Presentation of a multithreaded interval solver for nonlinear systems

Bartłomiej Jacek Kubica

Institute of Control and Computation Engineering
Warsaw University of Technology

SWIM 2015

Prague, Czech Republic

Background

- Interval methods provide us several powerful tools for solving nonlinear systems, e.g.:
  - various kinds of interval Newton operator,
  - various consistency operators,
  - other constraint propagation/satisfaction tools,
  - ...
- Question: **What is crucial for the efficiency (or its lack) of an interval method for solving a specific problem?**

Background

- Interval methods provide us several powerful tools for solving nonlinear systems, e.g.:
  - various kinds of interval Newton operator,
  - various consistency operators,
  - other constraint propagation/satisfaction tools,
  - ...
- Question: **What is crucial for the efficiency (or its lack) of an interval method for solving a specific problem?**
  - Answer: developing a proper heuristic for **choosing**, **parameterizing** and **arranging** adequate tools to process specific data.

Overall b&p algorithm

```
Lpos = {}; Lverif = {};
L = {x0};
(optionally) preprocess the list L; // prior to the actual b&p procedure
while (there are boxes to consider) do
        pop (x);
        process (x);
        if (x was verified to contain a solution) then
                push (Lverif, x);
        else if (x is verified not to contain solutions) then
                discard x;
        end if
        if (x was discarded or stored) then
                pop (x);
        else if (diam (x) < ε) then
                push (Lpos, x);
        else
                bisect (x, x1, x2);   push (x2);   x = x1;
        end if
end while
```

Features and focus

- Multithreaded implementation – different boxes can be processed by different threads.
  - Synchronization & load balancing.
  - Some popular tools are not as adequate, e.g., LP-preconditioners, LP-narrowing, hull consistency(?).
  - Focus on MT-safe (or easy to parallelize) tools.
  - Tuning for various architectures (in particular, MIC).
- Efficiency – as for well-determined, as for underdetermined problems.
  - Proper tools and heuristics development.

Selected previous papers

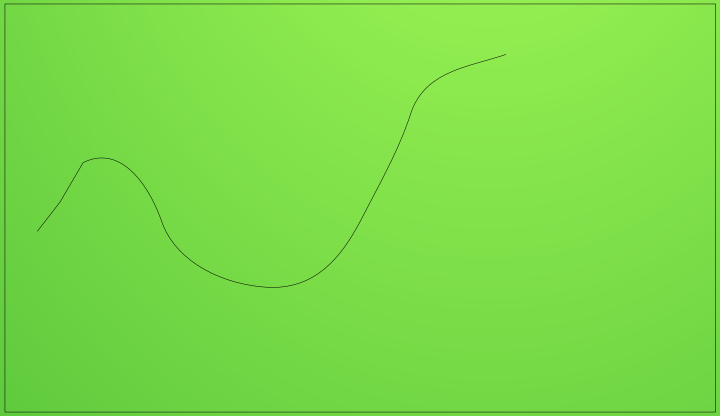
- B. J. Kubica, *Interval methods for solving underdetermined nonlinear equations systems*, SCAN 2008, Reliable Computing, Vol. 15, pp. 207 – 217 (2011).
- B. J. Kubica, *Tuning the multithreaded interval method for solving underdetermined systems of nonlinear equations*, PPAM 2011, LNCS, Vol. 7204, pp. 467 – 476 (2012).
- B. J. Kubica, *Excluding regions using Sobol sequences in an interval branch-and-prune method for nonlinear systems*, SCAN 2012, Reliable Computing, Vol. 19(4), pp. 385 – 397 (2014).
- B. J. Kubica, *Using quadratic approximations in an interval method of solving underdetermined and well-determined nonlinear systems*, PPAM 2013, LNCS 8385, pp. 623 – 633 (2014).
- B. J. Kubica, *Presentation of a highly tuned multithreaded interval solver for underdetermined and well-determined nonlinear systems*, Numerical Algorithms, published online, http://dx.doi.org/10.1007/s11075-015-9980-y, 2015.
- B. J. Kubica, *Parallelization of a bound-consistency enforcing procedure and its application in solving nonlinear systems*, submitted to PPAM 2015.

Used tools

- Initial exclusion phase – prior to the actual b&p:
  - Sobol sequence as a basis.
  - solving the tolerance problem.
  - computing the completion of a set of boxes.
- Interval Newton operator:
  - switching between the componentwise version and GS.
- 2nd order approximation and quadratic equation solving.
- Box consistency enforcing.
- Bound consistency enforcing.
- Advanced heuristics to choose and parameterize these tools.

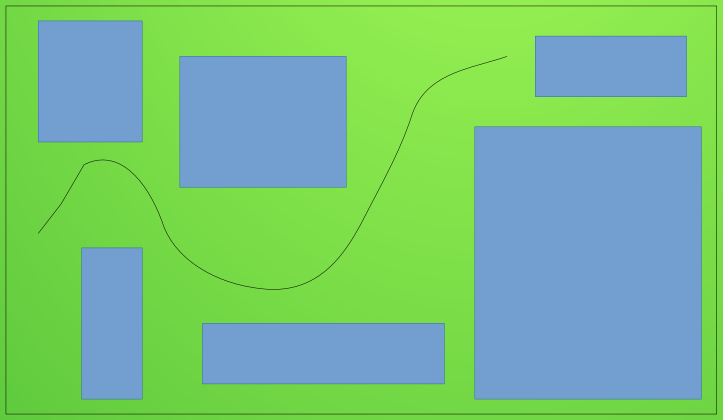
Initial exclusion phase

- Initial exclusion phase – motivation:
  - Interval Newton operators are powerful, but relatively expensive.
  - Large boxes, encountered in the early stages of the b&p algorithm can rarely be reduced by the Newton operator.
  - We should apply these operators only for boxes close to the solution set.
  - Large regions of the domain can be discarded using function values, only.



Initial exclusion phase

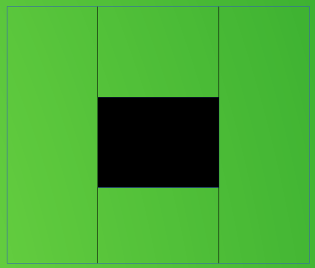
- Initial exclusion phase – motivation:
  - Interval Newton operators are powerful, but relatively expensive.
  - Large boxes, encountered in the early stages of the b&p algorithm can rarely be reduced by the Newton operator.
  - We should apply these operators only for boxes close to the solution set.
  - Large regions of the domain can be discarded using function values, only.



Initial exclusion phase

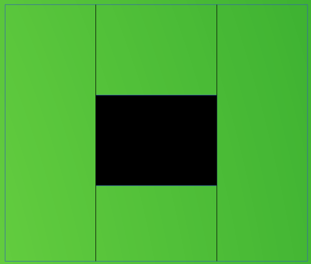
- Remove areas not containing solutions – not using the Newton operator or other higher order info:
  - Prior to starting the actual branch-and-bound type method, we generate a number (e.g., n^2) of points, using the Sobol sequence.
  - Generate solution-free boxes around them, using the procedure of Сергей П. Шарый for the linearized equation and ε-inflation; if $f(x) \in [-\varepsilon, \varepsilon]$, the point is ignored.
  - Exclude the boxes from the domain and perform the b&b type algorithm on their completion.
- Comments:
  - The procedure is cheap – no derivatives.
  - Sobol sequences can be generated efficiently and simply (there are Open Source libraries).

Initial exclusion phase

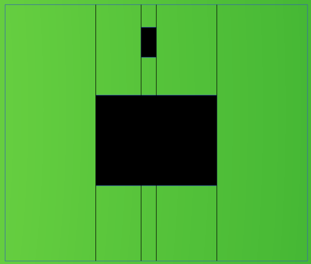


- An issue – proper implementation of the procedure computing the completion.
- There is the procedure of R.B. Kearfott for a single box; it generates at most 2 *n* boxes.
- It can be applied several times subsequently, but...

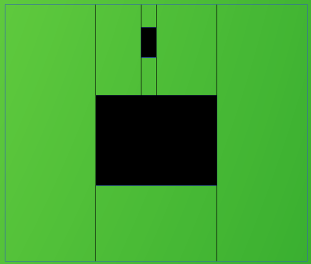
Initial exclusion phase



- An issue – proper implementation of the procedure computing the completion.
- There is the procedure of R.B. Kearfott for a single box; it generates at most 2 n boxes.



- It can be applied several times subsequently, but:
  - No parallelism.
  - The result would depend on the order of boxes exclusion.
  - A great deal of boxes can get generated.
  - Often, boxes have peculiar shapes (long and flat) and their shapes are unrelated to function values.
  - Hence, actually, sometimes expanding the exclusion boxes decreases the performance.



Initial exclusion phase

- Boxes might be sorted with respect to decreasing Lebesgue measure, but it solves the problem rarely.
- The satisfying solution:
  - We use task parallelism. Each task is to cut from a specific box a list of excluded boxes.
  - From this list we choose the box with the largest (wrt the Lebesgue measure) intersection with the box from which we do the exclusion.
  - Boxes, created in the exclusion process, become basis for new tasks (obviously, their lists of excluded boxes are shorter by one than for the parent task).
  - Far fewer boxes are created and the parallelism is natural.
  - All functions $f_i(\cdot)$ are used for exclusion.

Initial exclusion phase

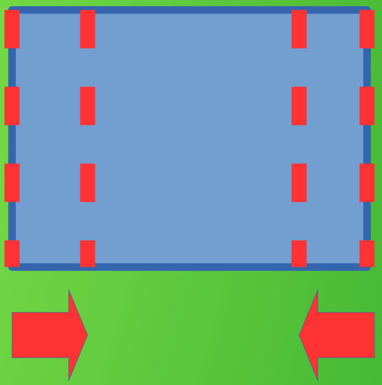
- For each function, after the ε-inflation, variables, not occurring in its formula, are set to their whole domain.
- We exclude the box for $f_i(\cdot)$, for which we obtained the largest Lebesgue measure.
- There is a threshold value not to exclude to many boxes (1024 currently; it is a magical constant, obviously).

• Intel TBB allows an elegant implementation:

- We use the concept of **`tbb::parallel_do`**.
- Boxes, created in the exclusion process, become basis for new tasks – using **`tbb::parallel_do_feeder`**.
- Lists of boxes are represented as **`std::vector`** (**`tbb::concurrent_vector`** does not have the method **`pop_back`**).
- Counter of excluded boxes it represented as atomic integers.

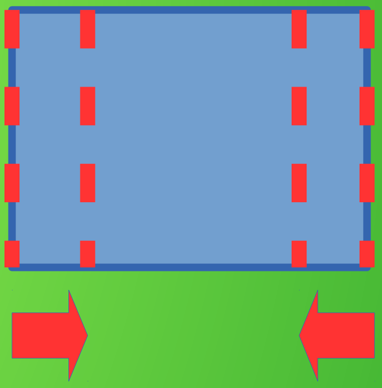
Box consistency

- A box $\boldsymbol{x}$ is **box consistent** iff all its facets are pseudo solutions.
- Enforcing box consistency – the *bc3revise* procedure:
  - For each variable i, compute leftmost and rightmost pseudo-solutions, using the unidimensional interval Newton operator.
  - Update bounds on $\boldsymbol{x}_i$.
  - Repeat the above steps while at least one of the variables gets modified.



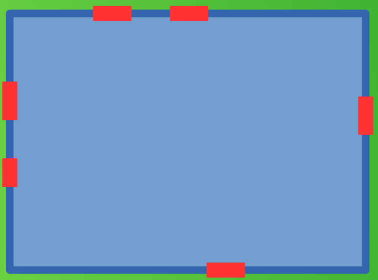
Box consistency

- A box $\boldsymbol{x}$ is **box consistent** iff all its facets are pseudo solutions.
- Enforcing box consistency – the *bc3revise* procedure:
  - For each variable i, compute leftmost and rightmost pseudo-solutions, using the unidimensional interval Newton operator.
  - Update bounds on $\boldsymbol{x}_i$.
  - Repeat the above steps while at least one of the variables gets modified.
- Parallelization possibilities:
  - Concurrent computing of different pseudo-solutions and updating different variables.



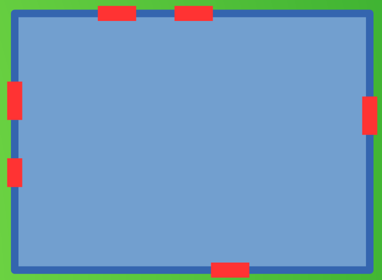
Bound consistency

- A box $\boldsymbol{x}$ is **bound consistent** iff all its facets contain a non-empty box consistent subbox.
- Enforcing bound consistency:
  - Consider slices of each box wrt. all variables; there are 2 n such slices for each box:
    $\left(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, [\underline{x}_i, c_1], \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_n\right), \left(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, [c_2, \overline{x}_i], \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_n\right).$
  - Apply the *bc3revise* procedure for each of them.
  - If the proper bound of the i-th component has been reduced, update the proper bound of $\boldsymbol{x}$.



Bound consistency

- A box $\boldsymbol{x}$ is **bound consistent** iff all its facets contain a non-empty box consistent subbox.
- Enforcing bound consistency:
  - Consider slices of each box wrt. all variables; there are $2\,n$ such slices for each box:
    $(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_{i-1},[\underline{x}_i,c_1],\boldsymbol{x}_{i+1},\ldots,\boldsymbol{x}_n)$, $(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_{i-1},[c_2,\overline{x}_i],\boldsymbol{x}_{i+1},\ldots,\boldsymbol{x}_n)$.
  - Apply the *bc3revise* procedure for each of them.
  - If the proper bound of the i-th component has been reduced, update the proper bound of $\boldsymbol{x}$.



- Parallelization possibilities:
  - Concurrent processing of both slices for a single variable.
  - Concurrent updating of different variables – synchronization needed!

Bisection or multisection?

- Some authors claim multisection outperforms bisection:
  - It is more netural in some cases.
  - It is more adequate for parallelization – more boxes generated, hence more parallelism.
- According to my experiences:
  - It is rarely better than bisection.
  - The influence on parallelism is negligible.
  - Using the inital exclusion phase (that generates several boxes) reduces this impact even more.
  - I was going to come up with a heuristic on when to use trisection, but... up to now my heuristic is: use bisection always. ;-)
  - But... (?)

Numerical experiments: environment I – laptop

- Intel Core i7-3632QM, 2.2GHz; 4 cores wth HT.
- 64-bit Manjaro 0.8.8 GNU/Linux.
- Kernel 3.10.22-1-MANJARO.
- C++, compiler: GCC 4.8.2.
- Glibc 2.18.
- Libraries:
  - C-XSC,
  - Intel TBB,
  - OpenBLAS.

Numerical experiments: environment II – host

- 2 × Intel Xeon E5-2695 v2, 2.4GHz; 12 cores with 2 hyper-threads each (48 HT in total).
- Turbo frequency non-uniform! (2.9GHz – 3.2GHz).
- GNU/Linux.
- Kernel 3.10.0-123.el7.x86_64.
- C++, Intel compiler: ICC 15.0.2.
- Glibc 2.17.
- Libraries:
  - C-XSC,
  - Intel TBB,
  - MKL.

Numerical experiments: environment III – MIC

- Intel Xeon Phi 7120P, 1.238GHz; 61 cores with 4 hyper-threads each.
- Micro-OS GNU/Linux.
- Kernel 2.6.38.8+mpss3.4.1.
- C++, Intel compiler: ICC 15.0.2 (used by cross-compilation from host, i.e., environment II).
- Glibc 2.14.90.
- Libraries:
  - C-XSC,
  - Intel TBB,
  - MKL.

Example times

Problem	Laptop (8 threads)	Host (32 threads)	MIC (61 threads)	MIC (122 threads)
Broyden 16	1.9s	0.6s	11.4s	12.2s
Brent 10	14.8s	7.0s	120.9s	124.9s
Academic	10.5s	4.6s	19.1s	16.6s
Puma 6	30.9s	11.5s	72.7s	62.6s
5R planar	102.6s	35.1s	241s	207.4s

Results

- The solver parallelizes pretty well on 61 threads.
  - The serial part is below 1.3% of the total time.
- Parallelizing box- and bound consistency enforcing operators is significant for a high number of threads.
- Results on the Xeon Phi coprocessor are still much worse than on CPU.
- It will be beneficial to utilize the hyper-threads on MIC, but this requires careful tuning, wrt. cache utilization and vectorization.
- It seems, for underdetermined problems, using HT (i.e., more than one thread per core) is worthwile.
  - It is **not** because of post-processing of the list of boxes (longer than for well-determined problems)???

Solver

- The solver is available at my ResearchGate profile:

  https://www.researchgate.net/profile/Bartlomiej_Kubica?ev=hdr_xprf.

- The currently developed version is not available yet – will update it in a few weeks.
- The available version does not use bound consistency – this version has been described in the paper in Numerical Algorithms journal.
- Feel encouraged to use and test it!
- And stay tuned for updates!

Future research

- Explaining the mysterious behavior of the solver on the MIC architecture (i.e., Intel Xeon Phi).
- Tuning the solver for use with hyper-threads on this platform.
- Preparing the hybrid version, using both host and MIC (Intel compiler's directive: `#pragma offload target(mic)`).
- Exploring other tools (hull consistency?).
- ...

Acknowledgements

- Thanks to professor Roman Wyrzykowski (and his co-workers), from Częstochowa University of Technology, thanks to whom I have access to the MIC machine (environments II and III).