CANDO - A Compiled Programming Language for Computer-Aided Nanomaterial Design and Optimization Based on Clasp Common Lisp

Christian E. Schafmeister
Chemistry Department
Temple University
1901 N. 13th Street
Philadelphia, PA
meister@temple.edu

ABSTRACT

CANDO is a compiled programming language designed for rapid prototyping and design of macromolecules and nanometer-scale materials. CANDO provides functionality to write programs that assemble atoms and residues into new molecules and construct three-dimensional coordinates for them. CANDO also provides functionality for searching molecules for substructures, automatically assigning atom types, identifying rings, carrying out conformational searching, and automatically determining stereochemistry, among other things. CANDO extends the Clasp implementation of the dynamic language Common Lisp. CANDO provides classes for representing atoms, residues, molecules and aggregates (collections of molecules) as primitive objects that are implemented in C++ and subject to automatic memory management, like every other object within the language. CANDO inherits all of the capabilities of Clasp, including the easy incorporation of C++ libraries using a C++ template programming library. This automatically builds wrapper code to expose the C++ functionality to the CANDO Common Lisp environment and the use of the LLVM library[1] to generate fast native code. A version of CANDO can be built that incorporates the Open Message Passing Interface C++ library,[2] which allows CANDO to be run on supercomputers, in order to automatically setup, start, and analyze molecular mechanics simulations on large parallel computers. CANDO is currently available under the LGPL 2.0 license.

Categories and Subject Descriptors

D.2.12 [**Software and its engineering**]: Interoperability; D.3.4 [**Software and Programming Languages**]: Incremental compilers

Keywords

Computational chemistry, Common Lisp, LLVM, C++, interoperation

1. INTRODUCTION

Using computers to design molecules and materials presents many challenges. There are almost no software tools that allow a chemist to write efficient programs that automatically construct molecules according to a design and optimize those designs based on their ability to organize functional groups in desired three-dimensional constellations. Software like this would greatly facilitate the creation of designer macromolecules and materials.

My laboratory is a synthetic organic chemistry group that has developed a unique approach to synthesizing large molecules with well defined three-dimensional structures.[3] These molecules can be designed to act as new therapies, to create new catalysts, and to create channels that purify water and separate small molecules from each other. These molecules are assembled like peptides, DNA and carbohydrates, using a common set of interchangeable building blocks linked through common linkage chemistry. By assembling them in different sequences an enormous number of different well defined, three-dimensional structures are created, some of which could have extremely valuable properties. The challenge is to find those active molecules and with the right software, together with large, parallel, modern supercomputers we may be able to find them.

We synthesize stereochemically pure bis-amino acids that we connect together through pairs of amide bonds. These create large molecules with programmable three-dimensional shapes and functional groups called "spiroligomers". In the last several years, we have demonstrated that we can design small spiroligomers that bind a protein[4], and we have demonstrated three spiroligomer based catalysts that accelerate chemical reactions.[5][6][7] Recently we have demonstrated that we can connect multiple spiroligomers together to create macromolecules that present large surfaces with complex constellations of functional groups.[8] These molecules can now be synthesized to a size where it is impossible to design them using existing tools and so I have developed CANDO, a molecular programming environment that will enable the design of large, functional spiroligomers as well as any other foldamers[9] such as peptides, peptiods, beta-peptides and any other large molecules assembled from modular building blocks.

CANDO is a compiled, dynamic programming language

that implements objects essential to molecular design including atoms, residues, molecules and aggregates as memory-efficient C++ classes. CANDO manages these objects using modern memory management (garbage collection), freeing the programmer from having to deal with memory management for exploratory code. CANDO is written in C++ and makes it easy to integrate existing C++, C, and Fortran libraries that efficiently carry out molecular simulations and tightly integrate them with high-level code that sets up calculations and carries out analyses of results. The idea is to do everything within CANDO: build new molecules, assign force-field atom types and atomic charges, run molecular dynamics simulations, and analyze the results of the simulations, all within a single compute node, without having to leave the CANDO programming environment. This is done using external C++ libraries like OpenMM from within CANDO, using the exposed C++ application programming interface (API) provided by the library. CANDO is designed to be for chemistry what Mathematica is for symbolic mathematics or R is for statistial analysis - a general purpose programming language, tailored for the specific problem domain of computational chemistry and molecular and nanomaterials design. CANDO is an implementation of the Common Lisp dynamic programming language, a compiled, industrial strength language with a well-defined standard that includes incremental typing and can interactively compile code in a way that is both highly interactive and dynamic while generating fast native code.

2. RELATED WORK

CANDO is not the only programming language that specifically targets chemistry. The Scientific Vector Language (SVL) is embedded within the Molecular Operating Environment (MOE) software package developed by the Chemical Computing Group Inc.[10] SVL contains primitive objects for atoms, residues, molecules and systems of molecules. SVL is touted as being both a compiled and interpreted language. MOE is a commercial product and the source code to the SVL compiler is not open source.[11] SVL uses a syntax that could be termed "C-like". More generally, Scientific Python (SciPy),[12] and scientific programming languages like Julia could be used to implement chemistry objects, but they do not implement the classes required to represent molecular systems.[13]

There are significant differences between CANDO and SVL. CANDO builds on and extends the existing language Clasp Common Lisp, therefore CANDO is able to use the large library of existing software that has been developed in Common Lisp.[14] CANDO is open source, and is freely available to the scientific community under the LGPL 2.1 license. This is to facilitate its use in research and allow it to be adapted to run on large academic parallel supercomputers without paying a per-processor fee. SVL and MOE are closed source and commercial products and licensing must be negotiated with the Chemical Computing Group.

In the realm of molecular design, CANDO is related to the software package Rosetta.[**?**] Rosetta is a suite of command line applications and bindings for languages like Python, that carry out different operations such as protein folding prediction and protein/protein interface design. Rosetta is highly tuned to work with natural proteins and uses the Brookhaven Protein Database to construct rotamer libraries that describe the preferred conformations of amino acids.

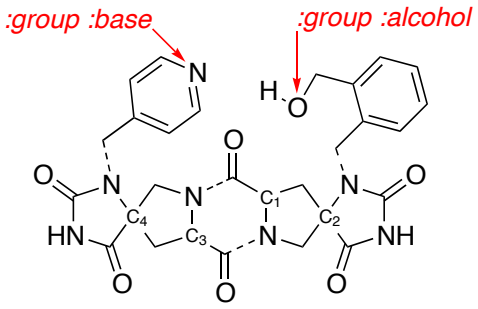

Figure 1: A bond line drawing of a molecule that can form an internal hydrogen bond between the base and alcohol if it has the correct stereochemistry at C_1, C_2, C_3 and C_4. The annotations in red add properties to the atom's property list.

This structural information is not available for unnatural building blocks like spiroligomers and other foldamers and non-proteogenic building blocks require a great deal of work to incorporate into Rosetta. CANDO provides tools for carrying out conformational analysis on molecules constructed from unnatural building blocks and enables the construction of conformational databases for building blocks. CANDO is more general than Rosetta with respect to the chemistry that it can deal with and while it was designed with spiroligomers in mind, it supports molecular design with any molecular building blocks.

3. BUILDING MOLECULAR STRUCTURES

One of the ways that chemists communicate with each other is using "bond line formalism", a graphical notation that describes the chemical structure of molecules. A molecule is represented with a structure like Figure 1. Structures like this can be drawn using commercial packages such as ChemDraw[15] and ChemDoodle[16]. These programs are able to save these structures in an XML-based format called cdxml. CANDO can read cdxml files directly and construct the molecular graph of the molecule with atom names, atom elements, bond orders, connectivity and other annotations attached. CANDO interprets some of the bond types as codes to annotate atom, residue, and molecule properties with Common Lisp symbols and value pairs so that programs can easily identify features within the molecules. The red keyword symbols and arrows in Figure 1 are examples of this. The red arrows indicate that the property/value pairs (:group :base) and (:group :alcohol) are attached to the pyridine nitrogen and benzyl alcohol oxygen respectively.

The code to read a cdxml file into CANDO is shown in listing 1. In bond line formalism (and thus cdxml), hydrogen atoms are not drawn on carbon atoms because their number can be inferred from the structure. Carbon atoms are represented by vertices and ends of lines and other atoms are labeled with their element name or a name that starts with the element name (eg: C1). CANDO automatically identifies the number of implicit hydrogens and adds them to the molecules graph. The resulting structure obtained by loading the cdxml file is put in the dynamic variable *agg*, which is an instance of the CHEM:AGGREGATE class containing one instance of the CHEM:MOLECULE class, which contains four instances of the CHEM:RESIDUE

class. Each residue contains a collection of CHEM:ATOM instances that contain lists of pointers to CHEM:BOND objects. The CHEM:BOND objects create a bidirectional graph by pointing to pairs of CHEM:ATOM objects. A CHEM:RESIDUE object contains multiple atoms that are bonded to each other and form a common unit of molecular structure like an amino acid, DNA nucleotide, or monosaccharide. A collection of CHEM:RESIDUE objects can be bonded to each other to form a CHEM:MOLECULE object. A collection of CHEM:MOLECULEs can be grouped together into a CHEM:AGGREGATE instance, used to describe a collection of molecules that interact with each other.

Listing 1: Load cdxml file

```
(defparameter *cd*
  (with-open-file
    (fin #P"base-alcohol.cdxml"
      :direction :input)
    (chem:make-chem-draw fin)))
(defparameter *agg* (chem:as-aggregate *cd*))
```

Carbon atoms that have sp^3 hybridization are bonded to four other atoms that roughly point to the corners of a tetrahedron with the sp^3 carbon atom at its center. If the four bonded groups are all unique then there are two possible three-dimensional arrangements of those four groups, termed the "stereochemical configuration" of the central carbon. CANDO can automatically identify the carbons that can have such stereochemical configurations, and allows the programmer to define their configurations using the keyword symbols `:S` or `:R`. In the structure above, the four stereocenters were labeled C_1, C_2, C_3, and C_4 and in Listing 2 the four stereocenters are all set to the `:S` configuration with the Listing 3. The assignment of stereochemical configuration sets a field in the carbon's CHEM:ATOM instance which will be used later to add a stereochemical configuration restraint when three-dimensional coordinates of the molecule are constructed.

Listing 2: Set stereocenter configurations

```
(defparameter *stereocenters*
  (sort (cando:gather-stereocenters *agg*)
        #'string< :key #'chem:get-name))
(cando:set-stereoisomer-func
  *stereocenters*
  (constantly :S) :show t)
```

Listing 3: Output of Listing 2.

```
C1      S
C2      S
C3      S
C4      S
```

CANDO can load files of various formats that describe the three-dimensional structure of molecules (PDB, MOL2). However, CANDO's purpose is to build three-dimensional structures from a simple graph description of the molecule with no prior knowledge of the three-dimensional structure. To do this CANDO uses a very robust approach to building chemically reasonable three-dimensional structures for molecules from first principles. Quantum mechanics and the Schroedinger equation can be used to build chemically reasonable three-dimensional structures of molecules, however the calculations for even simple molecules are extremely time consuming. A less expensive approximation is to use "molecular mechanics," which approximates bonds as mechanical springs and describes molecules using a "molecular force-field function" that contains many empirically-determined parameters including ideal bond lengths, angles, torsion angles, and through-space interactions describing the electrostatic and van Der Waals interactions between non-bonded atoms. CANDO has built in the popular AMBER force-field,[17] which has the form (1). CANDO adds several additional terms to the force-field equation that restrain atoms in different ways. For instance, the stereochemical configuration defined above is maintained with a special stereochemical configuration restraint term that penalizes molecular structures where the four groups attached to the stereochemical carbon are in the wrong tetrahedral arrangement.

$$E_{\text{total}} = \sum_{\text{bonds}} K_r(r - r_{eq})^2 + \sum_{\text{angles}} K_\theta(\theta - \theta_{eq})^2 + \sum_{\text{dihedrals}} \frac{V_n}{2}[1 + \cos(n\phi - \gamma)] + \sum_{i<j} \left[\frac{A_{ij}}{R_{ij}^{12}} - \frac{B_{ij}}{R_{ij}^{6}} + \frac{q_i q_j}{\epsilon R_{ij}} \right] + Restraints \quad (1)$$

To build the three-dimensional structure of a molecule from first principles, the programmer first instructs CANDO to assign random coordinates to every atom within a 20x20x20 Angstrom box using the `(cando:jostle *agg* 20)` command (Figure 2). Then CANDO is instructed to minimize the force-field energy using non-linear optimization. CANDO implements three non-linear optimizers in C++ (a steepest descent, a conjugate gradient, and a truncated Newton-Raphson[20] optimizer), switching between them as the optimization proceeds. Prior to carrying out the non-linear optimization, CANDO assigns force-field atom types using substructure pattern recognition and assigns force-field parameters. Currently CANDO assigns all atomic charges to zero for model building. In the future, atomic-charge approximation algorithms will be implemented in CANDO Common Lisp. The non-linear-optimization is then initated using the `(my-minimize *agg* *ff*)` function (Listing 4).

Listing 4: Energy minimization

```
(cando:jostle *agg* 20)
(defparameter *ff* (energy:setup-amber))
(my-minimize *agg* *ff*)
(cando:chimera *agg*)
```

The energy minimization function first minimizes the energy with non-bonded interactions turned off, and then with non-bonded interactions turned on. This is to allow atoms and bonds to initially pass through each other so that rings do not get tangled up with each other. The resulting structure is generated in a few seconds and shown in Figure 3. The structure can then be displayed using the molecular visualization program Chimera[19] by calling the `(cando:chimera *agg*)` function, which writes the structure to a temporary file and calls the Unix `system` function to open it (Figure 3).

Listing 5: Definition of my-minimizer function

```
(defun my-minimize (agg force-field)
  "Minimize the conformational energy"
  (let* ((energy-func
          (chem:make-energy-function
```

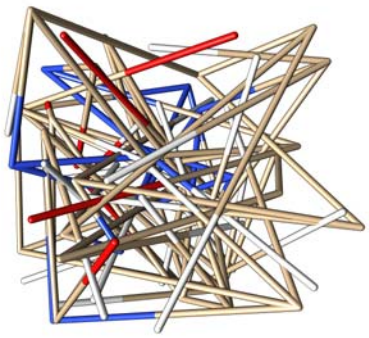

Figure 2: A molecule with random coordinates.

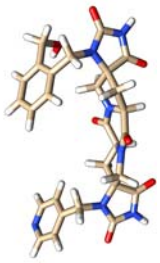

Figure 3: A molecule after energy minimization.

```
        agg force-field))
      (minimizer
       (chem:make-minimizer
        :energy-function energy-func)))
(cando:configure-minimizer
 minimizer
 :max-steepest-descent-steps 1000
 :max-conjugate-gradient-steps 5000
 :max-truncated-newton-steps 0)
(chem:enable-print-intermediate-results
 minimizer)
(chem:set-option energy-func
                 'chem:nonbond-term nil)
(cando:minimize-no-fail minimizer)
(chem:set-option energy-func
                 'chem:nonbond-term t)
(cando:minimize-no-fail minimizer)))
```

CANDO is capable of many other manipulations to prepare structures for calculations such as molecular dynamics simulations. For example, Figure 4 shows a model of a potential membrane channel that was designed using CANDO.

Listing 6: Loading a channel from a mol2 file and a model membrane

```
(defparameter *channel*
  (cando:load-mol2 "trichannel.mol2"))
(cando:chimera *channel*)
(defparameter *membrane* (load-membrane))
(cando:chimera *membrane*)
```

This molecule is designed to create a pore in a cell membrane. To prepare it for a simulation, it needs to be embedded within a model of a fragment of cell membrane in water. To do this, the channel needs to be superimposed onto the membrane and the lipid and water molecules that overlap

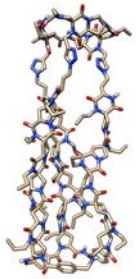

Figure 4: A potential membrane channel designed using CANDO.

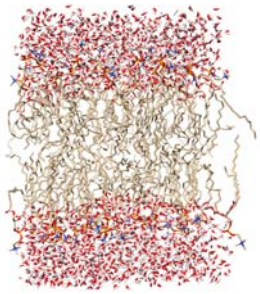

Figure 5: A model of a small segment of cell membrane.

with the channel atoms need to be removed. A problem is that lipid molecules are long and almost always overlap the channel while water molecules are small and so different criteria for overlap are needed for the two types of molecules. If the same distance criterion was used for each, then there would be large gaps where too many lipids were removed or there would be lots of close contacts between many water molecules and the channel where not enough water molecules were removed. This is a problem that is easily solved by writing a bit of code. The function `load-membrane` (Listing 7), loads the membrane and then uses the `chem:map-molecules` function to count the number of atoms in each molecule of the membrane and assign the symbol `:solvent` or `:lipid` to each molecule based on the number of atoms it contains.

Listing 7: Load a membrane from a pair of psf/pdb files and assign each molecule in the membrane to be :solvent or :lipid.

```
(defun load-membrane ()
  "Load the membrane from the psf/pdb files
and name each of the molecules
within it based on whether they
are solvent (<= 3 atoms) or lipid."
  (let ((agg-membrane
    (cando:load-psf-pdb "POPC36.psf"
                        "POPC36.pdb")))
    (chem:map-molecules
     nil
     (lambda (m)
       (if (<= (chem:number-of-atoms m) 3)
           (chem:set-name m :solvent)
           (chem:set-name m :lipid)))
     agg-membrane)
    agg-membrane))
```

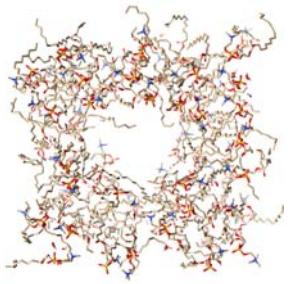

Figure 6: The top view of the cell membrane after overlaps with the channel are removed. Water molecules have been hidden for clarity.

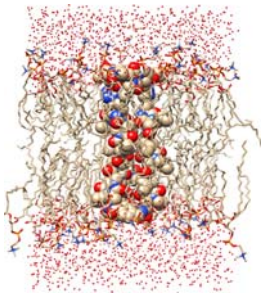

Figure 7: The side view of the channel embedded within the cell membrane. The channel is rendered using spheres for atoms and waters as dots for clarity.

Removing the various molecules from the membrane model that overlap with the channel is carried out using the function `cando:remove-overlaps`. This function accepts two AGGREGATE objects and a Common Lisp function object that returns the minimim overlap distance depending on the type of molecule that is overlapping with an atom in the channel (Listing 8). The ability to pass functions and closures to other functions is a very useful feature of the Common Lisp language that CANDO makes extensive use of. The `cando:remove-overlaps function` is written in CANDO Common Lisp and it carries out a very common but time-consuming operation in programming for chemistry. It needs to compare every atom in the first aggregate to every atom in the second aggregate, an expensive NxM calculation that would normally be implemented in C++. Since CANDO Common Lisp is compiled, the algorithm can be implemented in Common Lisp and the calculation is fast. After the molecules are removed from the membrane model, a new aggregate, `*merged*`, is created. The molecules from the membrane and the molecule of the channel are added to it and the result is saved to a mol2 file (Listing 9).

Listing 8: Removing membrane molecules that overlap with the channel.

```
(defun overlap-func (molecule)
  "Small waters (:solvent) overlap
if within 3A of a channel atom while
:lipid overlaps if within 0.6A"
  (if (eq (chem:get-name molecule) :solvent)
      3.0
      0.6))
(cando:remove-overlaps
 *membrane*
 *channel*
 :distance-function #'overlap-func)
(cando:chimera *membrane*)
```

Listing 9: Create a new aggregate and add the membrane and channel molecules to it.

```
(let ((agg (chem:make-aggregate)))
  (chem:map-molecules
   nil
   (lambda (m) (chem:add-molecule agg m))
   *membrane*)
  (chem:map-molecules
   nil
   (lambda (c) (chem:add-molecule agg c))
   *channel*)
  (defparameter *merged* agg))
```

```
(cando:save-mol2 *merged* "merged.mol2")
(cando:chimera *merged*)
```

At this point the structure in `*merged*` in the file merged.mol2 can be used to construct the input files for a molecular dynamics simulation using AMBER[18]. Molecular dynamics simulates the motions of molecules and is used by chemistry researchers to understand many molecular phenomena. CANDO does not currently generate the input files for a molecular dynamics simulation directly but relies on an external program called LEaP (written by this author), which is the primary tool used by computational chemists to set up their calculations for AMBER.[18] Future plans are to write the code to generate AMBER input files in CANDO Common Lisp and to incorporate molecular dynamics packages directly into CANDO and communicate with them through an application programming interface (API) using CANDO's built-in abilities to interface with external libraries.

4. CHEMICAL PATTERN RECOGNITION

CANDO incorporates a modified version of SMARTS,[21] which is a language that allows a programmer to specify a chemical substructure and then search for that substructure within the graph of a molecule. SMARTS searches chemical structures for substructures in the same way that regular expressions are used to search text. With SMARTS, a string such as `"CCCC"` will match a continuous chain of carbon atoms connected by single bonds. `"$(N1(C2)(~[#1]3)~C4(=O5)C6)"` will match a secondary amide bond and also bind the six atoms that form the amide bond to numerical keys so that the atoms can be recovered from the match. This capability has myriad uses and is the basis of the automatic atom-type assignment required for structure building. In the example below, a cyclic peptide is searched using a substructure matcher that recognizes secondary amide bonds (Listing 10). All five of the amide bonds are found and the nitrogen atoms of the amides are displayed (Listing 11).

Listing 10: Find all secondary amide bonds within a cyclic peptide.

```
(defparameter *cyclorgd*
  (load-cdxml #P"cyclorgd.cdxml"))
(defparameter *smarts*
  (make-cxx-object 'chem:chem-info))
(chem:compile-smarts
 *smarts*
```

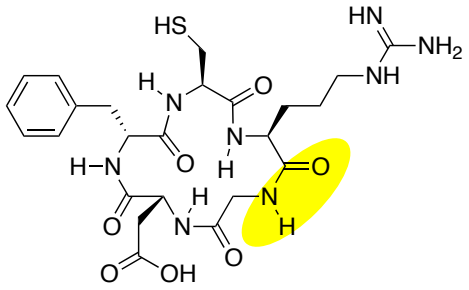

Figure 8: A cyclic peptide with one secondary amide bond highlighted in yellow.

```
 "$(N1(C2)(~[#1]3)~C4(=O5)C6)")
(chem:map-atoms
 nil
 (lambda (a)
   (let ((match (chem:matches *smarts* a)))
     (when match
       (format t "Amide nitrogen: ~a~%" a))))
 *cyclorgd*)
```

Listing 11: Found five amide bonds within the cyclic peptide.

```
Amide nitrogen: #<ATOM :N/:N@0x118032610>
Amide nitrogen: #<ATOM :N/:N@0x118032190>
Amide nitrogen: #<ATOM :N/:N@0x118032c10>
Amide nitrogen: #<ATOM :N/:N@0x117991d90>
Amide nitrogen: #<ATOM :N/:N@0x117991790>
```

5. MOLECULAR MECHANICS ENERGY FUNCTIONS

CANDO uses a very flexible approach to implementing non-linear optimization for structure building with molecular mechanics. CANDO allows the programmer to define new energy function terms by providing the functional form of the energy term and then automatically generates optimized computer code to evaluate the function and its analytical first and second derivatives. CANDO uses automatic symbolic differentiation and a compiler that optimizes the resulting symbolic expressions to remove common subexpressions and minimize the number of expensive reciprocal and square-root operations. The energy terms, the analytical gradient terms, and the analytical Hessian terms of the AMBER force field and the additional restraint terms are all implemented within CANDO in this way. Currently this facility is implemented in Mathematica code[22] and it generates efficient C code. The Mathematica and resulting C code is included in the CANDO source code. In the future this facility will be implemented in CANDO Common Lisp and generate efficient static single assignment (SSA) LLVM-IR. This will allow users to add new energy terms to the molecular mechanics force-field of CANDO to support other force fields and add new restraint terms and to use CANDO's non-linear optimizers to optimize things like partial charges in residues.

6. SERIALIZATION OF OBJECTS

CANDO has a built-in serialization format based on Common Lisp S-expressions. It uses the Common Lisp printer and reader facilities to serialize any collection of objects, including those containing internal cross-referencing pointers, into a compact, human-readable format. This serialization facility is used within CANDO to store and retrieve objects to files and to communicate data and code between processing nodes on large parallel clusters. The serialization facility is extended to CANDO C++ classes by adding one C++ method to the definition of the C++ class.

7. MEMORY MANAGEMENT OF CHEMISTRY OBJECTS

Programs that implement molecular design algorithms need to repeatedly allocate memory to represent atoms, residues, molecules, and other objects. They then need to allocate other objects to generate three-dimensional coordinates for molecular designs, predict the molecular properties *in silico*, and then release those memory resources to advance to the next design. Molecules are most conveniently represented as bi-directional graphs of atoms connected to each other through bonds. These form enormous numbers of reference loops and primitive reference-counting techniques are inadequate to manage memory. Robust garbage collection is therefore necessary to avoid crippling memory leaks.

CANDO is based on the Common Lisp implementation Clasp[14] and it uses the memory management facilities that Clasp provides to manage all chemistry objects. Clasp supports both the Boehm[23] and the Memory Pool System (MPS)[24] garbage collectors with the MPS garbage collector intended for use in production and the Boehm garbage collector used for bootstrapping and building the CANDO executable. The additional 239 C++ classes that CANDO adds to Clasp are all managed by the garbage collectors in the same way that the standard Common Lisp objects are managed. This means that objects that are no longer used are automatically discarded and objects that remain are continuously compacted in memory to release memory for further use. For algorithms where the overhead of memory management is considered to be too high for performance code, the algorithms can be implemented in C++ and the responsibility for memory management is taken over by the programmer.

In order to allow the MPS library to work with CANDO's C++ core, every pointer to every object that will move in memory needs to be updated whenever that object is moved. CANDO fully automates the identification of C++ pointers that need to be updated by MPS, using a static analyzer written in Clasp Common Lisp. The static analyzer uses the Clang C++ compiler front end to parse the more than 360 C++ source files of CANDO and uses the Clang AST-Matcher library to search the C++ Abstract Syntax Tree, in order to identify every class and global pointer that needs to be managed by MPS.

8. CANDO, CLASP AND COMMON LISP

CANDO is a full implementation of the Common Lisp language including the Common Lisp Object System. It supports the Common Lisp software packages: ASDF, the Superior Lisp Interaction Mode for Emacs (SLIME), and Quicklisp. It is a superset of the Clasp implementation of Common Lisp and has every feature that Clasp has, including the C++ template programming library "clbind", which makes it easy to integrate C++ libraries with CANDO.[14]

Table 1: Calculating the 78th Fibonacci number 10^7 times

Language	Seconds	Factor	Stdev(sec)	Rel. cclasp
clang C++	0.76	1	0.016	0.3
sbcl 1.2.11	0.63	1	0.008	0.2
cclasp	2.9	4	0.022	1.0
Python 2.7	77.7	102	0.36	26.8
bclasp	639	839	n/c	221

In the past year, Clasp (and CANDO) have been enhanced with tagged pointers, and immediate fixnums, characters, and single-float types. Additionally, the Cleavir compiler[25] has been fully integrated into Clasp and the speed of the most optimized code generated by Clasp is within a few factors of C++ (Table 1). The "cclasp" compiler is the Cleavir compiler within Clasp. The "bclasp" compiler is a less sophisticated compiler within Clasp that bootstraps cclasp and was Clasp's only compiler when reported last year.[14] So, by one measure, the performance of Clasp has improved by a factor of 221 in the past year.

9. THE CANDO SOURCE CODE

CANDO adds 155,000 logical source lines of C++ code to the 186,000 logical source lines that makes up Clasp (version 0.4). CANDO extends Clasp with 239 additional C++ classes, 948 additional C++ instance methods and 66 C++ functions that implement objects and algorithms related to chemistry. In addition, CANDO adds a growing body of CANDO Common Lisp code that implements algorithms important to chemistry and molecular design. CANDO is freely available at http://github.com/drmeister/cando.

10. CONCLUSIONS AND FUTURE WORK

CANDO is a general, compiled programming language designed for rapid prototyping and design of macromolecules and nanometer-scale materials. Future work will include developing bindings external molecular modeling packages and the OpenGL graphics library and developing a rich library of molecular modeling tools within CANDO Common Lisp.

11. LICENSE

Clasp is currently licensed under the GNU Library General Public License version 2.

12. ACKNOWLEDGMENTS

Thanks to Robert Strandh for providing Cleavir and for many fruitful discussions. Thanks to Stephanie Schneider for helpful feedback.

13. REFERENCES

[1] *Lattner, C. (2005)* "Masters Thesis: LLVM: An Infrastructure for Multi-Stage Optimization", Computer Science Dept., University of Illinois at Urbana-Champaign, http://llvm.cs.uiuc.edu

[2] *Richard L. Graham and Timothy S. Woodall and Jeffrey M. Squyres. (2005)* "Open MPI: A Flexible High Performance MPI"; Proceedings, 6th Annual International Conference on Parallel Processing and Applied Mathematics

[3] *Schafmeister, C. E., Brown, Z. Z., and Gupta, S.* "Shape-Programmable Macromolecules" Accounts Of Chemical Research 41, no. 10 (2008): 1387-1398. doi:10.1021/ar700283y

[4] *Brown, Z. Z., Akula, K., Arzumanyan, A., Alleva, J., Jackson, M., Bichenkov, E., Sheffield, J. B., Feitelson, M. A., and Schafmeister, C. E.* "A Spiroligomer alpha-Helix Mimic That Binds HDM2, Penetrates Human Cells and Stabilizes HDM2 in Cell Culture." Plos One 7, no. 10 (2012): e45948. doi:10.1371/journal.pone.0045948

[5] *Zhao, Q., Lam, Y.-H., Kheirabadi, M., Xu, C., Houk, K. N., and Schafmeister, C. E.* "Hydrophobic Substituent Effects on Proline Catalysis of Aldol Reactions in Water." The Journal of Organic Chemistry 77, no. 10 (2012): 4784-4792. doi:10.1021/jo300569c

[6] *Kheirabadi, M., Celebi-Olcum, N., Parker, M. F. L., Zhao, Q., Kiss, G., Houk, K. N., and Schafmeister, C. E.* "Spiroligozymes for Transesterifications: Design and Relationship of Structure to Activity." Journal Of The American Chemical Society 134, no. 44 (2012): 18345-18353. doi:10.1021/ja3069648

[7] *Parker, M. F. L., Osuna, S., Bollot, G., Vaddypally, S., Zdilla, M. J., Houk, K. N., and Schafmeister, C. E.* "Acceleration of an Aromatic Claisen Rearrangement via a Designed Spiroligozyme Catalyst That Mimics the Ketosteroid Isomerase Catalytic Dyad." Journal Of The American Chemical Society 136, no. 10 (2014): 3817-3827. doi:10.1021/ja409214c

[8] *Zhao, Q. and Schafmeister, C. E.* "Synthesis of Spiroligomer-Containing Macrocycles" Journal Of Organic Chemistry 80, no. 18 (2015): 8968-8978. doi:10.1021/acs.joc.5b01109

[9] *Hill, D. J., Mio, M. J., Prince, R. B., Hughes, T. S., and Moore, J. S.* "A Field Guide to Foldamers" Chemical Reviews 101, no. 12 (2001): 3893-4012. doi:10.1021/cr990120t

[10] https://www.chemcomp.com/journal/svl.htm

[11] *Molecular Operating Environment (MOE), 2013.08*; Chemical Computing Group Inc., 1010 Sherbooke St. West, Suite #910, Montreal, QC, Canada, H3A 2R7, 2016.

[12] *Jones E, Oliphant E, Peterson P, et al.* "SciPy: Open Source Scientific Tools for Python", 2001-, http://www.scipy.org/

[13] *Jeff Bezanson, Alan Edelman, Stefan Karpinski, Viral B. Shah (2014)*, "Julia: A fresh approach to numerical computing". http://arxiv.org/abs/1411.1607

[14] *Christian E. Schafmeister, (2015)* "Clasp - A Common Lisp that Interoperates with C++ and Uses the LLVM Backend"; Proceedings of the 8th European Lisp Symposium, pg 90. http://github.com/drmeister/clasp

[15] *Mills, N. (2006).* "ChemDraw Ultra 10.0". J. Am. Chem. Soc. 128 (41): 13649âĂŞ13650.

doi:10.1021/ja0697875

[16] *Todsen, W.L. (2014).* "ChemDoodle 6.0"; J. Chem. Inf. Model., 2014, 54 (8), pp 2391âĂŞ2393 doi: 10.1021/ci500438j

[17] *Cornell, W. D., Cieplak, P., Bayly, C. I., Gould, I. R., Merz, K. M., Ferguson, D. M., Spellmeyer, D. C., Fox, T., Caldwell, J. W., and Kollman, P. A.* "A Second Generation Force Field for the Simulation of Proteins, Nucleic Acids, and Organic Molecules" J. Am. Chem. Soc. 118, no. 9 (1996): 2309âĂŞ2309. doi:10.1021/ja955032e

[18] *D.A. Case, et. al and P.A. Kollman (2015),* "AMBER 2015", University of California, San Francisco.

[19] *Pettersen EF, Goddard TD, Huang CC, Couch GS, Greenblatt DM, Meng EC, Ferrin TE.* "UCSF Chimera–a visualization system for exploratory research and analysis." J Comput Chem. 2004 Oct;25(13):1605-12.

[20] *Nash, Stephen G. (2000).* "A survey of truncated-Newton methods". Journal of Computational and Applied Mathematics 124 (1âĂŞ2): 45âĂŞ59. doi:10.1016/S0377-0427(00)00426-X

[21] "SMARTS Theory Manual", Daylight Chemical Information Systems, Santa Fe, New Mexico. http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html

[22] Wolfram Research, Inc., Mathematica, Version 10.3, Champaign, IL (2015).

[23] *Boehm, H.,* "Simple Garbage-Collector-Safety", Proceedings of the ACM SIGPLAN '96 Conference on Programming Language Design and Implementation.

[24] *Richard Brooksby. (2002)* The Memory Pool System: Thirty person-years of memory management development goes Open Source. ISMM02.

[25] https://github.com/robert-strandh/SICL