Automatic Source Code Generation of Complicated Models For Deterministic Global Optimization With Parallel Architectures

Robert X. Gottlieb, Pengfei Xu, and Matthew D. Stuber[1]

Process Systems and Operations Research Laboratory, Department of Chemical and Biomolecular Engineering, University of Connecticut, Storrs, CT, USA

Abstract

Trends over the past two decades indicate that the majority of progress to accelerate solving optimization problems is due to improvements in x86 hardware. If we are to make progress beyond our manufacturing capabilities, it is imperative that alternative architectures be considered, especially those that can make use of massively parallelized operation. In this paper, we show how complex symbolic models can be generated for use in global optimization routines and then introduce an approach wherein these symbolic models are used to enable parallelized deterministic global optimization on GPUs. In two examples, we separately demonstrate the symbolic automatic differentiation of a complex thermodynamic model and the parallelization of a kinetic parameter estimation problem. A comparatively weak lower-bounding routine is run simultaneously on 5×10^4 branch-and-bound nodes at a time on a GPU, resulting in 95% problem convergence in 64% of the time of a standard, state-of-the-art CPU implementation of the problem. This work paves the way for the utilization of alternative hardware architectures that can compete with—or potentially outclass—the most powerful serial CPU implementations of deterministic global optimizers, and to the best of the authors' knowledge, represents the first successful demonstration of deterministic global optimization using GPUs.

Keywords

dynamical systems, parameter estimation, deterministic global optimization, open-source software

Introduction

The increased solution speed of optimization problems over the past two decades has been primarily due to advances in CPU hardware (Koch, 2022). Yet, despite this bottleneck, significant advances that exploit alternative architectures have largely been missing in optimization research. Parallel programming, for instance, is a way to exploit well-known architectures such as general-purpose graphics processing units (GPGPUs) that has the potential to outpace progress from Moore's law alone. Although the successes of distributed programming have been well demonstrated (Garland et al., 2008), to date, and to the best of the authors' knowledge, there has been no publication documenting the use of GPGPUs in deterministic global optimization. This paper details the first successful implementation, to the best of our knowledge, of performing deterministic global optimization using GPGPUs for massive parallelization.

Motivation

Process systems engineers are constantly faced with the curse of dimensionality, such as when solving parameter estimation problems involving first-principles models to global dynamic optimization. In parameter estimation problems, first-principles models of practical interest may have a large number of parameters that need to be estimated for model fitting and validation against experimental data. In dynamic optimization problems, the dimensionality of solvable problems is severely limited by the high complexity of the problem class. Without a method to accelerate the solution speed of these classes of problems, formulations with more than a few variables are unsolvable in a reasonable time span. This barrier limits the practicable applications of these classes and hinders the use of global optimization for these problems in the broader community.

A problem of specific interest in our research is the fitting of the electrolyte nonrandom two-liquid (eNRTL) model to experimental data to accurately simulate the performance of industrial water treatment systems. The purpose of the model is to represent the thermodynamic properties of multisolvent/multielectrolyte systems (Chen and Evans, 1986), and recent work has extended the model to better match high-concentration electrolyte systems (Bollas et al., 2008). This extension is critical for water treatment applications, where understanding the properties of high-concentration brines can lead to the development of better water separation techniques. However, the number of parameters to fit in this model grows polynomially with the number of species. For

[1] Corresponding author: M.D. Stuber (E-mail: stuber@alum.mit.edu).

such a system, finding guaranteed ε-global optimal parameters, even for industrially simple solutions, may take an impractically long time using traditional approaches. Our goal is to overcome this limitation through innovations in software tools for hardware-specific parallelization.

Dynamic optimization problems—those constrained by a system of ordinary differential equation initial value problems (ODE-IVPs)—represent another class whose solutions can only be achieved in practice for problems with low dimensionality. As an example to illustrate the complexity, one method of solving this type of problem is to discretize the system of ODE-IVPs, which reformulates them into a large number of algebraic equality constraints, and then solve the resulting NLP with a global optimizer. However, the resulting NLP may contain hundreds of thousands of decision variables and nonlinear equality constraints. Even the best-in-class commercial global solvers (*e.g.*, BARON (Sahinidis, 1996), ANTIGONE (Misener and Floudas, 2014)) have difficulty solving such high-dimensional problems in a reasonable time frame (Wilhelm et al., 2019). However, with parallel programming, doing so may become possible and extremely useful in practice.

Background

Adapting the B&B framework to exploit parallel computing requires restructuring the entire routine. Typically, this framework begins with a root node that covers the decision variables' interval domain. The lower-bounding problem proceeds by creating a convex relaxation of the objective function and obtaining the relaxation's infimum, which transitively applies to the objective function on the node's subdomain. The node is then partitioned (*i.e.*, *branched*) to create a pair of new nodes, and a similar lower-bounding problem is solved on the resulting nodes to obtain progressively improved lower bounds. This process is repeated until the greatest global lower bound converges to within some ε distance from the problem's lowest upper bound, which can be found using a variety of methods. In practice, this framework is applied in serial: a single node is selected as the working node, the lower-bounding problem is solved for this node, and following branching, the resulting nodes are added to a stack from which future working nodes will be selected. Parallelizing this process using multiple CPU cores is relatively straightforward: each CPU thread takes a working node from the stack and independently solves its lower-bounding problem. Using a hardware architecture different from x86, such as GPGPUs, creates additional complications.

To parallelize the processing of B&B nodes on a GPGPU, several criteria must be met that are not satisfied by standard B&B methods. First, GPGPU cores operate in lockstep—unlike in a CPU, where different cores can execute entirely independent sets of instructions—and all parallel GPGPU cores must execute the same sets of instructions simultaneously. That is, if a calculation is performed on one GPGPU core, an identical calculation must be performed on all parallel cores. An example of this is the computation of $\mathbf{C} = \mathbf{A} + \mathbf{B}$, with $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{R}^{m \times n}$, where GPGPUs simultaneously execute the elementwise addition. This restriction is problematic for lower-bounding problems because the mathematical structure of each node's convex relaxations may be different and the process of finding each relaxation's lower bound generally requires a unique and variable number of iterations. Second, the performance of GPGPUs is slowed by control flow. Any process that requires different evaluations to be made depending on some set of criteria—such as when calculating the structure of node-dependent relaxations—will not be fast on a GPGPU. To fully exploit GPGPU architectures and the speed-up that comes from parallelization, these hurdles must be overcome.

The first step in addressing these criteria is to develop a method for constructing relaxations that can be calculated in parallel for multiple nodes. A typical method of calculating relaxations is to represent variables as McCormick objects with associated lower and upper bounds and convex and concave relaxations and then evaluate the nonconvex functions of interest using the rules of McCormick (McCormick, 1976; Scott et al., 2011). This operation results in pointwise evaluations of convex relaxations and their associated (sub)gradients in the search space, which can be used to determine global lower bounds through convex optimization algorithms such as IPOPT (Wächter and Biegler, 2005). This method is not appropriate for parallelization on a GPGPU because the McCormick-based relaxation rules are dependent on the lower and upper bounds on the variables, which necessarily differ between nodes.

This paper introduces a novel method implemented as the open-source `SourceCodeMcCormick.jl` package, which applies the McCormick-based relaxation rules to symbolic functions without *a priori* knowledge of the variable bounds. This is accomplished via a source-code transformation approach that begins by automatically decomposing the input symbolic function into a factorable representation (*i.e.*, a sequence of elementary operations and univariate intrinsic functions). The McCormick rules associated with each factor are applied symbolically (with the variable bounds expressed symbolically), and new symbolic functions representing interval extensions and convex/concave relaxations of the original function are constructed from the factors. Critically, the symbolic versions of the McCormick rules applied here are designed without typical control flow, satisfying the second GPGPU requirement and making the rules fast to evaluate on GPGPUs. Since the variable bounds are symbolic, the output from `SourceCodeMcCormick.jl` is partition independent. Consequently, convex/concave relaxations can be calculated on any subdomain using the same set of instructions, which satisfies the first GPGPU requirement. Additional details can be found in the Software Toolkit section.

The second point to address is the variable number of steps required for the lower-bounding problem. This hurdle is addressed by the black-box sampling technique of Song et al. (2021). In summary, given a B&B node bounded in $\mathbb{R}^n$, a rigorous lower bound for the node can be calculated using $2n+1$ pointwise evaluations of the objective function's convex relaxation, at points that can be determined *a priori* based on the bounds of the node. Thus, to calculate rigorous lower bounds for a *set* of nodes $\mathcal{F}_k$, all that is needed is to produce a *set* of $2n+1$ evaluation points corresponding to each node $X \in \mathcal{F}_k$, and then evaluate the objective function's convex relaxation at those points in parallel using the output from `SourceCodeMcCormick.jl`. Evaluations of these points can then be connected back to their original nodes and the lower bounds can be calculated using a single algebraic expression per node for box-constrained problems (Song et al., 2021).

Source-code transformation of symbolic functions has broad applicability for global optimization beyond only the scope of `SourceCodeMcCormick.jl`. One example is our interest in modeling water treatment systems, where the relevant thermodynamic properties can be obtained by differentiating Gibbs free energy expressions. Although packages for automatic differentiation exist and have abundant use cases, such techniques may be inappropriate in a global optimization context where knowing the mathematical structure of the derivative is often necessary to calculate tight relaxations. This paper presents the use of source-code transformation to obtain symbolic derivatives of a Gibbs free energy expression for the eNRTL model of interest, to construct a representation of the derivative that is more suited for global optimization routines such as the newly developed `SourceCodeMcCormick.jl`. Source-code transformation approaches such as this are of special importance in cases where the underlying expression is complicated, as manually calculating a derivative expression and inputting it into an optimization model can be highly error-prone.

Parameter Estimation in Process Systems Engineering

The problem we are focused on in this work is formulated generally as:

$$\begin{aligned} \mathbf{p}^* \in \arg \min_{\mathbf{p}\in P\subset\mathbb{R}^{n_p}} & \sum_{i=1}^{n_d} (y_i(\mathbf{p}) - y_i^{\text{data}})^2 \\ \text{s.t.} \quad & \mathbf{h}(\mathbf{p}) = \mathbf{0} \\ & \mathbf{g}(\mathbf{p}) \leq \mathbf{0}. \end{aligned} \tag{1}$$

The objective function is the sum of squared error (SSE) between the predictions of a model of interest $\mathbf{y}(\mathbf{p})$ and a set of experimental data $\mathbf{y}^{\text{data}}$, where $\mathbf{p}$ is the uncertain parameter vector and n_d is the number of experimental data points.

Refined Electrolyte Nonrandom Two-Liquid Model

Predicting the thermodynamic behavior of multielectrolyte/multisolvent systems has historically been challenging due to the complexity of the first-principles models. In this work, we are interested in the *refined* eNRTL developed by Bollas et al. (2008), and the reader is directed to that work for the full details. From the model's Gibbs free energy expression, all other thermodynamic state properties, such as heat capacity, enthalpy, and entropy can be derived.

To illustrate the source-code transformation approach, we will focus on the contribution of the short-range interaction to the molar excess Gibbs free energy $\underline{G}^{\text{SR}}$, defined as

$$\begin{aligned} \frac{\underline{G}^{\text{SR}}}{RT} = & \sum_{j=1}^{n_m} X_{m_j} \left(\frac{\sum_{s\in\{m,a,c\}} \sum_{l=1}^{n_s} X_{s_l} F_{m_j,s_l,m_j} \tau_{m_j,s_l,m_j}}{\sum_{s\in\{m,a,c\}} \sum_{l=1}^{n_s} X_{s_l} F_{m_j,s_l,m_j}} \right) \\ & + \sum_{j=1}^{n_a} X_{a_j} \sum_{k=1}^{n_c} \left(\frac{X_{c_k}}{\sum_{k'=1}^{n_c} X_{c_{k'}}} \right) \left(\frac{\sum_{s\in\{m,c\}} \sum_{l=1}^{n_s} X_{s_l} F_{a_j,s_l,c_k} \tau_{a_j,s_l,c_k}}{\sum_{s\in\{m,c\}} \sum_{l=1}^{n_s} X_{s_l} F_{a_j,s_l,c_k}} \right) \\ & + \sum_{j=1}^{n_c} X_{c_j} \sum_{k=1}^{n_a} \left(\frac{X_{a_k}}{\sum_{k'=1}^{n_a} X_{a_{k'}}} \right) \left(\frac{\sum_{s\in\{m,a\}} \sum_{l=1}^{n_s} X_{s_l} F_{c_j,s_l,a_k} \tau_{c_j,s_l,a_k}}{\sum_{s\in\{m,a\}} \sum_{l=1}^{n_s} X_{s_l} F_{c_j,s_l,a_k}} \right). \end{aligned} \tag{2}$$

Here, τ_{t_j,o_k,r_l} refers to the local interaction strength difference between a center species t_j and an objective species o_k (the k^{th} species of type o), in reference to species r_l, where the type r is determined based on t (Chen and Evans, 1986). The term $F_{t_j,o_k,r_l} = \exp(-\alpha\tau_{t_j,o_k,r_l})$ follows the same structure, with $\alpha = 0.3$ for $t,o,r = m$ and $\alpha = 0.2$, otherwise. Additionally, X_{t_j} represents the effective mole fraction of species j of type $t \in \{m,a,c\}$ in the solution, defined as

$$X_{t_j} = \frac{N_{t_j} C_{t_j}}{\sum_{t\in\{m,a,c\}} \sum_{j=1}^{n_t} N_{t_j} - \sum_{t\in\{a,c\}} \sum_{j=1}^{n_t} N_{t_j} h_{t_j}}, \tag{3}$$

where m represents neutral solvent molecules, a the anions, and c the cations, with the cardinality of each type represented as n_m, n_a, and n_c, respectively. *E.g.*, X_{c_4} would correspond to the fourth unique species of cation in solution. This indexing is also used for the number of moles N_{t_j}, the hydration number h_{t_j}, the magnitude of the charge number C_{t_j} (or 1 for neutral species), and the short-range interaction contribution to the activity coefficient $\gamma_{t_j}^{\text{SR}}$, defined as

$$\log \gamma_{t_j}^{\text{SR}} = \frac{\partial}{\partial N_{t_j}} \left(\sum_{\hat{t}\in\{m,a,c\}} \sum_{\hat{j}=1}^{n_{\hat{t}}} N_{\hat{t}_{\hat{j}}} \frac{\underline{G}^{\text{SR}}}{RT} \right). \tag{4}$$

For the optimization problem given by (1), $y_i(\mathbf{p}) = \gamma_{t_j}^{\text{SR}}(\mathbf{p},\mathbf{u}_i)$, where $\mathbf{u}_i$ represents the temperature and concentrations of each species for experiment i, *i.e.*, $\mathbf{u}_i = (T, N_{m_1}, \ldots, N_{m_{n_m}}, N_{a_1}, \ldots, N_{a_{n_a}}, N_{c_1}, \ldots, N_{c_{n_c}})_i$. Given the multiplicative dimensionalities of τ, it is clear that as the number of species grows, the number of parameters to estimate increases polynomially. *E.g.*, when there are 10 unique species of anions and cations, which is common for brines and industrial wastewater, there are 1140 parameters to fit. This presents an extreme challenge if a globally optimal solution is required, as even small practical systems may have an intractable number of parameters. Additionally, even in situations where the unique species count is low, generating a derivative and inputting the expression manually into a global optimizer is both difficult and error-prone. For this reason, we seek to use source-code transformation to automatically generate a full symbolic expression of the derivative that can be directly inputted into an optimization routine.

Transient Absorption Kinetics Model

Another example is the kinetic parameter estimation problem originally described by Taylor (2005). This problem consists of a system of ODEs that describes the concentrations of several species after an initial laser flash pyrolysis, as given by:

$$
\begin{aligned}
\frac{dx_A}{dt} =& k_1 x_Z x_Y - c_{O_2}(k_{2f}+k_{3f})x_A + \frac{k_{2f}}{K_2}x_D + \frac{k_{3f}}{K_3}x_B \\
& - k_5 x_A^2, \\
\frac{dx_B}{dt} =& c_{O_2}k_{3f}x_A - \left(\frac{k_{3f}}{K_3}+k_4\right)x_B, \\
\frac{dx_D}{dt} =& c_{O_2}k_{2f}x_A - \frac{k_{2f}}{K_2}x_D, \\
\frac{dx_Y}{dt} =& -k_{1s}x_Z x_Y, \\
\frac{dx_Z}{dt} =& -k_1 x_Z x_Y, \\
& x_A(0) = x_B(0) = x_D(0) = 0,\ x_Y(0) = 0.4,\ x_Z(0) = 140.
\end{aligned} \tag{5}
$$

Here, x_j is the concentration of species $j \in \{A,B,D,Y,Z\}$, and the given constants are $T = 273$, $K_2 = 46\exp(6500/T - 18)$, $K_3 = 2K_2$, $k_1 = 53$, $k_{1s} = k_1 \times 10^{-6}$, $k_5 = 1.2 \times 10^{-3}$, and $c_{O_2} = 2 \times 10^{-3}$. The uncertain model parameters are the rate constants $\mathbf{p} = (k_{2f}, k_{3f}, k_4)$ with $k_{2f} \in [10, 1200]$, $k_{3f} \in [10, 1200]$, and $k_4 \in [0.001, 40]$. The objective is to minimize the sum-squared error between this model and experimentally measured intensity data, where the intensity has a known dependency on concentrations as $I = x_A + \frac{2}{21}x_B + \frac{2}{21}x_D$, which comes from the Beer-Lambert law for relating measured absorbance to concentration with a correction for multiple species (Singer, 2004).

Software Toolkit

The software contributions described in this paper are designed as core components and as extensions of the open-source deterministic global optimizer EAGO (Wilhelm and Stuber, 2020). The base EAGO package embeds a novel McCormick-based relaxation library to construct convex and concave relaxations of a nonconvex functions and uses the relaxations in an implementation of B&B. Most notably, in the context of the present paper, one of EAGO's design principles is to function as a research platform, in the sense that almost all of the core functionality can be extended and modified to address any problem-specific complexities or irregularities. This is shown in the documentation and in numerous Jupyter notebook examples on the group GitHub page. This extensibility enables new work, such as what is presented in this paper, to be incorporated seamlessly into EAGO.

McCormick-Based Relaxation Library

A key component of the EAGO ecosystem is the `McCormick.jl` package that contains a library of forward-mode McCormick-based operators to support the computation of convex/concave relaxations of complicated factorable expressions. There is support for operators ranging from common algebraic expressions such as `min` and `sqrt` to activation functions for machine learning applications such as `leaky_relu` and `gelu`.

In particular, the capabilities of the `McCormick.jl` library are accessed using *McCormick objects* that are tuples of the form $\{L,U,CV,CC,SCV,SCC\}$, representing a variable's lower bound, upper bound, convex/concave relaxations, and (sub)gradients of its convex/concave relaxations, respectively. In a manner analogous to using intervals of the form $[L,U]$ with interval arithmetic, these McCormick objects can be used to evaluate factorable functions comprising elementary arithmetic operations and transcendental functions, which will return McCormick object representations of the evaluated functions. This interaction is made possible by Julia's *multiple dispatch* paradigm, which is a feature of the language that allows specialized versions of functions to be called based on the runtime types of the inputs; somewhat similar to *operator overloading* in other languages such as C++. In `McCormick.jl`, elementary and transcendental functions are defined that operate on McCormick objects and automatically apply the appropriate McCormick rules associated with the overloaded function. A similar operator overloading approach is used in C++ implementations of the McCormick rules, such as those used in MC++ (Mitsos et al., 2009) and the MAiNGO solver (Bongartz et al., 2018). McCormick objects that represent relaxations of objective functions or constraints can then be used directly in global optimization algorithms.

Source-Code Generation/Transformation

Our novel approach is implemented in the software package `SourceCodeMcCormick.jl`, which parses the nonconvex functions we wish to relax for optimization and generates source code representations of the convex/concave relaxations for *any* domain. This is in contrast to `McCormick.jl` which uses multiple dispatch to evaluate *at runtime* the McCormick-based relaxation rules on a prescribed domain for a function that we wish to relax, for a single evaluation point. `SourceCodeMcCormick.jl` gives up the flexibility to evaluate different functions at runtime in exchange for the flexibility to evaluate any points and bounds within the search space for a single, pre-defined function.

To create this functionality, `SourceCodeMcCormick.jl` utilizes `Symbolics.jl` to represent symbolically the factorable functions to be evaluated. This symbolic representation is then internally factorized to create a list of subexpressions corresponding to the factorable representation. McCormick rules are applied to each subexpression, and the results are symbolically recombined through substitution to generate full symbolic expressions of convex/concave relaxations and interval bounds of the original expression. A key part of this implementation is how bounds-dependent McCormick rules are handled. In `McCormick.jl`, rules have different cases depending on the bounds of the input McCormick objects. In `SourceCodeMcCormick.jl`, these rules, with all their possible cases, are embedded within single expressions using `if-else` statements. Julia evaluates `if-else` statements without branching, which enables fast

GPGPU performance as the same expression (albeit a complicated one) is evaluated regardless of the conditions.

Although the resulting expressions from `SourceCodeMcCormick.jl` are cumbersome, part of their benefit is that the McCormick rules have been applied upfront in their creation. This eliminates the need to spend time determining which rules to follow and which cases to use at runtime, as is the case with `McCormick.jl` every time an expression is evaluated. Thus, `SourceCodeMcCormick.jl` incurs an upfront cost to generate a source-code-transformed evaluation function, but then significantly lowers runtime costs when the expression is evaluated.

To demonstrate this benefit, consider the simple function $f(x,y,z) = (5-x)^2 + (y+2)^2 + xy/z$ with $x \in [3.0, 7.0]$, $y \in [-3.0, 3.0]$, and $z \in [1.0, 5.0]$. Using `McCormick.jl`, calculating a convex relaxation of f at 10^5 randomly selected points in these bounds takes $\approx$ 43ms. Using `SourceCodeMcCormick.jl`, creating the callable convex evaluation function takes $\approx$ 200ms (the high upfront cost), but evaluating the same 10^5 randomly selected points takes only $\approx 28\mu$s, or about a 1500x speedup when utilizing an NVIDIA Quadro GV100 GPGPU. In a global optimization routine, hundreds of millions of pointwise evaluations of the convex relaxations of the objective function may be required. With such a large number of evaluations, the upfront cost of creating the evaluation function is more than offset by the speed of evaluating the relaxations at these points.

Exploiting GPGPU Architectures

The ability to parallelize lower-bounding problems for a GPGPU requires restructuring EAGO's normal B&B routine, which is easily accomplished by exploiting EAGO's extensibility with a custom extension. In the normal B&B routine, each iteration proceeds by fathoming nodes from the main problem stack, selecting a node from the stack to evaluate, solving the lower- and upper-bounding problems for that node, and branching. To support parallelization, an extension was created that contains a "substack" of nodes. The parallel B&B routine then proceeds first by fathoming, then by repeatedly selecting nodes from the main stack and adding them to the substack until a predefined limit has been reached, and then processing all nodes in the substack simultaneously to determine their lower and upper bounds. Nodes are then removed from the substack in sequence, given their calculated bounds, and branched on to be added back onto the main stack. Unlike the standard *serial* B&B routine, where each iteration processes a single node at a time, this parallel routine processes many thousands of nodes simultaneously in each iteration. By adjusting how nodes are selected from the main stack, both breadth-first and depth-first branching heuristics can be adopted within this framework.

Preliminary Results

eNRTL Model

To illustrate the functional complexity necessitating the automatically-generated source-code approach, we present here the results for a simple solution of aqueous NaCl. In this case, there are 6 decision variables that must be determined through optimization: $\mathbf{p} = (h_{a_1}, h_{c_1}, \tau_{m_1,a_1,m_1}, \tau_{m_1,c_1,m_1}, \tau_{a_1,m_1,c_1}, \tau_{c_1,m_1,a_1})$, and there are 2 terms to compare with experimental data: $\log\gamma_{c_1}^{SR}$ and $\log\gamma_{a_1}^{SR}$, representing the activity coefficients of Na^+ and Cl^-, respectively. As a demonstrative example, $\log\gamma_{c_1}^{SR}$ is generated as:

```
Nm1_f=Nm1-Na1*ha1-Nc1*hc1
log_gamma_c1=(tC1M1A1*Nm1_f*exp(-0.2tC1M1A1))
  / (Na1+Nm1_f*exp(-0.2tC1M1A1))
  + (tM1C1M1*Nm1_f*exp(-0.2tM1C1M1))
  / (Nm1_f+Na1*exp(-0.2tM1A1M1)+Nc1*exp(-0.2tM1C1M1))
  + (-Na1*tA1M1C1*Nm1_f*exp(-0.2tA1M1C1))
  / ((Nc1+Nm1_f*exp(-0.2tA1M1C1))^2)
  - (((Na1*tM1A1M1*exp(-0.2tM1A1M1)
  + Nc1*tM1C1M1*exp(-0.2tM1C1M1))*Nm1_f)
  / ((Nm1_f+Na1*exp(-0.2tM1A1M1)
  + Nc1*exp(-0.2tM1C1M1))^2))*exp(-0.2tM1C1M1)
```

Although complicated, this symbolic expression can then be used directly for simulation or sent to `SourceCodeMcCormick.jl` to generate code for convex/concave relaxations and interval bounds. While we present only this relatively simple term, the source-code transformation approach can be used to generate symbolic expressions for an arbitrary number of m, a, c species.

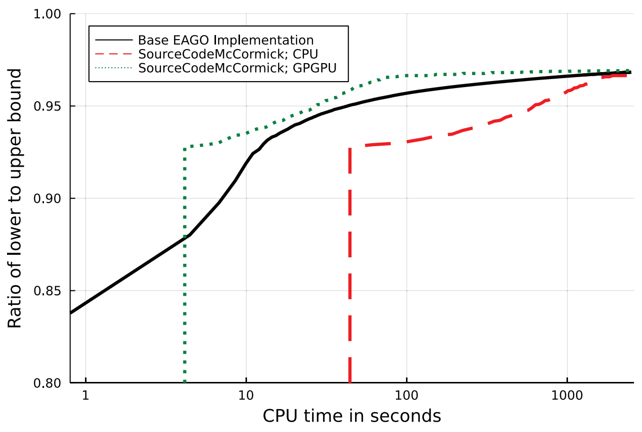

Figure 1: Convergence plot for the kinetic parameter estimation problem.

Transient Absorption Kinetics Model

To solve the kinetic parameter estimation problem, an explicit Euler discretization was used to approximate the solution of the initial value problem. The problem was then sent to the base version of EAGO, which achieved 95% convergence in 43.1s, as shown in Figure 1. The problem was then solved using the `SourceCodeMcCormick.jl` source-code transformation approach which used the explicit Euler discretization to generate a function representation of the objective function's convex relaxation. As described previously, this approach employs a black-box sampling approach to solve the lower-bounding problem for multiple nodes simultaneously. The black-box approach produces weaker bounds than convex optimization methods that make use of (sub)gradients, but with massive parallelization, solving a large number of these problems simultaneously may lead to faster global convergence than tighter, single-node methods.

The `SourceCodeMcCormick.jl` method was used in two ways: first with an Intel W-2195 (*i.e.*, single core of a CPU), and second with an NVIDIA Quadro GV100 GPGPU. In each case, 5×10^4 nodes were solved simultaneously using the source-code-generated function to evaluate the convex relaxations of the objective function. As shown in Figure 1, the `SourceCodeMcCormick.jl` method, when performed on the x86 architecture, performs slower than the base EAGO implementation of the problem. Although many nodes are evaluated simultaneously, the weaker bounds of the black-box method cannot compete with the (sub)gradient techniques on standard x86 hardware, reaching 95% convergence in 633.3s. When the same algorithm is paired with a GPGPU, however, the benefits of the alternative architecture's parallelization becomes clear: 95% convergence is achieved in 27.6s, or roughly two-thirds of the time of the base EAGO algorithm.

Conclusion

Source-code transformation and generation approaches are of special importance in cases where the mathematical structure of an expression plays a role in the solution technique. Deterministic global optimization falls in this category, as convex/concave relaxations of nonconvex functions are dependent on both the operations that comprise those functions as well as the overall composition/structure of those functions. This paper documented two cases where source-code transformation was useful for global optimization. First, in a problem rooted in thermodynamics where the objective function was described by the derivative of a complicated expression, and second, in the implementation of a method that used the problem structure to exploit an alternative hardware architecture. In both cases, source-code transformation was critical to arriving at a final result that was directly useful in a deterministic global optimization context.

Acknowledgements

Funding is provided in part by the National Alliance for Water Innovation (NAWI) funded by the U.S. Department of Energy, Office of Energy Efficiency and Renewable Energy (EERE), Advanced Manufacturing Office, under Funding Opportunity Announcement Number DE-FOA-0001905, and by the National Science Foundation under Grant No. 1932723. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, NAWI, or the Department of Energy.

References

Bollas, G. M., C. C. Chen, and P. I. Barton (2008). Refined electrolyte-NRTL model: Activity coefficient expressions for application to multi-electrolyte systems. *AIChE Journal 54*(6), 1608–1624.

Bongartz, D., J. Najman, S. Sass, and A. Mitsos (2018). MAiNGO - McCormick-based Algorithm for mixed-integer Nonlinear Global Optimization.

Chen, C.-C. and L. B. Evans (1986, mar). A local composition model for the excess gibbs energy of aqueous electrolyte systems. *AIChE Journal 32*(3), 444–454.

Garland, M., S. L. Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, and V. Volkov (2008, jul). Parallel computing experiences with CUDA. *IEEE Micro 28*(4), 13–27.

Koch, T. (2022, July). Progress in mathematical programming solvers from 2001 to 2020. Presentation at EURO 2022, Espoo, Finland.

McCormick, G. P. (1976, dec). Computability of global solutions to factorable nonconvex programs: Part I — convex underestimating problems. *Mathematical Programming 10*(1), 147–175.

Misener, R. and C. A. Floudas (2014, mar). ANTIGONE: Algorithms for coNTinuous / integer global optimization of nonlinear equations. *Journal of Global Optimization 59*(2-3), 503–526.

Mitsos, A., B. Chachuat, and P. I. Barton (2009, jan). McCormick-based relaxations of algorithms. *SIAM Journal on Optimization 20*(2), 573–601.

Sahinidis, N. V. (1996, mar). BARON: A general purpose global optimization software package. *Journal of Global Optimization 8*(2), 201–205.

Scott, J. K., M. D. Stuber, and P. I. Barton (2011, feb). Generalized McCormick relaxations. *Journal of Global Optimization 51*(4), 569–606.

Singer, A. B. (2004). *Global Dynamic Optimization*. Ph. D. thesis, Massachusetts Institute of Technology.

Song, Y., H. Cao, C. Mehta, and K. A. Khan (2021, oct). Bounding convex relaxations of process models from below by tractable black-box sampling. *Computers & Chemical Engineering 153*, 107413.

Taylor, J. W. (2005, September). *Direct Measurement and Analysis of Cyclohexadienyl Oxidation*. Ph. D. thesis, Massachusetts Institute of Technology.

Wilhelm, M. E., A. V. Le, and M. D. Stuber (2019, nov). Global optimization of stiff dynamical systems. *AIChE Journal 65*(12).

Wilhelm, M. E. and M. D. Stuber (2020, aug). EAGO.jl: easy advanced global optimization in Julia. *Optimization Methods and Software*, 1–26.

Wächter, A. and L. T. Biegler (2005, apr). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming 106*(1), 25–57.