Computational technologies for the optimization of ultra-large atomic-molecular clusters

Anton Anikin,
Pavel Sorokovikov, Aleksander Gornov

ISDCT SB RAS, Irkutsk, Russia

April 21, 2021

The problem

A large number of models focused on different types and microstates of substances:

- Metal clusters
- Gas clusters
- Molecular clusters
- Biomolecules
- Ionic clusters
- Binary clusters
- Supercooled liquids
- ...

The Cambridge Cluster Database (CCD)
`http://www-wales.ch.cam.ac.uk/CCD.html`

The problem

- "Unstructured" clusters
  - Morse potential
  - Gupta potential
  - Sutton-Chen potential
  - Z1
- "Structured" clusters
  - Keating potential

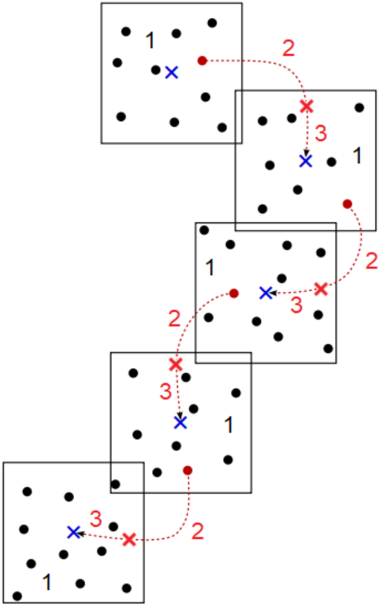
"Unstructured" clusters

- No cluster structure - "any-to-any" interaction
- Global optimization problem
- No "good" start point
- Huge number of extremes
- Large dimension of the problem

Three-phase computational technology

1. Generation of initial points (a set of algorithms-generators is implemented)
2. Descent from the first-phase points using one of the "starter algorithms"
3. Local descent from the point obtained at the second phase (using the L-BFGS method)

Three-phase computational technology



Phases:

1. Generator
2. "Starter"
3. Local descent

Three-phase computational technology

Phase 1 - generators:

1. "Level" Generator (LG)
2. "Averaged" Generator (AG)
3. "Gradient" Generator (GG)
4. combined "Level-Averaged" Generator (LAG)
5. combined "Level-Gradient" Generator (LGG)
6. combined "Averaged-Gradient" Generator (AGG)
7. combined "Level-Averaged-Gradient" Generator (LAGG)

Three-phase computational technology

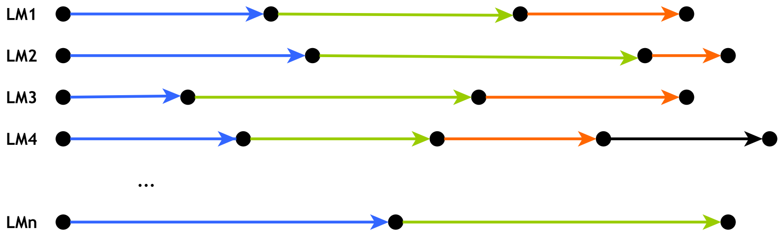
Phase 2 - starters:

1. Modification of Polyak method ("Polyak")
2. Gradient-type decomposition method ("Raider")
3. Modification of the multidimensional dichotomy method (DHTM)

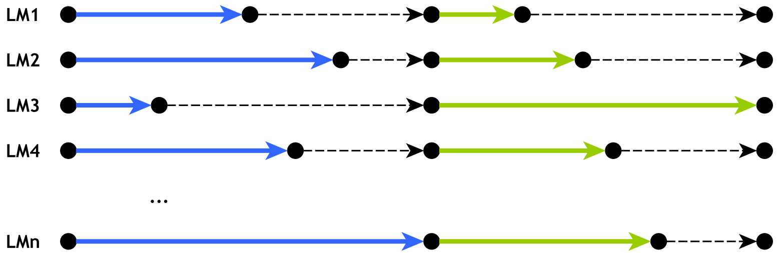
Phase 3 - local descent:

1. Limited memory Broyden–Fletcher–Goldfarb–Shanno method (L-BFGS)

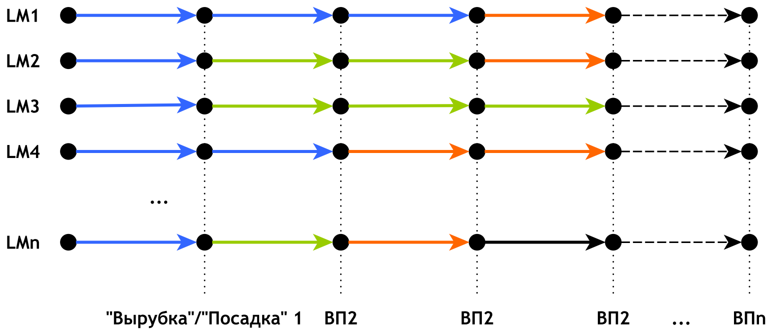
MSBH method (CPU)



MSBH method (GPU)



"Forest" method (CPU, GPU)



The method was developed by authors for efficiently executing on modern GPUs.

Numerical experiments

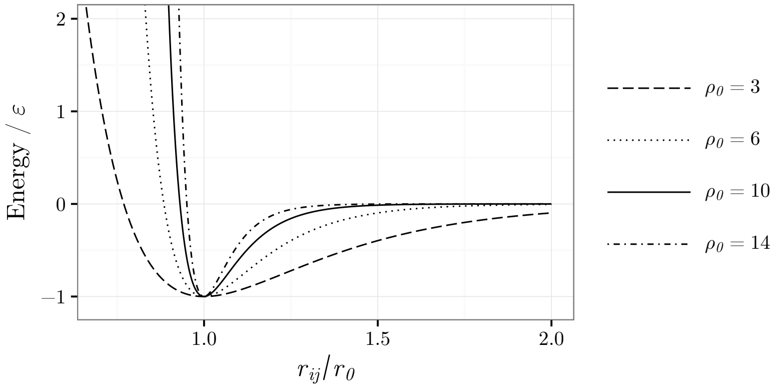
- 2 x Intel E5-2680 v2 2.8 GHz; total 20 cores, 40 threads
- 128 Gb DDR3 1866 MHz
- 3 x GeForce GTX 1060 6GB (1280 CUDA Cores)
- gcc-9.3.0
- CUDA toolkit 11.2.152

Morse potential

$$E(x) = \varepsilon \sum_{i=1}^{N} \sum_{j>i} e^{\rho_0 \left(1 - \frac{r_{ij}}{r_0}\right)} \left(e^{\rho_0 \left(1 - \frac{r_{ij}}{r_0}\right)} - 2 \right),$$

where
N - atoms count,
$r_{ij} = \|x_i - x_j\|_2$ - interatomic distance between atoms i and j,
$x_i \in \mathbb{R}^3$ - coordinate of atom i.

Morse potential



Morse potential

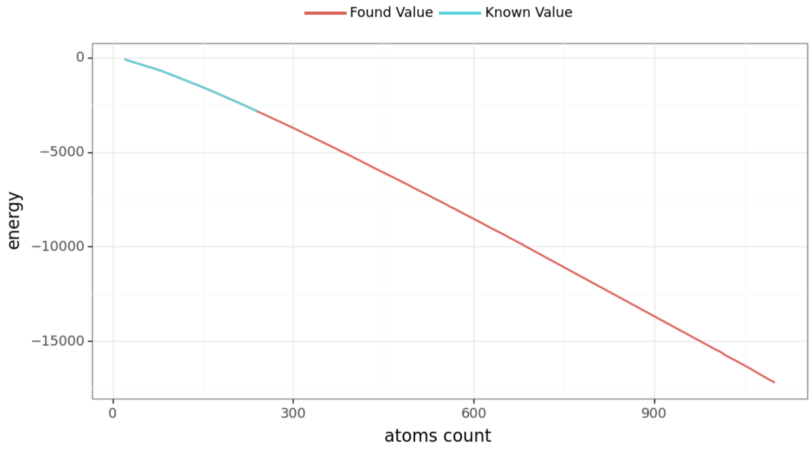
n	UK (CCD)	ISDCT
20	-97.417393	-97.41739307417
80	-690.577890	-690.5778902004155952
147	-1531.498857	-1531.498857189995761

n	CN	ISDCT
150	-1570.956895	-1570.956894507743300
170	-1842.786500	-1842.786499541551848
190	-2119.104888	-2119.104888297832076
210	-2400.884161	-2400.884161410538582
230	-2691.174648	-2691.174648208746930
240	-2839.054430	-2839.099924748702961

Morse potential

241	-2852.824893
300	-3728.226966
400	-5280.028313
500	-6893.650168
600	-8544.026103
...	
1000	-15424.2310457
1010	-15578.99495368
1020	-15792.99574419
1030	-15954.77107289
1040	-16125.34627164
1050	-16300.25804006
1060	-16472.51659678
1070	-16668.24019803
1080	-16846.84468081
1090	-17031.37978206
1100	-17199.62917956

Morse potential

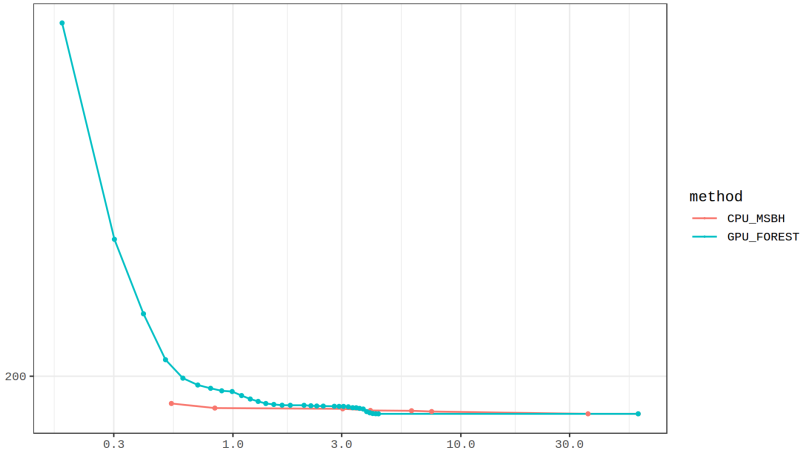


GPU-accelerated Forset method

Test case:

- Morse cluster with 90 atoms (270 optimized variables)
- Same start point for both methods
- Work time – 60 seconds
- CPU algorithm – MSBH + LBFGS with history parameter $m = 3$. 40 CPU cores are used
- GPU algorithm – Forest + LBFGS with history parameter $m = 3$. 1280 CUDA cores are used

GPU-accelerated Forset method



Function values are increased (+1000) for logarithmic scale, first iteration is removed for a more visual presentation

GPU-accelerated Forset method

- CPU MSBH $f_* = -807.442$, value found in 37 seconds
- GPU Forest $f_* = -807.442$, value found in 4.3 seconds

Both methods found point with same function value (same potential energy) but GPU version done this much faster

CUDA code tricks

- Simplify the CUDA kernels code
- Avoid "`if .. then ... else ...`" with complex branches body
- Use `float` instead `double` if possible
- Provide concurrent CPU and GPU code execution
- Fix memory access patterns
- Fix CPU/GPU memory allocations (pinned memory, etc)
- Reduce data exchange between CPU and GPU
- ...

Gupta potential (Zn)

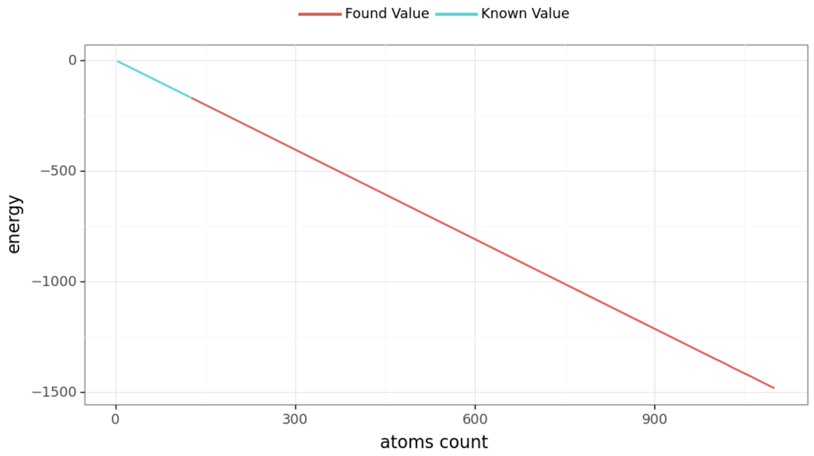
$$E(x) = \sum_{i=1}^{N} \left(A \sum_{j>i} e^{p\left(1-\frac{r_{ij}}{r_0}\right)} - \sqrt{\sum_{j>i} e^{2q\left(1-\frac{r_{ij}}{r_0}\right)}} \right),$$

$$A = 0.1477, \quad p = 9.689, \quad q = 4.602, \quad r_0 = 1.$$

Gupta potential (Zn)

126	-168.8758040477
127	-170.1994387092
128	-171.5821078823
129	-172.8529959138
130	-174.2539299691
...	
1000	-1348.558410283
1010	-1360.966877096
1020	-1374.326979101
1030	-1389.157734306
1040	-1401.817725396
1050	-1415.482970466
1060	-1428.123259985
1070	-1441.912303785
1080	-1455.839102625
1090	-1469.074670291
1100	-1482.238525141

Gupta potential (Zn)



Sutton-Chen potential

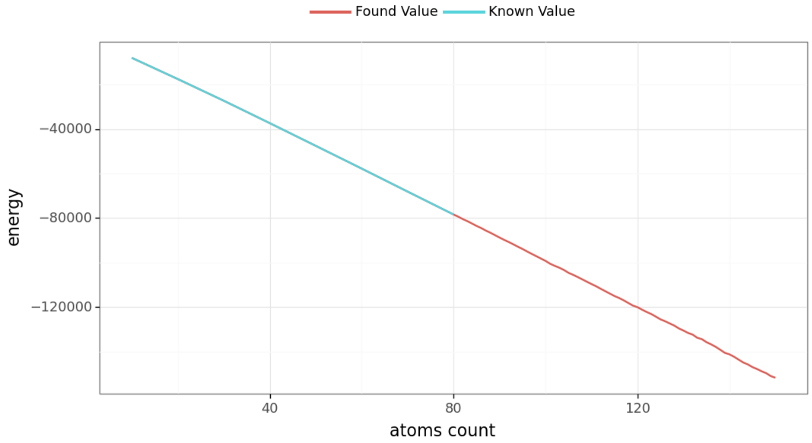
$$E(x) = \varepsilon \sum_{i=1}^{N} \left(\sum_{j>i} \left(\frac{a}{r_{ij}} \right)^{n} - 2c \sqrt{\sum_{j>i} \left(\frac{a}{r_{ij}} \right)^{m}} \right),$$

$$\varepsilon = 1, \quad c = 144.41, \quad a = 1, \quad n = 12, \quad m = 6.$$

Sutton-Chen potential

81	-79382.0486
82	-80478.4711
83	-81403.5270
84	-82468.2286
...	
140	-141271.7880524
141	-142357.5062035
142	-143658.4130983
143	-144933.391532
144	-145805.6373165
145	-147004.2697545
146	-147875.0114186
147	-148892.0975901
148	-149753.7148217
149	-151053.4417848
150	-151808.6281567

Sutton-Chen potential



Z1 potential

$$E(x) = \sum_{i=1}^{N} \sum_{j>i}^{N} V(r_{ij}),$$

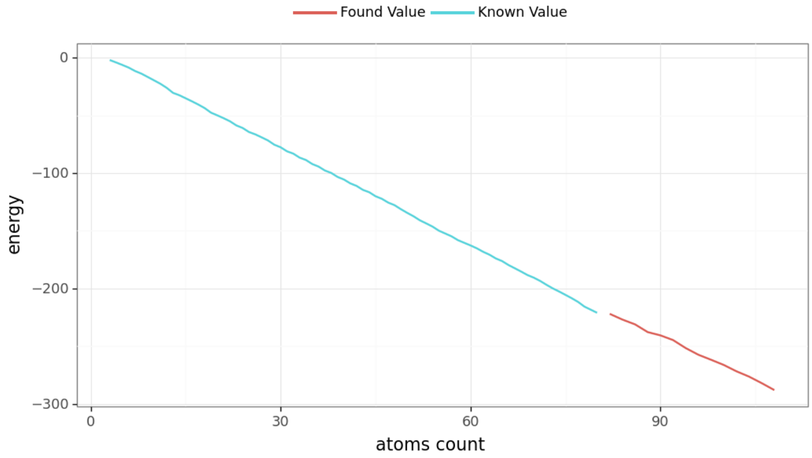
$$V(r_{ij}) = \begin{cases} a \dfrac{e^{\alpha r_{ij}}}{r_{ij}^3} \cos(2k_F r_{ij}) + b \left(\dfrac{\sigma}{r_{ij}}\right)^p + V_0 & , r_{ij} < r_c, \\ 0 & , r_{ij} \geq r_c \end{cases},$$

$$a = 1.58, \quad \alpha = -0.22, \quad k_F = 4.12, \quad b = 4.2 \cdot 10^8,$$
$$\sigma = 0.331, \quad p = 18, \quad r_c = 2.64909, \quad V_0 = 0.04682632.$$

Z1 potential

82	-221.899112
84	-226.839896
86	-231.014856
88	-237.705236
90	-240.507964
92	-244.586360
94	-251.526160
96	-257.295601
98	-261.636343
100	-266.109069
102	-271.594312
104	-276.220837
106	-281.869872
108	-287.898120

Z1 potential



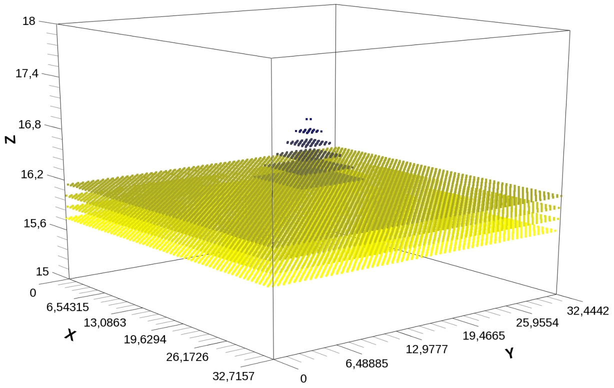
Keating potential

$$E = \frac{3}{8} \sum_{i=1}^{n} \left[\frac{1}{2} \sum_{j=1}^{4} \frac{\alpha_{ij}}{d_{ij}^2} \left(\|r_i - r_j\|_2^2 - d_{ij}^2 \right)^2 + \right.$$

$$\left. + \sum_{j=1}^{4} \sum_{k=j+1}^{4} \frac{\beta_{ijk}}{d_{ij} \cdot d_{ik}} \left(\langle r_i - r_j, r_i - r_k \rangle + \frac{d_{ij} \cdot d_{ik}}{3} \right)^2 \right]$$

where:
n – number of atoms in the crystal lattice;
α_{ij}, β_{ijk}, d_{ij}, d_{ik} – predefined constants (bond properties);
$r_i = (x_i, x_{2i}, x_{3i})$ – position of i-th atom (optimized variables).

Si-Ge quantum point



Problem properties

- The cluster structure is present
- The function is non-convex
- The function is smooth
- The analytical form for the function and it's gradient is known
- High dimensions – more than a million variables for actual problems
- Computational complexity (for "big" problems)

Optimization methods

- Conjugate gradients methods (CG)

- Quasi-Newton methods (L-BFGS)

Methods speedup

For the selected methods, the main task was to speed up the local descent procedure:

- Function/gradient calculation speedup
- Line-search (LS) procedures speedup
- Linear algebra (BLAS) procedures speedup

Function/gradient speedup

The function/gradient calculation procedures are accelerated with OpenMP parallel programming technology:

- When calculating the function, parallelization is performed for the "main" loop
- When calculating a gradient, parallelization is performed by independently calculating the individual components $\partial f / \partial x_i$

Linear algebra (BLAS) procedures speedup

All used BLAS procedures (level 1) also accelerated with OpenMP technology

Line-search methods

- Brent's method
- Simple adaptive method – uses $(0 < k^{(-)} < 1)$ and $(k^{(+)} > 1)$ to correct start step h
- Parabolic approximation 2-point method – uses $f(x^k)$, $\nabla f(x^k)$ and $f(x^k + h \cdot d^k)$ to build approximation parabola
- Parabolic approximation 3-point method – uses $f(x^k)$, $f(x^k + h \cdot d^k)$ and $f(x^k + 2 \cdot h \cdot d^k)$ to build approximation parabola

Line-search methods

The tests performed showed that the best results (in terms of speed) are achieved when using the 2-point parabolic method. This is achieved by the minimum number of function calculations – usually only **1** additional one is required.

Additional speed-up is also achieved by using the **step history** – the starting value of the line-search step is the value found in the previous iteration.

Numerical experiments

- 8-core Intel Core i7-9700KF 3.6 GHz (Turbo Boost up to 4.9 GHz)
- 64 Gb DDR4 2666 MHz
- Ubuntu 20.04.2 LTS
- gcc-9.3.0
- Release build, O2-optimization

Numerical experiments

The same criterion for stopping the calculation is set for all tested problem/method combinations:

$$\|\nabla f(x^k)\|_2 \leq \varepsilon_g$$

$$\varepsilon_g = 10^{-4}$$

Numerical experiments, LBFGS, OpenMP

Problem has $3 \cdot 10^6$ variables
8 cores are used for OpenMP acceleration

Acceleration			time, sec.	iterations
$f(x)$	$\nabla f(x)$	BLAS		
-	-	-	270.15	742
-	-	+	258.82	758
+	-	-	251.18	816
+	-	+	231.36	816
-	+	-	124.18	751
-	+	+	108.06	749
+	+	-	88.77	817
+	+	+	68.64	816

The final acceleration – almost 4 times

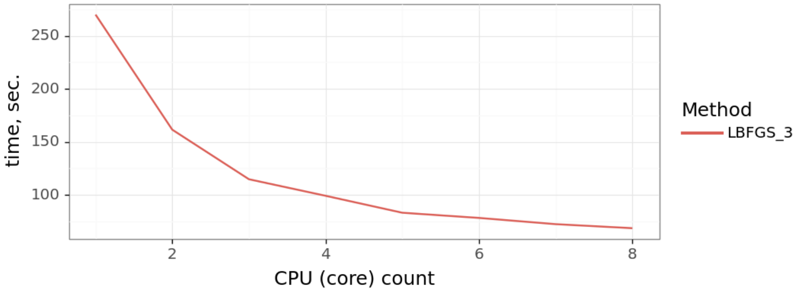
Numerical experiments, LBFGS, OpenMP

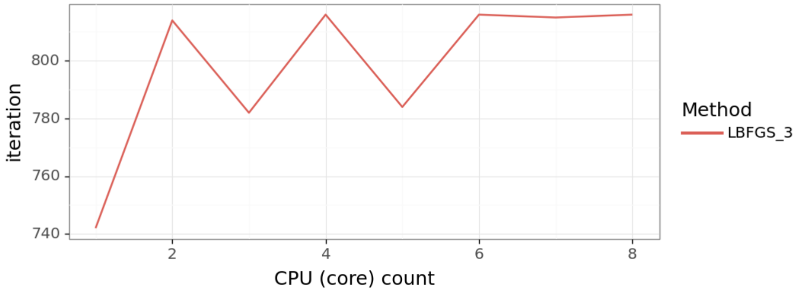
Problem has $3 \cdot 10^6$ variables
$f(x)$, $\nabla f(x)$ and BLAS are OpenMP-accelerated

CPU (core) count	time, sec.	iterations
1	270.15	742
2	161.59	814
3	114.82	782
4	99.20	816
5	83.22	784
6	78.33	816
7	72.44	815
8	68.64	816

The final acceleration – almost 4 times

Numerical experiments, LBFGS, OpenMP





Numerical experiments, convergence time (seconds)

method	problem size (millions of variables)					
	1m	3m	6m	12m	25m	52m
CG(CD)	13.10	56.99	156.82	426.42	1282.86	3336.30
CG(DY)	14.95	63.96	182.96	470.20	1433.57	3866.97
CG(FR)	12.81	55.55	155.99	425.14	1273.31	3264.12
CG(HS)	15.29	68.56	180.26	477.31	1434.82	3823.97
CG(LS)	14.20	62.47	174.64	450.12	1338.52	3444.25
CG(PRP+)	13.80	60.25	166.90	440.91	1258.06	3373.99
CG(PRP)	13.83	60.28	166.09	442.91	1256.28	3374.93
LBFGS(3)	15.11	68.64	180.10	420.18	1108.94	2906.96
LBFGS(5)	17.09	78.11	207.721			
LBFGS(10)	21.05	102.65	274.546			
LBFGS(50)	50.84	289.17	801.471			
LBFGS(100)	87.08	502.37	1430.386			

Numerical experiments, CG methods

$$\beta_k^{HS} = \frac{\langle y_k, g_{k+1} \rangle}{\langle y_k, s_k \rangle},$$

$$\beta_k^{FR} = \frac{\langle g_{k+1}, g_{k+1} \rangle}{\langle g_k, g_k \rangle} = \frac{\|g_{k+1}\|_2^2}{\|g_k\|_2^2},$$

$$\beta_k^{PRP} = \frac{\langle y_k, g_{k+1} \rangle}{\langle g_k, g_k \rangle} = \frac{\langle y_k, g_{k+1} \rangle}{\|g_k\|_2^2}, \qquad \beta_k^{PRP+} = \max\left\{0, \beta_k^{PRP}\right\},$$

$$\beta_k^{CD} = -\frac{\langle g_{k+1}, g_{k+1} \rangle}{\langle g_k, d_k \rangle} = -\frac{\|g_{k+1}\|_2^2}{\langle g_k, d_k \rangle},$$

$$\beta_k^{LS} = -\frac{\langle y_k, g_{k+1} \rangle}{\langle g_k, d_k \rangle},$$

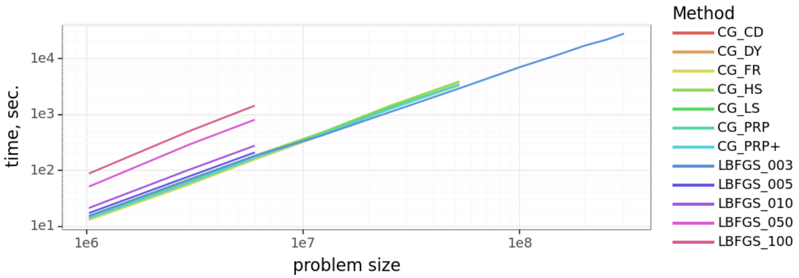
$$\beta_k^{DY} = \frac{\langle g_{k+1}, g_{k+1} \rangle}{\langle y_k, s_k \rangle} = \frac{\|g_{k+1}\|_2^2}{\langle y_k, s_k \rangle},$$

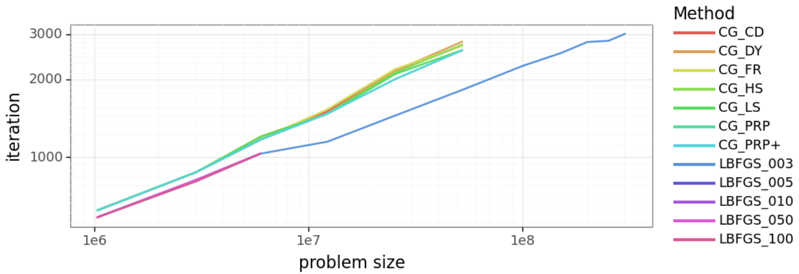
$$g_k = \nabla f(x_k), \qquad s_k = x_{k+1} - x_k, \qquad y_k = g_{k+1} - g_k.$$

Numerical experiments, LBFGS(3) convergence

problem size	time, sec.	iterations	$\|\nabla f(x^*)\|_2$
$1029000 \approx 10^6$	15.11	584	$9.63 \cdot 10^{-5}$
$3 \cdot 10^6$	68.64	816	$9.74 \cdot 10^{-5}$
$6001128 \approx 6 \cdot 10^6$	180.10	1032	$9.95 \cdot 10^{-5}$
$12288000 \approx 12 \cdot 10^6$	420.18	1148	$9.82 \cdot 10^{-5}$
$25468992 \approx 25 \cdot 10^6$	1108.94	1450	$9.67 \cdot 10^{-5}$
$52728000 \approx 52 \cdot 10^6$	2906.96	1828	$9.90 \cdot 10^{-5}$
$100158744 \approx 100 \cdot 10^6$	6961.68	2259	$1.05 \cdot 10^{-4}$
$151959000 \approx 152 \cdot 10^6$	11743.77	2538	$8.69 \cdot 10^{-5}$
$200770248 \approx 200 \cdot 10^6$	17027.95	2797	$1.21 \cdot 10^{-4}$
$252083016 \approx 252 \cdot 10^6$	21666.72	2827	$2.17 \cdot 10^{-4}$
$303584088 \approx 303 \cdot 10^6$	27748.17	3017	$1.60 \cdot 10^{-4}$

Numerical experiments





Numerical experiments

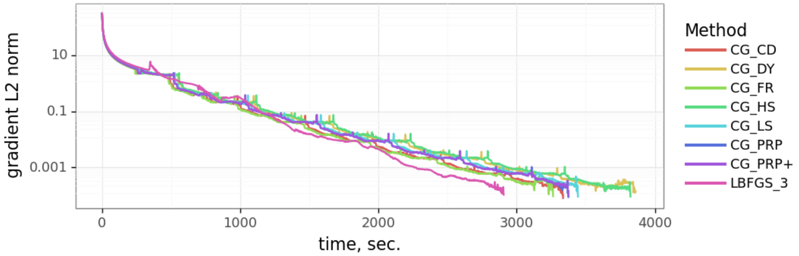
In some cases the methods failed to reach the required gradient norm value. But a closer analysis of the results showed that this is not a problem and the value $\varepsilon_g = 10^{-4}$ is unnecessarily strict.

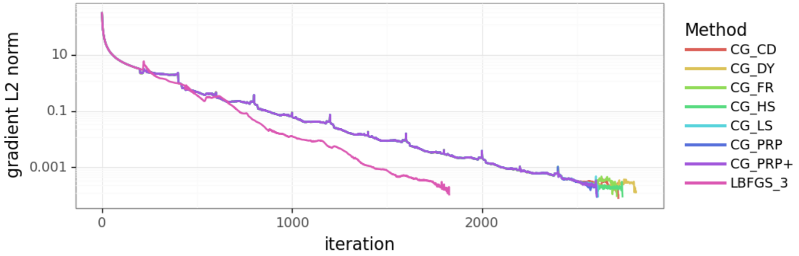
For example, for a problem with 152 million (151959000) variables and LBFGS(3) we have:

time, sec.	iteration	$f(x)$	$\nabla f(x)$
2.25	0	2664.156	$4.77 \cdot 10^2$
371.90	80	1187.127	$9.95 \cdot 10^0$
2573.88	556	1155.868	$9.99 \cdot 10^{-1}$
4733.54	1023	1155.444	$9.96 \cdot 10^{-2}$
6916.99	1495	1155.441	$9.97 \cdot 10^{-3}$
9410.24	2034	1155.440	$9.79 \cdot 10^{-4}$
11743.77	2538	1155.440	$8.69 \cdot 10^{-5}$

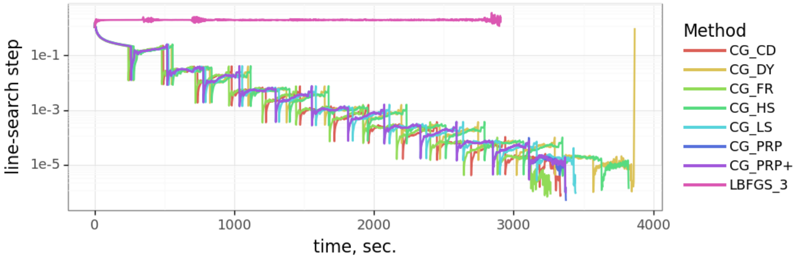
Therefore, $\varepsilon_g = 10^{-3}$ can be used in the future.

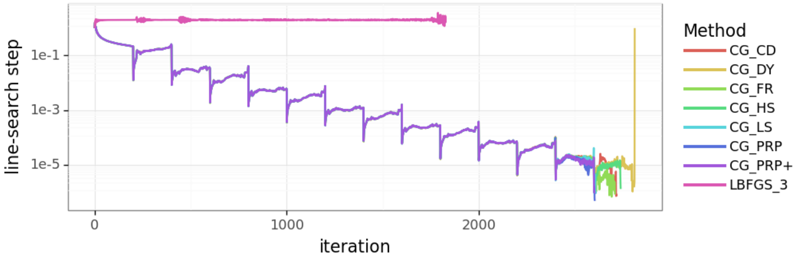
Problem with 52 million (52728000) variables





Problem with 52 million (52728000) variables





Conclusions

- Three-phase computational techniques was proposed for the study of atomic-molecular clusters
- The performance of the developed methods for problems with small, medium and large dimensions was tested
- Numerous computational experiments were performed for comparing the proposed initial approximation generators and "starter algorithms"
- Probable optimal configurations were obtained for clusters of extremely large dimensions

Conclusions

- Choosing the "correct" optimization method allows you to successfully solve large-scale optimization problems
- Simple (sometimes even "primitive") line-search algorithms in some cases can significantly speed up the local descent. Applying simple ideas like step history can also give you a significant performance boost
- Many problems can be (and should be!) accelerated with using (not too complex) technologies such as OpenMP
- For huge-scale problems, it is important to use suitable data structures and optimize algorithms in terms of memory allocation, since a single vector of optimized variables can take up several gigabytes of RAM

Computational technologies for the optimization of ultra-large atomic-molecular clusters

Anton Anikin,
Pavel Sorokovikov, Aleksander Gornov

ISDCT SB RAS, Irkutsk, Russia

April 21, 2021