New Timestepper for Explicit Thermonuclear C++ Code

Mike Guidry
Department of Physics and Astronomy, University of Tennessee
Knoxville, TN 37996-1200, USA

These notes outline the implementation and testing of a new timestepper for the C++ explicit algebraic thermonuclear code that parallels the implementation of a new timestepper for neutrino transport developed by Eirik Endeve.

1 Introduction

We are presently developing a C++ implementation of the explicit algebraic approximation for solving large sets of extremely stiff differential equations that was introduced in Refs. [1, 2, 3, 4, 5, 6]. These methods are based on three complementary approaches to solving stiff sets of equations explicitly:

- the *explicit asymptotic algorithm* (Asy),
- the *quasi-steady-state algorithm* (QSS), and
- the *partial equilibrium algorithm* (PE),

that stabilize explicit integration through algebraic constaints and permit much larger timesteps than normally are possible with explicit methods (often by many orders of magnitude). Using a very simple timestepper we demonstrated that these methods can be much faster than standard implicit approaches to large sets of stiff equations, with the advantage of the explicit methods growing with network size and with deployment on limited capability devices like GPUs. The new C++ code is intended to be a general-purpose community code that can be used in a variety astrophysical applications where large sets of stiff ordinary differential equations must be integrated that are coupled dynamically to a description of the fluid dynamics (typically 3D radiation hydrodynamics) and neutrino transport.

Although the code is presently being developed for serial application, once that is debugged we intend to port the code to GPUs since we have already demonstrated that these new algorithms work extremely efficiently on GPUs [5, 6]. Because the largest present petascale and coming exascale machines derive much of their speed from GPU accelerators, we expect that this codebase will be particularly useful in large-scale astrophysical simulations on those machines.

The present code is aimed at astrophysical applications but the algorithms themselves should be applicable with relatively small modification for a variety of scientific and technological applications where large stiff sets of equations must be solved. For example, we are already deploying them for the solution of neutrino transport equations under supernova or neutron star merger conditions [7].

One goal in the new code is the implementation of a more sophisticated timestepper than used in our original proof of principle work, which should lead to even more computational

efficiency for these methods. The new timestepper is based on two basic ideas: (1) an iteration that ensures that particle number is conserved to acceptable tolerance (implemented through checking the sum of mass fractions at each step),[1] and (2) a prediction of the next trial timestep based on estimating local errors (by comparing results for two half-steps to one full step) and choosing a timestep that is projected to satisfy a specified local tolerance. The new timestepper is implemented primarily in the function `doIntegrationStep()` of the class `Integrate` and the functions in `Integrate` that it calls,

- `computeTimeStep_EA()`,
- `computeTimeStep_EA()`, and
- `computeNextTimeStep()`.

Presently this code is in lines $\sim 3439 - 3638$ of `explicitMatrix.cpp`. The method has been shown to work very well for the asymptotic approximation (Asy) for alpha-networks. There is presently a bug in the asymptotic + partial equilibrium approximation (Asy+PE) that I am tracking down.

2 Results for Asymptotic Approximation

Results for several alpha-networks have been computed under constant temperature and density conditions typical of a zone in a hydrodynamical simulation of a Type Ia supernova explosion.

2.1 A 3-alpha Network

Results with the new timestepper for evolution of the mass fractions X for a 3-alpha network (^{4}He, ^{12}C, and ^{16}O) omitting partition functions [2] are displayed in Fig. 1, and the corresponding timestepping is illustrated in Fig. 2. Also illustrated in Fig. 2 are the fractions of isotopes that are asymptotic and the fraction of reaction groups that satisfy the partial equilibrium (PE) conditions, as estimated in the asymptotic calculation. We see that there is excellent agreement between the mass fractions found in the original Java Asy calculation[3] and in the present C++ calculation using the new timestepper in Fig. 1, but the timestepping for the Asy calculation in Fig. 2 is much better than in the original Java calculation (which used a very crude timestepping algorithm since the goal was only to establish proof of principle). Indeed, the original Java Asy calculation required 29,638 integration steps while the C++ Asy calculation required only 3,922 integration

[1] This is a crucial aspect of the timestepper for these methods because the asymptotic and quas-steady-state approximations do not conserve particle number intrinsically . Thus particle number conservation at an acceptable level must be imposed through control of the timestep if those methods are employed.

[2] The partition function algorithm in the C++ code has not yet been fully checked, so we omit partition functions from the calculation for comparison purposes. This has no impact on evaluation of the algorithms since the effect of including partition functions is only to modify effective rates for reactions, in most cases by relatively small amounts.

[3] In general all Java Asy results used in this document have been shown to be equivalent to a forward Euler calculation with sufficiently small timesteps that it is stable, so we may take the Java Asy calculations as benchmarks for comparisons.

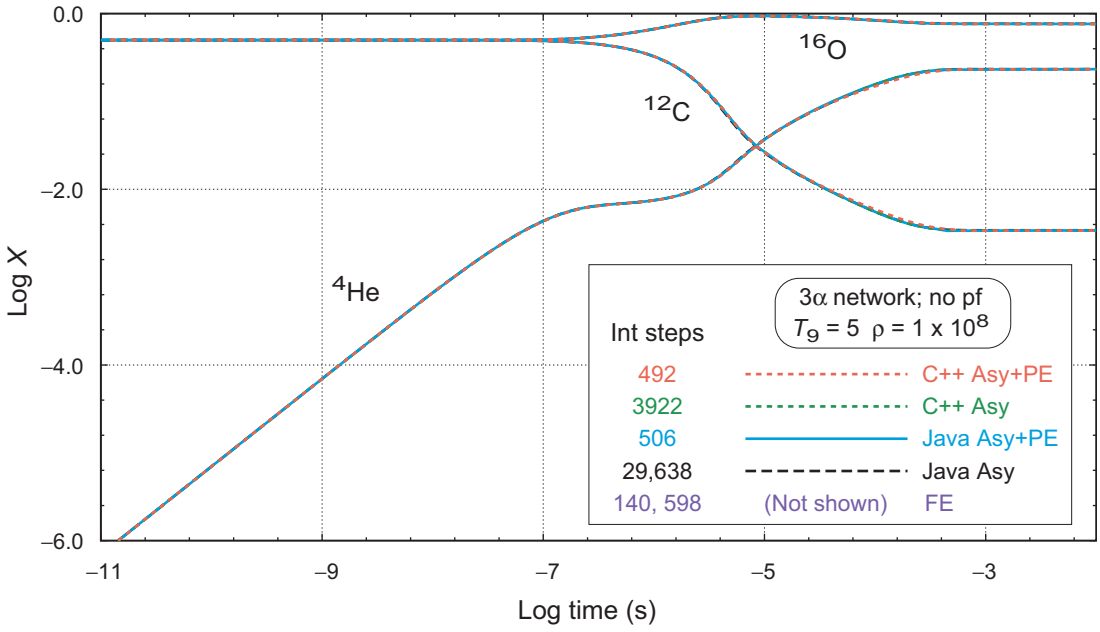

Figure 1: Mass fractions for 3α network with $T_9 = 5$ and $\rho = 1 \times 10^8$ g cm^{-3}. Curves and total integration steps are indicated for forward Euler (FE), the original Java code with Asy approximation, the original Java code with Asy+PE approximation, the C++ code using Asy approximation with the new timestepper, and the C++ code using Asy+PE approximation with the new timestepper. For comparison purposes all calculations ignored the role of partition functions (no pf).

steps. Both are a huge improvement over an explicit forward Euler calculation (FE; not shown), which required 140,598 integration steps.

Also shown in Figs. 1 and 2 are the results of asymptotic plus partial equilibrium calculations (Asy+PE) for both the original Java code and for the C++ code with the old timestepper. in Fig. 1 we see that the computed mass fractions are almost identical to those calculated for the Asy method (and forward Euler with timesteps small enough to be stable), but in Fig. 2 we see that the timestepping is much more efficient for the Asy+PE approximation relative to the Asy calculation. Furthermore, the C++ Asy+PE timestepping with the new timestepper (492 integration steps) is better than the original Java Asy+PE result with crude timestepping (506 timesteps), and both are large improvements over the asymptotic approximation (and of course over forward Euler).

Three important timescales for the explicit approximation are also illustrated in Fig. 2 as dotted black curves. All are proportional to the inverse of the fastest reaction rate in the system at a given time, $R_{\rm max}(t)$. We often say loosely that the maximum stable explicit forward Euler timestep is given by $\sim 1/R_{\rm max}$, which is indicated by the lowest black dotted curve in Fig. 2. Indeed, there is a suggestion in Fig. 2 that the Java Asy timestepping (solid blue curve labeled Java Asy) is limited by this curve until some species begin to satisfy the asymptotic condition

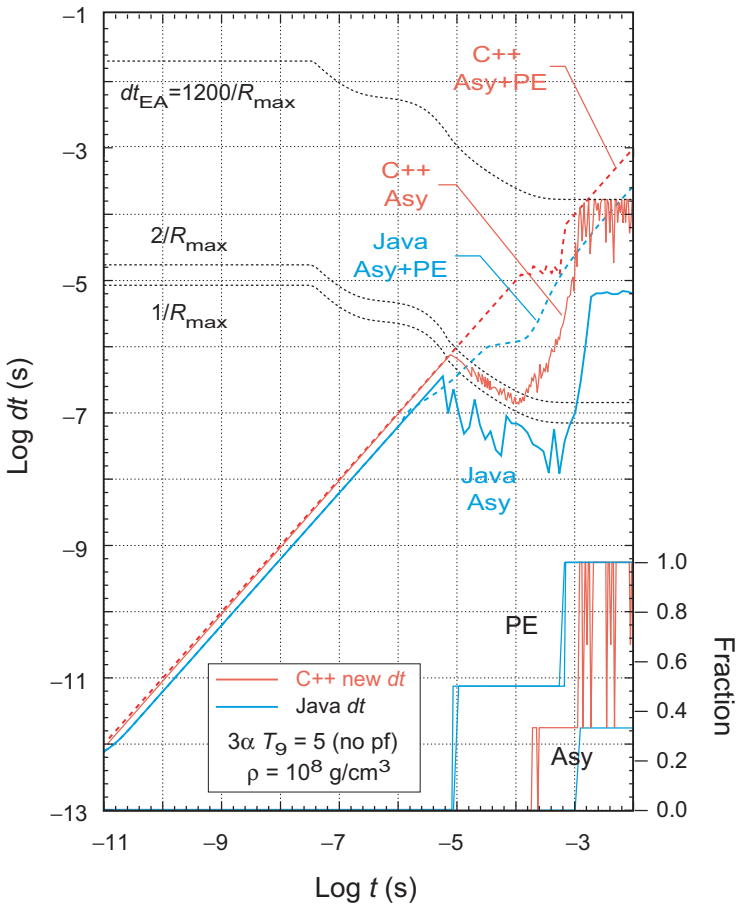

Figure 2: Timestepping and fraction asymptotic and equilibrated for a 3α network with $T_9 = 5$ and $\rho = 1 \times 10^8$ g cm^{-3}. Curves and total integration steps are indicated for the original Java code with Asy approximation, the original Java code with Asy+PE approximation, the C++ code with Asy approximation, and the C++ code with Asy+PE approximation, with the C++ calculations using the new timestepper. For comparison purposes all calculations ignored the role of partition functions (no pf).

around $\log t = -3$ (the Java+Asy approximation uses forward Euler for those isotopes that do not satisfy the asymptotic condition, so for Java+Asy the calculation is effectively forward Euler where no isotopes satisfy the asymptotic condition).

However, more careful considerations indicate that the maximum stable forward Euler timestep is given by $2/R_{\rm max}$ for many cases. Indeed, one sees clearly that dt for the C++ Asy calculation with the new timestepper in Fig. 2 (solid red curve) is limited by the $2/R_{\rm max}$ curve shown in Fig. 4 before $\log t \sim -4$, where no isotopes are asymptotic and the calculation is effectively forward Euler. Furthermore, one sees that the control of the timestepping is much better for the new timestepper than for the old crude timestepper used in the Java calculation, with much

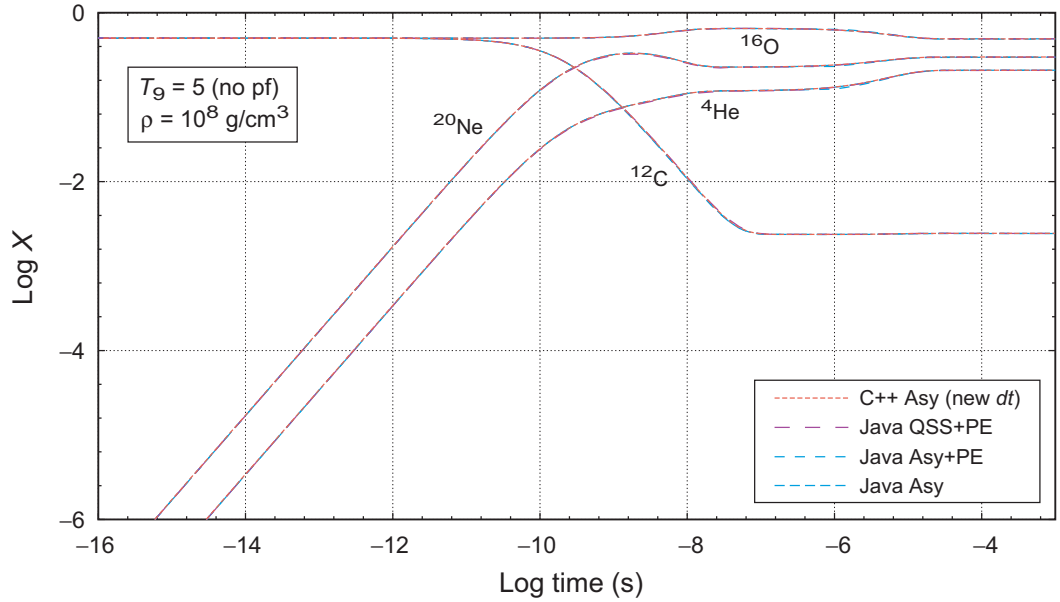

Figure 3: Mass fractions for 4α network with $T_9 = 5$ and $\rho = 1 \times 10^8$ g cm^{-3}. Curves and total integration steps are indicated for the original Java code with Asy approximation, the original Java code with Asy+PE approximation, the C++ code with Asy approximation and new timestepper, and the Java code with QSS+PE approximation. For comparison purposes all calculations ignored the role of partition functions (no pf).

smaller fluctutions at the $2/R_{\rm max}$ limit. Finally, we see from Fig. 2 that the curve $dt_{\rm EA} = A/R_{\rm max}$, which is a parameter defined in the new timestepper where generally one can choose the constant $A \gg 1$, plays the role of limiting dt when many isotopes satisfy the asymptotic condition in the new C++ timestepper: see how the upper dotted black curve in Fig. 2 limits the solid red C++ Asy curve at late times when many isotopes are aymptotic). In this example we have chosen $A = 1200$. [4]

2.2 A 4-alpha Network

Results with the new timestepper for evolution of the mass fractions X for a 4α network (^{4}He, ^{12}C, ^{16}O, and ^{20}Ne) omitting partition functions are displayed in Fig. 3, and the corresponding timestepping is illustrated in Fig. 4. Also illustrated in Fig. 4 are the fraction of isotopes that are asymptotic and the fraction of reaction groups that satisfy the partial equilibrium (PE) conditions, as estimated in the asymptotic calculation. There is excellent agreement between the mass fractions found in the original Java Asy calculation and in the present C++ calculation using the new timestepper in Fig. 3, but the timestepping for the Asy calculation in Fig. 4 is much better

[4]This has the consequence that if $A = 1$ is chosen in the new timestepper, the asymptotic condition is generally never satisfied and the entire calculation is forced to use the forward Euler method.

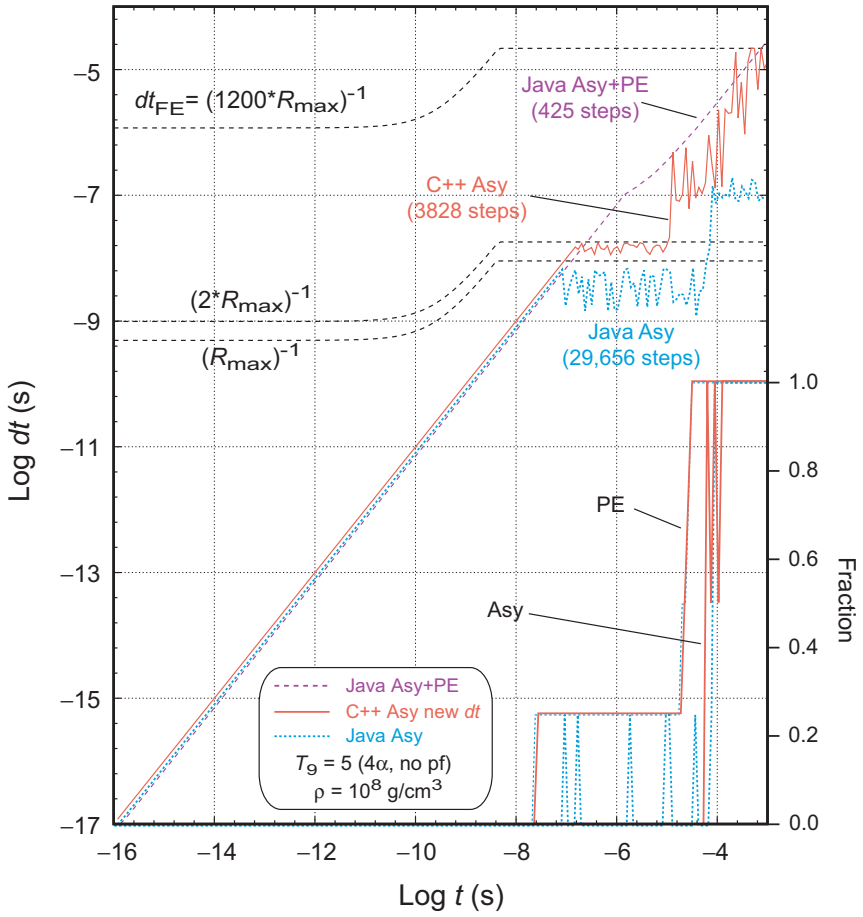

Figure 4: Timestepping and fraction asymptotic and equilibrated for 4α network with $T_9 = 5$ and $\rho = 1 \times 10^8$ g cm^{-3}. Curves and total integration steps are indicated for the original Java code with Asy approximation, the original Java code with Asy+PE approximation, and the C++ code with Asy approximation and new timestepper. For comparison purposes all calculations ignored the role of partition functions (no pf).

than in the original Java calculation. The original Java Asy calculation required 29,656 integration steps while the C++ Asy calculation required only 3,828 integration steps. Also shown in Figs. 3 and 4 are the results of asymptotic plus partial equilibrium calculations (Asy+PE) for the original Java code that we will address later, for which the required number of integration steps is reduced to 425. We will address extending the C++ calculation to the Asy+PE approximation in later sections. In Fig. 3 we see that the computed mass fractions are almost identical to those calculated for the Asy method, but Fig. 4 shows that the timestepping is much more efficient for the Asy+PE approximation relative to the Asy calculation. As for the 3α example in Section 2.1, we see that the timescales indicated by the dotted black curves play a similar role:

1. $1/R_{\rm max}$ is the limiting forward Euler dt using the original Java Asy timestepper.

2. $2/R_{\rm max}$ is the limiting forward Euler dt using the new C++ timestepper.

3. $dt_{\rm EA} \equiv A/R_{\rm max}$ with $A = 1200$ defines the maximum dt for the new timestepper when the asymptotic condition is satisfied for all isotopes.

Indeed, we see an indication that the timestepping before any isotopes become asymptotic is limited by $1/R_{\rm max}$ for the Java Asy calculation and $2/R_{\rm max}$ for the C++ Asy calculation, while the C++ Asy calculation reaches timesteps at late time that begin to approach those of the Java Asy+PE calculation because the asymptotic timestepping is not limited by the value chosen for $dt_{\rm EA}$.

2.3 Full Alpha Network

The preceding tests with 3α and 4α networks allow transparent understanding of the explicit algebraic methods because there are only a few moving parts. Let's now consider some more realistic calculations, beginning with a full 16-isotope alpha network under temperature and density conditions characteristic of a zone in a Type Ia supernova simulation.

2.3.1 Alpha Network for $T_9 = 5$ and $\rho = 1 \times 10^8$ g cm^{-3}

Figure 5 shows the mass fractions for a full alpha network (16 isotopes) as a function of time for constant $T_9 = 5$ and constant $\rho = 1 \times 10^8$ g cm^{-3}. Calculations are shown for the Asy approximation in the original Java code and in the C++ code with new timestepper. Figure 6 shows the differential energy release and the total energy release corresponding to the calculation of Fig. 5. Because the plot is log–log and dE/dt or E could be negative for particular regions, the absolute values of these quantities have been plotted. Positive and negative regions of dE/dt are indicated by $\pm$ signs; in this particular calculation both dE/dt and E remain positive over the whole region. For reference, we have also indicated the fastest reaction in the network in each time region, with its associated Q-value. Initially the fastest reaction under these temperature and density conditions is the burning of carbon through $^{12}\mathrm{C} + {}^{12}\mathrm{C} \rightarrow {}^{4}\mathrm{He} + {}^{20}\mathrm{Ne}$, which is exothermic with energy release $Q = 4.621$ MeV. For a short period beginning at $t = 5.5 \times 10^{-10}$ the fastest reaction becomes $^{4}\mathrm{He} + {}^{20}\mathrm{Ne} \rightarrow {}^{24}\mathrm{Mg}$, which is exothermic with $Q = +9.312$ MeV, but after $t = 7.9 \times 10^{-9}$ s the fastest reaction switches to the photodisintegration of the neon produced from the initial carbon burning through the reaction $^{20}\mathrm{Ne} \rightarrow {}^{4}\mathrm{He} + {}^{16}\mathrm{O}$, which is endothermic with $Q = -4.734$ MeV.

Results for the mass fractions in Fig. 5 are in excellent agreement except for some small discrepances at very small values for ^{36}Ar, ^{40}Ca, ^{44}Ti, and ^{48}Cr. These small descrepancies in such small mass fractions are negligible in hydrodynamical coupling since they contribute essentially nothing to energy. However, I think it likely that they have their origin in these

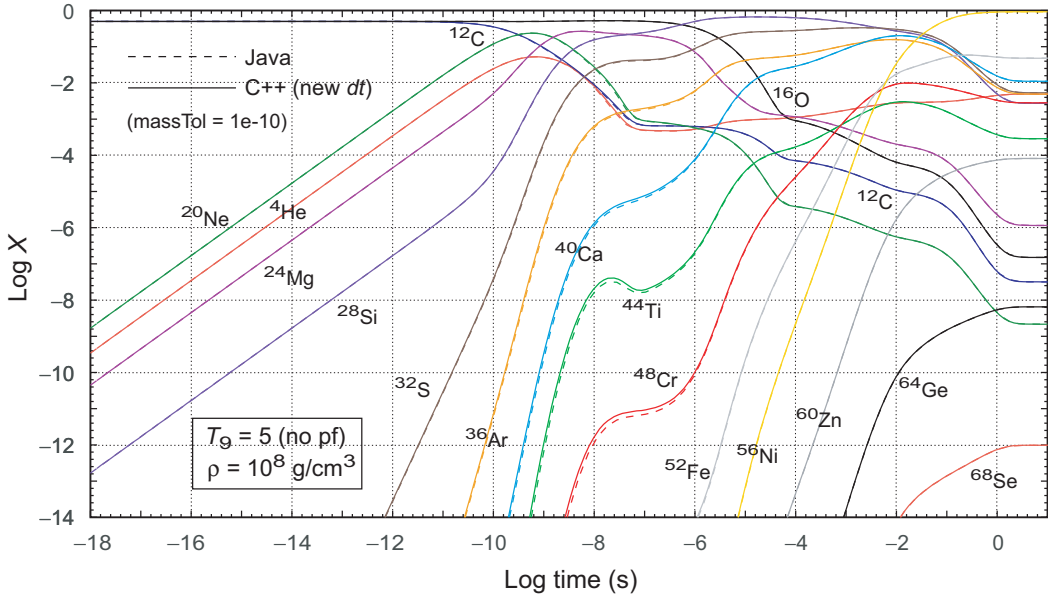

Figure 5: Mass fractions for a full α network with $T_9 = 5$ and $\rho = 1 \times 10^8$ g cm^{-3}. Curves are shown for the original Java code with Asy approximation, and the C++ code with Asy approximation and new timestepper. For comparison purposes all calculations ignored the role of partition functions (no pf).

isotopes being part of the silicon burning quasi-equilibrium

$$\begin{aligned} \alpha + {}^{28}\mathrm{Si} &\rightleftharpoons {}^{32}\mathrm{S} + \gamma \\ \alpha + {}^{32}\mathrm{S} &\rightleftharpoons {}^{36}\mathrm{Ar} + \gamma \\ &\vdots \\ \alpha + {}^{52}\mathrm{Fe} &\rightleftharpoons {}^{56}\mathrm{Ni} + \gamma \end{aligned} \tag{1}$$

that is burning silicon to the iron-group nuclei (^{52}Fe, ^{56}Ni, ^{60}Zn, ...) under these conditions. It could be that adding partial equilibrium to the calculation, or looking more carefully at this quasi-equilibrium in the asymptotic approximation, could improve even these small discrepancies.

The differential energy release dE/dt and the integrated energy release E shown in Fig. 6 display excellent agreement between the C++ Asy calculation and the Java Asy benchmark calculation. The fluctuations at the very last times in the differential energy release for the Java benchmark calculation, and small discrepancy between C++ and Java calculations at those times, are caused by the particular choices of parameter controlling the level of particle number conservation. These fluctuations occur at a level 12 orders of magnitude below the peak energy release rate, so they are irrelevant for the coupling to fluid dynamics. However, a tighter value chosen for the parameter controlling particle number conservation in the Java calculations would both erase the fluctuation and bring the C++ and Java benchmark calculations into approximate

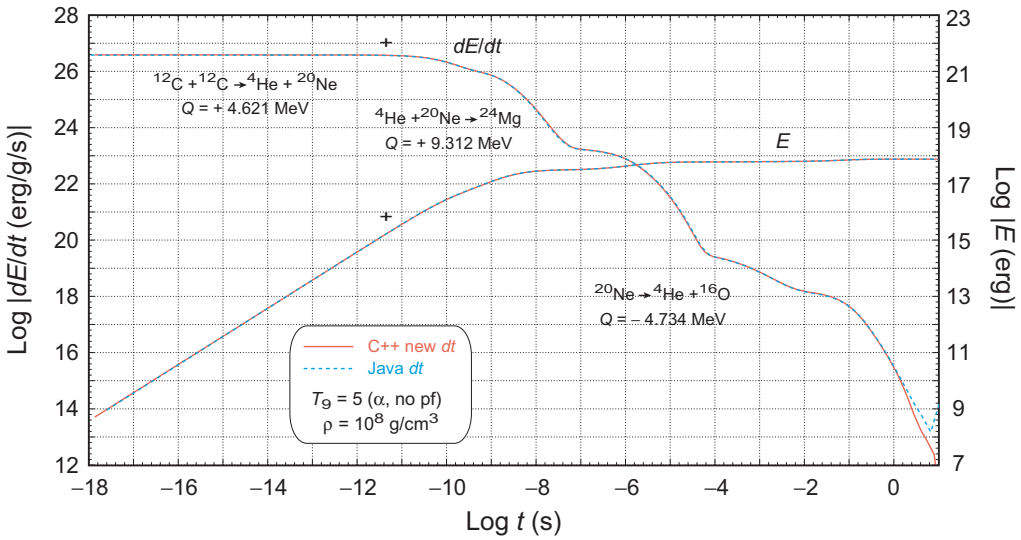

Figure 6: Differential energy release dE/dt and total energy release E for the calculation given in Fig. 5. Curves compare the original Java code with Asy approximation, and the C++ code with Asy approximation and new timestepper. Because the plot is log–log and dE/dt or E could be negative for particular regions, the absolute values of these quantities have been plotted, and positive and negative regions of dE/dt and E are indicated by signs (for this particular example both dE/dt and E are positive over the whole plot). For reference we have also indicated the fastest reaction in each time region with its associated Q-value.

agreement at very late times.

The timestepping used to obtain the results shown in Fig. 5 is shown in Fig. 7 for the Java Asy results (dotted blue curve) and for the new C++ timestepper (solid red curve). Also shown in Fig. 7 is the fraction of isotopes that are asymptotic (purple curves labeled Asy) and fraction of reaction groups that would satisfy the equilibrium condition if we invoked the partial equilibrium approximation (orange curves labeled PE), with dotted curves coming from the Java Asy calculation and solid curves coming from the C++ calculation with new timestepper.

We see that the Java timestepping and the new C++ timestepper are similar in behavior, but generally the new timestepper gives larger values of dt by almost an order of magnitude. We also see that neither the old Java timestepper nor the new C++ timestepper can increase dt substantially once the stability limit for forward Euler imposed by the $1/R_{\max}$ or $2/R_{\max}$ curves is encountered until some isotopes begin to satisfy the asymptotic condition, which doesn't occur until around $\log t \sim -1$. We also note that the estimate of this calculation is that reaction groups begin to satisfy the partial equilibrium condition at much earlier times ($\sim \log t = -6$) than this, which suggests that timestepping could be greatly improved by adding the partial equilibrium approximation to the asymptotic approximation, as we shall discuss in later sections.

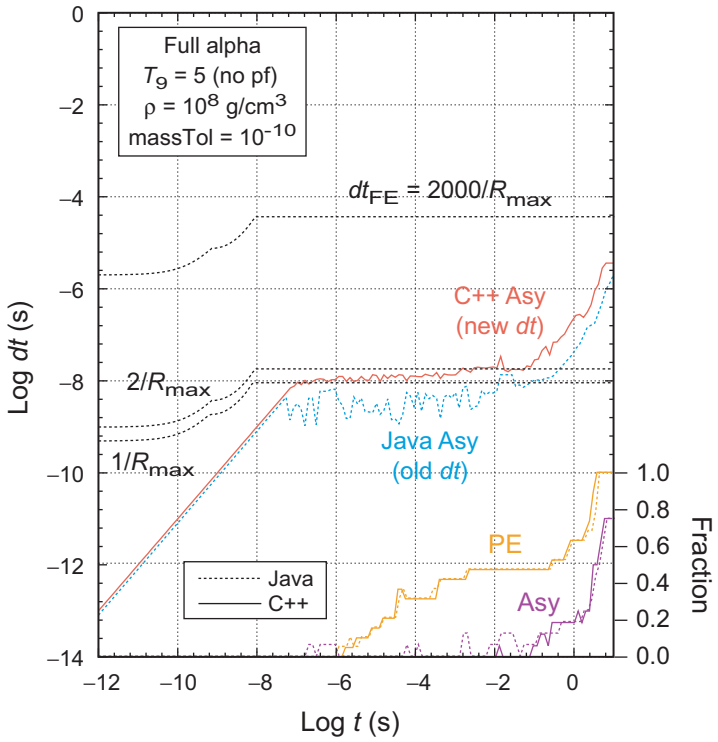

Figure 7: Timestepping for a full α network with $T_9 = 5$ and $\rho = 1 \times 10^8$ g cm^{-3}. Curves and total integration steps are indicated for the original Java code with Asy approximation, and the C++ code with Asy approximation and new timestepper. For comparison purposes all calculations ignored the role of partition functions (no pf).

2.3.2 Alpha Network for $T_9 = 7$ and $\rho = 1 \times 10^8$ g cm^{-3}

Figure 8 Shows the mass fractions for a full alpha network (16 isotopes) as a function of time for constant $T_9 = 7$ and $\rho = 1 \times 10^8$ g cm^{-3}. Calculations are shown for the Asy approximation in the original Java code and with the C++ code with new timestepper. Figure 9 shows the differential energy release dE/dt and the integrated energy release E as a function of time for the simulation in Fig. 8. Because dE/dt or E could be negative and the plot is logarithmic, the log of the absolute value for E and dE/dt have been plotted. In this calculation the integrated energy E is positive over the whole range but the sign of dE/dt switches several times, with positive and negative signs indicating the sign of dE/dt in various regions. For reference, we have also indicated the fastest reaction in the network in each time region, with its associated Q-value. Initially the fastest reaction under these temperature and density conditions ia the burning of carbon through $^{12}\mathrm{C} + {}^{12}\mathrm{C} \rightarrow {}^{4}\mathrm{He} + {}^{20}\mathrm{Ne}$, which is exothermic with energy release $Q = 4.621$ MeV, but after $t = 2.86 \times 10^{-11}$ s the fastest reaction switches to the photodisintegration of the neon produced from the initial carbon burning through $^{20}\mathrm{Ne} \rightarrow {}^{4}\mathrm{He} + {}^{16}\mathrm{O}$, which is endothermic with

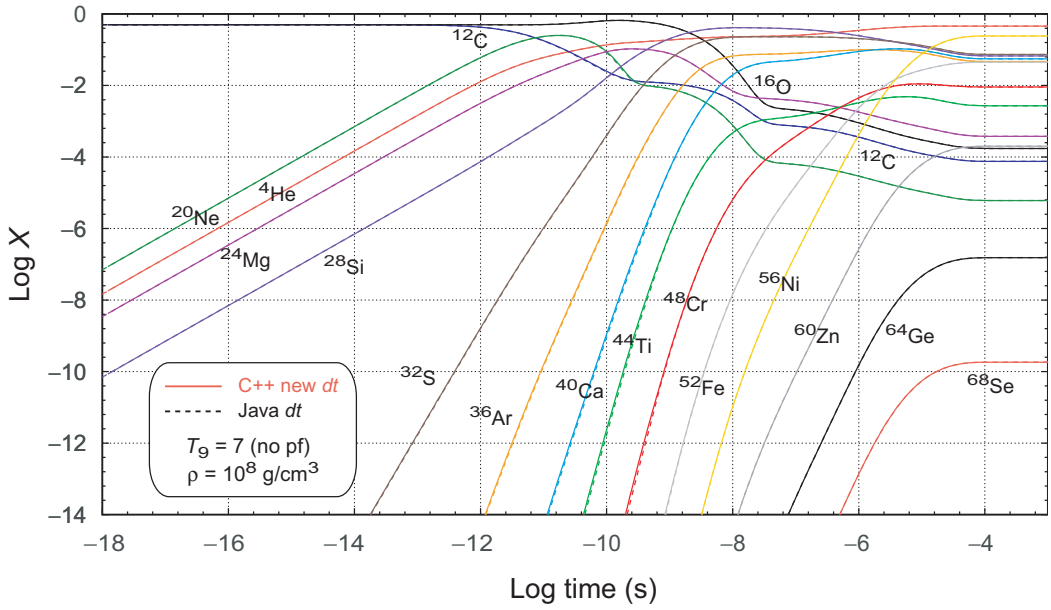

Figure 8: Mass fractions for a full α network with $T_9 = 7$ and $\rho = 1 \times 10^8$ g cm^{-3}. Curves and total integration steps are indicated for the original Java code with Asy approximation, and the C++ code with Asy approximation and new timestepper. For comparison purposes all calculations ignored the role of partition functions (no pf).

$Q = -4.734$ MeV. [5]

Results are in excellent agreement except for some discrepances at width-of-line levels for very small values for ^{36}Ar, ^{40}Ca, ^{44}Ti, and ^{48}Cr. As noted in Section 2.3.1, these small descrepancies in such small mass fractions are negligible in hydrodynamical coupling and likely have their origin in these isotopes being part of the silicon burning quasi-equilibrium that is burning silicon to the iron-group nuclei (^{52}Fe, ^{56}Ni, ^{60}Zn, ...) under these conditions. Thus it could be that adding partial equilibrium to the calculation, or looking more carefully at this quasi-equilibrium in the asymptotic approximation could improve even these small discrepancies.

The timestepping used to obtain the results shown in Fig. 8 is shown in Fig. 10 for the Java Asy results (dotted blue curve) and for the new C++ timestepper (solid red curve). Also shown in Fig. 10 is the fraction of isotopes that are asymptotic (labeled Asy) and fraction of reaction groups that would satisfy the equilibrium condition if we invoked the partial equilibrium approximation (labeled PE), with dotted blue curves coming from the Java Asy calculation and solid red curves coming from the C++ calculation with new timestepper.

We see that the new C++ timestepper give larger values of dt by almost an order of magnitude than the old Java timestepper. We also see that neither the old Java timestepper nor the new C++

[5] Note that the photodistigration reaction $^{20}\mathrm{Ne} \rightarrow {}^4\mathrm{He} + {}^{16}\mathrm{O}$ is actually $\gamma + {}^{20}\mathrm{Ne} \rightarrow {}^4\mathrm{He} + {}^{16}\mathrm{O}$ but it is common to suppress the photon γ in the notation since photons are not conserved particles. Likewise, the radiative capture reaction $^4\mathrm{He} + {}^{16}\mathrm{O} \rightarrow {}^{20}\mathrm{Ne} + \gamma$ is commonly abbreviated to $^4\mathrm{He} + {}^{16}\mathrm{O} \rightarrow {}^{20}\mathrm{Ne}$.

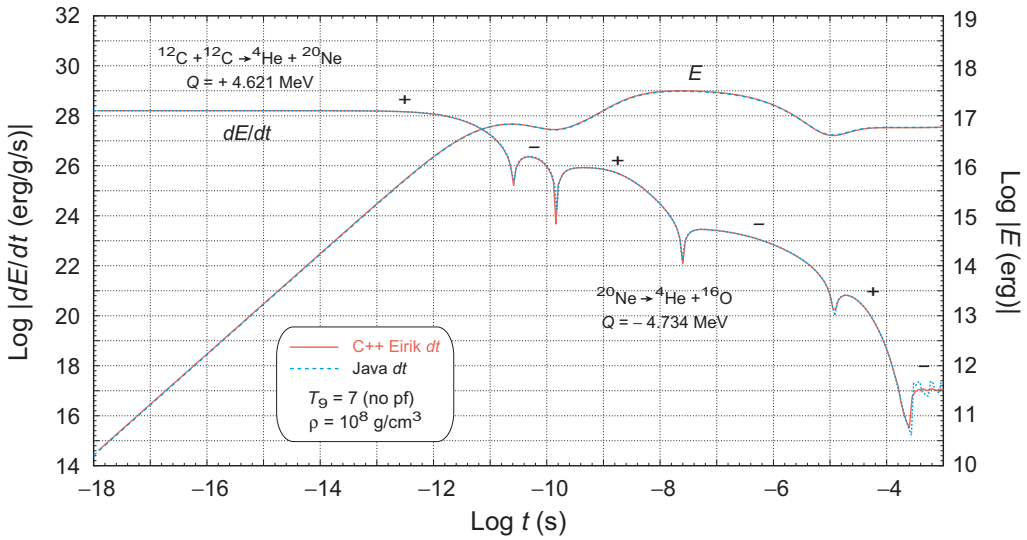

Figure 9: Differential energy release dE/dt and total energy release E for the calculation given in Fig. 8. Curves compare the original Java code with Asy approximation, and the C++ code with Asy approximation and new timestepper. For reference we have also indicated the fastest reaction in each time region with its associated Q-value, and marked the sign of dE/dt and E in each region.

timestepper can increase dt substantially once the stability limit for forward Euler imposed by the $1/R_{\max}$ or $2/R_{\max}$ curves is encountered, until some isotopes begin to satisfy the asymptotic condition, which doesn't occur until around $\log t \sim -6$. We also note that the estimate of this calculation is that reaction groups begin to satisfy the partial equilibrium condition at much earlier times ($\sim \log t = -8$) than the onset of asymptotic behavior, which suggests that timestepping could be greatly improved by adding the partial equilibrium approximation to the asymptotic condition.

2.4 Asymptotic Plus Partial Equilibrium (Asy+PE) Approximation

Let us now consider the addition of the partial equilibrium approximation to the asymptotic approximation illustrated in preceding sections (Asy+PE approximation). The use of the partial equilibrium approximation has been discussed and its efficacy demonstrated in Refs. [1, 4]. There we have shown with a simple timestepper that the Asy+PE approximation is generally more powerful than either Asy or PE alone, and is essential to getting competitive explicit timestepping in the approach to complete equilibrium.

Figure 10 contains a curve for the Java Asy+PE approximation at $T_9 = 7$ and $\rho = 1 \times 10^8$ g cm^{-3}. There we see that the timestepping at late times is greatly improved by the Asy+PE approximation relative to the Asy approximation. This improvement was demonstrated using a simple timestepper employing global constraints. We may hope that the new timestepper with local

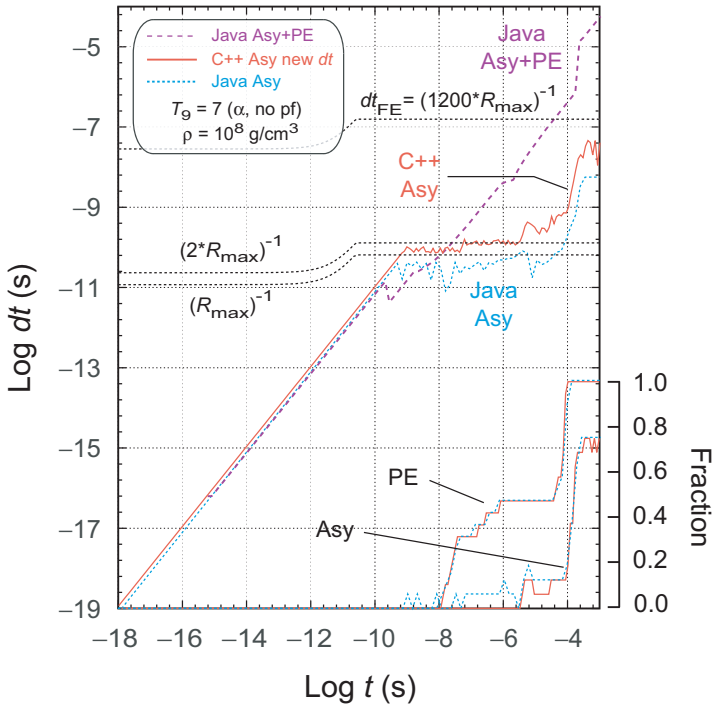

Figure 10: Timestepping for a full α network with $T_9 = 7$ and $\rho = 1 \times 10^8$ g cm^{-3}. Curves and total integration steps are indicated for the original Java code with Asy approximation, and for the C++ code with Asy approximation and new timestepper. Also shown for future reference is a calculation in Asy+PE approximation using the benchmark Java code. For comparison purposes all calculations ignored the role of partition functions (no pf).

constraints on dt could improve on the Java Asy+PE results. Alas, as we now discuss there appears to be a bug in the C++ Asy+PE implementation that we must locate and fix before we can proceed. Since I will show that the bug appears for both the old and new C++ timesteppers at about the same place (the time when the PE approximation is first implemented), the suggestion is that the bug is in the original C++ implementation of the Asy+PE approximation and not the timesteppers.

To see where things stand, let's look at Asy+PE with the new C++ timestepper (that was shown earlier to work very well for Asy approximation) for a full α network. Figure 11 compares Asy and Asy+PE approximations in a full α network for $T_9 = 7$ and $\rho = 1 \times 10^8$ g cm^{-3}. Shortly after onset of partial equilibrium (dashed vertical line) the Asy+PE mass fraction curves undergo large fluctuations (for clarity the strongly fluctuating region has been cut out for some curves), but all curves converge to their correct values (the dashed curves) as full equilibrium is reached. Tests with other full α networks yield similar results. Thus the C++ Asy+PE approximation with the new timestepper gives results that are clearly incorrect at the onset of partial equilibrium, but these curves converge again to the expected values at late time as full equilibrium is approached. This behavior suggests that the basic Asy+PE is correct (because the required values at full

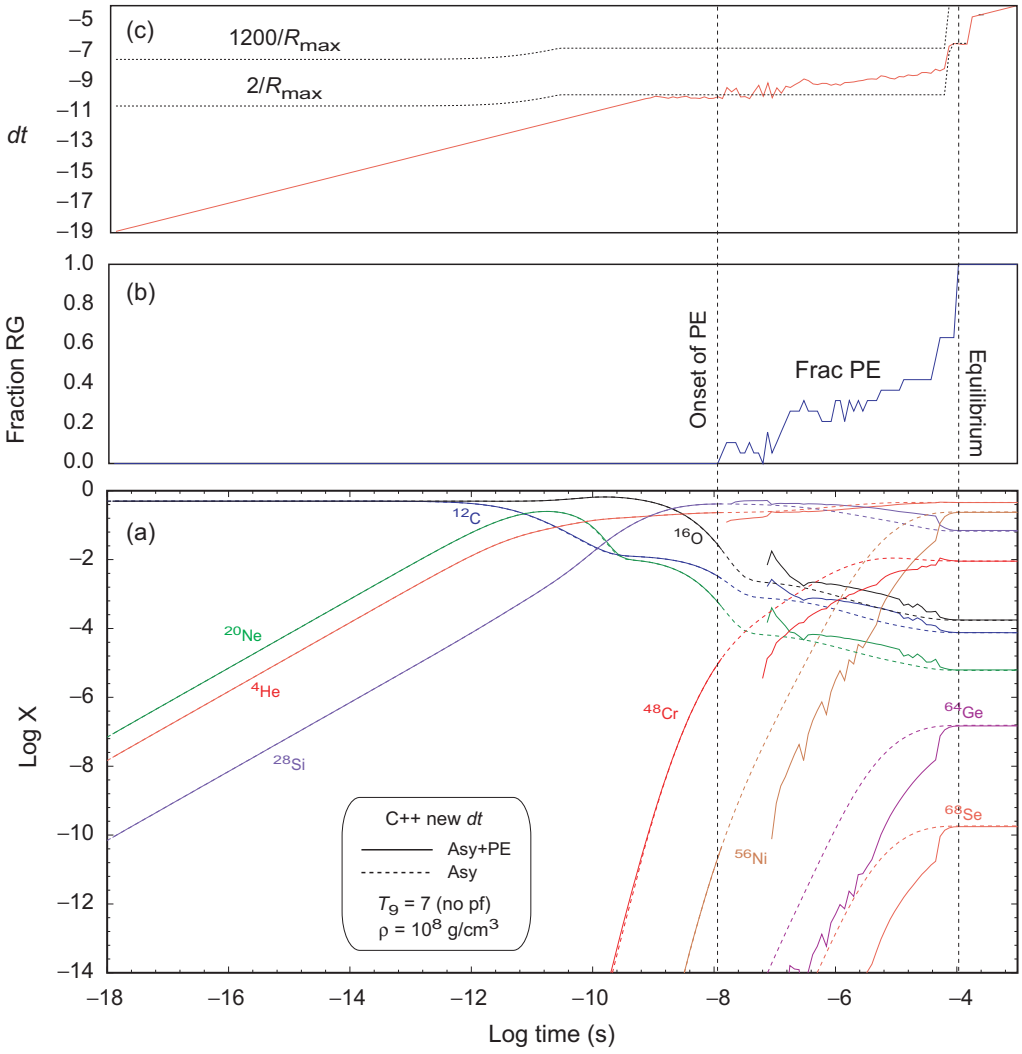

Figure 11: Asy+PE (solid curves) compared with Asy (dashed curves) for full α network with $T_9 = 7$ and $\rho = 1 \times 10^8$ g cm^{-3} and the new C++ timestepper. (a) Mass fractions; (b) fraction of reaction groups satisfying the PE condition; (c) timestepping. Only selected isotopes are shown to avoid clutter. Note that very shortly after onset of partial equilibrium the Asy+PE curves go berserk (to avoid confusing clutter the strongly fluctuating region has been cut out for some curves), but all curves eventually converge to their correct values (the dashed curves) as full equilibrium is reached.

equilibrium are obtained) but that something is wrong with either the PE approximation or the timestepping in the region where partial equilibrium first appears.

To understand this behavior and identify the error let us turn to calculations for the 4α system, since this is a large enough network to be realistic but small enough to make analysis easier. Figure 12 shows a comparison for a 4α network with $T_9 = 5$ and $\rho = 1 \times 10^8$ g cm^{-3} using both the old timestepper (the one in the Java benchmark code) as implemented in the C++ code, and the new C++ timestepper. Again we see glitches appearing in the mass fraction curves as partial equilibrium is first implement (vertical dashed curve), but all curves then converge to the correct final equilibrium value.

Figure 13 shows the region where the first reaction group equilibrates for the new C++ timestepper and we begin to see deviations from expected for the mass fractions, but blown up to show the behavior near the onset of partial equilibrium more clearly. We see that the initial deviation in the mass fraction for the C++ calculation occurs where a reaction group that was just put in equilibrium falls out of equilbrium (dashed vertical line).[6] Conversely, in the Java benchmark calculation the reaction group remains in equilbrium at this time. Furthermore, we see in the C++ calculation that immediately after the number of RG in equilibrium falls from one to zero it jumps to complete equilibration, whereas over this small time range the fraction equilbrated jumps from zero to $\frac{1}{4}$ and remains there.

Since the glitches occur almost exactly at the beginning of equilbration, and similar behavior is observed for both the old and new C++ timesteppers, it is suggested strongly that the problem is in the equilibration algorithm for the C++ code and not in the timestepper (especially since the new timestepper works very well for the pure Asy calculation). Furthermore, since the newest version of the C++ code with the new timestepper gives excellent results for the Asy approximation, this suggests that things like the rates and fluxes are being calculated correctly, and the error must be specifically in the part of the code calculating and implementing partial equilibrium, which greatly narrows the lines of code that need to be checked. I am presently comparing the C++ code with new timestepper and the original Java code in the partial equilibrium algorithm line by line, and using the debugger to check the values of variables in the C++ code, to determine where the error is.

[6] There are four reaction groups for this case so equilibration of each reaction group increases the fraction by 0.25.

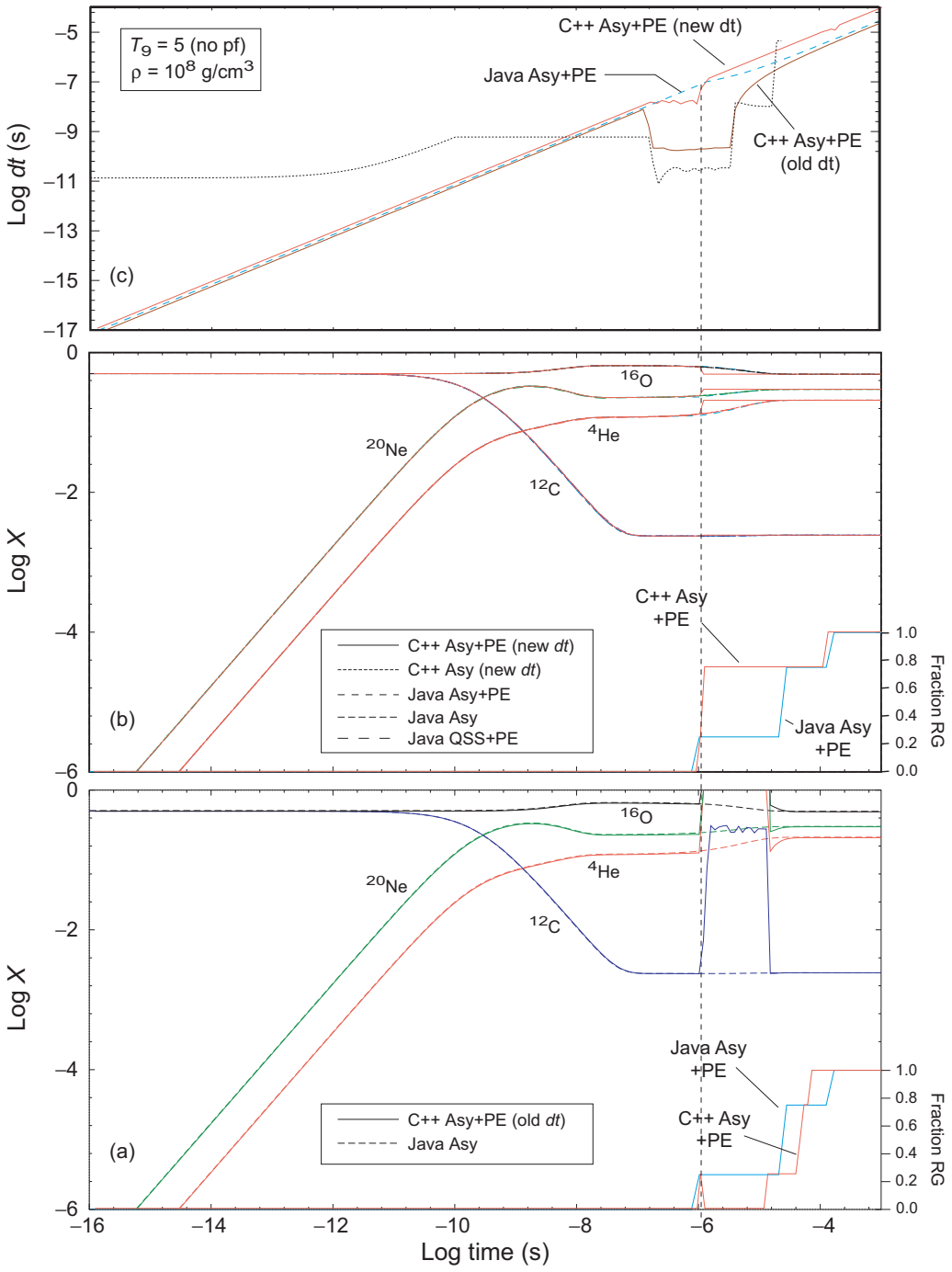

Figure 12: Mass fractions, timestepping, and Asy and PE fractions for a 4α network in Asy+PE approximation, for $T_9 = 5$ and $\rho = 1 \times 10^8$ g cm^{-3}. (a) Mass fractions and partial equilibration fractions for the old C++ timestepper; (b) mass fractions and partial equilibration fractions for the new C++ timestepper; (c) timestepping for the Java benchmark, for C++ with the old timestepper, and for C++ with the new timestepper, all in Asy+PE approximation.

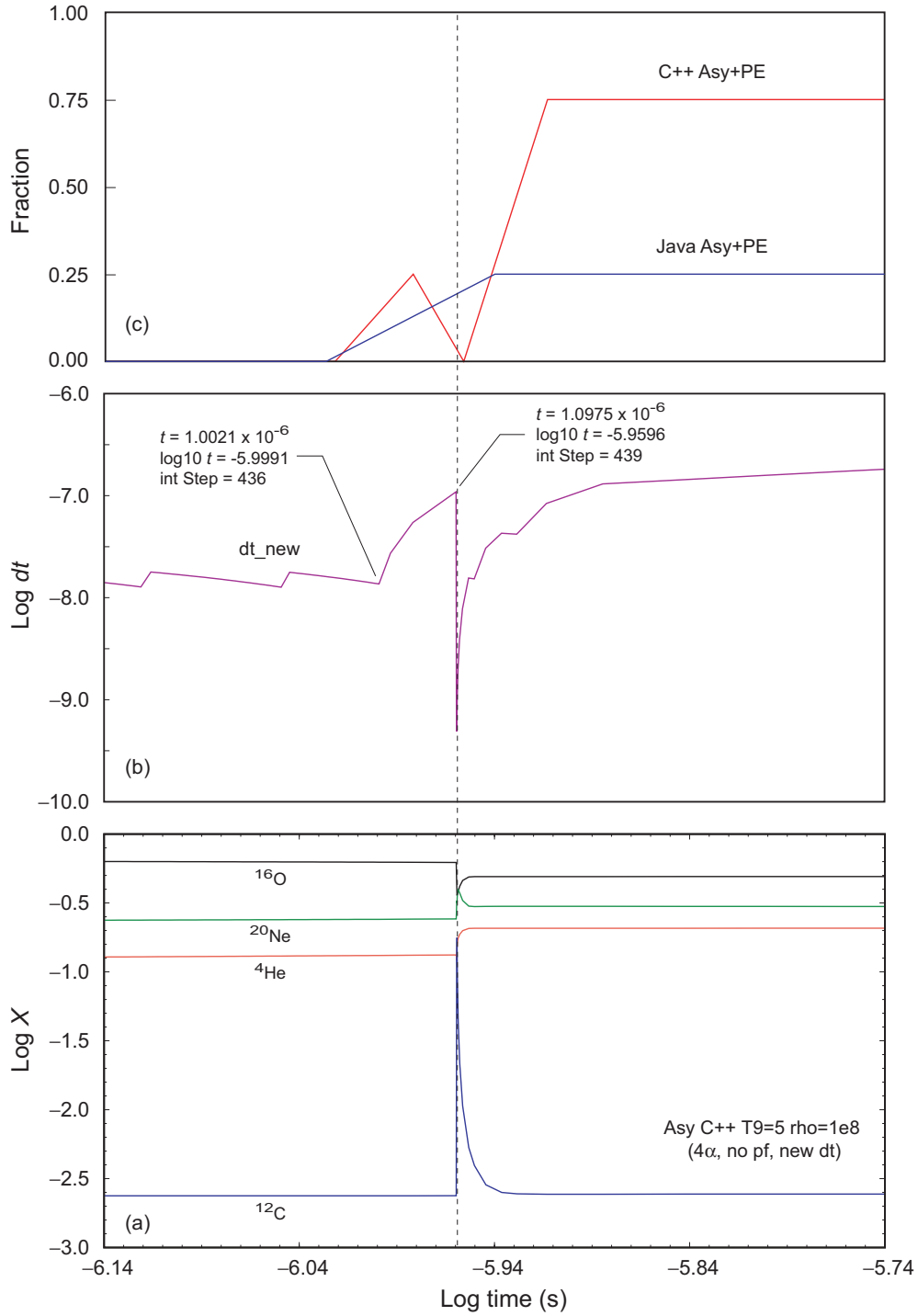

Figure 13: Blow-up of results in Fig. 12 around the region where the calculation goes bad for the new timestepper using C++ Asy+PE. (a) Mass fractions, new timestepper; (b) timestep; (c) fraction of reaction groups equilibrated.

References

[1] Mike Guidry, J. Comp. Phys. **231**, 5266-5288 (2012). [arXiv:1112.4778].

[2] M. W. Guidry, R. Budiardja, E. Feger, J. J. Billings, W. R. Hix, O. E. B. Messer, K. J. Roche, E. McMahon, and M. He, Comput. Sci. Disc. **6**, 015001 (2013) [arXiv: 1112.4716].

[3] M. W. Guidry and J. A. Harris, Comput. Sci. Disc. **6**, 015002 (2013) [arXiv: 1112.4750]

[4] M. W. Guidry, J. J. Billings, and W. R. Hix, Comput. Sci. Disc. **6**, 015003 (2013) [arXiv: 1112.4738]

[5] Benjamin Brock, Andrew Belt, Jay Jay Billings, and Mike Guidry, J. Comp. Phys. **302**, 591-602 (2015).

[6] A. Haidar, B. Brock, S. Tomov, M. Guidry, J. J. Billings, D. Shyles, and J. Dongarra, "Performance Analysis and Acceleration of Explicit Integration for Large Kinetic Networks using Batched GPU Computations" 2016 IEEE High Performance Extreme Computing Conference, HPEC (2016).

[7] E. Endeve et al, unpublished.