Thermo4PFM: Facilitating Phase-field simulations of alloys with thermodynamic driving forces *

Jean-Luc Fattebert[a,*], Stephen DeWitt[a], Aurelien Perron[b], John Turner[a]

[a] *Computational Sciences and Engineering Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA*
[b] *Lawrence Livermore National Laboratory, Livermore, CA 94550, USA*

Abstract

Phase-field modeling is a popular front-tracking approach used to model solidification. Its time-evolution equations are often coupled to alloy composition and/or thermal diffusion in high-resolution multiphysics approaches. Materials thermodynamic properties tabulated in CALPHAD databases can be used for phase-field modeling to parameterize bulk energies of alloys. In addition, they can be naturally integrated into models such as the Kim-Kim-Suzuki (KKS) model where driving forces depend on the the differences between chemical potentials of co-existing phases. In that case, a small system of coupled nonlinear equations needs to be solved at every point in space where the phase-field order parameter is to be updated and evolved in time. We present Thermo4PFM, a solver for the KKS equations for binary and ternary alloys, with two or three phases, and parameterized with CALPHAD models. Thermo4PFM is open source, written in C++, and can take advantage of Graphics Processing Units (GPU) accelerators. Using OpenMP offload capabilities for C++ classes, an excellent performance is demonstrated on GPU using the LLVM compiler. CALPHAD data is read from simple JSON files using an open source parser from the *boost* library.

Keywords: Phase-field model, CALPHAD, Graphics Processing Units (GPU)

PROGRAM SUMMARY

Program Title: Thermo4PFM
CPC Library link to program files: (to be added by Technical Editor)

*Corresponding author.
E-mail address: fattebertj@ornl.gov

Developer's repository link: https://github.com/ORNL/Thermo4PFM
Code Ocean capsule: (to be added by Technical Editor)
Licensing provisions: BSD 3-clause
Programming language: C++/OpenMP
Nature of problem(approx. 50-250 words):
Accurate modeling of solidification in metallic alloys requires thermodynamic data associated with possible material phases. That data can be used in phase-field modeling (PFM) to evaluate the driving force responsible for phase changes and solidification front motion. Integrating that data into PFM requires the solution of a small system of coupled nonlinear differential equations — the Kim-Kim-Suzuki equations — that needs to be solved at every point of a discretization mesh.
Solution method(approx. 50-250 words):
Thermo4PFM implements a Newton-based solver for the Kim-Kim-Suzuki equations for a few special cases (binary and ternary alloys with two or three phases) for CALPHAD-based models of the bulk energy of the phases. The software is written in C++ and uses the Curiously Recurring Template Pattern and OpenMP offload capabilities to take advantage of GPU accelerators, when available, on modern High Performance Computing resources. It also uses the *boost* Property Tree library to parse input CALPHAD data.

1. Introduction

The Kim-Kim-Suzuki (KKS) model [1] is very popular in phase-field modeling (PFM) of alloys. It introduces an auxiliary set of variables, the alloy composition in each phase, that allows for a smooth transition for the alloy composition field through the PFM diffuse interface. An earlier model by Tiaden et al. [2] had already introduced the concept of "internal" phase composition in the dilute alloy case. That model assumed a constant ratio between the phase compositions given by the partition coefficient. The more general KKS model case however comes with the cost of having to solve a system of nonlinear equations at every mesh point of the discretization mesh to calculate the values of these phase compositions.

In parallel, the PFM community started to embrace the use of CALPHAD databases to model the bulk energies of various phases [3, 4, 5, 6, 7], leading to more complicated expressions for the bulk energies of various phases. This leads to more complex equations to solve in the KKS model, which can become the dominant part of the computational time in a PFM simulation. To reduce the cost of these calculations, various schemes have been proposed. Eiken et al. [8] suggest not to solve the thermodynamic equations exactly at every time step, but only after a certain interval when significant changes have occurred and use extrapolations in between. With CALPHAD databases, parabolic fit of the data have also been proposed for binary alloys [9]. These fits are easy to work with around equilibrium compositions in each phase for binary alloys, and lead to a linear system of equations, avoiding the use of a non-linear iterative solver. For ternary and higher multi-component alloys, this is more complicated since there exist multiple equilibrium compositions, in contrast to binary alloys for

which there exists a unique equilibrium composition at a given temperature for each phase [10]. A recent computational framework was developed by Noubary et al. [11] to enable a systematic improvement of quadratic fits to better match original CALPHAD data as needed. A completely different idea was presented by Joshi et al. [12] who proposed to discretize the composition space with a uniform mesh and use linear interpolation functions between node values to represent any thermodynamic quantity in that space.

In this paper, we describe Thermo4PFM capabilities and their implementation. Thermo4PFM was designed to accurately solve the KKS equations for binary and ternary alloys with CALPHAD thermodynamic data. It also provides some additional functionalities useful in PFM, such as computing equilibrium compositions. While some of the functionalities provided by Thermo4PFM may be found in other software such as Thermo-Calc™ [13], they are typically not designed to be at the core of intensive numerical kernels as used in a PFM code. Thermo4PFM on the other hand is designed with that purpose in mind: solving numerous system of equations on the fly at every mesh point of a discretization, for many timesteps. That is, Thermo4PFM is not itself a PFM simulation code, it is a thermodynamic library meant to be called pointwise from a PFM simulation code. We start by describing the KKS approach in PFM for binary and ternary alloys, for two or three phases, as well as the CALPHAD approach in Section 2. Section 3 then describes the software, its implementation as well as some performance results, and an application example.

2. KKS-based Phase-field model

In this section, we describe solvers for four cases currently supported in Thermo4PFM: binary alloys, ternary alloys, dilute binary alloys for two phases, as well as binary alloys with three phases. With N atomic species, there is always the constraint that the sum of the N components adds up to one,

$$\sum_{i=1}^{N} c_i = 1. \tag{1}$$

In this paper, as in Thermo4PFM in general, we always use Eq. (1) to eliminate the last components, working with $c_i, i = 1, \ldots, N-1$ only and assuming $c_N = 1 - \sum_{i=1}^{N-1} c_i$. Also, in the following, alloy compositions always refer to molar fractions.

The binary alloy with two phases (liquid + solid) will be used to illustrate the general ideas in this section, and details for other models will be described later. Let us first write a PFM energy functional in the general form

$$F\left[\phi, c, T\right] \equiv \int_V \{f_{int}(\phi) + f_{bulk}(\phi, c, T)\} dv, \tag{2}$$

composed of an "interfacial" and a "bulk" energy density integrated over a computational domain V. The interfacial energy density f_{int} is defined as

$$f_{int}(\phi(x)) = \frac{\epsilon_\phi^2}{2} |\nabla \phi(x)|^2 + f_{dw}(\phi(x)), \tag{3}$$

where in general the first term can be replaced with an anisotropic expression and f_{dw} denotes here the "double well" potential given by

$$f_{dw}(\phi(x)) = \phi(x)^2(1-\phi(x))^2. \tag{4}$$

The choice of the double well potential here is not necessary, and a "double obstacle" [14] could also be used.

We use the convention $\phi = 0$ for liquid and $\phi = 1$ for solid. We focus here on the model proposed by Kim, Kim and Suzuki [1, 15] which assumes coexistence of solid and liquid (or more) phases at each point in space. Each phase is then associated with a composition such that the weighted sum of all the phase concentrations is equal to the local concentration where the weights are given by the phase fractions (or by functions of those). For instance, for a binary alloy of composition c and two phases (liquid and solid),

$$c = (1 - h_r(\phi))c_L + h_r(\phi)c_S \tag{5}$$

where h_r is an interpolation polynomial that satisfies $h_r(0) = 0$, $h_r(1) = 1$, and $h_r(\phi) = 1 - h_r(1-\phi)$. In this context, it is appropriate to write the bulk energy f_{bulk} as a sum of contributions from each phase. For a binary alloy and two phases (liquid + solid), it is given by

$$f_{bulk}(\phi, c, T) = (1 - h_p(\phi))f_L(c_L, T) + h_p(\phi)f_S(c_S, T). \tag{6}$$

where h_p is also an interpolation polynomial that satisfies $h_p(0) = 0$, $h_p(1) = 1$, and $h_p(\phi) = 1 - h_p(1-\phi)$. We use the notations h_p, h_r as introduced by Kim [16] to distinguish between the interpolation polynomials used for the energy and the composition. The KKS model assumes that the two co-existing phases are at equilibrium, that is their chemical potentials satisfy

$$\mu = \frac{\partial f_L}{\partial c_L} = \frac{\partial f_S}{\partial c_S}. \tag{7}$$

This equation, together with Eq. (5), define a system of two equations to solve to determine the two unknowns c_L and c_S.

As usual in PFM, we assume that the phase variable ϕ evolves in time following an Allen-Cahn equation that drives the energy F towards a minimum,

$$\dot{\phi} = -M\frac{\delta F}{\delta \phi} \tag{8}$$

where M is a phase field mobility coefficient. We split the driving force $\delta F/\delta\phi$ into two components: a PFM interface component, responsible for maintaining the smooth PFM interface, and a thermodynamic component, responsible for the solidification front propagation,

$$\frac{\partial f_{int}}{\partial \phi} + \frac{\partial f_{bulk}}{\partial \phi} \tag{9}$$

We denote by g_{thermo} the second one, the thermodynamic component. For the specific case of a binary alloy again, it takes the form

$$g_{thermo} = h_p'(\phi)\left[f_L(c_L, T) - f_S(c_S, T) - \mu(c_L - c_S)\right]. \tag{10}$$

This formulation specifically requires an energy evaluation in each phase using its own composition. This is precisely the type of energy one can find in databases using CALPHAD parameterizations as will be described in Section 2.5. Note that the phase compositions obtained by solving the KKS equations may also be used to evaluate the composition time-evolution (diffusion) [8, 17]

$$\dot{c} = \nabla\left[\phi D^S \nabla c_S + (1-\phi) D^L \nabla c_L\right]. \tag{11}$$

2.1. Binary alloys

As noted above, for two phases (liquid and solid) and a binary alloy, the KKS model is characterized by c_L and c_S, the compositions in the liquid and solid phases, that satisfy the system of equations

$$\begin{aligned} c &= (1 - h_r(\phi))c_L + h_r(\phi)c_S, \\ \frac{\partial f^S}{\partial c_S} &= \frac{\partial f^L}{\partial c_L} \end{aligned} \tag{12}$$

with the constraint $0 \leq c_S, c_L \leq 1$. Using c_S and c_L solutions of (12), one can readily evaluate the energy density (6).

In some cases the equilibrium composition in each phases is necessary to calculate, such as for evaluting phase-field mobilities according to Kim's formula [16]. The computation of equilibrium compositions at a given temperature also requires an equal chemical potential equation. For binary alloys, the equilibrium compositions in each phase must satisfy the system of two equations

$$\begin{aligned} f^L(c_L^e, T) - f^S(c_S^e, T) &= (c_L^e - c_S^e)\left.\frac{\partial f^S}{\partial c_S}\right|_{c_S = c_S^e}, \\ \left.\frac{\partial f^S}{\partial c_S}\right|_{c_S = c_S^e} &= \left.\frac{\partial f^L}{\partial c_L}\right|_{c_L = c_L^e} \end{aligned} \tag{13}$$

2.2. Ternary alloys

The KKS model generalizes to multi-component alloys. For a ternary alloy, we have to solve a system of four equations [15]

$$\begin{aligned} c_0 &= (1 - h_r(\phi))c_{L,0} + h_r(\phi)c_{S,0}, \\ c_1 &= (1 - h_r(\phi))c_{L,1} + h_r(\phi)c_{S,1}, \\ \frac{\partial f^S}{\partial c_{S,0}} &= \frac{\partial f^L}{\partial c_{L,0}}, \\ \frac{\partial f^S}{\partial c_{S,1}} &= \frac{\partial f^L}{\partial c_{L,1}} \end{aligned} \tag{14}$$

where $c_{L,0}, c_{S,0}$ and $c_{L,1}, c_{S,1}$ are the two independent components of the composition field in each phase ($c_{L,2} = 1 - c_{L,0} - c_{L,1}$, $c_{S,2} = 1 - c_{S,0} - c_{S,1}$).

For ternary alloys, we also consider the equations that give the compositions in a two-phase mixture along a tie line passing through a nominal composition (c_0, c_1). While not directly used to solve PFM equations, these can be helpful in understanding and checking the thermodynamics of a specific CALPHAD parameterization. We can compute the compositions $c^t_{L,0}, c^t_{L,1}, c^t_{S,0}, c^t_{S,1}$ by solving the set of five equations

$$\begin{aligned}
\left.\frac{\partial f^S}{\partial c_{S,0}}\right|_{c_{S,0}=c^t_{S,0}} &= \left.\frac{\partial f^L}{\partial c_{L,0}}\right|_{c_{L,0}=c^t_{L,0}}, \\
\left.\frac{\partial f^S}{\partial c_{S,1}}\right|_{c_{S,1}=c^t_{S,1}} &= \left.\frac{\partial f^L}{\partial c_{L,1}}\right|_{c_{L,1}=c^t_{L,1}}, \\
f^L(c_{L,0}, c_{L,1}, T) - f^S(c_{S,0}, c_{S,1}, T) &= (c^t_{L,0} - c^t_{S,0}) \left.\frac{\partial f^S}{\partial c_{S,0}}\right|_{c_{S,0}=c^t_{S,0}} \\
&\quad + (c^t_{L,1} - c^t_{S,1}) \left.\frac{\partial f^S}{\partial c_{S,1}}\right|_{c_{S,1}=c^t_{S,1}}, \\
c_0 &= (1-\phi)c^t_{L,0} + \phi c^t_{S,0}, \\
c_1 &= (1-\phi)c^t_{L,1} + \phi c^t_{S,1}.
\end{aligned} \tag{15}$$

Note that unlike Eq.(14), the solid fraction ϕ here is a variable.

2.3. Binary alloy with three phases

Let us now consider a system with two components and three phases. This is specifically a case that is used to model eutectic solidification [18]. We will assume one phase is liquid, denoted by L, and the other two phases are solid denoted by A and B. The corresponding phase fractions, noted ϕ_L, ϕ_A and ϕ_B, have to satisfy the constraint $\phi_L + \phi_A + \phi_B = 1$. To interpolate between phases, we consider more general functions h_L, h_A, and h_B that depend on all three arguments ϕ_L, ϕ_A, and ϕ_B [18]. To solve for the three unknowns c_L, c_A, and c_B, one has to solve the following system of three equations

$$\begin{aligned}
c = h_L(\phi_L, \phi_A, \phi_B)c_L + h_A(\phi_L, \phi_A, \phi_B)c_A + h_B(\phi_L, \phi_A, \phi_B)c_B, \\
\frac{\partial f^A}{\partial c_A} = \frac{\partial f^L}{\partial c_L}, \\
\frac{\partial f^B}{\partial c_B} = \frac{\partial f^L}{\partial c_L}
\end{aligned} \tag{16}$$

2.4. Dilute Binary alloy

For many binary alloys, one can approximate the low concentration part of the phase diagram by two straight solidus and liquidus lines. Such a "linearized" phase diagram can then easily be described by three parameters: (i) T_m, the

temperature where the solidus and liquidus lines intersect the vertical axis at $c = 0$, (ii) m_L, the slope of the liquidus line, and (iii) k, the partition coefficient that is given in this case by the ratio of the liquidus and solidus slopes. This dilute binary approximation is very popular in the PFM community (see for instance [19, 20, 21]).

We consider here the dilute KKS model proposed in [1]. We write the energy densities for each phase as ideal solutions [22]

$$\begin{aligned} f_L(c_L, T) &= \frac{RT}{V}\left[c_L \ln(c_L) + (1 - c_L)\ln(1 - c_L)\right], \\ f_S(c_S, T) &= \frac{RT}{V}\left[c_S \ln(c_S) + (1 - c_S)\ln(1 - c_S) + c_S f_A + (1 - c_S) f_B\right] \end{aligned} \tag{17}$$

where f_A and f_B are sometimes interpreted as the free energies of the pure components A and B (*i.e.* for c=1 and c=0). As in the non-dilute case, the thermodynamic driving force is given by Eq. (10).

Using the fact that $g_{thermo} = 0$ for $c_L = c_L^e$ and $c_S = c_S^e$, and that $\mu_L = \mu_S$ when the two phases are at equilibrium, one obtains explicit expressions for f_A and f_B (see Appendix A) and subsequently the thermodynamic driving force given by

$$g_{thermo} = h'_p(\phi)\frac{RT}{V}\left[\ln(\frac{(1 - c_L)(1 - c_S^e)}{(1 - c_S)(1 - c_L^e)})\right] \tag{18}$$

where $c_L^e = (T - T_m)/m_L$ and $c_S^e = c_L^e \cdot k$ are the liquidus and solidus compositions at temperature T, respectively. This is Eq. (35) in Ref. [1].

The equations to be solved to determine c_L and c_S are the same as for the non-dilute case (12), albeit for simpler expressions of f_L and f_S. The equations to be solved to determine the equilibrium compositions at a given temperature are trivial in this case since those quantities are given directly by the linearized phase diagram parameters.

2.5. CALPHAD data

In the CALPHAD approach (see [23] for instance), the Gibbs energy of individual phases are parameterized as functions of temperature and alloy composition. For pure elements, the most commonly used model is the one proposed by the Scientific Group Thermodata Europe (SGTE). The SGTE data for the most common pure elements of the periodic table have been compiled by Dinsdale [24]. The Gibbs energy values are given relative to a standard element reference state (SER)

$$G_m - H_m^{SER} = a + bT + cTln(T) + \sum_j d_j T^j. \tag{19}$$

where H_m^{SER} is the enthalpy of the element in its stable phase at 298.15 K and ambient pressure, and the list of j-indices can include negative and positive integers. In this paper we ignore the pressure dependence and the magnetic contribution for simplicity. The coefficients a, b, c, and d_j are the model parameters and are collected in a database.

For multi-component systems, the Gibbs energy can be decomposed into the following form,

$$G = {}^{0}G + {}^{ideal}G_{mix} + {}^{xs}G_{mix} \tag{20}$$

where ${}^{0}G$ denotes the contribution from the mixing of the pure components, ${}^{ideal}G_{mix}$ is the ideal mixing contribution, and ${}^{xs}G_{mix}$ is the so-called excess Gibbs energy of mixing that accounts for non-ideal interactions. Models for the Gibbs energy are then parameterized for each phase of interest.

Gibbs energies are typically expressed in J/mol in CALPHAD databases. To obtain PFM energy densities, we can simply divide the corresponding CALPHAD Gibbs energies by the molar volume V_m expressed in m^3/mol. In this section we assume that molar volumes V_m are independent of composition, a common assumption in PFMs. We write the PFM free energy density for a phase Φ as

$$\begin{aligned} f^{\Phi}(\boldsymbol{c},T) &= {}^{0}f^{\Phi}(\boldsymbol{c},T) + {}^{ideal}f_{mix}(\boldsymbol{c},T) \\ &+ {}^{xs}f^{\Phi}_{mix}(\boldsymbol{c},T) \end{aligned} \tag{21}$$

where $\boldsymbol{c}$ denote the $N-$component vector of alloy compositions $\boldsymbol{c} = (c_0, \ldots, c_{N-1})^T$. The three terms on the right-hand side are

$${}^{0}f^{\Phi}(\boldsymbol{c},T) = \sum_{i=1}^{N} c_i \frac{{}^{0}G_i^{\Phi}(T)}{V_m} \tag{22}$$

$${}^{ideal}f^{\Phi}(\boldsymbol{c},T) = \frac{RT}{V_m} \sum_{i=1}^{N} c_i \ln(c_i) \tag{23}$$

and the excess Gibbs energy of mixing for a multicomponent alloy which can be written as

$${}^{xs}f^{\Phi}_{mix}(\boldsymbol{c},T) = \frac{1}{V_m} \sum_{i=1}^{N} \sum_{j=i+1}^{N} c_i c_j L_{ij}^{\Phi} \tag{24}$$

The L_{ij}^{Φ} coefficients are expanded as a Redlich–Kister power series

$$L_{ij}^{\Phi} = \sum_{\nu=0}^{k} (c_i - c_j)^{\nu} \cdot {}^{\nu} L_{ij}^{\Phi} \tag{25}$$

where the interaction coefficients ${}^{\nu}L_{ij}^{\Phi}$ are expressed as polynomial of T, and are tabulated in a multicomponent alloy CALPHAD databases. Typically, this expansion is limited to a maximum of three terms per phase (e.g. ${}^{0}L_{ij}$, ${}^{1}L_{ij}$ and ${}^{2}L_{ij}$), and the polynomials in T are linear, e.g.

$${}^{\nu}L_{ij} = a + b \cdot T \tag{26}$$

3. Software

3.1. Solvers and their implementations

To solve the nonlinear set of equations in the KKS model, we use a standard Newton solver. A trial solution vector $\mathbf{c}$ is updated until convergence where each step is given by

$$\mathbf{c}^{k+1} = \mathbf{c}^k - J^{-1}\mathbf{r}(\mathbf{c^k}) \tag{27}$$

where $\mathbf{r}$ is the residual of the KKS equations, and J its Jacobian. $\mathbf{c}$ denotes the vector of the N compositions we are solving for. The size N of the vectors and the Jacobian matrix in this formulation depend on which specific case we are dealing with: N=2 for a binary alloy (dilute or not), N=4 for a ternary alloy, and N=3 for a binary alloy with three phases (see Sections 2.1, 2.2, 2.3, and 2.4). The specific residuals are given by the residuals of the respective KKS equations for each system, and the Jacobian matrix is the matrix of the derivatives of these residuals,

$$J_{ij} = \frac{\partial \mathbf{r_i}}{\partial c_j} \tag{28}$$

where $c_j, j = 1, \ldots, N$ are the composition we are solving for in the various phases.

In C++, it is natural to design a solver using a base class that implements a common Newton solver for all the cases considered here, with one derived class for each specific set of equations. It is then the responsibility of the derived classes to compute the specific residuals vectors and Jacobian matrices.

In Thermo4PFM, this is done following the Curiously Recurring Template Patterns [25]. That is a derived class "D" derives from a templated base class using "D" itself as the template argument. In our case, we have for instance (for the ternary alloy)

```
class CALPHADConcSolverTernary
    : public NewtonSolver<4, CALPHADConcSolverTernary,
                          JacobianDataType>
{
...
    // setup model parameter values to be used by solver,
    // including composition "c0" and phase fraction "phi"
    // to solve for
    void setup(const double c0, const double c1,
        const double phi, ...);

    // evaluate RHS of the system of eqautions to solve for
    // specific to this solver
    void RHS(const double* const c, double* const fvec);

    // evaluate Jacobian of system of equations
    // specific to this solver
    void Jacobian(const double* const c,
                  JacobianDataType** const fjac);
}
```

Note that in addition to being templated on the derived class itself, the base class is also templated on the number of equations in the system (four for a ternary alloy) and the data type used for the Jacobian. The main reason for using this C++ pattern rather than traditional C++ inheritance is to avoid

virtual functions and enable offloading objects of that class to the GPU using OpenMP (see Section 3.3). The template parameter "JacobianDataType" is set to float since double precision is not required for the Jacobian inversion. This leads to a mixed precision Newton solver [26]. The Jacobian inversion is done using Cramer's rule [27].

Listing 1 illustrates how to instantiate a solver and solve the KKS equations within an outer loop. In practice, this outer loop would be over the discretization mesh points for instance. At every point, a solver object is constructed, then initialized based on the temperature at this point and the values of the phase variable and local composition. Specifically it means that quantities that are function of the temperature only such as the CALPHAD interaction coefficients ${}^0L_{AB}$, ${}^1L_{AB}$, ..., (see Eq. (24)) and the single species energies f_A and f_B are evaluated for the temperature at that point and ready to be used in the solver. Then the solve function is called to solve for the phase compositions c_L, c_S,...

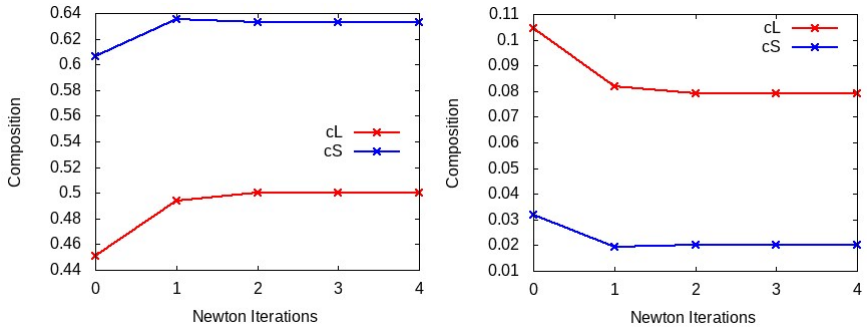
Listing 1: Code illustrating the use of Thermo4PFM solver within a loop for a ternary alloy

```
for (int i = 0; i < N; i++) // loop over mesh cells, ...
{
    // initialize x with initial guess
    x[4 * i]     = ...;
    x[4 * i + 1] = ...;
    x[4 * i + 2] = ...;
    x[4 * i + 3] = ...;

    // set local phase variable and composition
    double phi = ...;
    double c0  = ...;
    double c1  = ...;

    // declare solver object
    Thermo4PFM::CALPHADConcSolverTernary solver;

    // initialize solver with physical parameters,
    // including CALPHAD
    solver.setup(c0, c1, phi, RTinv, L_AB_L, L_AC_L, L_BC_L,
                 L_AB_S, L_AC_S, L_BC_S, L_ABC_L, L_ABC_S,
                 fA, fB, fC);

    // solve for x, tolerance 1e-8, max. 10 Newton steps
    // return number of iterations used
    nits = solver.ComputeConcentration(&xdev[4 * i], 1.e-8, 10);
}
```

3.2. Solver robustness and convergence

Figure 1 shows a typical case of convergence of the compositions variables when solving the KKS equations for a ternary alloy. This use case is for a surrogate ternary alloy for Inconel 718 (IN718), Ni–0.60Fe–0.035Nb (compositions in atom fraction), [28] using the CALPHAD data from Mathon *et al.* [29]. Note here that the Nb content of the ternary alloy was kept at 3.5 at. %, within the range of Nb content in IN718, and the Fe content adjusted such that the equilibrium solidification range for the surrogate ternary alloy was close to that of IN718 [28].

Robustness of the KKS solver is important in a PFM code since it is called at many spatial locations for many time steps, for a variety of compositions,



(a) Fe composition (b) Nb composition

Figure 1: Illustration of Newton solver convergence for the KKS equations of a ternary alloy Ni–0.60Fe–0.035Nb. Convergence of c_L and c_S is show for the Fe and Nb concentrations.

phase fraction, and temperature values. One issue that arises is that functions describing the energy as a function of composition are not necessarily defined outside the interval [0,1] of possible physical compositions. This is in particular the case for the ideal mixing function

$$ {}^{ideal}f_{mix}(c) = \frac{RT}{V_m}\left[c\ln(c) + (1-c)\ln(1-c)\right] \tag{29}$$

which is even numerically problematic at c =0 and c =1. While we typically do not expect a solution for the KKS equations to be outside of (0,1), the Newton solver may take intermediate steps with values outside of this interval. Thus to enhance the robustness of our solver, as also proposed by Schwen et al. [30], we extend ${}^{ideal}f_{mix}(c)$ beyond its definition interval. In practice, for c smaller than 10^{-5}, we extend the $c\ln(c)$ function with a quadratic function that match the values and its first two derivatives at $c = 10^{-5}$. Figure 2 illustrates this case where indeed a composition value smaller than 0 is reached during an intermediate step of a Newton iterative solve. This is a situation that is often observed for alloy compositions close to 0 or 1. This use case is again for Ni–0.60Fe–0.035Nb (compositions in atom fraction), where the Nb composition is very small.

3.3. Graphics Processing Units (GPU) implementation

Modern GPUs offer substantial computational throughput, with thousands of threads available for concurrent computations. However, making good use of that computational power is not always an easy task, as a high level of parallelism is required. The problem we are considering here, solving the KKS equations at each point of a discretization mesh, is actually very parallelizable: KKS solves are independent of each other and can be carried out concurrently. So ideally one would like to associate one KKS problem to each thread of a GPU.

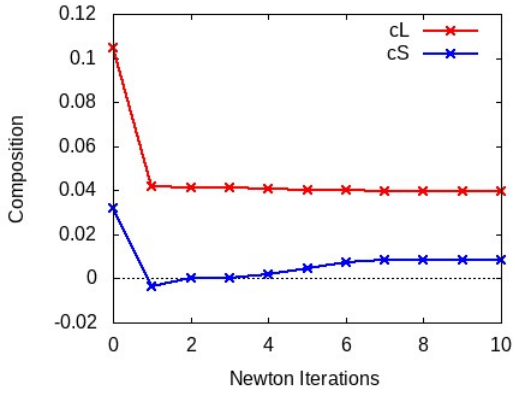


Figure 2: Newton convergence

To offload computational work to the GPU, we chose to use OpenMP. One of the main advantages of OpenMP is that it is portable, supported by modern C++ compilers, and does not require linking with any extra external package. OpenMP offloading capabilities have been used by various other applications on modern High Performance Computing platforms [31]. The task to be carried out here is however much more complicated than what is typically done with OpenMP offloading and shown in demonstrations. It requires each thread to solve a nonlinear system of several (between two and four here) equations, parameterized with a dozen or more (CALPHAD) parameters. In addition, this task is implemented using C++ derived and templated classes. While this turned out to be a successful vision, it was not obvious it would work when that development started.

While the final code looks deceivingly simple, it took significant trial and error and consultation with OpenMP compiler developers to reach that state. Listing 2 illustrates the use of a Thermo4PFM on GPU, where the main difference compared to Listing 1 is the addition/modification of the OpenMP pragmas at the beginning of the loop. It shows how the C++ solver object is instantiated directly on the device, within an OpenMP target loop. For OpenMP offloading to work, the initial C++ implementation had to be simplified to avoid any construct not supported by this programming model, specifically avoiding the use of the standard template library (STL) and calls to functions such as assert() or abort() in offloaded regions, as well as use trivial constructors for the classes to be fully offloaded. While some restrictions of OpenMP offload are well known, other are not and sometimes depend on specific compilers. The current status of debuggers for OpenMP offloading has also not reached the level of maturity one would like, making it even more difficult to track down these various issues. Once all these issues were resolved, the resulting additions to offload code inside Thermo4PFM essentially consist of pairs of pragmas (#pragma omp declare target ... #pragma omp end declare target) surrounding the functions that need to be executed on the GPU.

Listing 2: Code illustrating the use of a Thermo4PFM solver on GPU within a loop for a ternary alloy

```
#pragma omp target map(from : x[:4*N]) \
                    map(from : nits[:N])
#pragma omp teams distribute parallel for
for (int i = 0; i < N; i++) // loop over mesh cells, ...
{
    // initialize x with initial guess
    ...
    // set local phase variable and composition
    ...

    // declare solver object
    Thermo4PFM::CALPHADConcSolverTernary solver;

    // initialize solver
    solver.setup(c0, c1, phi, RTinv, L_AB_L, L_AC_L, L_BC_L,
                 L_AB_S, L_AC_S, L_BC_S, L_ABC_L, L_ABC_S,
                 fA, fB, fC);

    // solve for x
    nits = solver.ComputeConcentration(&xdev[4 * i], 1.e-8, 10);
}
```

GPU performance was evaluated on the Summit supercomputer at the Oak Ridge Leadership Computing Facility (OLCF), using a single node which is composed of two IBM Power9 processors and six NVIDIA Tesla V100 accelerators. A recent version of the clang++ compiler from LLVM was used (LLVM15) [32]. Performance was evaluated for the code described in Listing 2, that is for solving N times the KKS equations for a ternary alloy. The loop is representative of a loop one would have over mesh points in a finite difference or finite volume approach, with N mesh points. To establish a CPU baseline, timings measurements were carried out on the CPU using OpenMP for a various number of CPU threads (1, 4, and 8) with one thread/core. Scaling with the number of threads and the loop dimension is almost perfect on the CPU, and is shown in Figure 3. Performance in that figure is measured by the number of KKS problems solved in 1 ms wall-clock time. For the GPU performance evaluation, a single GPU was used. As expected, the problem size needs to be large enough to fully take advantage of the accelerator as shown in Figure 3. The best performance is achieved for N=1 million, with a minor drop in performance beyond that.

The maximum speedup between one CPU core and one GPU is estimated to be 277 X by comparing the GPU performance with the single CPU thread result. Although one-to-one comparisons are one standard comparison for GPU speedup, taking into account of the ratio seven CPU cores/GPU per node on the machine is more representative of the GPU speedup to be expected on a per node basis. In that case, one can divide that number by seven to obtain a per-node speedup of 40 X. These are excellent speedup numbers, close to what one would expect by comparing the peak performances of the V100 NVIDIA GPU and the Power9 CPU. Thermo4PFM benefits particularly from the link-time optimization (LTO) support from the LLVM clang compiler [33].

Three other compilers were tried to offload to the NVIDIA GPU and none performed as well as LLVM clang: the GNU compiler was about 10X slower and IBM xlC and NVHPC simply failed to build the code.

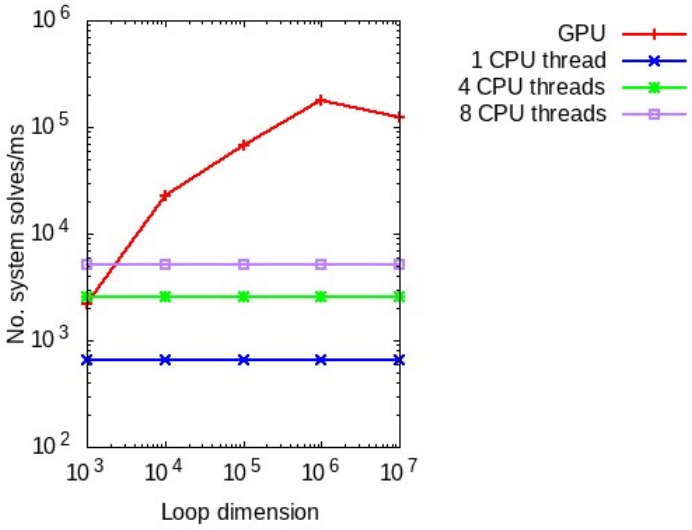


Figure 3: Thermo4PFM performance on a Summit node at OLCF using the clang++ compiler. GPU performance is shown for one NVIDIA V100 GPU. CPU threads run on the IBM Power9 cores.

3.4. Parsing CALPHAD data

Parsing data from a file is a common operation used by many software. In addition, various standard file formats exist and can be used to facilitate this operation. We chose to store CALPHAD parameters in JSON (JavaScript Object Notation) files. A standard format like JSON presents the advantage of not having to rewrite our own parser, but instead allowing us to use a parser provided by a third party library. We chose to use the widely available and open source *Property Tree* library from *boost* [34]. An example of a CALPHAD database in JSON format readable by Thermo4PFM is shown in Figure 4 and 5, for the energies of a single species (Au here) and the mixing of two species (Au and Ni) respectively. These contain the minimal information necessary for a PFM simulation using CALPHAD energies.

3.5. Code building, verification and testing

Thermo4PFM build system relies on CMake [35]. It is relatively simple since the list of dependencies is small: essentially just boost and OpenMP. In the "scripts" directory a few build scripts for some specific platforms are available as examples. Testing of the code is done using the tools provided by CMake [35] and Catch2 [36]. The tests include testing the proper functioning of the *boost* library, testing the offloading by OpenMP alone, and verifying the analytical form of function derivatives by comparison with finite differences. The test suite also include regression tests for the solvers, including convergence towards previously computed and known solutions, as well as tests verifying the thread safety of the various solvers when embedded in OpenMP loops.

```
{
"SpeciesA" :
{
   "name" : "Au",
   "PhaseL":
   {
      "Tc" : [298.15, 933.51, 1337.58, 1735.8, 3000.0],
      "a"  : [5613.147, -81023.261, 326614.987, 413.343],
      "b"  : [97.446385, 1012.21732, -2025.7579, 155.893158],
      "c"  : [-22.75455, -155.6947, 263.2523, -30.9616],
      "d2" : [-0.00385924, 0.08756015, -0.11821685, 0.0],
      "d3" : [3.79625e-7, -1.1518713e-05, 8.923845e-06, 0.0],
      "dm1": [-25097.0, 10637210.0, -67999850.0, 0.0]
   },
   "PhaseA":
   {
      "type" : "fcc",
      "Tc" : [298.15, 933.51, 1337.58, 1735.8, 3000.0],
      "a"  : [-6938.853, -93575.261, 314062.987, -12138.657],
      "b"  : [106.830495, 1021.60143, -2016.37379, 165.277268],
      "c"  : [-22.75455, -155.6947, 263.2523, -30.9616],
      "d2" : [-0.00385924, 0.08756015, -0.11821685, 0.0],
      "d3" : [3.79625e-7, -1.1518713e-05, 8.923845e-06, 0.0],
      "dm1": [-25097.0, 10637210.0, -67999850.0, 0.0]
   }
}
}
```


Figure 4: CALPHAD parameterization of the Gibbs energies of Au in liquid and solid (fcc) phases as functions of temperature T, with H_{Au}^{SER} =6016.6 and S_{Au}^{SER} =47. Notations refer to Eq. (19). Data is given for four temperature intervals, with the interval bounds given by the five entries for "Tc" (note that "Tc" has five entries while the other variables have four). Units: temperatures are in K and energies in J/mol.


```
{
"LmixPhaseL":
{
   "L0" : [9500.0, -5.429],
   "L1" : [1614.0,  0.0]
}
}
```


Figure 5: CALPHAD values of the Redlich–Kister parameters that define the excess Gibbs energy in Eq.(25) as functions of temperature T for the Au-Ni alloy in liquid phase. Each row corresponds to an interaction coefficient ${}^{\nu}L$ for which the values of a and b are given as in Eq. (26). Units: J/mol.

3.6. Code structure

Beside a main source code directory "src", and the "scripts" directory with example build scripts, Thermo4PFM includes a "tests" directory for the unit and regression tests. In addition, the "drivers" directory contains several small codes that use Thermo4PFM and perform various tasks complementary to the main code. There is for instance a driver to calculate the equilibrium composition of a binary alloy, given a CALPHAD database and a temperature. There are also drivers designed to evaluate performance on a GPU and used to measure the performance shown in Section 3.3. Another directory "thermodynamic_data" includes a few CALPHAD databases in the JSON format expected by Thermo4PFM parser. Finally, Doxygen code documentation can be generated with an input file provided in the "docs" directory.

3.7. Application example

Thermo4PFM is currently used by the PFM code AMPE [37]. AMPE phase-field models and their implementations are documented in [38] and [17]. AMPE features a finite volume discretization and an implicit and adaptive time-stepper using a backward difference formula (BDF) temporal discretization. This implicit scheme typically allows longer time steps than an explicit time integration scheme, and thus require fewer calls to a KKS solver. The implicit scheme uses a Jacobian-free Newton-Krylov (JFNK) approach [39]. In JFNK, the Generalized Minimal Residual (GMRES) iterative solver used for the system of equations associated with BDF requires the action of the Jacobian J only in the form of matrix–vector products $J\mathbf{v}$, which may be approximated by finite difference as

$$J\mathbf{v} \approx [\mathbf{F}(\mathbf{u} + \epsilon\mathbf{v}) - \mathbf{F}(\mathbf{u})] \tag{30}$$

when solving for $F(\mathbf{u}) = \mathbf{0}$. As such, JFNK requires consistency between subsequent evaluations of the right-hand side of the PFM time-evolution equations when finite differences are being evaluated. To ensure consistency and avoid random errors, we compute an accurate solution for the KKS equations.

We illustrate here the use of Thermo4PFM in AMPE with a directional solidification simulation for the ternary alloy Ni–0.60Fe–0.035Nb. We use a 2D domain of 3.5 μm $\times$ 2.5 μm discretized with a mesh of 700 $\times$ 500 (350,000) cells. We start with the leftmost 25% of the domain being solid, with a solid-liquid interface position perturbed with a small random fluctuation, and a uniform composition at the alloy nominal composition throughout the domain. We use a frozen temperature approximation with a cooling rate of 10^4 K/s, and a uniform temperature gradient of 1 K/μm in the x-direction. We observe the instability of the interface developing over the first $3 \cdot 10^{-5}$ seconds in a moving frame that matches the pulling velocity, moving at $10^4 \mu$m/s. With that combination of cooling rate, temperature gradient, and moving frame velocity, the temperature profile within the simulation domain remains constant over time.

Figure 6 shows the time evolution of the phase variable ϕ, as well as the Ni composition profile for the last snapshot and the temperature profile. That

simulation took a total of 1427 time-steps, including a total of 2776 Newton-Krylov iterations (about 2 per time-step) and 4422 linear iterations. For that run, a total of about $2.2 \cdot 10^9$ Newton steps were used to solve the KKS equations using Thermo4PFM, that is an average of 4.4 Newton steps per cell and per time-step. The run took a total of 577 seconds wall-clock time using 32 MPI tasks on an Intel Xeon CPU node. Of that, about 27% of the time is spent in the KKS solver.

4. Concluding remarks

In this paper, we describe a library that will facilitate the integration and usage of CALPHAD data into PFM simulations, specifically using the KKS model. Thermo4PFM is meant to be used inside of a PFM simulation code to calculate pointwise thermodynamic quantities. The design of the library takes use of some advantages of C++ such as templating and class derivations, without sacrificing performance. We demonstrated up to 277 X GPU speedsups on NVIDIA GPUs compared to an IBM Power9 core using OpenMP and the LLVM compiler. The currently supported models include binary and ternary alloys, as well as binary alloys with three phases. A demonstration case was presented where Thermo4PFM was used in the AMPE PFM simulation code for a directional solidification simulation of a Ni-Fe-Nb alloy. Possible future extensions of Thermo4PFM could include adding support for general multi-component alloys (beyond three species) as well as support for sublattice CALPHAD models.

From an implementation point of view, an extension to multi-component system is in principle not technically difficult, although maybe tedious. But difficulties in converging to the right solution should be expected, depending on the specific physical system of interest, and in particular when no good initial guess is available and with an increase in number of species. Note that a different approach was recently proposed to deal with high-dimensional and prohibitively large data sets[40]. On the other hand, an extension to general sublattice models, an important category of models, is much less straightforward. While new approaches are being discussed in the literature (see *e.g.* [41, 42]), we do not have a design ready for an implementation in Thermo4PFM at this time.

Finally, in addition to the specific target of solving equations used in phase-field models, this paper also demonstrates how one can offload non-trivial C++ classes to a GPU accelerator using OpenMP and achieve excellent performance. This implementation approach is of general interest beyond the PFM community. The OpenMP-based GPU-offloading approach taken here is meant to performance portable – being able to run with reasonable performance across various computing hardware and compilers. However, due the current lack of maturity OpenMP for GPUs, the portability of Thermo4PFM across non-LLVM compilers and non-NVIDIA GPUs remains to be demonstrated. As OpenMP GPU offload capabilities mature, we expect that the excellent GPU performance seen in this work will be replicatable across a variety of compiler/hardware combinations.

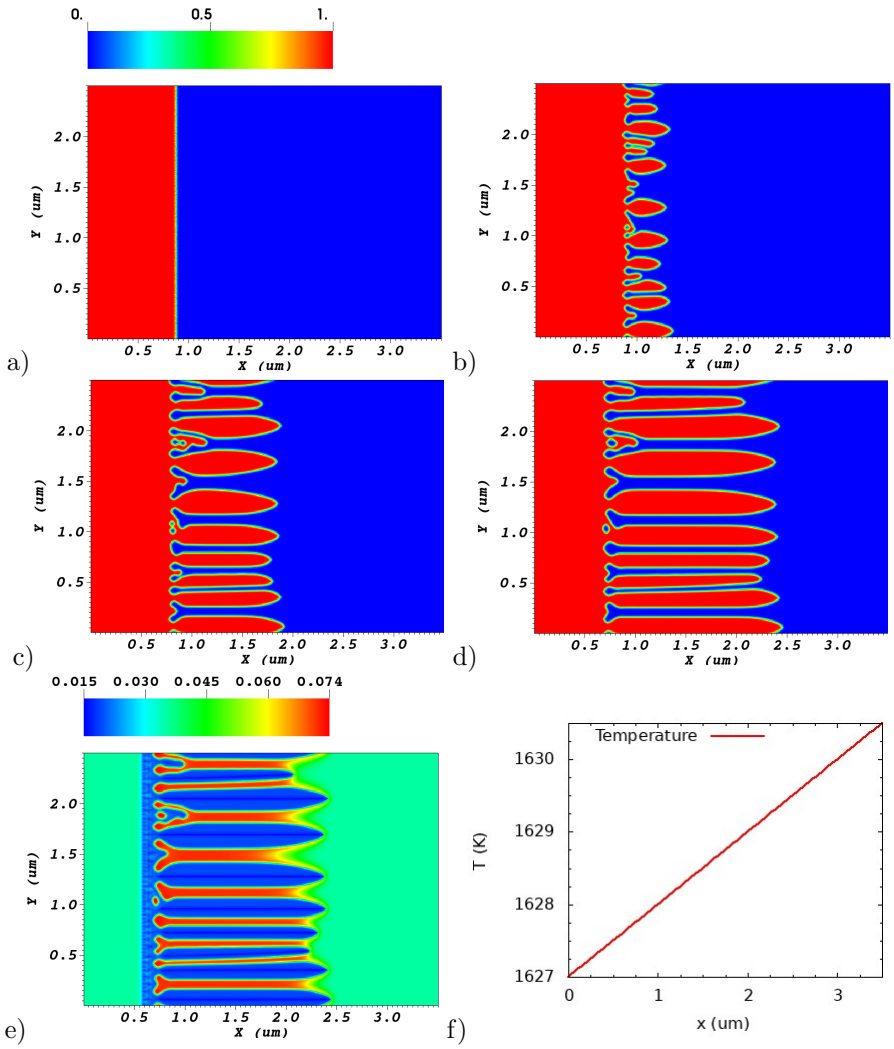


Figure 6: Directional solidification of Ni–0.60Fe–0.035Nb in 2D using AMPE. a) Initial conditions for phase variable ϕ, b) ϕ at $t = 10^{-5}$ s, c) ϕ at $t = 2 \cdot 10^{-5}$ s, d) ϕ at $t = 3 \cdot 10^{-5}$ s, e) Nb composition at $t = 3 \cdot 10^{-5}$ s, f) temperature profile in x-direction.

Acknowledgements

We would like to thank Johannes Doefert and Joseph Huber for their help with the LLVM compiler infrastructure. 2D visualizations in Figure 6 were generated using the VisIt software [43]. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This research was partially supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the DOE Office of Science and the National Nuclear Security Administration (NNSA). This research was also partially supported by the U.S. Department of Energy, Office of Energy Efficiency and Renewable Energy, Vehicle Technologies Office Propulsion Materials Program and the Advanced Manufacturing Office High-Performance Computing for Manufacturing (HPC4MFG) program. Work by A.P. was performed under the auspices of the U.S Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52- 07NA27344, and supported by the Critical Materials Institute, an Energy Innovation Hub funded by the US Department of Energy, Office of Energy Efficiency and Renewable Energy, Advanced Materials and Manufacturing Technologies Office.

Appendix A. KKS dilute alloy model

We consider here the dilute KKS model proposed in [1]. We write the energy densities for each phase as ideal solutions [22]

$$\begin{aligned} f_L(c_L,T) &= \frac{RT}{V}\left[c_L\ln(c_L)+(1-c_L)\ln(1-c_L)\right], \\ f_S(c_S,T) &= \frac{RT}{V}\left[c_S\ln(c_S)+(1-c_S)\ln(1-c_S)+c_S f_A+(1-c_S)f_B\right] \end{aligned} \tag{A.1}$$

where f_A and f_B are sometimes interpreted as the free energies of the pure components A and B (*i.e.* for c=1 and c=0). In that case, the driving force is given by

$$g_{thermo} = h'_p(\phi)\left[f_L(c_L,T)-f_S(c_S,T)-\mu(c_L-c_S)\right]. \tag{A.2}$$

Using

$$\mu_L = \frac{\partial f_L}{\partial c_L} = \frac{RT}{V}\left(ln(c_L)-ln(1-c_L)\right) \tag{A.3}$$

and

$$\mu_S = \frac{\partial f_S}{\partial c_S} = \frac{RT}{V}\left(ln(c_S)-ln(1-c_S)+f_A-f_B\right), \tag{A.4}$$

and rewriting Eq. (A.2) as

$$g_{thermo} = h'_p(\phi)\left[f_L(c_L,T)-f_S(c_S,T)-\mu_L c_L+\mu_S c_S\right] \tag{A.5}$$

we obtain

$$g_{thermo} = h'_p(\phi)\frac{RT}{V}\left[\ln(\frac{(1-c_L)}{(1-c_S)}) - f_B\right]. \tag{A.6}$$

Because $g_{thermo} = 0$ for $c_L = c_L^e$ and $c_S = c_S^e$, one obtains

$$f_B = \ln(1 - c_L^e) - \ln(1. - c_S^e) \tag{A.7}$$

where $c_L^e = (T - T_m)/m_L$ and $c_S^e = c_L^e \cdot k$ are the liquidus and solidus compositions at temperature T, respectively.

Thus the PFM driving force is given by

$$g_{thermo} = h'_p(\phi)\frac{RT}{V}\left[\ln(\frac{(1-c_L)(1-c_S^e)}{(1-c_S)(1-c_L^e)})\right] \tag{A.8}$$

which is Eq. (35) in [1].

In addition, because $\mu_L = \mu_S$ when the two phases are at equilibrium, from Eqs. (A.3) and (A.4) we obtain

$$f_A = -\ln(c_S^e) + \ln(c_L^e) = \ln(\frac{1}{k}). \tag{A.9}$$

References

[1] S. G. Kim, W. T. Kim, T. Suzuki, Phase-field model for binary alloys, Physical Review E 60 (6) (1999) 7186–7197. doi:10.1103/PhysRevE.60.7186.

[2] J. Tiaden, B. Nestler, H. Diepers, I. Steinbach, The multiphase-field model with an integrated concept for modelling solute diffusion, Physica D: Nonlinear Phenomena 115 (1) (1998) 73–86. doi:https://doi.org/10.1016/S0167-2789(97)00226-1.

[3] J. Zhu, Z. Liu, V. Vaithyanathan, L. Chen, Linking phase-field model to CALPHAD: application to precipitate shape evolution in ni-base alloys, Scripta Materialia 46 (5) (2002) 401–406. doi:https://doi.org/10.1016/S1359-6462(02)00013-1.

[4] I. Steinbach, B. Böttger, J. Eiken, N. Warnken, S. G. Fries, CALPHAD and phase-field modeling: A successful liaison, J. Phs. Eqil. and Diff. 28 (2007) 101–106. doi:10.1007/s11669-006-9009-2.

[5] N. Moelans, B. Blanpain, P. Wollants, An introduction to phase-field modeling of microstructure evolution, Calphad 32 (2) (2008) 268–294. doi:https://doi.org/10.1016/j.calphad.2007.11.003.

[6] A. Perron, J. D. Roehling, P. E. A. Turchi, J.-L. Fattebert, J. T. McKeown, Matching time and spatial scales of rapid solidification: dynamic tem experiments coupled to calphad-informed phase-field simulations, Modelling and Simulation in Materials Science and Engineering 26 (1) (2017) 014002. doi:10.1088/1361-651X/aa9a5b.
URL https://dx.doi.org/10.1088/1361-651X/aa9a5b

[7] J. Berry, A. Perron, J.-L. Fattebert, J. D. Roehling, B. Vrancken, T. T. Roehling, D. L. Rosas, J. A. Turner, S. A. Khairallah, J. T. McKeown, M. J. Matthews, Toward multiscale simulations of tailored microstructure formation in metal additive manufacturing, Materials Today 51 (2021) 65–86. doi:https://doi.org/10.1016/j.mattod.2021.09.024.
URL https://www.sciencedirect.com/science/article/pii/S1369702121003370

[8] J. Eiken, B. Böttger, I. Steinbach, Multiphase-field approach for multicomponent alloys with extrapolation scheme for numerical application, Phys. Rev. E 73 (2006) 066122. doi:10.1103/PhysRevE.73.066122.

[9] C. Yang, J. Wang, H. Xing, H. Huang, A parabolic approximation scheme for multi-phase-field simulation of non-isothermal solidification, Materials Today Communications 28 (2021) 102712. doi:https://doi.org/10.1016/j.mtcomm.2021.102712.

[10] A. Choudhury, M. Kellner, B. Nestler, A method for coupling the phase-field model based on a grand-potential formalism to thermodynamic databases, Current Opinion in Solid State and Materials Science 19 (5) (2015) 287–300. doi:https://doi.org/10.1016/j.cossms.2015.03.003.

[11] D. Noubary, K. Kellner, M. Hötzer, J., M. Seiz, H. J. Seifert, B. Nestler, Data workflow to incorporate thermodynamic energies from Calphad databases into grand-potential-based phase-field models, J Mater Sci 56 (2021) 11932–11952. doi:https://doi.org/10.1007/s10853-021-06033-7.

[12] K. Joshi, S. S. Quek, Y. Zeng, D. T. Wu, An efficient implementation for the solution of auxiliary composition fields in multicomponent phase field models, Computational Materials Science 197 (2021) 110608. doi:https://doi.org/10.1016/j.commatsci.2021.110608.

[13] Thermo-Calc Software.
URL https://thermocalc.com

[14] I. Steinbach, Phase-field models in materials science, Modelling and Simulation in Materials Science and Engineering 17 (7) (2009) 073001. doi:10.1088/0965-0393/17/7/073001.

[15] M. Ode, J. S. Lee, S. G. Kim, W. T. Kim, T. Suzuki, Phase-field model for solidification of ternary alloys, ISIJ International 40 (9) (2000) 870–876. doi:10.2355/isijinternational.40.870.

[16] S. G. Kim, A phase-field model with antitrapping current for multicomponent alloys with arbitrary thermodynamic properties, Acta Materialia 55 (13) (2007) 4391–4399. doi:https://doi.org/10.1016/j.actamat.2007.04.004.

[17] J.-L. Fattebert, M. Wickett, P. Turchi, Phase-field modeling of coring during solidification of Au–Ni alloy using quaternions and CALPHAD input, Acta Materialia 62 (2014) 89–104. doi:10.1016/j.actamat.2013.09.036.

[18] R. Folch, M. Plapp, Quantitative phase-field modeling of two-phase growth, Phys. Rev. E 72 (2005) 011602. doi:10.1103/PhysRevE.72.011602.

[19] A. Clarke, D. Tourret, Y. Song, S. Imhoff, P. Gibbs, J. Gibbs, K. Fezzaa, A. Karma, Microstructure selection in thin-sample directional solidification of an Al-Cu alloy: In situ X-ray imaging and phase-field simulations, Acta Materialia 129 (2017) 203–216. doi:https://doi.org/10.1016/j.actamat.2017.02.047.

[20] S. Sakane, T. Takaki, T. Aoki, Parallel-gpu-accelerated adaptive mesh refinement for three-dimensional phase-field simulation of dendritic growth during solidification of binary alloy, Mater Theory 6 (3). doi:https://doi.org/10.1186/s41313-021-00033-5.

[21] S. Elahi, R. Tavakoli, A. Boukellal, T. Isensee, I. Romero, D. Tourret, Multiscale simulation of powder-bed fusion processing of metallic alloys, Computational Materials Science 209 (2022) 111383. doi:https://doi.org/10.1016/j.commatsci.2022.111383.

[22] A. A. Wheeler, W. J. Boettinger, G. B. McFadden, Phase-field model for isothermal phase transitions in binary alloys, Phys. Rev. A 45 (1992) 7424–7439. doi:10.1103/PhysRevA.45.7424.

[23] H. Lukas, S. G. Fries, B. Sundman, Computational Thermodynamics: The Calphad Method, Cambridge University Press, 2007. doi:10.1017/CBO9780511804137.

[24] A. Dinsdale, SGTE data for pure elements, Calphad 15 (4) (1991) 317–425. doi:https://doi.org/10.1016/0364-5916(91)90030-N.

[25] J. O. Coplien, Curiously recurring template patterns, C++ Report.

[26] C. T. Kelley, Newton's method in mixed precision, SIAM Review 64 (1) (2022) 191–211. doi:10.1137/20M1342902.

[27] T. Shores, Applied Linear Algebra and Matrix Analysis, Undergraduate Texts in Mathematics, Springer International Publishing, 2018.

[28] B. Radhakrishnan, S. B. Gorti, J. A. Turner, R. Acharya, J. A. Sharon, A. Staroselsky, T. El-Wardany, Phase field simulations of microstructure evolution in IN718 using a surrogate Ni–Fe–Nb alloy during laser powder bed fusion, Metals 9 (1). doi:10.3390/met9010014.

[29] M. Mathon, D. Connetable, B. Sundman, J. Lacaze, Calphad-type assessment of the Fe–Nb–Ni ternary system, Calphad 33 (1) (2009) 136 – 161. doi:10.1016/j.calphad.2008.10.005.

[30] D. Schwen, L. Aagesen, J. Peterson, M. Tonks, Rapid multiphase-field model development using a modular free energy based approach with automatic differentiation in MOOSE/MARMOT, Computational Materials Science 132 (2017) 36–45. doi:https://doi.org/10.1016/j.commatsci.2017.02.017.

[31] S. Bak, C. Bertoni, S. Boehm, R. Budiardja, B. M. Chapman, J. Doerfert, M. Eisenbach, H. Finkel, O. Hernandez, J. Huber, S. Iwasaki, V. Kale, P. R. Kent, J. Kwack, M. Lin, P. Luszczek, Y. Luo, B. Pham, S. Pophale, K. Ravikumar, V. Sarkar, T. Scogland, S. Tian, P. Yeung, OpenMP application experiences: Porting to accelerated nodes, Parallel Computing 109 (2022) 102856. doi:https://doi.org/10.1016/j.parco.2021.102856.

[32] The LLVM compiler infrastructure.
URL https://llvm.org

[33] S. Tian, J. Huber, J. Tramm, B. Chapman, J. Doerfert, Just-in-time compilation and link-time optimization for OpenMP target offloading, in: M. Klemm, B. R. de Supinski, J. Klinkenberg, B. Neth (Eds.), OpenMP in a Modern World: From Multi-device Support to Meta Programming, Springer International Publishing, Cham, 2022, pp. 145–158.

[34] boost c++ libraries, (accessed: 2020-02-15).
URL https://www.boost.org

[35] CMake, (accessed: 2022-11-14).
URL https://cmake.org

[36] Catch2, (accessed: 2022-11-14).
URL https://github.com/catchorg/Catch2

[37] AMPE.
URL https://github.com/LLNL/AMPE

[38] M. Dorr, J.-L. Fattebert, M. Wickett, J. Belak, P. Turchi, A numerical algorithm for the solution of a phase-field model of polycrystalline materials, Journal of Computational Physics 229 (3) (2010) 626–641. doi:https://doi.org/10.1016/j.jcp.2009.09.041.

[39] D. Knoll, D. Keyes, Jacobian-free Newton–Krylov methods: a survey of approaches and applications, Journal of Computational Physics 193 (2) (2004) 357–397. doi:https://doi.org/10.1016/j.jcp.2003.08.010.

[40] Y. Coutinho, N. Vervliet, L. D. Lathauwer, N. Moelans, Combining thermodynamics with tensor completion techniques to enable multicomponent microstructure prediction, npj Comput Mater 6 (2). doi:https://doi.org/10.1038/s41524-019-0268-y.

[41] D. Schwen, C. Jiang, L. Aagesen, A sublattice phase-field model for direct calphad database coupling, Computational Materials Science 195 (2021) 110466. doi:https://doi.org/10.1016/j.commatsci.2021.110466.

[42] Y. Ji, H. W. Abernathy, L.-Q. Chen, Thermodynamic models of multicomponent nonstoichiometric solution phases using internal process order parameters, Acta Materialia 223 (2022) 117462. doi:https://doi.org/10.1016/j.actamat.2021.117462.

[43] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, M. Miller, C. Harrison, G. H. Weber, H. Krishnan, T. Fogal, A. Sanderson, C. Garth, E. W. Bethel, D. Camp, O. Rübel, M. Durant, J. M. Favre, P. Navrátil, Visit: An end-user tool for visualizing and analyzing very large data, in: High Performance Visualization–Enabling Extreme-Scale Scientific Insight, 2012, pp. 357–372. doi:10.1201/b12985.