Rev. A **90** (2014) 055601;
S. K. Adhikari, Stable matter-wave solitons in the vortex core of a uniform condensate, J. Phys. B: At. Mol. Opt. Phys. **48** (2015) 165303;
S. K. Adhikari, Stable spatial and spatiotemporal optical soliton in the core of an optical vortex, Phys. Rev. E **92** (2015) 042926;
T. Khellil, A. Balaž, and A. Pelster, Dirty bosons in a quasi-one-dimensional harmonic trap, e-print arXiv:1510.04985 (2015).

[4] M. Harris, CUDA Pro Tip: Write Flexible Kernels with Grid-Stride Loops, Parallel Forall Blog, http://devblogs.nvidia.com/parallelforall/cuda-pro-tip-write-flexible-kernels-grid-stride-loops/ (2013).

[5] G. E. Blelloch, Prefix Sums and Their Applications, In J. H. Reif (Ed.), Synthesis of Parallel Algorithms, Morgan Kaufmann, San Francisco (1990).

[6] M. Harris, Parallel Prefix Sum (Scan) with CUDA, EECS 570 Parallel Computer Architecture Course, University of Michigan, http://www.eecs.umich.edu/courses/eecs570/hw/parprefix.pdf (2007).

[7] cuFFT, CUDA API References, CUDA Toolkit Documentation v7.5, http://docs.nvidia.com/cuda/cufft/ (2015).

© 2015 Elsevier B.V. All rights reserved.

Computer Physics Communications 209 (2016) 190–196

Contents lists available at ScienceDirect

Computer Physics Communications

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

OpenMP, OpenMP/MPI, and CUDA/MPI C programs for solving the time-dependent dipolar Gross–Pitaevskii equation

Vladimir Lončar[a,*], Luis E. Young-S.[b,c], Srdjan Škrbić[d], Paulsamy Muruganandam[e], Sadhan K. Adhikari[c], Antun Balaž[a]

[a] *Scientific Computing Laboratory, Center for the Study of Complex Systems, Institute of Physics Belgrade, University of Belgrade, Pregrevica 118, 11080 Belgrade, Serbia*

[b] *Departamento de Ciencias Básicas, Universidad Santo Tomás, 150001 Tunja, Boyacá, Colombia*

[c] *Instituto de Física Teórica, UNESP—Universidade Estadual Paulista, 01.140-70 São Paulo, São Paulo, Brazil*

[d] *Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia*

[e] *School of Physics, Bharathidasan University, Palkalaiperur Campus, Tiruchirappalli – 620024, Tamil Nadu, India*

A R T I C L E I N F O

Article history:
Received 13 July 2016
Accepted 15 July 2016
Available online 6 September 2016

Keywords:
Bose–Einstein condensate
Dipolar atoms
Gross–Pitaevskii equation
Split-step Crank–Nicolson scheme
C program
OpenMP
GPU
CUDA program
MPI

A B S T R A C T

We present new versions of the previously published C and CUDA programs for solving the dipolar Gross–Pitaevskii equation in one, two, and three spatial dimensions, which calculate stationary and non-stationary solutions by propagation in imaginary or real time. Presented programs are improved and parallelized versions of previous programs, divided into three packages according to the type of parallelization. First package contains improved and threaded version of sequential C programs using OpenMP. Second package additionally parallelizes three-dimensional variants of the OpenMP programs using MPI, allowing them to be run on distributed-memory systems. Finally, previous three-dimensional CUDA-parallelized programs are further parallelized using MPI, similarly as the OpenMP programs. We also present speedup test results obtained using new versions of programs in comparison with the previous sequential C and parallel CUDA programs. The improvements to the sequential version yield a speedup of 1.1–1.9, depending on the program. OpenMP parallelization yields further speedup of 2–12 on a 16-core workstation, while OpenMP/MPI version demonstrates a speedup of 11.5–16.5 on a computer cluster with 32 nodes used. CUDA/MPI version shows a speedup of 9–10 on a computer cluster with 32 nodes.

New version program summary

Program Title: DBEC-GP-OMP-CUDA-MPI: (1) DBEC-GP-OMP package: (i) imag1dX-th, (ii) imag1dZ-th, (iii) imag2dXY-th, (iv) imag2dXZ-th, (v) imag3d-th, (vi) real1dX-th, (vii) real1dZ-th, (viii) real2dXY-th, (ix) real2dXZ-th, (x) real3d-th; (2) DBEC-GP-MPI package: (i) imag3d-mpi, (ii) real3d-mpi; (3) DBEC-GP-MPI-CUDA package: (i) imag3d-mpicuda, (ii) real3d-mpicuda.

Program Files doi: http://dx.doi.org/10.17632/j3z9z379m8.1

Licensing provisions: Apache License 2.0

Programming language: OpenMP C; CUDA C.

Computer: DBEC-GP-OMP runs on any multi-core personal computer or workstation with an OpenMP-capable C compiler and FFTW3 library installed. MPI versions are intended for a computer cluster with a recent MPI implementation installed. Additionally, DBEC-GP-MPI-CUDA requires CUDA-aware MPI implementation installed, as well as that a computer or a cluster has Nvidia GPU with Compute Capability 2.0 or higher, with CUDA toolkit (minimum version 7.5) installed.

Number of processors used: All available CPU cores on the executing computer for OpenMP version, all available CPU cores across all cluster nodes used for OpenMP/MPI version, and all available Nvidia GPUs across all cluster nodes used for CUDA/MPI version.

Journal reference of previous version: Comput. Phys. Commun. **195** (2015) 117; *ibid.* **200** (2016) 406.

Does the new version supersede the previous version?: Not completely. OpenMP version does supersede previous AEWL_v1_0 version, while MPI versions do not supersede previous versions and are meant for execution on computer clusters and multi-GPU workstations.

* Corresponding author.
E-mail addresses: vladimir.loncar@ipb.ac.rs (V. Lončar), luisevery@gmail.com (L.E. Young-S.), srdjan.skrbic@dmi.uns.ac.rs (S. Škrbić), anand@cnld.bdu.ac.in (P. Muruganandam), adhikari@ift.unesp.br (S.K. Adhikari), antun.balaz@ipb.ac.rs (A. Balaž).

http://dx.doi.org/10.1016/j.cpc.2016.07.029
0010-4655/© 2016 Elsevier B.V. All rights reserved.

Nature of problem: These programs are designed to solve the time-dependent nonlinear partial differential Gross–Pitaevskii (GP) equation with contact and dipolar interaction in a harmonic anisotropic trap. The GP equation describes the properties of a dilute trapped Bose–Einstein condensate. OpenMP package contains programs for solving the GP equation in one, two, and three spatial dimensions, while MPI packages contain only three-dimensional programs, which are computationally intensive or memory demanding enough to require such level of parallelization.

Solution method: The time-dependent GP equation is solved by the split-step Crank–Nicolson method by discretizing in space and time. The discretized equation is then solved by propagation, in either imaginary or real time, over small time steps. The contribution of the dipolar interaction is evaluated by a Fourier transformation to momentum space using a convolution theorem. MPI parallelization is done using the domain decomposition. The method yields the solution of stationary and/or non-stationary problems.

Reasons for the new version: Previously published C and Fortran programs [1] for solving the dipolar GP equation are sequential in nature and do not exploit the multiple cores or CPUs found in typical modern computers. A parallel implementation exists, using Nvidia CUDA [2], and both versions are already used within the ultra-cold atoms community [3]. However, CUDA version requires special hardware, which limits its usability. Furthermore, many researchers have access to high performance computer clusters, which could be used to either further speed up the computation, or to work with problems which cannot fit into a memory of a single computer. In light of these observations, we have parallelized all programs using OpenMP, and then extended the parallelization of three-dimensional programs using MPI to distributed-memory clusters. Since the CUDA implementation uses the same algorithm, and thus has the same structure and flow, we have applied the same data distribution scheme to provide the distributed-memory CUDA/MPI implementation of three-dimensional programs.

Summary of revisions:

Package DBEC-GP-OMP: Previous serial C programs [1] are here improved and then parallelized using OpenMP (package DBEC-GP-OMP). The main improvement consists of switching to real-to-complex (R2C) Fourier transform, which is possible due to the fact that input of the transform is purely real. In this case the result of the transform has Hermitian symmetry, where one half of the values are complex conjugates of the other half. The fast Fourier transformation (FFT) libraries we use can exploit this to compute the result faster, using half the memory.

To parallelize the programs, we have used OpenMP with the same approach as described in [4], and extended the parallelization routines to include the computation of the dipolar term. The FFT, used in computation of the dipolar term, was also parallelized in a straightforward manner, by using the built-in support for OpenMP in FFTW3 library [5]. With the introduction of multiple threads memory usage has increased, driven by the need to have some variables private to each thread. To reduce the memory consumed, we resorted to using techniques similar to the ones used in our CUDA implementation [2], i.e., we have reduced the memory required for FFT by exploiting the aforementioned R2C FFT, and reused the memory with pointer aliases whenever possible.

Package DBEC-GP-MPI: Next step in the parallelization (package DBEC-GP-MPI) was to extend the programs to run on distributed-memory systems, i.e., on computer clusters using domain decomposition with MPI programming paradigm. We chose to use the newly-implemented threaded versions of the programs as the starting point. Alternatively, we could have used serial versions, and attempt a pure MPI parallelization, however we have found that OpenMP-parallelized routines better exploit the data locality and thus outperform the pure MPI implementation. Therefore, our OpenMP/MPI-parallelized programs are intended to run one MPI process per cluster node, and each process would spawn the OpenMP threads as needed on its cluster node. Note that this is not a requirement, and users may run more than one MPI process per node, but we advise against it due to performance reasons. With the suggested execution strategy (one MPI process per cluster node, each spawning as many threads as CPU cores available), OpenMP threads perform most of the computation, and MPI is used for data exchanges between processes.

There are numerous ways to distribute the data between MPI processes, and we decided to use a simple one-dimensional data distribution, also known as slab decomposition. Data is distributed along the first (slowest changing) dimension, which corresponds to NX spatial dimension in our programs (see Fig. 1). Each process is assigned a different portion of the NX dimension, and contains the entire NY and NZ spatial dimensions locally. This allows each process to perform computation on those two dimensions in the same way as before, without any data exchanges. In case the computation requires whole NX dimension to be local to each process, we transpose the data, and after the computation, we transpose the data back.

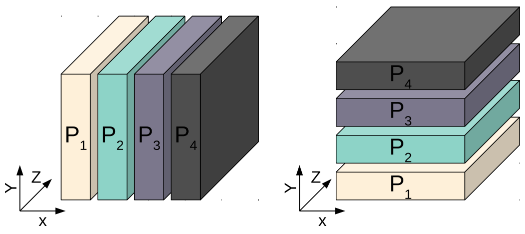

Fig. 1. Illustration of data distribution between MPI processes. On the left, the data are distributed along the NX dimension, while on the right the same data are redistributed along the NY dimension.

Transpose routine can be implemented in many ways using MPI, most commonly using `MPI_Alltoall` function, or using transpose routines from external libraries, like FFTW3 [5] or 2DECOMP&FFT [6]. Since we already rely on FFTW3 library for FFT, we have utilized its dedicated transpose interface to perform the necessary transformations. To speed up transpose operation, we do not perform full transposition of data, but rather leave it *locally* transposed. That is, we transform from local_NX × NY × NZ, stored in row-major order, to NX × local_NY × NZ in row-major order (where local_NX = NX / number_of_processes, and equivalently for local_NY). This approach has an additional benefit that we do not have to make significant changes in the way array elements are processed, and in most cases we only have to adjust the loop limit of the non-local dimension.

Package DBEC-GP-MPI-CUDA: The aforementioned data distribution scheme can be also applied to the CUDA version of programs [2]. However, there is no support for CUDA in FFTW3, and cuFFT (used in CUDA programs for FFT) does not provide equivalent MPI or transpose interface. Instead, we developed our own transpose routines, and used them in FFT computation. One example of manual implementation of transpose routines is shown in Ref. [7], and while we could readily use the same code, we wanted to have the same result as when using FFTW3. To achieve this, we use the same basic principle as in Ref. [7], first we create a custom MPI data type that maps to portions of the data to be exchanged, followed by an all-to-all communication to exchange the data between processes, see Fig. 2 for details.

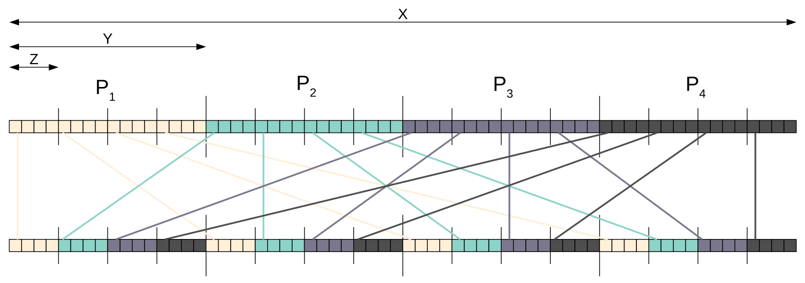

Fig. 2. Example of a transpose routine of a 4 × 4 × 4 data between four MPI processes. Initially, all processes have 1/4 of the NX dimension, and whole NY and NZ dimensions. After transposing, each process has full NX and NZ dimensions, and 1/4 of the NY dimension.

The implemented transpose routines are also used to compute a distributed-memory FFT, performed over all MPI processes. To divide the computation of a multidimensional FFT, in our case three-dimensional, we use a well-known row–column algorithm. The basic idea of the algorithm is perhaps best explained on a two-dimensional FFT of $N \times M$ data, stored in row-major order, illustrated in Fig. 3. First the N one-dimensional FFTs of length M are performed (along the row of data), followed by a transpose, after which data are stored as $M \times N$ in row-major format. Now M FFTs of length N can be performed along what used to be a column of original data, but are stored as rows after transposing. Finally, an optional transpose can be performed to return the data in their original $N \times M$ form. In three dimensions, we can perform a two-dimensional FFT, transpose the data, and perform the FFT along the third dimension. This algorithm can be easily adapted for distributed memory systems. We use advanced cuFFT interface for local computation of FFT, and use our transpose routine to redistribute the data.

Note that DBEC-GP-MPI-CUDA programs can be easily modified to work on a single workstation with multiple GPU cards, or a computer cluster with multiple GPU cards per node. In that case, for each GPU card a separate MPI process should be launched and the programs should be modified to assign a separate GPU card for processes on the same cluster node.

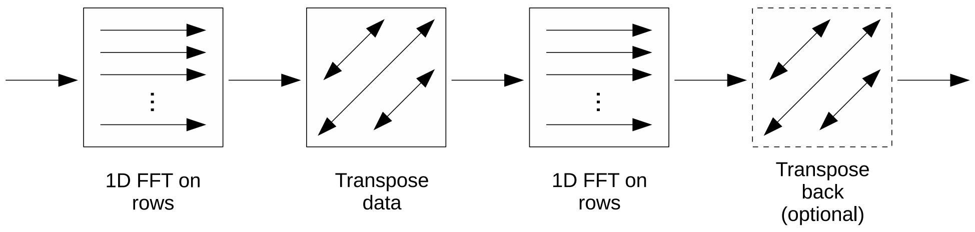

Fig. 3. Illustration of four stages of row–column FFT algorithm. The last transpose operation may be omitted, and often yields better performance.

MPI output format: Given that the distributed memory versions of the programs can be used for much larger grid sizes, the output they produce (i.e., the density profiles) can be much larger and difficult to handle. To alleviate this problem somewhat, we have switched to a binary output instead of the textual. This allowed us to reduce the size of files, while still retaining precision. All MPI processes will write the output to the same file, at the corresponding offset, relieving the user of the task of combining the files. The binary output can be subsequently converted to textual, for example by using `hexdump` command on UNIX-like systems. We have developed a simple script which converts the output from binary to textual format and included it in the software package.

Testing results: We have tested all programs on the PARADOX supercomputing facility at the Scientific Computing Laboratory of the Institute of Physics Belgrade. Nodes used for testing had two Intel Xeon E5-2670 CPUs (with a total of 2 × 8 = 16 CPU cores) with 32 GB of RAM and one Nvidia Tesla M2090 GPU with 6 GB of RAM, each connected by Infiniband QDR interconnect. The presented results are obtained for arbitrary grid sizes, which are not tailored to maximize performance of the programs. We also stress that execution times and speedups reported here are calculated for critical parallelized parts of the programs performing iterations over imaginary or real time steps, and they exclude time spent on initialization (threads initialization, MPI environment, allocation/deallocation of memory, creating/destroying FFTW plans, I/O operations). As a part of its output, each program separately prints initialization time and time spent on iterations for GP propagation. The latter time is used to calculate a speedup, as a speedup obtained this way does not depend on the number of iterations and is more useful for large numbers of iterations.

The testing of OpenMP versions of programs DBEC-GP-OMP was performed with the number of threads varying from 1 to 16. Table 1 and Fig. 4 show the obtained absolute wall-clock times, speedups, and scaling efficiencies, as well as comparison with the previous serial version of programs [1]. As we can see from the table, improvements in the FFT routine used already yield a speedup of 1.3 to 1.9 for single-threaded ($T = 1$) 2d and 3d programs compared to the previous serial programs, and somewhat smaller speedup for 1d programs, 1.1 to 1.3. The use of additional threads brings about further speedup of 2 to 2.5 for 1d programs, and 9 to 12 for 2d and 3d programs. From Fig. 4 we see that for 1d programs, although speedup increases with the number of threads used, the efficiency decreases due to insufficient size of the problem, and one can achieve almost maximal value of speedup already with $T = 4$ threads, while still keeping the efficiency around 50%. We also see, as expected, that speedup and efficiency of 2d and 3d programs behave quite well as we increase the numbers of threads. In particular, we note that the efficiency is always above 60%, making the use of all available CPU cores worthwhile.

Table 1
Wall-clock execution times of DBEC-GP-OMP programs compiled with Intel's icc compiler, compared to the execution times of previously published serial versions. The execution times given here are for 1000 iterations (in seconds, excluding initialization and input/output operations, as reported by each program) with grid sizes: 10^5 for 1d programs, $10^4 \times 10^4$ for 2d programs, and $480 \times 480 \times 480$ for 3d programs. Columns $T = 1$, $T = 2$, $T = 4$, $T = 8$, and $T = 16$ correspond to the number of threads used, while the last column shows the obtained speedup with 16 OpenMP threads ($T = 16$) compared to one OpenMP thread ($T = 1$). Note that the reduction in the execution time is not solely due to the introduction of multiple threads, as the improvements in the FFT routine used also have noticeable impact. This is most evident when comparing execution times of serial versions to OpenMP versions with one thread. Execution times and speedups of imag1dZ-th, real1dZ-th, imag2dXZ-th, and real2dXZ-th (not reported here) are similar to those of imag1dX-th, real1dX-th, imag2dXY-th, and real2dXY-th, respectively.

	Serial [1]	$T = 1$	$T = 2$	$T = 4$	$T = 8$	$T = 16$	Speedup
imag1dX-th	9.1	7.1	4.7	3.4	2.9	2.8	2.5
real1dX-th	15.2	14.2	10.5	8.2	7.3	7.2	2.0
imag2dXY-th	13657	7314	4215	2159	1193	798	9.2
real2dXY-th	17281	11700	6417	3271	1730	1052	11.1
imag3d-th	16064	9353	5201	2734	1473	888	10.5
real3d-th	22611	17496	9434	4935	2602	1466	11.9

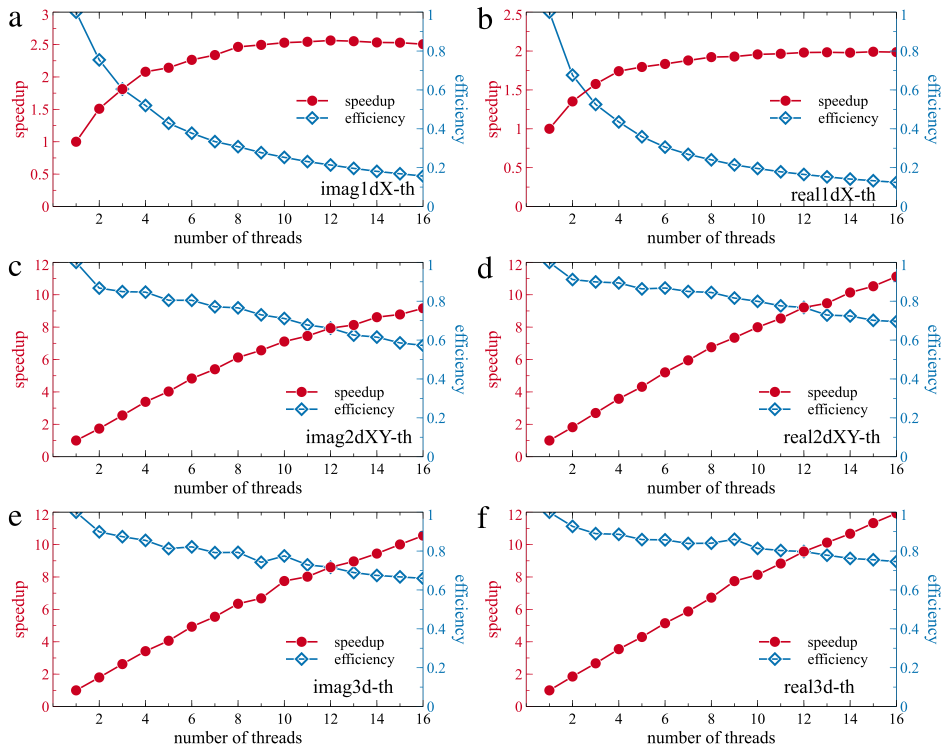

Fig. 4. Speedup in the execution time and scaling efficiency of DBEC-GP-OMP programs compared to single-threaded runs: (a) imag1dX-th, (b) real1dX-th, (c) imag2dXY-th, (d) real2dXY-th, (e) imag3d-th, (f) real3d-th. Scaling efficiency is calculated as a fraction of the obtained speedup compared to a theoretical maximum. Grid sizes used for testing are the same as in Table 1. Speedups and efficiencies of imag1dZ-th, real1dZ-th, imag2dXZ-th, and real2dXZ-th (not reported here) are similar to those of imag1dX-th, real1dX-th, imag2dXY-th, and real2dXY-th, respectively.

For testing of MPI versions we have used a similar methodology to measure the strong scaling performance. For OpenMP/MPI programs DBEC-GP-MPI, the obtained wall-clock times are shown in Table 2, together with the corresponding wall-clock times for the OpenMP programs DBEC-GP-OMP that served as a baseline to calculate speedups. The testing was done for varying number of cluster nodes, from 4 to 32, and the measured speedup ranged from 11 to 16.5. The corresponding graphs of speedups and efficiencies are shown in Fig. 5, where we can see that the speedup grows linearly with the number of nodes used, while the efficiency remains mostly constant in the range between 40% and 60%, thus making the use of OpenMP/MPI programs highly advantageous for problems with large grid sizes.

Table 2
Wall-clock execution times of DBEC-GP-MPI programs compiled with mpicc compiler from OpenMPI implementation of MPI, backed by Intel's icc compiler, compared to the execution times of OpenMP (DBEC-GP-OMP) versions on a single-node ($T = 16, N = 1$). The execution times given here are for 1000 iterations (in seconds, excluding initialization and input/output operations, as reported by each program) with the grid size $480 \times 480 \times 500$. Columns $N = 4$, $N = 8$, $N = 16$, $N = 24$, and $N = 32$ correspond to the number of cluster nodes used (each with $T = 16$ threads), while the last column shows the obtained speedup with $N = 32$ nodes compared to single-node runs.

	OpenMP	$N = 4$	$N = 8$	$N = 16$	$N = 24$	$N = 32$	Speedup
imag3d-mpi	1124	653	352	167	128	96	11.5
real3d-mpi	2140	979	513	277	220	129	16.5

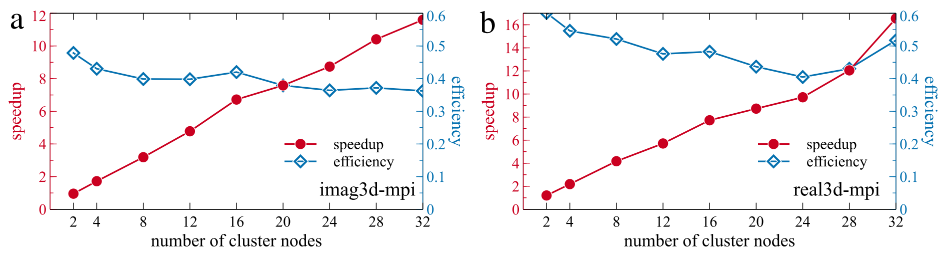

Fig. 5. Speedup in the execution time and scaling efficiency of DBEC-GP-MPI programs compared to single-node OpenMP runs: (a) imag3d-mpi, (b) real3d-mpi. Scaling efficiency is calculated as a fraction of the obtained speedup compared to a theoretical maximum. Grid size used for testing is the same as in Table 2.

For CUDA/MPI programs DBEC-GP-MPI-CUDA we observe similar behavior in Table 3 and in Fig. 6. The obtained speedup with $N = 32$ nodes here ranges from 9 to 10, with the efficiency between 30% and 40%. While the efficiency is slightly lower than in the case of OpenMP/MPI programs, which could be expected due to a more complex memory hierarchy when dealing with the multi-GPU system distributed over many cluster nodes, the speedup still grows linearly and makes CUDA/MPI programs ideal choice for use on GPU-enabled computer clusters. Additional benefit of using these programs is their low CPU usage (up to one CPU core), allowing for the possibility that same cluster nodes are used for other CPU-intensive simulations.

Table 3
Wall-clock execution times of DBEC-GP-MPI-CUDA programs compiled with Nvidia's nvcc compiler, with CUDA-aware OpenMPI implementation of MPI, backed by Intel's icc compiler, compared to the execution times of previous CUDA [2] versions on a single-node with one GPU card ($N = 1$). The execution times given here are for 1000 iterations (in seconds, excluding initialization and input/output operations, as reported by each program) with the grid size $480 \times 480 \times 250$. Columns $N = 4$, $N = 8$, $N = 16$, $N = 24$, and $N = 32$ correspond to the number of cluster nodes used (each with one GPU card), while the last column shows the obtained speedup with $N = 32$ nodes compared to single-node runs.

	CUDA [2]	$N = 4$	$N = 8$	$N = 16$	$N = 24$	$N = 32$	Speedup
imag3d-mpicuda	579	447	212	103	71	61	9.5
real3d-mpicuda	800	619	295	142	96	80	9.9

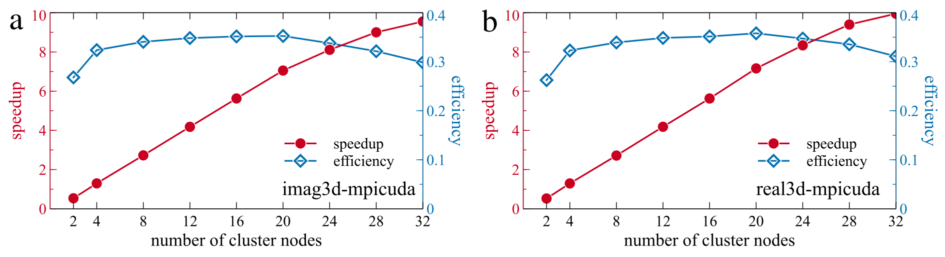

Fig. 6. Speedup in the execution time and scaling efficiency of DBEC-GP-MPI-CUDA programs compared to single-node runs of previous CUDA programs [2]: (a) imag3d-mpicuda, (b) real3d-mpicuda. Scaling efficiency is calculated as a fraction of the obtained speedup compared to a theoretical maximum. Grid size used for testing is the same as in Table 3.

The introduction of distributed transposes of data creates some overhead, which negatively impacts scaling efficiency. This is more evident in the CUDA/MPI version, as the transpose algorithm is inferior to the one provided by FFTW3. In our tests, both MPI versions of programs failed to achieve speedup on less than 4 nodes, due to the introduction of the transpose routines. We therefore recommend using MPI versions only on 4 or more cluster nodes.

The MPI versions are highly dependent not only on the configuration of the cluster, mainly on the speed of interconnect, but also on the distribution of processes and threads, NUMA configuration, etc. We recommend that users experiment with several different configurations to achieve the best performance. The results presented are obtained without extensive tuning, with the aim to show the base performance.

Finally, we note that the best performance can be achieved by evenly distributing the workload among the MPI processes and OpenMP threads, and by using grid sizes which are optimal for FFT. In particular, the programs in DBEC-GP-OMP package have the best performance if NX, NY, and NZ are divisible by the number of OpenMP threads used. Similarly, for DBEC-GP-MPI programs the best performance is achieved if NX and NY are divisible by a product of the number of MPI processes and the number of OpenMP threads used. For DBEC-GP-MPI-CUDA programs, the best performance is achieved if NX and NY are divisible by a product of the number of MPI processes and the number of Streaming Multiprocessors (SM) in the GPU used. For all three packages, the best FFT performance is obtained if NX, NY and NZ can be expressed as $2^a 3^b 5^c 7^d 11^e 13^f$, where e and f are either 0 or 1, and the other exponents are non-negative integer numbers [8].

Additional comments, restrictions, and unusual features: MPI programs require that grid size (controlled by input parameters NX, NY and NZ) can be evenly distributed between the processes, i.e., that NX and NY are divisible by the number of MPI processes. Since the data is never distributed along the NZ dimension, there is no such requirement on NZ. Programs will test if these conditions are met, and inform the user if not (by reporting an error). Additionally, MPI versions of CUDA programs require CUDA-aware MPI implementation. This allows the MPI runtime to directly access GPU memory pointers and avoid having to copy the data to main RAM. List of CUDA-aware MPI implementations can be found in Ref. [9].

Acknowledgments

V.L., S.Š. and A.B. acknowledge support by the Ministry of Education, Science, and Technological Development of the Republic of Serbia under projects ON171017, OI1611005, and III43007, as well as SCOPES project IZ74Z0-160453. L.E. Y.-S. acknowledges support by the FAPESP of Brazil under project 2012/21871-7 and 2014/16363-8. P.M. acknowledges support by the Science and Engineering Research Board, Department of Science and Technology, Government of India under project No. EMR/2014/000644. S.K.A. acknowledges support by the CNPq of Brazil under project 303280/2014-0, and by the FAPESP of Brazil under project 2012/00451-0.

References:

[1] R. Kishor Kumar, L. E. Young-S., D. Vudragović, A. Balaž, P. Muruganandam, and S. K. Adhikari, Comput. Phys. Commun. **195** (2015) 117.

[2] V. Lončar, A. Balaž, A. Bogojević, S. Škrbić, P. Muruganandam, S. K. Adhikari, Comput. Phys. Commun. **200** (2016) 406.

[3] R. Kishor Kumar, P. Muruganandam, and B. A. Malomed, J. Phys. B: At. Mol. Opt. Phys. **46** (2013) 175302;
H. Al-Jibbouri, I. Vidanović, A. Balaž, and A. Pelster, J. Phys. B: At. Mol. Opt. Phys. **46** (2013) 065303;
R. R. Sakhel, A. R. Sakhel, and H. B. Ghassib, J. Low Temp. Phys. **173** (2013) 177;
B. Nikolić, A. Balaž, and A. Pelster, Phys. Rev. A **88** (2013) 013624;
X. Antoine and R. Duboscq, Comput. Phys. Commun. **185** (2014) 2969;
J. Luo, Commun. Nonlinear Sci. Numer. Simul. **19** (2014) 3591;
K.-T. Xi, J. Li, and D.-N. Shi, Physica B **436** (2014) 149;
S. K. Adhikari, Phys. Rev. A **90** (2014) 055601;
M. C. Raportaru, J. Jovanovski, B. Jakimovski, D. Jakimovski, and A. Mishev, Rom. J. Phys. **59** (2014) 677;
A. I. Nicolin, A. Balaž, J. B. Sudharsan, and R. Radha, Rom. J. Phys. **59** (2014) 204;
A. Balaž, R. Paun, A. I. Nicolin, S. Balasubramanian, and R. Ramaswamy, Phys. Rev. A **89** (2014) 023609;
A. I. Nicolin and I. Rata, High-Performance Computing Infrastructure for South East Europe's Research Communities: Results of the HP-SEE User Forum 2012, in Springer Series: Modeling and Optimization in Science and Technologies **2** (2014) 15;
S. K. Adhikari, J. Phys. B: At. Mol. Opt. Phys. **48** (2015) 165303;
S. K. Adhikari, Phys. Rev. E **92** (2015) 042926;
T. Khellil and A. Pelster, arXiv:1512.04870 (2015);
H. L. C. Couto and W. B. Cardoso, J. Phys. B: At. Mol. Opt. Phys. **48** (2015) 025301;
R. R. Sakhel, A. R. Sakhel, and H. B. Ghassib, Physica B **478** (2015) 68;
L. Salasnich and S. K. Adhikari, Acta Phys. Pol. A **128** (2015) 979;
X. Antoine and R. Duboscq, Lecture Notes Math. **2146** (2015) 49;
E. Chiquillo, J. Phys. A: Math. Theor. **48** (2015) 475001;
S. Sabari, C. P. Jisha, K. Porsezian, and V. A. Brazhnyi, Phys. Rev. E **92** (2015) 032905;
W. Wen, T. K. Shui, Y. F. Shan, and C. P. Zhu, J. Phys. B: At. Mol. Opt. Phys. **48** (2015) 175301;
P. Das and P. K. Panigrahi, Laser Phys. **25** (2015) 125501;
Y. S. Wang, S. T. Ji, Y. E. Luo, and Z. Y. Li, J. Korean. Phys. Soc. **67** (2015) L1504;
A. I. Nicolin, M. C. Raportaru, and A. Balaž, Rom. Rep. Phys. **67** (2015) 143;
V. S. Bagnato, D. J. Frantzeskakis, P. G. Kevrekidis, B. A. Malomed, and D. Mihalache, Rom. Rep. Phys. **67** (2015) 5;
J. B. Sudharsan, R. Radha, H. Fabrelli, A. Gammal, and B. A. Malomed, Phys. Rev. A **92** (2015) 053601;
K.-T. Xi, J. Li, and D.-N. Shi, Physica B **459** (2015) 6;
E. J. M. Madarassy and V. T. Toth, Phys. Rev. D **91** (2015) 044041;
F. I. Moxley III, T. Byrnes, B. Ma, Y. Yan, and W. Dai, J. Comput. Phys. **282** (2015) 303;
D. Novoa, D. Tommasini, J. A. Nóvoa-López, Phys. Rev. E **91** (2015) 012904;

Y. H. Wang, A. Kumar, F. Jendrzejewski, R. M. Wilson, M. Edwards, S. Eckel, G. K. Campbell, and C. W. Clark, New J. Phys. **17** (2015) 125012;
T. Khellil, A. Balaž, and A. Pelster, New J. Phys. **18** (2016) 063003;
T. Khellil and A. Pelster, J. Stat. Mech.-Theory Exp. (2016) 063301;
J. Akram and A. Pelster, Phys. Rev. A **93** (2016) 023606;
S. K. Adhikari, Laser Phys. Lett. **13** (2016) 035502;
J. Akram and A. Pelster, Phys. Rev. A **93** (2016) 033610;
J. Akram, B. Girodias, and A. Pelster, J. Phys. B: At. Mol. Opt. Phys. **49** (2016) 075302;
S. K. Adhikari and S. Gautam, Phys. Rev. A **93** (2016) 013630;
Ž. Marojević, E. Göklü, and C. Lämmerzahl, Comput. Phys. Commun. **202** (2016) 216;
A. Paredes and H. Michninel, Phys. Dark Universe **12** (2016) 50;
J. Akram and A. Pelster, Laser Phys. **26** (2016) 065501;
T. Mithun, K. Porsezian, and B. Dey, Phys. Rev. A **93** (2016) 013620;
C.-Y. Lai and C.-C. Chien, Phys. Rev. Appl. **5** (2016) 034001;
S. K. Adhikari, Laser Phys. Lett. **13** (2016) 085501;
K. Manikandan, P. Muruganandam, M. Senthilvelan, and M. Lakshmanan, Phys. Rev. E **93** (2016) 032212;
R. R. Sakhel, A. R. Sakhel, H. B. Ghassib, and A. Balaž, Eur. Phys. J. D **70** (2016) 66;
W. Bao, Q. Tang, and Y. Zhang, Commun. Comput. Phys. **19** (2016) 1141;
R. Kishor Kumar, T. Sriraman, H. Fabrelli, P. Muruganandam, and A. Gammal, J. Phys. B: At. Mol. Opt. Phys. **49** (2016) 155301;
A. Bogojević, A. Balaž, and A. Belić, Phys. Rev. E **72** (2005) 036128;
A. Bogojević, I. Vidanović, A. Balaž, and A. Belić, Phys. Lett. A **372** (2008) 3341;
I. Vidanović, A. Bogojević, A. Balaž, and A. Belić, Phys. Rev. E **80** (2009) 066706;
A. Balaž, A. Bogojević, I. Vidanović, and A. Pelster, Phys. Rev. E **79** (2009) 036701;
A. Balaž, I. Vidanović, A. Bogojević, and A. Pelster, Phys. Lett. A **374** (2010) 1539;
A. I. Nicolin, Physica A **391** (2012) 1062;
I. Vasić and A. Balaž, arXiv:1602.03538 (2016);
O. Voronych, A. Buraczewski, M. Matuszewski, and M. Stobińska, arXiv:1603.02570 (2016);
A. M. Martin, N. G. Marchant, D. H. J. O'Dell, and N. G. Parker, arXiv:1606.07107 (2016).

[4] D. Vudragović, I. Vidanović, A. Balaž, P. Muruganandam, and S. K. Adhikari, Comput. Phys. Commun. **183** (2012) 2021.

[5] FFTW3 library, http://www.fftw.org/ (2016).

[6] 2DECOMP&FFT library, http://www.2decomp.org/ (2016).

[7] B. Satarić, V. Slavnić, A. Belić, A. Balaž, P. Muruganandam, S. K. Adhikari, Comput. Phys. Commun. **200** (2016) 411.

[8] Real-data DFTs with FFTW3, http://www.fftw.org/fftw3_doc/Real_002ddata-DFTs.html (2014);
Nvidia's cuFFT accuracy and performance, http://docs.nvidia.com/cuda/cufft/#accuracy-and-performance (2015).

[9] Nvidia's MPI Solutions for GPUs, https://developer.nvidia.com/mpi-solutions-gpus (2016).

© 2016 Elsevier B.V. All rights reserved.

Parallelization of Minimum Spanning Tree Algorithms Using Distributed Memory Architectures

Vladimir Lončar, Srdjan Škrbić and Antun Balaž

Abstract Finding a minimum spanning tree of a graph is a well known problem in graph theory with many practical applications. We study serial variants of Prim's and Kruskal's algorithm and present their parallelization targeting message passing parallel machine with distributed memory. We consider large graphs that can not fit into memory of one process. Experimental results show that Prim's algorithm is a good choice for dense graphs while Kruskal's algorithm is better for sparse ones. Poor scalability of Prim's algorithm comes from its high communication cost while Kruskal's algorithm showed much better scaling to larger number of processes.

Keywords Distributed memory · Kruskal · MPI · MST · Paralellization · Prim

1 Introduction

A minimum spanning tree (MST) of a weighted graph $G = (V, E)$ is a subset of E that forms a spanning tree of G with minimum total weight. MST problem has many applications in computer and communication network design, as well as indirect applications in fields such as computer vision and cluster analysis [12].

V. Lončar · S. Škrbić (✉)
Faculty of Science, University of Novi Sad, Trg Dositeja Obradovica 4, Novi Sad, Serbia
e-mail: srdjan.skrbic@dmi.uns.ac.rs

V. Lončar
e-mail: vladimir.loncar@dmi.uns.ac.rs

A. Balaž
Scientific Computing Laboratory, Institute of Physics Belgrade, University of Belgrade, Pregrevica 118, Belgrade, Serbia
e-mail: antun.balaz@scl.rs

In this paper we implement two parallel algorithms for finding MST of a graph, based on classical algorithms of Prim [23] and Kruskal [18], building upon our previous work in [19]. Algorithms target message passing parallel machine with distributed memory. Primary characteristic of this architecture is that the cost of inter-process communication is high in comparison to cost of computation. Our goal was to develop algorithms which minimize communication, and to measure the impact of communication on the performance of algorithms. Our primary interest were graphs which have significantly larger number of vertices than processors involved in computation. Since graphs of this size cannot fit into the memory of a single process, we use a partitioning scheme to divide the input graph among processes. We consider both sparse and dense graphs.

First algorithm is a parallelization of Prim's serial algorithm. Each process is assigned a subset of vertices and in each step of computation, every process finds a candidate minimum-weight edge connecting one of its vertices to MST. The root process collects those candidates and selects one with minimum weight which it adds to MST and broadcasts result to other processes. This step is repeated until every vertex is in MST.

Second algorithm is based on Kruskal's approach. Processes get a subset of G in the same way as in first algorithm, and then find local minimum spanning tree (or forest). Next, processes merge their MST edges until only one process remains, which holds edges that form MST of G.

Implementations of these algorithms are done using C programming language and MPI (Message Passing Interface) and tested on a parallel cluster PARADOX using up to 256 cores and 256 GB of distributed memory.

Section 2 contains references to the most important related papers. In Sect. 3 we continue with the description and analysis of algorithms—both serial and parallel versions, and their implementation. In the last section we describe experimental results, analyze them and draw our conclusions.

2 Related Work

Algorithms for MST problem have mostly been based on one of three approaches, that of Boruvka [3], Prim [23] and Kruskal [18], however, a number of new algorithms has been developed. Gallager et al. [10] presented an algorithm where processor exists at each node of the graph (thus $n = p$), useful in computer network design. Katriel and Sanders designed an algorithm exploiting cycle property of a graph targeting dense graph, [17], while Ahrabian and Nowzari-Dalini's algorithm relies on depth first search of the graph [1].

Due to its parallel nature, Boruvka's algorithm (also known as Sollin's algorithm) has been the subject to most research related to parallel MST algorithms. Examples of algorithms based on Boruvka's approach include Chung and Condon [4], Wang and Gu [14] and Dehne and Götz [7].

Parallelization of Prim's algorithm has been presented by Deo and Yoo [8]. Their algorithm targets shared memory computers. Improved version of Prim's algorithm has been presented by Gonina and Kale [11]. Their algorithm adds multiple vertices per iteration, thus achieving significant speedups. Another approach targeting shared memory computers presented by Setia et al. [24] uses the cut property of a graph to grow multiple trees in parallel. Hybrid approach, combining both Boruvka's and Prim's approaches has been developed by Bader and Cong [2].

Examples of parallel implementation of Kruskal's algorithm can be found in work of Jin and Baker [16], and Osipov et al. [21]. Osipov et al. proposes a modification to Kruskal's algorithm to avoid edges which certainly are not in a graph. Their algorithm runs in near linear time if graph is not too sparse.

Bulk of the research into parallel MST algorithms has targeted shared memory computers like PRAM, i.e. computers where entire graph can fit into memory. Our algorithms target distributed memory computers and use partitioning scheme to divide the input graph evenly among processors. Because no process contains info about partition of other processes, we designed our algorithms to use predictable communication patterns, and not depend on the properties of input graph.

3 The Algorithms

Let us assume that graph $G = (V, E)$, with vertex set V and edge set E is connected and undirected. Without loss of generality, it can be assumed that each weight is distinct, thus G is guaranteed to have only one MST. This assumption simplifies implementation, otherwise a numbering scheme can be applied to edges with same weight, at the cost of additional implementation complexity.

Let n be the number of vertices, m the number of edges ($|V| = n$, $|E| = m$), and p the number of processes involved in computation of MST. Let $w(v, u)$ denote weight of edge connecting vertices v and u. Input graph G is represented as $n \times n$ adjacency matrix $A = (a_{i,j})$ defined as:

$$a_{i,j} = \begin{cases} w(v_i, v_j) & \text{if } (\mathrm{v_i}, \mathrm{v_j}) \in \mathrm{E} \\ 0 & \textit{otherwise} \end{cases} \tag{1}$$

3.1 Prim's Algorithm

Prim's algorithm starts from an arbitrary vertex and then grows the MST by choosing a new vertex and adding it to MST in each iteration. Vertex with an edge with lightest weight incident on the vertices already in MST is added in every iteration. The algorithm continues until all the vertices have been added to the MST. This algorithm requires $O(n^2)$ time. Implementations of Prim's algorithm commonly use auxiliary array d of length n to store distances (weight) from each

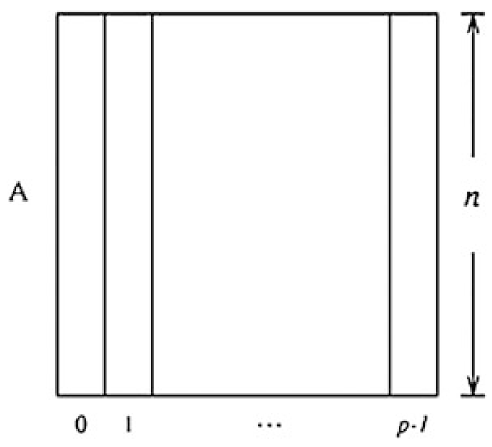

Fig. 1 Partitioning of adjacency matrix among p processes

vertex to MST. In every iteration a lightest weight edge in d is added to MST and d is updated to reflect changes.

Parallelizing the main loop of Prim's algorithm is difficult [13], since after adding a vertex to MST lightest edges incident on MST change. Only two steps can be parallelized: selection of the minimum-weight edge connecting a vertex not in MST to a vertex in MST, and updating array d after a vertex is added to MST. Thus, parallelization can be achieved in the following way:

1. Partition the input set V into p subsets, such that each subset contains n/p consecutive vertices and their edges, and assign each process a different subset. Each process also contains part of array d for vertices in its partition. Let V_i be the subset assigned to process p_i, and d_i part of array d which p_i maintains. Partitioning of adjacency matrix is illustrated in Fig. 1.
2. Every process p_i finds minimum-weight edge e_i (*candidate*) connecting MST with a vertex in V_i.
3. Every process p_i sends its e_i edge to the root process using all-to-one reduction.
4. From the received edges, the root process selects one with a minimum weight (called global minimum-weight edge e_{min}), adds it to MST and broadcasts it to all other processes.
5. Processes mark vertices connected by e_{min} as belonging to MST and update their part of array d.
6. Repeat steps 2–5 until every vertex is in MST.

Finding a minimum-weight edge and updating of d_i during each iteration costs $O(n/p)$. Each step also adds a communication cost of all-to-one reduction and all-to-one broadcast. These operations complete in $O(\log p)$. Combined, cost of one iteration is $O(n/p + \log p)$. Since there are n iterations, total parallel time this algorithm runs in is:

$$T_p = O\left(\frac{n^2}{p}\right) + O(n \log p) \tag{2}$$

Prim's algorithm is better suited for dense graphs and works best for complete graphs. This also applies to its parallel formulation presented here. Ineffectiveness of the algorithm on sparse graphs stems from the fact that Prim's algorithm runs in $O(n^2)$, regardless of the number of edges. A well-known modification [5] of Prim's algorithm is to use binary heap data structure and adjacency list representation of a graph to reduce the run time to $O(m \log n)$. Furthermore, using Fibonacci heap asymptotic running time of Prim's algorithm can be improved to $O(m + n \log n)$. Since we use adjacency matrix representation, investigating alternative approaches for Prim's algorithm was out of the scope of this paper.

3.2 Kruskal's Algorithm

Unlike Prim's algorithm which grows a single tree, Kruskal's algorithm grows multiple trees in parallel. Algorithm first creates a forest F, where each vertex in the graph is a separate tree. Next step is to sort all edges in E based on their weight. Algorithm then chooses minimum-weight edge e_{min} (i.e. first edge in sorted set). If e_{min} connects two different trees in F, it is added to the forest and two trees are combined into a single tree, otherwise e_{min} is discarded. Algorithm loops until either all edges have been selected, or F contains only one tree, which is the MST of G. This algorithm is commonly implemented using Union-Find algorithm [22]. *Find* operation is used to determine which tree a particular vertex is in, while *Union* operation is used to merge two trees. Kruskal's algorithm runs in $O(m \log n)$ time, but can be made even more efficient by using more sophisticated Union-Find data structure, which uses *union by rank* and *path compression* [9]. If the edges are already sorted, using improved Union-Find data structure Kruskal's algorithm runs in $O(m\alpha(n))$, where $\alpha(n)$ is the inverse of the Ackerman function.

Our parallel implementation of Kruskal's algorithm uses the same partitioning scheme of adjacency matrix as in Prim's approach and is thus bounded by $O(n^2)$ time to find all edges in matrix. Having that in mind, our parallel algorithm proceeds through the following steps:

1. Every process p_i first sorts edges contained in its partition V_i.
2. Every process p_i finds a local minimum spanning tree (or forest, MSF) F_i using edges in its partition V_i applying the Kruskal's algorithm.
3. Processes merge their local MST's (or MSF's). Merging is performed in the following manner. Let a and b denote two processes which are to merge their local trees (or forests), and let F_a and F_b denote their respective set of local MST edges. Process a sends set F_a to b, which forms a new local MST (or MSF) from $F_a \cup F_b$. After merging, process a is no longer involved in computation and can terminate.
4. Merging continues until only one process remains. Its MST is the end result.

Creating a new local MSF during merge step can be performed in a number of different ways. Our approach is to perform Kruskal's algorithm again on $F_a \cup F_b$.

Computing the local MST takes $O(n^2/p)$. There is a total of log p merging stages, each costing $O(n^2\log p)$. During one merge step one process transmits maximum of $O(n)$ edges for a total parallel time of:

$$T_p = O(n^2/p) + O(n^2 \log p) \tag{3}$$

Based on speedup and efficiency metrics, it can be shown that this parallel formulation is efficient for $p = O(n/\log n)$, same as the first algorithm.

3.3 Implementation

Described algorithms were implemented using ANSI C and Message Passing Interface (MPI). Fixed communication patterns in parallel formulation of the algorithms map directly to MPI operations. Complete source code can be found in [25].

4 Experimental Results

Implementations of algorithms were tested on a cluster of up to 32 computing nodes. Each computer in the cluster had two Intel Xeon E5345 2.33 GHz quad-core CPUs and 8 GB of memory, with Scientific Linux 6 operating system installed. We used OpenMPI v1.6 implementation of the MPI standard. The cluster nodes are connected to the network with a throughput of 1 Gbit/s. Both implementations were compiled using GCC 4.4 compiler. This cluster has enabled testing algorithms with up to 256 processes as shown in Table 1.

We tested graphs with densities of 1, 5, 10, 15 and 20 % with number of vertices ranging from 10,000 to 100,000, and number of edges from 500,000 to 1,000,000,000. Distribution of edges in graphs was uniformly random, and all edge weights were unique. Due to the high memory requirements of large graphs, not every input graph could be partitioned in a small number of cluster nodes, as can be seen in Table 1.

4.1 Results

Due to the large amount of obtained test results, we only present the most important ones here. Complete set of results can be found in [25].

In the Table 2 we show the behavior of algorithms with increasing number of processes on input graph of 50,000 vertices and density of 10 %.

Results show poor scalability of Prim's algorithm, due to its high communication cost. Otherwise, computation phase of Prim's algorithm is faster than that of

Table 1 Testing parameters

Processes	Nodes	Processes per node	No. of vertices (k)
4	4	1	10–50
8	8	1	10–60
16	16	1	10–80
32	32	1	10–100
64	32	2	10–100
128	32	4	10–100
256	32	8	10–100

Table 2 CPU time (in seconds) for algorithms with increasing number of processes

	4	8	16	32	64	128	256
Kruskal	38.468	19.94	10.608	5.342	2.958	1.796	1.382
Prim	16.703	15.479	25.201	30.382	30.824	32.661	39.737

Table 3 CPU time (in seconds) for algorithms with increasing density

	1 %	5 %	10 %	15 %	20 %
Kruskal	0.607	2.603	5.342	8.164	10.663
Prim	30.189	30.007	30.382	30.518	30.589

Kruskal's. Due to the usage of adjacency matrix graph representation, Prim's algorithm performs almost the same regardless of the density of the input graph. This can be seen from the results of input graph with 50,000 vertices and 32 processes with varying density shown at Table 3.

On the other hand, Kruskal's algorithm shows degradation of performance with increasing density. Results of Kruskal's algorithm show that majority of local computation time is spent sorting the edges of input graph, which grows with larger density. Increasing the number of processes makes local partitions smaller and faster to process, thus allowing this algorithm to achieve good scalability. If the edges of input graph were already sorted, Kruskal's algorithm would be significantly faster than other MST algorithms.

4.2 Impact of Communication Overhead

Cost of communication is much greater than the cost of computation, so it is important to analyse the time spent in communication routines. During tests we measured the time spent waiting for the completion of the communication operations. In case of Prim's algorithm, we measured the time that the root process spends waiting for the completion of MPI_Reduce and MPI_Bcast operations. Communication in Kruskal's algorithm is measured as total time spent waiting for messages received over MPI_Recv operation in the last active process (which will

Table 4 Communication versus computation time (in seconds)

Processes	4	8	16	32	64	128	256
Prim's algorithm							
Total	16.703	15.479	25.201	30.382	30.824	32.661	39.737
Communication	8.188	11.183	23.009	29.248	30.237	32.322	39.467
Kruskal's algorithm							
Total	38.468	19.94	10.608	5.342	2.958	1.796	1.382
Communication	0.171	0.356	0.371	0.288	0.317	0.253	0.256

contain the MST after last iteration of the merge operation). This gives us a good insight into the duration of communication routines because the last active process will have to wait the most.

The Table 4 shows communication times of processing input graph of 50,000 vertices with 10 % density.

When comparing communication time with a total computation time it can be noted that the Prim's algorithm spends most of time in communication operations, and by increasing number of processes almost all the running time of the algorithm is spent on communication operations. A bottleneck in Prim's algorithm is the cost of MPI_Reduce and MPI_Bcast communication operations. These operations require communication between all processes, and are much more expensive than local computation within each process, because all processes must wait until the operation is completed, or until the data are transmitted over the network. This prevents Prim's algorithm from achieving substantial speedup of running time with increasing number of processes. Therefore, this algorithm is most efficient on the fewest number of processes that the partitioned input graph can fit.

On the other hand Kruskal algorithm spends much less time in communication operations, but instead spends most of the time in local computation. These differences are illustrated in Figs. 2 and 3. The diagrams show that communication in Prim's algorithm rises sharply with increasing number of processes, while execution time slowly reduces. In Kruskal's algorithm, the situation is reversed.

4.3 Analysis of Results

The experimental results confirmed some of the assumptions made during the development and analysis of algorithms, but also made a couple of unexpected results. Results of these experiments gave us directions for further improvement of the described algorithms.

Prim's algorithm has shown excellent performance in computational part of the algorithm, but a surprisingly high cost of communication operations spoils its final score. Finding candidate edges for inclusion in MST can be further improved by using techniques described in [5], but it will not significantly improve the total

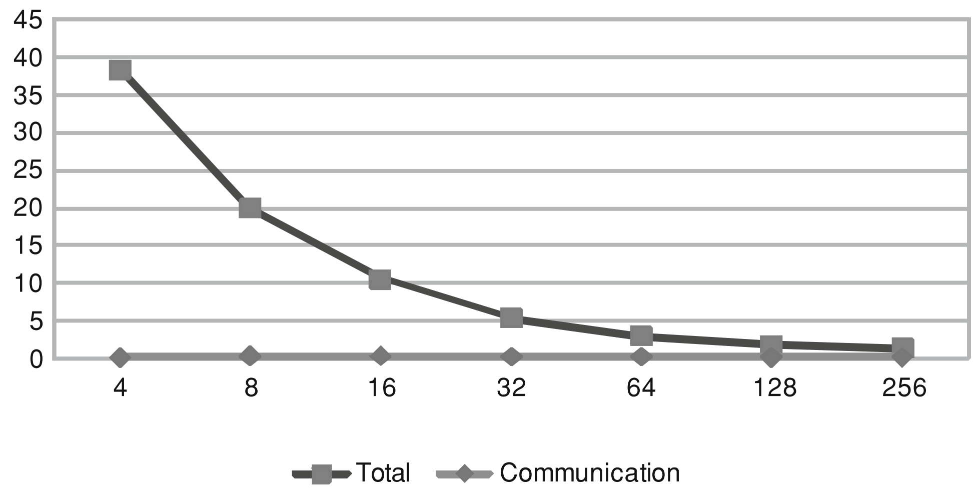

Fig. 2 Communication in Kruskal's algorithm

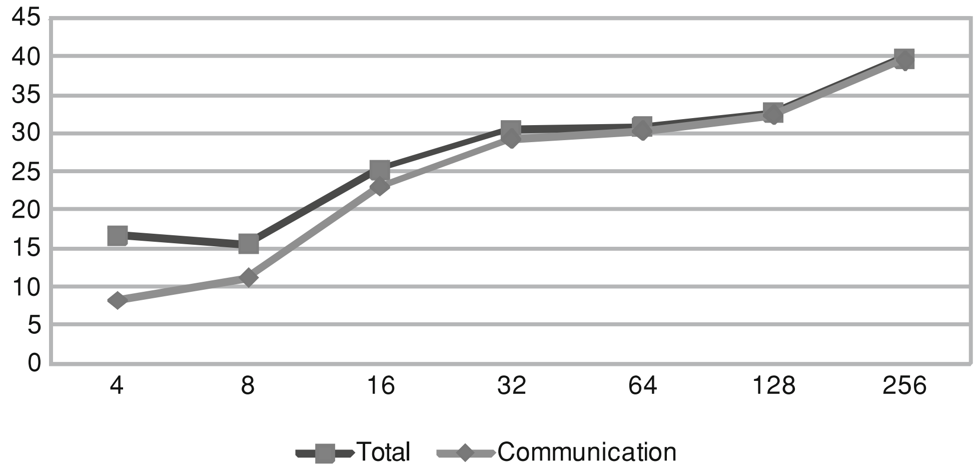

Fig. 3 Communication in Prim's algorithm

time of the algorithm, as communication routines will remain the same. Unfortunately, the communication can not be further improved by changing the algorithm. The only way to reduce the cost of communication is to use a cluster that has a better quality network, or to rely on the semantics of the implementation of the MPI operation MPI_Allreduce.

Kruskal's algorithm has shown good performance, especially for sparse graphs, while the performance degrades with increasing density. It is important to note that many real-world graphs have density much smaller than 1 % (for example, graph of roads as egdes and junctions as vertices has a density much smaller than 1 %). Also, this algorithm showed much better scaling to larger number of processes than Prim's algorithm. Cost of communication in Kruskal's algorithm is much smaller than in Prim's algorithm, but the local computation is slower. This can be improved by using more efficient Union-Find algorithms [9], or by improving merging of local trees between processes. Kruskal's algorithm does not use a lot of slow messages like Prim's algorithm, but can send very large messages depending on the number of processes and the size of the graph. This can be improved by introducing techniques for compressing messages, or changing the structure of the message.

5 Alternate Parallelization Approaches

In this section we will give a brief overview of two other parallelization approaches we considered using for implementation of these algorithms. One approach would be using graphics processing unit (GPU) technologies like Nvidia CUDA or OpenCL. Another would be using shared-memory parallelization API like OpenMP to utilize multi-core processors on cluster nodes. We will go over advantages and disadvantages of both approaches.

With the introduction of CUDA and OpenCL programming models, using GPU for general-purpose computing (GPGPU) has become a powerful alternative to traditional CPU programming models. Nowadays GPUs can be found in most high-ranking supercomputers and even ordinary clusters. GPUs have their own RAM, which is separate from main RAM of a computer and was not accessible for distributed-memory technologies like MPI. This made writing multi-GPU programs more difficult, since it required expensive copy operations between GPU memory and host (CPU) memory which MPI could access. However, recent developments in MPI implementations have alleviated this problem, and newer versions of popular MPI implementations like OpenMPI and MVAPICH can access GPU memory directly. This unfortunately still doesn't make GPU the perfect platform for implementations of our algorithms. GPUs still have much smaller amount of RAM when compared to main memory (recently released models like Tesla K10 have up to 8 GB of memory [20]). This means that GPU solution could only be used on much smaller graphs. Alternatively, a different graph representation (like adjacency lists) would allow graphs with greater number of vertices, but would still be only useful for sparser graphs. Primary part of Prim's algorithm which could be accelerated by GPU is finding local (and then global) vertex with the smallest distance to the tree. This could be achieved by slightly modifying well-known parallel reduction algorithm for GPU [15]. Communication pattern between nodes would remain the same. Kruskal's algorithm is more complex to implement on GPU due to Union-Find data structure. Other important portions of Kruskal's algorithm, like sorting of input could be done using various GPU libraries.

Unlike the relatively new technology that is GPGPU, OpenMP has been successfully used to parallelize serial code since the late 90s. In some cases, OpenMP allows developers to parallelize their with programs with minimal effort, using compiler directives around loops, often with good performance [6]. This technique could be used in parallelization of Prim's algorithm for finding local (and later global) vertex with the smallest distance to the tree. Graph would be partitioned in such a way that each node in cluster receives an equal part, then each node would use all it's processors and cores with OpenMP to find local minimum, and use MPI for communication between nodes. Kruskal's algorithm can be parallelized in similar way, although it would require a slightly greater effort for implementation of sorting and Union-Find data structure.

Acknowledgements Authors are partially supported by Ministry of Education, Science, and Technological Development of the Republic of Serbia, through projects no. ON174023: "Intelligent techniques and their integration into wide-spectrum decision support", and ON171017: "Modeling and numerical simulations of complex many-body systems", as well as European Commission through FP7 projects PRACE-2IP and PRACE-3IP.

References

1. H. Ahrabian, A. Nowzari-Dalini, Parallel algorithms for minimum spanning tree problem. Int. J. Comput. Math. **79**(4), 441–448 (2002)
2. D.A. Bader, G. Cong, Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs. J. Parallel Distrib. Comput. **66**(11), 1366–1378 (2006)
3. O. Boruvka, O Jistém Problému Minimálnm (about a certain minimal problem) (in Czech, German summary). *Práce Mor. Prrodoved. Spol. v Brne III*, vol. 3 (1926)
4. S. Chung, A. Condon, Parallel implementation of borvka's minimum spanning tree algorithm. in *Proceedings of the 10th International Parallel Processing Symposium, IPPS '96* (IEEE Computer Society, Washington, DC, 1996), pp. 302–308
5. T.H. Cormen, C. Stein, R.L. Rivest, C.E. Leiserson, *Introduction to Algorithms*, 2nd edn. (McGraw-Hill Higher Education, Boston, 2001)
6. M. Curtis-Maury, X. Ding, C.D. Antonopoulos, D.S. Nikolopoulos, *An Evaluation of Openmp on Current and Emerging Multithreaded/Multicore Processors*, ed. by M.S. Mueller, B.M. Chapman, B.R. Supinski, A.D. Malony, M. Voss. *OpenMP Shared Memory Parallel Programming*, vol 4315 (*Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2008), pp. 133–144
7. F. Dehne, S. Gtz, Practical Parallel Algorithms for Minimum Spanning Trees, *in Workshop on Advances in Parallel and Distributed Systems* (1998), pp. 366–371
8. N. Deo, Y.B. Yoo, Parallel algorithms for the minimum spanning tree problem, in *Proceedings of the International Conference on Parallel Processing* (1981), pp. 188–189
9. Z. Galil, G.F. Italiano, Data structures and algorithms for disjoint set union problems. ACM Comput. Surv. **23**(3), 319–344 (1991)
10. R.G. Gallager, P.A. Humblet, P.M. Spira, A distributed algorithm for minimum-weight spanning trees. ACM Trans. Program. Lang. Syst. **5**(1), 66–77 (1983)
11. E. Gonina, L.V. Kale, Parallel prim's algorithm on dense graphs with a novel extension, in *PPL Technical Report*, Oct 2007
12. R.L. Graham, P. Hell, On the history of the minimum spanning tree problem. IEEE Ann. Hist. Comput. **7**(1), 43–57 (1985)
13. A. Grama, G. Karypis, V. Kumar, A. Gupta, *Introduction to Parallel Computing, 2nd edn.* (Addison Wesley, Reading, 2003)
14. W. Guang-rong, G. Nai-jie, An efficient parallel minimum spanning tree algorithm on message passing parallel machine. J. Softw. **11**(7), 889–898 (2000)
15. M. Harris, Optimizing parallel reduction in CUDA. CUDA tips and tricks

16. M. Jin, J.W. Baker, Two graph algorithms on an associative computing model, in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2007*, vol 1, Las Vegas, Nevada, 25–28 June 2007, pp. 271–277
17. I. Katriel, P. Sanders, J.L. Trff, J.L. Tra, A practical minimum spanning tree algorithm using the cycle property, in *11th European Symposium on Algorithms (ESA),* vol. *2832 in LNCS* (Springer, New York, 2003), pp. 679–690
18. J.B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. Am. Math. Soc. **7**(1), 48–50 (1956)
19. V. Lončar, S. Škrbić, A. Balaž, Distributed memory parallel algorithms for minimum spanning trees, in *Lecture Notes in Engineering and Computer Science: Proceedings of the World Congress on Engineering 2013, WCE 2013*, London, 3–5 July 2013, pp. 1271–1275
20. Nvidia, Nvidia tesla GPU accelerators. Nvidia Tesla Product Datasheet (2012)
21. V. Osipov, P. Sanders, J. Singler, The filter-kruskal minimum spanning tree algorithm, in *ALENEX'09* (2009), pp. 52–61
22. D.-Z. Pan, Z.-B. Liu, X.-F. Ding, Q. Zheng, The application of union-find sets in kruskal algorithm, in *Proceedings of the 2009 International Conference on Artificial Intelligence and Computational Intelligence (AICI '09),* vol 2 (IEEE Computer Society, Washington, DC, 2009), pp. 159–162
23. R.C. Prim, Shortest connection networks and some generalizations. Bell Syst. Technol. J. **36**, 1389–1401 (1957)
24. R. Setia, A. Nedunchezhian, S. Balachandran, A new parallel algorithm for minimum spanning tree problem, in *Proceedings of the International Conference on High Performance Computing (HiPC)* (2009), pp. 1–5
25. S. Škrbić, Scientific Computing Seminar (2013)

Parallel implementation of minimum spanning tree algorithms using MPI

Vladimir Lončar* and Srdjan Škrbic**
Faculy of Science, Depatment for Mathematics an Informatics, University of Novi Sad, Serbia
* vlada.loncar@gmail.com
** shkrba@uns.ac.rs

***Abstract*—In this paper we study parallel algorithms for finding minimum spanning tree of a graph. We present two algorithms, based on sequential algorithms of Prim and Kruskal, targeting message passing parallel machine with distributed memory. First algorithm runs in $O(n^2/p+n\log p)$ and second algorithm runs in $O(n^2/p + n^2 \log p)$.**

***Index Terms*—Minimum spanning tree, parallel algorithms, message passing, distributed memory computer.**

I. Introduction

A minimum spanning tree (MST) of a graph $G = (V, E)$ is a subset of E that forms a spanning tree of G with minimum total weight. MST problem has many applications in computer and communication network design, as well as indirect applications in fields such as computer vision and cluster analysis [1].

In this paper we implement two parallel algorithms for finding MST of a graph, based on classical algorithms of Prim and Kruskal. Algorithms target message passing parallel machine with distributed memory. Primary characteristic of this architecture is that the cost of inter-process communication is high in comparison to cost of computation. Our goal was to develop algorithms which minimize communication, and to measure the impact of communication on the performance of algorithms. Our primary interest were graphs which have significantly larger number of vertices than processors involved in computation. Since graphs of this size cannot fit into a memory of single process, we use simple partitioning scheme to divide the input graph among processes. We considered both sparse and dense graphs.

First algorithm is a parallelization of Prim's sequential algorithm. Each process is assigned a subset of vertices and in each step of computation, every process finds a *candidate* minimum-weight edge connecting one of it's vertices to MST. Leader process collects those candidates and selects one with minimum weight which it adds to MST, and broadcasts result to other processes. This step is repeated until every vertex is in MST.

Second algorithm is based on Kruskal's approach. Processes get a subset of G in the same way as in first algorithm, and then find local minimum spanning tree (or forest). Next, processes merge their MST edges until only one process remains, which holds edges that form MST of G.

Algorithms we present are both easy to understand and implement, and since they use fixed communication patterns, their performance can easily be predicted.

II. Related Work

Algorithms for MST problem have mostly been based on one of three approaches, that of Boruvka [2], Prim [3] and Kruskal [4], however, a number of new algorithms has been developed. Gallager et al. presented an algorithm where processor exists at each node of the graph (thus $n = p$), useful in computer network design [5]. Katriel and Sanders designed an algorithm exploiting cycle property of a graph targeting dense graph, [6], while Ahrabian and Nowzari-Dalini's algorithm relies on depth first search of the graph [7].

Due to it's parallel nature, Boruvka's algorithm (also known as Sollin's algorithm) has seen the most research. Examples of algorithms based on Boruvka's approach include Chung and Condon [8], Wang and Gu [9] and Dehne and Götz [10].

Parallelization of Prim's algorithm has been presented by Deo and Yoo [11]. Their algorithm targets shared-memory computers. Improved version of Prim's algorithm has been presented by Gonina and Kale [12]. Their algorithm adds multiple vertices per iteration, thus achieving significant speedups. Another approach targeting shared-memory computers presented by Setia et al. [13] uses the cut property of a graph to grow multiple trees in parallel. Hybrid approach, combining both Boruvka's and Prim's approaches has been developed by Bader and Cong [14].

Examples of parallel implementation of Kruskal's algorithm can be found in work of Jin and Baker [15], and Osipov et al [16]. Osipov et al. proposes a modification to Kruskal's algorithm to avoid edges which certainly are not in a graph. Their algorithm runs in near linear time if graph is not too sparse.

Bulk of the research into parallel MST algorithms has targeted shared-memory computers like PRAM, i.e. computers where entire graph can fit into memory. Our algorithms target distributed-memory computers and use partitioning scheme to divide the input graph evenly among processors. Because no process contains info about partition of other processes, we designed our algorithms to use predictable communication patterns, and not depend on the properties of input graph.

III. THE ALGORITHMS

In the remainder of this paper, we will assume that graph $G = (V, E)$ is connected and undirected. Without loss of generality, it can be assumed that each weight is distinct, thus G is guaranteed to have only one MST. This assumption simplifies implementation, otherwise a numbering scheme can be applied to edges with same weight, at the cost of additional implementation complexity.

Let n be the number of vertices, m the number of edges ($|V| = n$, $|E| = m$), and p the number of processes involved in computation of MST. Let $w(v, u)$ denote weight of edge connecting vertices v and u. Input graph G is represented as $n \times n$ adjacency matrix $A = (a_{i,j})$ defined as:

$$a_{i,j} = \begin{cases} w(v_i, v_j) & \text{if } (v_i, v_j) \in E \\ 0 & otherwise \end{cases} \tag{1}$$

A. Prim's Algorithm

Prim's algorithm starts from an arbitrary vertex and then grows the MST by choosing a new vertex and adding it to MST in each iteration. Vertex with an edge with lightest weight incident on the vertices already in MST is added in every iteration. The algorithm continues until all the vertices have been added to the MST. This algorithm requires $O(n^2)$ time. Implementations of Prim's algorithm commonly use auxiliary array d of length n to store distances (weight) from each vertex to MST. In every iteration a lightest weight edge in d is added to MST and d is updated to reflect changes.

Parallelizing the main loop of Prim's algorithm is difficult [17], since after adding a vertex to MST lightest edges incident on MST change. Only two steps can be parallelized: selection of the minimum-weight edge connecting a vertex not in MST to a vertex in MST, and updating array d after a vertex is added to MST. Thus, parallelization can be achieved in the following way:

1) Partition the input set V into p subsets, such that each subset contains n/p consecutive vertices and their edges, and assign each process a different subset. Each process also contains part of array d for vertices in it's partition. Let V_i be the subset assigned to process p_i, and d_i part of array d which p_i maintains. Partitioning of adjacency matrix is illustrated in Fig. 1.
2) Every process p_i finds minimum-weight edge e_i (*candidate*) connecting MST with a vertex in V_i.
3) Every process p_i sends its e_i edge to leader process using all-to-one reduction.
4) From the received edges, leader process selects one with a minimum weight (called global minimum-weight edge e_{min}), adds it to MST and broadcasts it to all other processes.
5) Processes mark vertices connected by e_{min} as belonging to MST and update their part of array d.
6) Repeat steps 2-5 until every vertex is in MST.

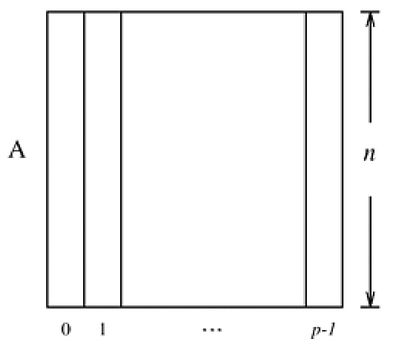

Fig. 1. Partitioning of adjacency matrix among p processes

Finding a minimum-weight edge and updating of d_i during each iteration costs $O(n/p)$. Each step also adds a communication cost of all-to-one reduction and all-to-one broadcast. These operations complete in $O(\log p)$. Combined, cost of one iteration is $O(n/p + \log p)$. Since there are n iterations, total parallel time this algorithm runs in is:

$$T_p = O\left(\frac{n^2}{p}\right) + O\left(n \log p\right) \tag{2}$$

In comparison to sequential algorithm, this algorithm achieves a speedup and efficiency of:

$$S = \frac{O(n^2)}{O(n^2/p + n \log p)} \tag{3}$$

$$E = \frac{1}{1 + O((p \log p)/n)} \tag{4}$$

From equations 3 and 4 we conclude that this formulation of Prim's algorithm is efficient only for $p = O(n/\log n)$ processes.

Prim's algorithm is better suited for dense graphs and works best for complete graphs. This also applies to it's parallel formulation presented here. Ineffectiveness of the algorithm on sparse graphs stems from the fact that Prim's algorithm runs in $O(n^2)$, regardless of the number of edges. A well-known modification [18] of Prim's algorithm is to use binary heap data structure and adjacency list representation of a graph to reduce the run time to $O(m \log n)$. Furthermore, using Fibonacci heap asymptotic running time of Prim's algorithm can be improved to $O(m + n \log n)$. Since we use adjacency matrix representation, investigating alternative approaches for Prim's algorithm was out of the scope of this paper.

B. Kruskal's Algorithm

Unlike Prim's algorithm which grows a single tree, Kruskal's algorithm grows multiple trees in parallel. Algorithm first creates a forest F, where each vertex in the graph is a separate tree. Next step is to sort all edges in E based on their weight. Algorithm then loops the sorted set and chooses minimum-weight edge e_{min} (i.e. first edge in sorted set). If e_{min} connects two different trees in F, add it to the forest and combine two trees into a single tree, otherwise discard e_{min}. Algorithm loops until either all edges have been selected, or F contains only one tree, which is the MST of G. This algorithm is commonly implemented using Union-Find algorithm [19]. *Find* operation is used to determine which tree a particular vertex is in, while *Union* operation is used to merge two trees. Kruskal's algorithm runs in $O(m \log n)$ time, but can be made even more efficient by using more sophisticated Union-Find data structure, which uses *union by rank* and *path compression* [20]. If the edges are already sorted, using improved Union-Find data structure Kruskal's algorithm runs in $O(m\alpha(n))$, where $\alpha(n)$ is the inverse of an Ackerman function.

Our parallel implementation of Kruskal's algorithm uses the same partitioning scheme of adjacency matrix as in Prim's approach and is thus bounded by $O(n^2)$ time to find all edges in matrix. Every process first sorts edges contained in its partition. From edges in partition V_i, every process p_i finds a local minimum spanning tree (or forest, MSF) T_i using Kruskal's algorithm. At the end of this step, local MSTs are merged. Merging is performed in the following manner. Let a and b denote two processes which are to merge their local trees (or forests), and let A and B denote their respective set of local MST edges. Process a sends set A to b, which forms a new local MST (or MSF) from $A \cup B$. After merging, process a is no longer involved in computation and can terminate. Merging continues until only one process remains, which will contain MST of G.

Example of parallel Kruskal's algorithm is illustrated in Fig. 2. Input graph in *(a)* is divided among processes p_1 and p_2 which compute local MST based on edges incident on vertices assigned to them (*(b)* and *(c)*). Next, processes merge their local MST-s to form a MST of input graph. The dashed lines represent edges which are in local MST of a process, but are removed after merging.

Creating a new local MSF during merge step can be performed in a number of different ways. One approach is to perform Kruskal's algorithm again on $A \cup B$. Alternatively, a modified *depth-first search* (DFS) can be used. For every edge in A, it is first determined if it is already in the same tree of B (using *find* operation). If it is not, it is added in MSF and *union* operation is called. Merging two trees can produce a cycle, so a modified DFS is run to eliminate edge with a heaviest weight.

Computing the local MST takes $O(n^2/p)$. There is a total of $\log p$ merging stages, each costing $O(n^2 \log p)$. During one merge step one process transmits maximum of $O(n)$ edges for a total parallel time of:

$$T_p = O(n^2/p) + O(n^2 \log p) \tag{5}$$

Based on speedup and efficiency metrics, it can be shown that this parallel formulation is efficient for $p = O(n/\log n)$, same as first algorithm.

IV. Implementation

Algorithms were simple to implement using ANSI C and MPI. Simplicity is the result of fixed communication patterns which directly map to MPI operations. During implementation we explored alternative communication patterns in order to grow multiple trees in parallel, similar to approaches of parallelization of Boruvka's algorithm. We have found that using data-dependant communication paths results in imbalanced computation due to arbitrary communication between processes. Also, implementation of arbitrary communication can be difficult with MPI, since number of messages each process sends or receives is not known in advance for every input. Overcoming this obstacle often requires adding additional communication complexity, at the cost of overall performance.

Communication pattern in Prim's algorithm can be improved by using MPI *MPI_Allreduce* operation instead of the standard combination of *MPI_Reduce* and *MPI_Bcast*. This optimization does not necessarily result in better performance, since MPI implementations can implement

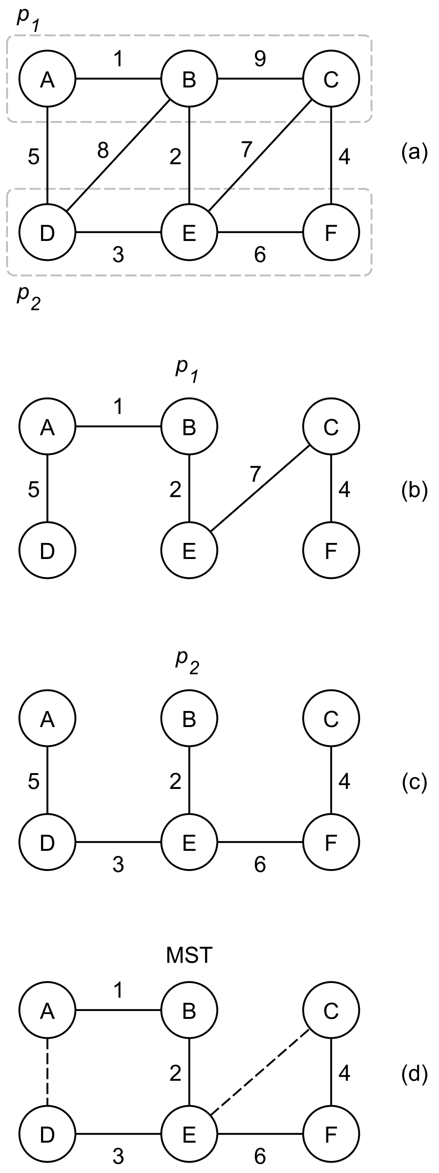

Fig. 2. Example of merge step for two processes

MPI_Allreduce operation as a simple all-to-one reduce, followed by a broadcast, without any performance improvements [21].

Main performance bottleneck of Prim's algorithm is communication overhead of all-to-one reduce operation. Reduce operation is costly in comparison to local computation, and all other processes are idle while waiting for reduce to complete. This prevents Prim's algorithm to achieve significant speedups on a larger number of processors. Therefore, Prim's algorithm is best used on a smallest number of processes on which partitioned input graph can fit.

Unlike Prim's algorithm, Kruskal's algorithm doesn't use collective communication operations during which all processes except one are idle. Performance-wise, critical part Kruskal's algorithm is merging of local MST-s. Merge part of Kruskal's algorithm is only fully efficient in case when p is a power of 2. Since merging is pairwise operation, in other cases at least one merge step will have a process without a pair. This process will be idle until a merge partner is available. In our implementation, idling can span multiple merge steps, thus causing a considerable efficiency degradation. For example, if there are 9 processes in computation, one process will be idle until the very last merge step. One approach to solving this issue would be introduction of a special 3-way merge (or in general a d-ary merge, where $d = 2, 3, 4 \ldots$) along with a load balancing logic to minimize or remove the effect on performance of algorithm.

V. Conclusion

We presented two parallel implementations of algorithms for finding minimum spanning tree of a graph. Our algorithms are parallelizations of classical sequential algorithms of Prim and Kruskal. Parallel processes work on a subset of input graph, and communicate using fixed communication pattern. First algorithm takes $O(n^2/p + n \log p)$ time, while second takes $O(n^2/p + n^2 \log p)$ time.

Our analysis has identified several bottlenecks in our implementations, and further work in this area would include minimizing communication cost by reducing the number and size of messages passed, as well as improving merge step of the second algorithm. Also, further experimental work would give us information about practical limitations of our algorithms for wider array of input graphs and uncover new areas for improvement.

Acknowledgment

Authors are partially supported by Ministry of Education and Science of the Republic of Serbia, through project no. III47003: " Infrastructure for technology enhanced learning in Serbia".

References

[1] R. L. Graham and P. Hell, "On the history of the minimum spanning tree problem," *IEEE Ann. Hist. Comput.*, vol. 7, no. 1, pp. 43–57, Jan. 1985. [Online]. Available: http://dx.doi.org/10.1109/MAHC.1985.10011

[2] O. Boruvka, " O Jistém Problému Minimálním (About a Certain Minimal Problem) (in Czech, German summary)," *Práce Mor. Prírodoved. Spol. v Brne III*, vol. 3, 1926.

[3] R. C. Prim, "Shortest connection networks and some generalizations," *Bell System Technology Journal*, vol. 36, pp. 1389–1401, 1957.

[4] J. B. Kruskal, "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, Feb. 1956. [Online]. Available: http://www.jstor.org/stable/2033241

[5] R. G. Gallager, P. A. Humblet, and P. M. Spira, "A distributed algorithm for minimum-weight spanning trees," *ACM Trans. Program. Lang. Syst.*, vol. 5, no. 1, pp. 66–77, Jan. 1983. [Online]. Available: http://doi.acm.org/10.1145/357195.357200

[6] I. Katriel, P. Sanders, J. L. Träff, and J. L. Tra, "A practical minimum spanning tree algorithm using the cycle property," in *In 11th European Symposium on Algorithms (ESA), number 2832 in LNCS*. Springer, 2003, pp. 679–690.

[7] H. Ahrabian and A. Nowzari-Dalini, "Parallel algorithms for minimum spanning tree problem," *International Journal of Computer Mathematics*, vol. 79, no. 4, pp. 441–448, 2002.

[8] S. Chung and A. Condon, "Parallel implementation of borvka's minimum spanning tree algorithm," in *Proceedings of the 10th International Parallel Processing Symposium*, ser. IPPS '96. Washington, DC, USA: IEEE Computer Society, 1996, pp. 302–308. [Online]. Available: http://dl.acm.org/citation.cfm?id=645606.661036

[9] W. Guang-rong and G. Nai-jie, "An efficient parallel minimum spanning tree algorithm on message passing parallel machine," *Journal of Software*, vol. 11, no. 7, pp. 889–898, 2000.

[10] F. Dehne and S. Götz, "Practical parallel algorithms for minimum spanning trees," in *In Workshop on Advances in Parallel and Distributed Systems*, 1998, pp. 366–371.

[11] Y. Y. B. Deo, Narsingh, "Parallel algorithms for the minimum spanning tree problem." in *Proceedings of the International Conference on Parallel Processing*, 1981, pp. 188–189.

[12] E. Gonina and L. V. Kale, "Parallel prim's algorithm on dense graphs with a novel extension," *PPL Technical Report*, October 2007.

[13] A. N. R. Setia and S. Balachandran, "A new parallel algorithm for minimum spanning tree problem," in *Proc.International Conference on High Performance Computing (HiPC)*, 2009, pp. 1–5.

[14] D. A. Bader and G. Cong, "Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs," *J. Parallel Distrib. Comput.*, vol. 66, no. 11, pp. 1366–1378, Nov. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2006.06.001

[15] M. Jin and J. W. Baker, "Two graph algorithms on an associative computing model," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2007, Las Vegas, Nevada, USA, June 25-28, 2007, Volume 1*, 2007, pp. 271–277.

[16] V. Osipov, P. Sanders, and J. Singler, "The filter-kruskal minimum spanning tree algorithm," in *ALENEX'09*, 2009, pp. 52–61.

[17] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *Introduction to Parallel Computing (2nd Edition)*, 2nd ed. Addison Wesley, Jan. 2003. [Online]. Available: http://www.worldcat.org/isbn/0201648652

[18] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.

[19] D.-Z. Pan, Z.-B. Liu, X.-F. Ding, and Q. Zheng, "The application of union-find sets in kruskal algorithm," in *Proceedings of the 2009 International Conference on Artificial Intelligence and Computational Intelligence - Volume 02*, ser. AICI '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 159–162. [Online]. Available: http://dx.doi.org/10.1109/AICI.2009.155

[20] Z. Galil and G. F. Italiano, "Data structures and algorithms for disjoint set union problems," *ACM Comput. Surv.*, vol. 23, no. 3, pp. 319–344, Sep. 1991. [Online]. Available: http://doi.acm.org/10.1145/116873.116878

[21] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *J. Parallel Distrib. Comput.*, vol. 69, no. 2, pp. 117–124, Feb. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2008.09.002

Quench Dynamics for Trapped Dipolar Fermi Gases

V. Veljić[1], A. Balaž[1] and A. Pelster[2]
[1]*Scientific Computing Laboratory, Institute of Physics Belgrade,*
University of Belgrade, Pregrevica 118, 11080 Belgrade, Serbia
[2]*Physics Department and Research center OPTIMAS,*
Technical University of Kaiserslautern, 67663 Kaiserslautern, Germany
e-mail: vveljic@ipb.ac.rs

A recent time-of-flight expansion experiment for polarized fermionic erbium atoms managed to detect a Fermi surface deformation which is due to the dipolar interaction [1]. Here we perform a systematic study of quench dynamics of trapped dipolar Fermi gases at zero temperature, which are induced by a sudden change of the magnetic field, which enforces the polarization of the magnetic moments of the erbium atoms. As this modifies the equilibrium configuration, oscillations of the fermionic erbium cloud emerge around the new equilibrium, which are characteristic for the presence of the dipole-dipole interaction. In order to analyze the emergent dynamics we follow Ref. [2] and solve analytically the underlying Boltzmann-Vlasov equation wihtin the relaxation approximation in the vicinity of the new equilibrium configuration by using a suitable rescaling of the equilibrium distribution [3]. The resulting ordinary differential equations of motion for the scaling parameters are solved numerically for experimentally relevant parameters all the way from the collisionless to the hydrodynamic regime. A comparison with a corresponding linear stability analysis reveals that the resulting quench dynamics can be understood in terms of the low-lying collective modes due to the smallness of the dipolar interaction strength. All our theoretical and numerical calculations can be tested in current experiments with ultracold dipolar fermionic atoms.

REFERENCES
[1] K. Aikawa et al., Science 345, 1484 (2014).
[2] F. Wächtler, A. R. P. Lima, A. Pelster, arXiv: 1311.5100 (2013).
[3] P. Pedri, D. Guery-Odelin, S. Stringari, Phys. Rev. A 68, 043608 (2003).

Trapped Bose-Einstein Condensates with Strong Disorder

V. Lončar[1], A. Balaž[1] and A. Pelster[2]
[1]*Scientific Computing Laboratory, Institute of Physics Belgrade,*
University of Belgrade, Pregrevica 118, 11080 Belgrade, Serbia
[2]*Physics Department and Research center OPTIMAS,*
Technical University of Kaiserslautern, 67663 Kaiserslautern, Germany
e-mail: vloncar@ipb.ac.rs

We work out a non-perturbative approach towards the dirty boson problem at zero temperature that is based on a Gaussian approximation for correlation functions of the disorder problem and the condensate wave function solving the Gross-Pitaevskii problem. For harmonically trapped Bose-Einstein condensates we apply, in addition, the

semiclassical approximation and derive with this self-consistency equations between the disorder ensemble-averages of particle density and condensate density. Invoking, furthermore, the Thomas-Fermi approximation we obtain results that reproduce for weak disorder the seminal results of a Bogoliubov theory of dirty bosons [1-3], but do not yield for strong disorder a Bose-glass phase. Afterwards, we go beyond the Thomas-Fermi approximation and perform a full numerical treatment of the self-consistency equations based on the Crank-Nicolson split-step semi-implicit imaginary-time propagation [4], which yields a quantum phase transition to a Bose-glass phase for strong disorder [5].

REFERENCES
[1] K. Huang, H.-F. Meng, Phys. Rev. Lett. 69, 644 (1992).
[2] G. M. Falco, A. Pelster, R. Graham, Phys. Rev. A 75, 063619 (2007).
[3] G. M. Falco, A. Pelster, R. Graham, Phys. Rev. A 76, 013624 (2007).
[4] D. Vudragović et al., Comput. Phys. Comm.. 183, 2021 (2012).
[5] P. Navez, A. Pelster, R. Graham, Appl. Phys. B 86, 395 (2007).

Faraday Waves in Dipolar Bose-Einstein Condensates

D. Vudragović and A. Balaž
Scientific Computing Laboratory, Institute of Physics Belgrade,
University of Belgrade, Pregrevica 118, 11080 Belgrade, Serbia
e-mail: dusan@ipb.ac.rs

We present the emergence of Faraday waves in cigar-shaped ^{52}Cr and ^{164}Dy Bose-Einstein condensates. These density waves are induced by periodic modulation of the frequency of the trapping potential. We study through extensive numerical simulations and detailed variational treatment the effects of the strong dipolar interaction on the spatial and time-period of the Faraday waves. Unlike in the case of homogeneous [1] or inhomogeneous contact interactions [2], the emergence of Faraday waves is found to further destabilize the condensate in the presence of strong dipolar interaction. The interesting effect of spatial period variation of generated density patterns is observed numerically and studied within the Gaussian variational approach.

REFERENCES
[1] A. Balaž, A. I. Nicolin, Phys. Rev. A 85, 023613 (2012).
[2] A. Balaž et al., Phys. Rev. A 89, 023609 (2014).

Q 17: Quantum Gases: Bosons I

Time: Tuesday 11:00–13:00 Location: e001

Group Report Q 17.1 Tue 11:00 e001

Rosensweig instability and solitary waves in a dipolar Bose-Einstein condensate — •Matthias Wenzel, Holger Kadau, Matthias Schmitt, Igor Ferrier-Barbut, and Tilman Pfau — 5. Physikalisches Institut and Center for Integrated Quantum Science and Technology, Universität Stuttgart, Pfaffenwaldring 57, 70569 Stuttgart, Germany

Ferrofluids show unusual hydrodynamic effects due to the magnetic nature of their constituents. For increasing magnetization a classical ferrofluid undergoes a Rosensweig instability and creates self-organized ordered surface structures or droplet crystals.
In the experiment we observe a similar behavior in a sample of ultracold dysprosium atoms, a quantum ferrofluid. By controlling the short-range interaction with a Feshbach resonance we can induce a finite-wavelength instability due to the dipolar interaction.
Subsequently, we observe the spontaneous transition from an unstructured superfluid to an ordered arrangement of droplets by in situ imaging. These patterns are surprisingly long-lived and show hysteretic behavior. When transferring the sample to a waveguide we observe mutually interacting solitary waves. Time-of-flight measurements allow us to show the existence of an equilibrium between dipolar attraction and short-range repulsion. In addition we observe interference between droplets.
In conclusion, our system shows both superfluidity and translational symmetry breaking. This novel state of matter is thus a possible candidate for a supersolid ground state.

Q 17.2 Tue 11:30 e001

Rosensweig instability due to three-body interaction or quantum fluctuations? — Vladimir Lončar[1], Dušan Vudragović[1], •Antun Balaž[1], and Axel Pelster[2] — [1]Scientific Computing Laboratory, Institute of Physics Belgrade, University of Belgrade, Serbia — [2]Physics Department and Research Center OPTIMAS, Technical University of Kaiserslautern, Germany

In the recent experiment [1], the Rosensweig instability was observed in a ^{164}Dy Bose-Einstein condensate, which represents a quantum ferrofluid due to the large atomic magnetic dipole moments. After a sudden reduction of the scattering length, which is realized by tuning the external magnetic field far away from a Feshbach resonance, the dipolar quantum gas creates self-ordered surface structures in form of droplet crystals. As the underlying Gross-Pitaevskii equation is not able to explain the emergence of that Rosensweig instability, we extend it by both three-body interactions [2-4] and quantum fluctuations [5]. We then use extensive numerical simulations in order to study the interplay of three-body interactions as well as quantum fluctuations on the emergence of the Rosensweig instability.
[1] H. Kadau, M. Schmitt, et al., `arXiv:1508.05007v2` (2015).
[2] H. Al-Jibbouri, I. Vidanović, A. Balaž, and A. Pelster, J. Phys. B **46**, 065303 (2013).
[3] R. N. Bisset and P. B. Blakie, `arXiv:1510.09013` (2015).
[4] K.-T. Xi and H. Saito, `arXiv:1510.07842` (2015).
[5] A. R. P. Lima and A. Pelster, Phys. Rev. A **84**, 041604(R) (2011); Phys. Rev. A **86**, 063609 (2012).

Q 17.3 Tue 11:45 e001

Phonon to roton crossover and droplet formation in trapped dipolar Bose-Einstein condensates — •Falk Wächtler and Luis Santos — Institut für Theoretische Physik, Leibniz Universität Hannover, Hannover, Germany

The stability, elementary excitations, and instability dynamics of dipolar Bose-Einstein condensates depend crucially on the trap geometry. In particular, dipolar condensates in a pancake trap with its main plane orthogonal to the dipole orientation are expected to present under proper conditions a roton-like dispersion minimum, which if softening induces the so-called roton instability. On the contrary, cigar-shape traps are expected to present no dispersion minimum, and to undergo phonon (global) instability if destabilized. In this talk we investigate by means of numerical simulations of the non-local non-linear Schrödinger equation and the corresponding Bogoliubov-de Gennes equations the stability threshold as a function of the trap aspect ratio, mapping the crossover between phonon and roton instability. We will discuss in particular how this crossover may be observed in destabilization experiments to reveal rotonization.
In a second part, motivated by recent experiments on droplet formation in Stuttgart, we introduce large conservative three-body interactions, and study how these forces affect the destabilization dynamics. We will discuss the ground-state physics of the individual droplets, and the crucial role that is played by the interplay between internal droplet energy, external center-of mass energy of the droplets, and energy dissipation in the nucleation of droplets observed in experiments.

Q 17.4 Tue 12:00 e001

Lattice Physics with Ultracold Magnetic Erbium — •Simon Baier[1], Manfred J. Mark[1,2], Daniel Petter[1], Kiyotaka Aikawa[1], Lauriane Chomaz[1,2], Zi Cai[2], Mikhail Baranov[2], Peter Zoller[2,3], and Francesca Ferlaino[1,2] — [1]Institut für Experimentalphysik, Universität Innsbruck, Technikerstraße 25, 6020 Innsbruck, Austria — [2]Institut für Quantenoptik und Quanteninformation, Österreichische Akademie der Wissenschaften, 6020 Innsbruck, Austria — [3]Institut für Theoretische Physik, Universität Innsbruck, Technikerstraße 21A, 6020 Innsbruck, Austria

Strongly magnetic atoms are an ideal systems to study many-body quantum phenomena with anisotropic and long-range interactions. Here, we report on the first observation of the manifestation of magnetic dipolar interaction in extended Bose-Hubbard (eBH) dynamics by studying an ultracold gas of Er atoms in a three-dimensional optical lattice. We drive the superfluid-to-Mott-insulator (SF-to-MI) quantum phase transition and demonstrate that the dipolar interaction can favor the SF or the MI phase depending on the orientation of the atomic dipoles. The system is well described by the individual terms of the eBH Hamiltonian. This includes the onsite interaction, which, additional to the isotropic contact interaction, can be tuned with the dipole-dipole interaction by changing the dipole orientation and the shape of the onsite Wannier functions. We find for the first time the presence of the nearest-neighbor interaction between two adjacent particles. Future work will investigate dipolar effects with erbium molecules and fermions as well as spin physics in our lattice system.

Q 17.5 Tue 12:15 e001

Strong-wave-turbulence character of non-thermal fixed points in Bose gases — •Isara Chantesana[1,2,3] and Thomas Gasenzer[2,3] — [1]Institut für Theoretische Physik, Ruprecht-Karls-Universität Heidelberg, Philosophenweg 16, 69120 Heidelberg, Germany — [2]Kirchhoff Institut für Physik, INF 227, 69120 Heidelberg, Germany — [3]ExtreMe Matter Institute EMMI, GSI Helmholtzzentrum für Schwerionenforschung GmbH, Planckstraße 1, 64291 Darmstadt, Germany

Far-from equilibrium dynamics of a dilute Bose gas is studied by means of the two-particle irreducible effective action formalism. We investigate the properties of non-thermal fixed points predicted previously, which are related to non-perturbative strong wave turbulence solutions of the many-body dynamic equations. Instead of using a scaling analysis, we study the Boltzmann equation of the scattering integral by means of direct integration equation for sound waves. In this way we obtain a direct prediction of the scaling behaviour of the possible fixed-point solutions in the context of sound-wave turbulence. Implication for the real-time dynamics of the non-equilibrium system are discussed.

Q 17.6 Tue 12:30 e001

Evidence of Non-Thermal Fixed Points in one-dimensional Bose gases — •Sebastian Erne[1,2,4], Robert Bücker[4], Wolfgang Rohringer[4], Thomas Gasenzer[1,2,3], and Jörg Schmiedmayer[4] — [1]Institut für Theoretische Physik, Ruprecht-Karls-Universität Heidelberg, Philosophenweg 16, 69120 Heidelberg, Germany — [2]ExtreMe Matter Institute EMMI, GSI Helmholtzzentrum für Schwerionenforschung GmbH, Planckstraße 1, 64291 Darmstadt, Germany — [3]Kirchhoff-Institut für Physik, INF 227, 69120 Heidelberg, Germany — [4]Vienna Center for Quantum Science and Technology (VCQ), Atominstitut, TU Wien, Vienna, Austria

This work investigates the rapid cooling quench over the dimensional- and quasicondensate-crossover. Analyzing experiments performed at the Atominstitut, we study the relaxation of such a far-from equilibrium system. The early stage of condensate formation is dominated by solitonic excitations, leading to a characteristic momentum distribution in agreement with a model of randomly distributed defects. The number of solitons increases with the quenchrate giving rise to an incompressible condensate. The isolated system follows a self-similar evolution governed by a universal time-independent nonthermal fixed point distribution. The dynamic universality classes of these nonequilibrium attractor solutions are relevant for a wide variety of physical systems ranging from relativistic high-energy physics to cold quantum gases. At later times of the evolution the system fully equilibrates leading to deviations from the self-similar evolution. Our results show a new way of condensation in far from equilibrium 1d Bose gases.

Q 17.7 Tue 12:45 e001

Spin phonon dynamics with classical statistical methods

— •Asier Piñeiro Orioli[1,2], Arghavan Safavi-Naini[2], Michael Wall[2], and Johannes Schachenmayer[2] — [1]Institute for Theoretical Physics, Heidelberg, Germany — [2]JILA, NIST and University of Colorado, Boulder, Colorado, USA

Systems with both spin and phonon degrees of freedom are ubiquitous in physical fields ranging from condensed matter to biophysics. However, methods to compute the dynamics of such systems are scarce, especially in high dimensions. In this work, we combine the Truncated Wigner Approximation (TWA) for bosons with its recently developed discrete version (dTWA) for spins to describe the dynamics of coupled spin-phonon systems. We benchmark the method by comparing to exact results and discuss applications to trapped-ion and cavity experiments.

УНИВЕРЗИТЕТ У НОВОМ САДУ
ПРИРОДНО-МАТЕМАТИЧКИ ФАКУЛТЕТ

ПРИМЉЕНО: 16 -10- 2013	
ОРГАНИЗ.ЈЕД	БРОЈ
0601	51/311

УНИВЕРЗИТЕТ У НОВОМ САДУ
ПРИРОДНО-МАТЕМАТИЧКИ ФАКУЛТЕТ
Департман за математику и информатику
Нови Сад, Трг Доситеја Обрадовића 4
Број: 01-2271/1
Датум: 14.10.2013.

На основу члана 70. став 2. Закона о научно-истраживачкој делатности („Сл.гласник РС" број 110/05, 50/06 – исправка и 18/2010) и члана 76. став 2. и члана 126. Статута Природно-математичког факултета у Новом Саду, Универзитета у Новом Саду, Научно веће Департмана за математику и информатику Природно-математичког факултета у Новом Саду, Универзитета у Новом Саду, донело је следећу

О Д Л У К У

ВЛАДИМИР ЛОНЧАР, бира се у звање **ИСТРАЖИВАЧА-САРАДНИКА за ужу научна област Информациони системи** на Природно-математичком факултету у Новом Саду на **три** године почевши **од 15. oktobra 2013.** године.

О б р а з л о ж е њ е

На предлог Већа **Департмана за математику и информатику од 12.06.2013.** године декан Природно-математичког факултета је именовао Комисију за писање извештаја за избор у звање једног **ИСТРАЖИВАЧА-САРАДНИКА за ужу научну област Информациони системи**, у саставу:

1. др Срђан Шкрбић, доцент ПМФ у Новом Саду – председник
2. др Данијела Боберић Крстићев, доцент ПМФ у Новом Саду – члан
3. др Милан Видаковић, ванредни професор ФТН у Новом Саду – члан

Комисија је извештај доставила **27.08.2013. године** у коме је навела да се на објављени конкурс пријавио кандидат **Владимир Лончар**, који испуњава услове утврђене конкурсом као и предлог да се кандидат **Владимир Лончар** изабере у звање **истраживача-сарадника за ужу научну област Информациони системи.**

Извештај је објављен у «Билтену» бр. **1448 od 15.09.2013**. године.

Научно веће **Департмана за математику и информатику** на својој седници дана **10. oktobra 2013. године** разматрало је извештај Комисије и донело одлуку да се **Владимир Лончар** изабере у звање **истраживача-сарадника.**

На основу свега изнетог донета је Одлука као у диспозитиву.

ПРЕДСЕДНИК НАУЧНОГ ВЕЋА
ДЕПАРТМАНА ЗА МАТЕМАТИКУ И ИНФОРМАТИКУ

[illegible]

Др Бранимир Шешеља, редовни професор

Доставити:
1. кандидату
2. Департману за **математику и информатику**
[✓ 3.] персонални досије
4. архиви.

УНИВЕРЗИТЕТ У НОВОМ САДУ
Др Зорана Ђинђића 1
21000 Нови Сад
Република Србија

UNIVERSITAS STUDIORUM NEOPLANTENSIS MCMLX

UNIVERSITY OF NOVI SAD
Dr. Zorana Đinđića 1
21000 Novi Sad
Republic of Serbia

Телефон / Phone: +381 21 485 2000, 485 2020 | Факс / Fax: +381 21 450 418 | E-mail: rektorat@uns.ac.rs | http://www.uns.ac.rs

Број: 04-29/11
Датум: 02. јун 2016. године

УНИВЕРЗИТЕТ У НОВОМ САДУ
ПРИРОДНО-МАТЕМАТИЧКИ ФАКУЛТЕТ

ПРИМЉЕНО	- 9 - 06 - 2016
ОРГАНИЗ.ЈЕД	БРОЈ
0603	193/10

На основу члана 55 став 6 Закона о високом образовању ("Службени гласник РС" број 76/05, 100/07 - аутентично тумачење, 97/08, 44/10, 93/12, 89/13, 99/14, 45/15-аутентично тумачење и 68/15) и члана 73 и 78 Статута Универзитета у Новом Саду (Савет Универзитета, 28.12.2010. године, 23.03.2012. године, 11.10.2012. године, 26.02.2013. године, 15.11.2013. године, 30.05.2014. године, 04.06.2015. године и 29.01.2016. године), Сенат Универзитета у Новом Саду на 11. седници одржаној 02. јуна 2016. године, једногласно доноси

ОДЛУКУ

Сенат Универзитета даје сагласност на Извештај о подобности теме, кандидата и ментора за израду докторске дисертације на Природно-математичком факултету Универзитета у Новом Саду под називом: „Хибридни паралелни алгоритми за решавање нелинеарне Шредингерове једначине" „*Hybrid parallel algorithms for solving nonlinear Schroedinger equation*", а кандидату **Владимиру Лончару** одобрава израду докторске дисертације.

Образложење

Наставно-научно веће Природно-математичког факултета Универзитета у Новом Саду на седници одржаној 12. маја 2016. године усвојило је извештај о подобности теме, кандидата и ментора за израду докторске дисертације Владимира Лончара под називом: „Хибридни паралелни алгоритми за решавање нелинеарне Шредингерове једначине" „*Hybrid parallel algorithms for solving nonlinear Schroedinger equation*".

Стручно веће за природно-математичке науке на седници одржаној 26. маја 2016. године дало је позитивно мишљење о испуњености услова за давање сагласности на наведени извештај.

На основу одлуке Наставно-научног већа Природно-математичког факултета Универзитета у Новом Саду и позитивног мишљења Стручног већа за природно-математичке науке, Сенат Универзитета у Новом Саду донео је одлуку као у диспозитиву.

Република Србија Аутономна Покрајина Војводина Универзитет у Новом Саду Нови Сад
Republika Srbija Autonomna pokrajina Vojvodina Univerzitet u Novom Sadu Novi Sad

ПРЕДСЕДНИК СЕНАТА

[illegible]

Проф. др Душан Николић

Republika Srbija - AP Vojvodina
Univerzitet u Novom Sadu
Prirodno-matematički fakultet
Novi Sad

Uverenje br.: 267/2016
Broj dosijea: 72d/11

Na osnovu čl.161 Zakona o opštem upravnom postupku (Službeni list SRJ br. 33/97) i po molbi Lončar (Milenko) Vladimir, rođenog 28.10.1985. u mestu Novi Sad, Republika Srbija, redovan student (samofinansiranje) 3. godine (godinu upisao 3. put) školske 2015/2016 na Departmanu za matematiku i informatiku, obrazovni profil Doktorske akademske studije informatike, izdaje se

U V E R E NJ E

kojim se potvrđuje da je imenovani položio ispite iz sledećih predmeta:

		Ocena	Bodovi
1.	Seminar 1..........................	10 (deset)	20.00 (dvadeset 00/100)
2.	Modeliranje sistema................	10 (deset)	5.00 (pet 00/100)
3.	Programske paradigme...............	10 (deset)	5.00 (pet 00/100)
4.	Metodi istraživanja................	9 (devet)	5.00 (pet 00/100)
5.	Razvoj zasnovan na komponentama....	10 (deset)	5.00 (pet 00/100)
6.	Bezbednost u računarskim mrežama...	10 (deset)	5.00 (pet 00/100)
7.	Algebarski i koalgebarski modeli računskih procesa.................	10 (deset)	5.00 (pet 00/100)
8.	Seminar 2..........................	10 (deset)	20.00 (dvadeset 00/100)
9.	Baze podataka*.....................	10 (deset)	5.00 (pet 00/100)
10.	Razvoj sistema.....................	10 (deset)	5.00 (pet 00/100)
11.	Seminar 3..........................	10 (deset)	20.00 (dvadeset 00/100)
12.	Seminar 4..........................	10 (deset)	20.00 (dvadeset 00/100)

Položeno 12 ispita sa pros. ocenom 9.92, i brojem bodova 120.00

Uverenje se izdaje na lični zahtev imenovanog.

Република Србија
Аутономна Покрајина Војводина
Универзитет у Новом Саду
Природно-математички факултет
XIV
Нови Сад

Novi Sad, 05.05.2016.

Referent
[illegible]
Saša Kočiš

UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI FAKULTET
Broj: 0603-3/930
Datum: 20.10.2011. godine
Novi Sad

Na osnovu člana 161. Zakona o opštem upravnom postupku ("Sl. list SRJ" br. 33/97, i 31/2001) i ("Sl. glasnik RS" br. 30/2010), u skladu sa članom 99. Zakona o visokom obrazovanju ("Sl. glasnik RS" br. 76/05, 100/07 autentično tumačenje 97/08 i 44/2010), uvida u matične knjige studenata master akademskih studija Prirodno-matematičkog fakulteta Univerziteta u Novom Sadu i zahteva Lončar Milenko Vladimira, iz Novog Sada izdaje se

U V E R E NJ E

O STEČENOM VISOKOM OBRAZOVANJU DRUGOG STEPENA MASTER AKADEMSKIH STUDIJA

Lončar (Milenko) Vladimir

rođen 28.10.1985. godine u Novom Sadu, opština Novi Sad, država Republika Srbija, završio je visoko obrazovanje drugog stepena-master akademskih studija, na studijskom programu Master akademske studije Informacione tehnologije Departmana za matematiku i informatiku Prirodno-matematičkog fakulteta Univerziteta u Novom Sadu, dana 19.10.2011. godine, sa prosečnom ocenom 9.50 (devet i 50/100), u toku studija i postignutim ukupnim brojem ESPB bodova 122.50 (slovima: sto dvadeset dva i 50/100) i stekao akademski naziv master informatičar-informacione tehnologije .

Uverenje se izdaje radi lične upotrebe i zamenjuje diplomu do izdavanja iste.

Na osnovu člana 19. stav 1. tačka 7. Zakona o republičkim administrativnim taksama ("Sl. glasnik RS" broj 43/2003, 51/2003 - ispr., 61/2005, 101/2005 - dr. zakon, 5/2009 i 54/2009) ovo uverenje je oslobođeno takse.

Šef Odseka za studentske i opšte poslove
Prirodno-matematičkog fakulteta

[illegible]

Tamara Zorić

Republika Srbija - AP Vojvodina
Univerzitet u Novom Sadu
Prirodno-matematički fakultet
Novi Sad

Uverenje br.: 1274/2014
Broj dosijea: 127m/09

Na osnovu čl.161 Zakona o opštem upravnom postupku (Službeni list SRJ br. 33/97) i po molbi Lončar (Milenko) Vladimir, rođenog 28.10.1985. u mestu Novi Sad, Republika Srbija, diplomiranog studenta-master na Departmanu za matematiku i informatiku, obrazovni profil Master akademske studije Informacione tehnologije (inf.sist), izdaje se

U V E R E NJ E

kojim se potvrđuje da je imenovani položio ispite iz sledećih predmeta:

		Ocena	Bodovi	
1.	Seminarski rad C...................	10 (deset)	6.00 (šest 00/100)	-priznat
2.	Informatički projekat..............	10 (deset)	10.00 (deset 00/100)	-priznat
3.	Teorija grafova....................	6 (šest)	6.00 (šest 00/100)	-priznat
4.	Matematička logika u računarstvu...	8 (osam)	6.00 (šest 00/100)	-priznat
5.	Upravljanje softverskim projektima.	10 (deset)	8.00 (osam 00/100)	-priznat
6.	Seminarski rad D...................	10 (deset)	6.00 (šest 00/100)	-priznat
7.	Obrazovni softver..................	10 (deset)	7.00 (sedam 00/100)	-priznat
8.	Integracija sistema................	10 (deset)	7.50 (sedam 50/100)	
9.	Proces razvoja informacionih sistema...........................	10 (deset)	7.50 (sedam 50/100)	-priznat
10.	Statistika.........................	10 (deset)	6.00 (šest 00/100)	-priznat
11.	Razvoj zasnovan na komponentama....	10 (deset)	7.50 (sedam 50/100)	
12.	Privatnost, etika i društvena odgovornost.......................	9 (devet)	7.50 (sedam 50/100)	
13.	Softversko inženjerstvo za sisteme baza podataka.....................	10 (deset)	7.50 (sedam 50/100)	

Položeno 13 ispita sa pros. ocenom 9.46, i brojem bodova 92.50

Minimalan broj bodova za odbranu završnog (master) rada je 90.00.

Uverenje se izdaje na lični zahtev imenovanog.

Napomena: Student je završio master akademske studije dana 19.10.2011. godine sa prosečnom ocenom 9.50 (devet i 50/100), brojem bodova završnog rada 30.00 (trideset i 00/100) i ocenom završnog rada 10 (deset).

Novi Sad, 27.11.2014.

Република Србија
XIII
Нови Сад

Referent
[illegible]
Saša Kočiš

UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI FAKULTET
Broj:0603-3/590
Datum:27.11.2009. godine
Novi Sad

Na osnovu čl.161 Zakona o opštem upravnom postupku (Službeni list SRJ br. 33/97) i na osnovu člana 126 Zakona o visokom obrazovanju i uvida u matičnu knjigu studenata Prirodno-matematičkog fakulteta Univerziteta u Novom Sadu i zahteva Lončar Milenko Vladimira izdaje se

U V E R E NJ E
O STEČENOM VISOKOM OBRAZOVANJU

Lončar (Milenko) Vladimir rođen 28.10.1985. godine u Novom Sadu, opština Novi Sad, Republika Srbija završio je visoko obrazovanje u trajanju od četiri godine, po Planu i programu obrazovnog rada Departmana za matematiku i informatiku smer Diplomirani informatičar-poslovna informatika Prirodno-matematičkog fakulteta Univerziteta u Novom Sadu, dana 14.10.2009. godine, sa prosečnom ocenom 8.79 (osam 79/100) i ocenom završnog ispita 10 (deset) i stekao stručni naziv Diplomirani informatičar-poslovna informatika.

Uverenje se izdaje radi lične upotrebe i zamenjuje diplomu do izdavanja iste.

Shodno članu 19 stav 1 tačka 7. Zakona o administrativnim taksama (Sl. glasnik Republike Srbije broj 43/03, 51/03 i 61/05) ovo uverenje je oslobođeno takse.

Novi Sad, 27.11.2014.

Република [illegible] XIII Нови Сад

Referent
[illegible]
Šaša Kočiš

Republika Srbija - AP Vojvodina
Univerzitet u Novom Sadu
Prirodno-matematički fakultet
Novi Sad

Uverenje br.: 124/2014
Broj dosijea: 407/04

Na osnovu čl.161 Zakona o opštem upravnom postupku (Službeni list SRJ br. 33/97) i po molbi Lončar (Milenko) Vladimir, rođenog 28.10.1985. u mestu Novi Sad, , diplomiranog studenta na Departmanu za matematiku i informatiku, obrazovni profil Diplomirani informatičar-poslovna informatika, izdaje se

U V E R E NJ E

kojim se potvrđuje da je imenovani položio ispite iz sledećih predmeta:

		Ocena	Bodovi
	1. godina		
1.	Strukture podataka i algoritmi I...	8 (osam)	6.00 (šest 00/100)
2.	Izborni predmet F-Finansijska matematika I......................	6 (šest)	5.00 (pet 00/100)
3.	Elektronsko poslovanje............	9 (devet)	3.00 (tri 00/100)
4.	Algebra za informatičare..........	7 (sedam)	6.00 (šest 00/100)
5.	Etički aspekti informatike........	8 (osam)	3.00 (tri 00/100)
6.	Teorijski osnovi informatike I....	8 (osam)	7.00 (sedam 00/100)
7.	Analiza za informatičare..........	8 (osam)	6.00 (šest 00/100)
8.	Uvod u programiranje..............	6 (šest)	8.00 (osam 00/100)
9.	Softverski praktikum I - Kancelarijsko poslovanje..........	9 (devet)	4.00 (četiri 00/100)
10.	Kombinatorika i teorija grafova....	6 (šest)	15.00 (petnaest 00/100)
11.	Softverski praktikum - Vizuelno programiranje.....................	10 (deset)	8.00 (osam 00/100)
12.	Sociologija.......................	6 (šest)	2.00 (dva 00/100)
	2. godina		
13.	Engleski jezik-produžni kurs.......	10 (deset)	4.00 (četiri 00/100)
14.	Objektno-orijentisano programiranje	8 (osam)	7.00 (sedam 00/100)
15.	Baze podataka I...................	8 (osam)	7.00 (sedam 00/100)
16.	Organizacija računara.............	8 (osam)	5.00 (pet 00/100)
17.	Izborni seminar I.................	10 (deset)	5.00 (pet 00/100)
18.	Izborni predmet iz grupe I-1-Baze podataka II......................	8 (osam)	5.00 (pet 00/100)
19.	Teorijski osnovi informatike III...	9 (devet)	5.00 (pet 00/100)
20.	Teorijski osnovi informatike II....	8 (osam)	5.00 (pet 00/100)
21.	Numerička analiza.................	10 (deset)	5.00 (pet 00/100)
22.	Računarske mreže..................	10 (deset)	5.00 (pet 00/100)
	3. godina		
23.	Informacioni sistemi I............	10 (deset)	7.00 (sedam 00/100)
24.	Izborni seminar II................	10 (deset)	6.00 (šest 00/100)
25.	Statistika........................	10 (deset)	6.00 (šest 00/100)
26.	Izborni predmet grupe I-3-Obrazovni softver..........................	10 (deset)	5.00 (pet 00/100)
27.	Izborni predmet grupe I-3-Informacioni sistemi II.......	10 (deset)	5.00 (pet 00/100)
28.	Izborni predmet grupe I-3-Operativni sistemi II.........	10 (deset)	6.00 (šest 00/100)
29.	Izborni predmet grupe I-3-Upravljanje softverskim projekt	10 (deset)	5.00 (pet 00/100)
30.	Softverski praktikum III - Multimedija.......................	10 (deset)	4.00 (četiri 00/100)
31.	Izborni predmet grupe I-2-Veštačka inteligencija.....................	10 (deset)	7.00 (sedam 00/100)
32.	Izborni predmet grupe I-2-Operativni sistemi I..........	8 (osam)	7.00 (sedam 00/100)
33.	Izborni predmet M-Analiza II......	8 (osam)	5.00 (pet 00/100)
	4. godina		
34.	Praksa............................	10 (deset)	15.00 (petnaest 00/100)
35.	Softversko inženjerstvo...........	8 (osam)	9.00 (devet 00/100)
36.	Izborni seminar III...............	10 (deset)	8.00 (osam 00/100)
37.	Napredni softverski praktikum II-Web dizajn....................	10 (deset)	8.00 (osam 00/100)

Република Србија
Аутономна Покрајина Војводина
Универзитет у Новом Саду
Природно-математички факултет
Нови Сад
XIII

```
--------------------------------------------------------------------------------
1. godina - Položeno 12 ispita sa pros. ocenom   7.58, i brojem bodova 73.00
2. godina - Položeno 10 ispita sa pros. ocenom   8.90, i brojem bodova 53.00
3. godina - Položeno 11 ispita sa pros. ocenom   9.64, i brojem bodova 63.00
4. godina - Položeno  4 ispita sa pros. ocenom   9.50, i brojem bodova 40.00
 Ukupno položeno 37 ispita sa pros. ocenom  8.76, i ukupnim brojem bodova 229.00
--------------------------------------------------------------------------------
```

Minimalan broj bodova za diplomiranje je 224.

Uverenje se izdaje na lični zahtev imenovanog.

Napomena: Student je diplomirao dana 14.10.2009. godine sa prosečnom ocenom 8.79 (osam 79/100), brojem bodova diplomskog rada 20.00 (dvadeset 00/100) i ocenom diplomskog rada 10 (deset).

Novi Sad, 27.11.2014.

Република Србија
Аутономна Покрајина Војводина
Универзитет у Новом Саду
Природно-математички факултет
Нови Сад
XIII

Referent

[illegible]

Saša Kočiš