



<b>Publication Year</b>	2018
<b>Acceptance in OA</b>	2020-12-28T16:05:30Z
<b>Title</b>	GAMER-2: a GPU-accelerated adaptive mesh refinement code - accuracy, performance, and scalability
<b>Authors</b>	Schive, Hsi-Yu, ZuHone, John A., Goldbaum, Nathan J., Turk, Matthew J., GASPARI, MASSIMO, Cheng, Chin-Yu
<b>Publisher's version (DOI)</b>	10.1093/mnras/sty2586
<b>Handle</b>	<a href="http://hdl.handle.net/20.500.12386/29235">http://hdl.handle.net/20.500.12386/29235</a>
<b>Journal</b>	MONTHLY NOTICES OF THE ROYAL ASTRONOMICAL SOCIETY
<b>Volume</b>	481

# GAMER-2: a GPU-accelerated adaptive mesh refinement code – accuracy, performance, and scalability

Hsi-Yu Schive,<sup>1,2★</sup> John A. ZuHone,<sup>3</sup> Nathan J. Goldbaum,<sup>1</sup> Matthew J. Turk,<sup>4,5</sup> Massimo Gaspari<sup>6</sup> and Chin-Yu Cheng<sup>7</sup>

<sup>1</sup>National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign, IL 61820, USA

<sup>2</sup>Institute of Astrophysics, National Taiwan University, 10617 Taipei, Taiwan

<sup>3</sup>Harvard-Smithsonian Center for Astrophysics, 60 Garden St., Cambridge, MA 02138, USA

<sup>4</sup>School of Information Sciences, University of Illinois, Urbana-Champaign, IL 61820, USA

<sup>5</sup>Department of Astronomy, University of Illinois, Urbana-Champaign, IL 61820, USA

<sup>6</sup>Department of Astrophysical Sciences, Princeton University, 4 Ivy Lane, Princeton, NJ 08544-1001, USA

<sup>7</sup>Department of Electrical and Computer Engineering, University of Illinois, Urbana-Champaign, IL 61820, USA

Accepted 2018 September 16. Received 2018 August 2; in original form 2017 December 19

## ABSTRACT

We present GAMER-2, a GPU-accelerated adaptive mesh refinement (AMR) code for astrophysics. It provides a rich set of features, including adaptive time-stepping, several hydrodynamic schemes, magnetohydrodynamics, self-gravity, particles, star formation, chemistry, and radiative processes with GRACKLE, data analysis with YT, and memory pool for efficient object allocation. GAMER-2 is fully bitwise reproducible. For the performance optimization, it adopts hybrid OpenMP/MPI/GPU parallelization and utilizes overlapping CPU computation, GPU computation, and CPU–GPU communication. Load balancing is achieved using a Hilbert space-filling curve on a level-by-level basis without the need to duplicate the entire AMR hierarchy on each MPI process. To provide convincing demonstrations of the accuracy and performance of GAMER-2, we directly compare with ENZO on isolated disc galaxy simulations and with FLASH on galaxy cluster merger simulations. We show that the physical results obtained by different codes are in very good agreement, and GAMER-2 outperforms ENZO and FLASH by nearly one and two orders of magnitude, respectively, on the Blue Waters supercomputers using 1–256 nodes. More importantly, GAMER-2 exhibits similar or even better parallel scalability compared to the other two codes. We also demonstrate good weak and strong scaling using up to 4096 GPUs and 65 536 CPU cores, and achieve a uniform resolution as high as  $10\,240^3$  cells. Furthermore, GAMER-2 can be adopted as an AMR + GPUs framework and has been extensively used for the wave dark matter simulations. GAMER-2 is open source (available at <https://github.com/gamer-project/gamer>) and new contributions are welcome.

**Key words:** methods: numerical.

## 1 INTRODUCTION

Many problems in computational astrophysics require resolving structures at a wide range of spatial scales. For this reason, the adaptive mesh refinement (AMR) method (Berger & Oliger 1984; Berger & Colella 1989) has played an indispensable role by enabling high dynamic range simulations of astrophysical phenomena. The fundamental principle in AMR is to allow the simulation resolution, in both space and time, to adaptively and locally adjust

so as to concentrate computational resources on regions requiring higher resolution. It is achieved by first covering the entire computational domain with a uniform-resolution grid, and then adding hierarchies of nested refined grid patches (also referred to as ‘patches’, ‘grids’, or ‘blocks’) with decreasing cell spacing over subvolumes of interest.

There have been many hydrodynamic AMR codes for astrophysics (e.g. Kravtsov, Klypin & Khokhlov 1997; Fryxell et al. 2000; Teyssier 2002; Cunningham et al. 2009; Schive, Tsai & Chiueh 2010; Almgren et al. 2010; Mignone et al. 2012; Almgren et al. 2013; Bryan et al. 2014; White, Stone & Gammie 2016). Among these, the AMR implementations can be broadly classified into three

★ E-mail: [hyschive@phys.ntu.edu.tw](mailto:hyschive@phys.ntu.edu.tw)

categories based on the basic units adopted for grid refinement. The first category uses rectangular patches of arbitrary aspect ratios as the refinement units, where different patches can have different sizes and shapes; for example, ENZO (Bryan et al. 2014) utilizes this scheme. The second category performs refinement on a cell-by-cell basis, for example, ART (Kravtsov et al. 1997) and RAMSES (Teyssier 2002). The third category performs refinement on patches of fixed size (e.g.  $8^3$  cells); in other words, all patches are restricted to be geometrically similar to each other. Examples include FLASH (Fryxell et al. 2000), GAMER (Schive et al. 2010), and ATHENA++ (Stone et al. 2008; White et al. 2016). In addition to AMR, it is also possible to achieve high resolution with Lagrangian particles (e.g. GADGET-2, Springel 2005), a moving unstructured mesh (e.g. AREPO, Springel 2010), or a meshless method (e.g. GIZMO, Hopkins 2015).

Use of graphic processing units (GPUs) has recently become a promising technique to achieve substantial speedups in simulation performance. However, compared to the uniform-resolution approaches, it remains extremely challenging for AMR codes to efficiently exploit the petascale computing power in heterogeneous CPU/GPU supercomputers, mainly due to the complicated AMR data structure, load imbalance, and the great amount of work required to isolate and convert existing physics modules to run with high efficiency on GPUs. So far, only a few astrophysical AMR codes have taken advantage of GPU acceleration. For example, both ENZO (Bryan et al. 2014) and RAMSES (Teyssier 2002) have ported the hydrodynamic and magnetohydrodynamic solvers to GPUs (Wang, Abel & Kaehler 2010; Kestener, Château & Teyssier 2010), and FLASH has ported an octree Poisson solver to GPUs (Lukat & Banerjee 2016). GAMER (GPU-accelerated Adaptive MEsh Refinement, Schive et al. 2010, hereafter referred to as GAMER-1), is the first astrophysical AMR code designed from scratch to exploit GPU acceleration. It supports both GPU hydrodynamic and Poisson solvers. However, the physical modules supported in GAMER-1 are much more limited compared to other widely adopted AMR codes, which restricts possible applications of the code.

Here, we present GAMER-2, a significant revision of GAMER-1 that includes much richer functionality, including adaptive time-stepping, several hydrodynamic schemes, magnetohydrodynamics, dual energy formalism, self-gravity, particles, star formation, chemistry, and radiative processes with the GRACKLE library, data analysis with the YT package, memory pool, bitwise reproducibility, test problem infrastructure, and the capability of being used as an AMR + GPUs framework. It also incorporates significant improvements in accuracy, stability, performance, and scalability.

For a GPU-accelerated astrophysical AMR code, there are at least three questions to be addressed in order to make a fair performance comparison with a CPU-only AMR code. First, does it sacrifice accuracy for performance? Second, does it still outperform CPUs by a large margin when enabling a rich set of physical modules? Third, how does the performance scale with the number of GPUs especially when running extremely large parallel simulations? To provide clear and convincing answers to these questions, we directly compare GAMER-2 with two widely adopted AMR codes, FLASH and ENZO, based on realistic astrophysical applications, namely, binary cluster merger simulations and isolated disc galaxy simulations, where we enable GPU acceleration for ENZO as well. These comparison simulations show that the physical results obtained by different codes are in very good agreement, and GAMER-2 outperforms ENZO and FLASH by nearly one and two orders of magnitude, respectively,

on the Blue Waters supercomputer<sup>1</sup> using 1–256 nodes. We also demonstrate good weak and strong scaling in GAMER-2 using up to 4096 GPUs and 65 536 CPU cores.

This paper is structured as follows. We describe the numerical algorithms in Section 2 and the performance optimizations in Section 3. Section 4 shows the code tests, especially focusing on the comparison simulations with GAMER-2, ENZO, and FLASH. Finally, we summarize our results and discuss future work in Section 5.

## 2 NUMERICAL ALGORITHMS

We provide in this section an overview of the numerical algorithms implemented in GAMER-2, including the AMR structure, hydrodynamic and gravity solvers, particle integration, and other miscellaneous features. Detailed descriptions of the performance optimizations and parallelization are given in Section 3.

### 2.1 Adaptive mesh refinement

The AMR structure in GAMER-2 is very similar to that in the original GAMER-1 code, and therefore we only provide a short summary here. Note that in this paper, we use the terms ‘patch’, ‘grid’, and ‘block’ interchangeably. GAMER-2 adopts a block-structured AMR where the simulation domain is covered by a hierarchy of patches with various resolutions. Patches with the same resolution are referred to as being on the same AMR level, where the root level,  $l = 0$ , has the coarsest resolution. The resolution ratio between two adjacent levels is currently fixed to 2. The levels of any two nearby patches can differ by at most 1, so the GAMER-2 AMR hierarchy is always properly nested.

All patches are restricted to be geometrically similar to each other, which is similar to the AMR implementation in FLASH. We assume a fixed patch size of  $8^3$  cells throughout this paper unless otherwise specified. The AMR hierarchy is manipulated by an octree data structure. A patch on level  $l$  can have zero or eight child patches on level  $l + 1$  that cover the same physical domain as their parent patch, and up to 26 sibling patches (including those along the diagonal directions) on the same level. For convenience, all patches store the patch identification numbers of their parent, child, and sibling patches (if any exist).

For advancing patches on different levels, GAMER-2 supports both the shared time-step and adaptive time-step integrations. For the former, all patches in the simulations are restricted to have the same evolution time-step, which is generally more robust but less efficient as patches on lower levels might have unnecessarily small time-steps. This fixed time-step scheme is used by the FLASH code. For the latter, patches on higher levels can have smaller time-steps. Furthermore, GAMER-2 does not require the time-step ratio between two adjacent levels to be a constant, which is similar to the implementation in ENZO. Different levels are advanced in a way similar to the W-cycle in a multigrid scheme, where the lower levels are advanced first and then wait until they are synchronized with higher levels. This approach can improve performance notably, especially when higher refinement levels only cover small subvolumes of the simulation domain.

For the adaptive time-step integration, in order to synchronize two adjacent levels, the finer level sometimes requires an extra update

<sup>1</sup><https://bluewaters.ncsa.illinois.edu>

with an extremely small time-step, which can have a non-negligible impact on the overall performance. To alleviate this issue, we allow the time-step on level  $l$ ,  $\Delta t_l$ , to increase by a small fraction (typically  $C_{\text{inc}} \sim 10$  per cent) if that helps synchronize with level  $l - 1$ . Moreover, and less intuitively, we also allow  $\Delta t_l$  to decrease by a small fraction (typically  $C_{\text{dec}} \sim 10$  per cent as well) if it could remove that extra small time-step on level  $l + 1$ . See Fig. 1 for an illustration of the second optimization. The procedure for computing the final optimum  $\Delta t_l$  can be described by the following pseudo-code:

```
// 'lv' is the abbreviation of 'level'
dt(lv) = ComputeTimeStep(lv)
dt_inc = t(lv-1) - t(lv)
dt_dec = 2*dt(lv+1)
// synchronize lv and lv-1
if (1+C_inc)*dt(lv) > dt_inc
    dt(lv) = dt_inc
// synchronize lv and lv + 1
else if dt(lv) > dt_dec and
(1-C_dec)*dt(lv) < dt_dec
    dt(lv) = dt_dec
```

Note that  $\Delta t_{l+1}$  is taken from the previous update under the assumption that this value does not change too quickly, and the coefficient 2 for computing  $\Delta t_{\text{dec}}$  further assumes that  $\Delta t_l$  is linearly proportional to the cell spacing on level  $l$ . Also note that this procedure can be directly applied to simulations with multiple levels. In principle, allowing  $\Delta t$  to vary by a larger fraction (by increasing  $C_{\text{inc}}$  or  $C_{\text{dec}}$ ) can improve performance further, but it could also deteriorate accuracy and stability.

A patch can be checked for refinement when synchronized with the child level. GAMER-2 supports a variety of refinement criteria, including, for example, the amplitude, first and second derivatives of simulation variables, vorticity, and Jeans length. The second derivative criterion follows the error estimator suggested by Löhner et al. (1987) and implemented in FLASH. For the simulations with particles, one can also check the number of particles in a patch, the number of particles in a cell, and the particle mass in a cell. The refinement thresholds on different levels can be set independently. A patch is flagged for refinement if any of its cells satisfy the refinement criteria. In addition, we add  $(1 + 2N_{\text{buf}})^3 - 1$  flag buffers, where  $N_{\text{buf}} = 0 - 8$ , around each flagged cell and refine the corresponding sibling patches as well if any of these flag buffers extend across the patch border. See fig. 2 in Schive et al. (2010) for an illustration.

GAMER-2 does not implement explicit grid derefinement criteria. A patch is removed if its parent patch does not satisfy any refinement criteria and if it has no child patches. In addition, a patch is not refined if it would violate the proper-nesting constraint. This approach makes grid refinement significantly more efficient, since only one refinement level is altered at a time and there is no need to reconstruct the entire AMR hierarchy, which would otherwise be very expensive. However, it could also cause an issue of insufficient resolution since patches might not be refined in time due to the proper-nesting check. To solve this issue, we typically set  $N_{\text{buf}} = 8$  on all levels except for  $l_{\text{max}} - 1$ , where  $l_{\text{max}}$  is the maximum refinement level, so that patches on lower levels are always pre-allocated. Having  $N_{\text{buf}} = 8$  on level  $l_{\text{max}} - 1$  is unnecessary as no patches will be allocated above level  $l_{\text{max}}$ , and thus we typically set  $N_{\text{buf}} = 2-4$  on level  $l_{\text{max}} - 1$  just to prevent the phenomenon of interest from leaving the highest resolution regions. As will be demonstrated in Section 4, this approach, although not optimal, is found to do a reasonable job.

## 2.2 Hydrodynamics

GAMER-2 supports four hydrodynamic schemes: the relaxing total variation diminishing scheme (RTVD; Jin & Xin 1995), the MUSCL–Hancock scheme (MHM; for an introduction, see Toro 2009), a variant of the MUSCL–Hancock scheme (VL; Falle 1991; van Leer 2006), and the corner transport upwind scheme (CTU; Colella 1990). There are several variants of the CTU scheme in the literature, and we adopt the one requiring six Riemann solvers per cell per time-step, similar to that implemented in ATHENA (Stone et al. 2008). The detailed implementation of these schemes have been described previously (Trac & Pen 2003; Stone et al. 2008; Stone & Gardiner 2009; Schive et al. 2010; Schive, Zhang & Chiueh 2012), which we do not repeat here. GAMER-2 also supports magnetohydrodynamics<sup>2</sup> (Zhang, Schive & Chiueh 2018) using the CTU scheme and the constrained transport (CT; Evans & Hawley 1988) technique to ensure the divergence-free constraint on the magnetic field, for which we closely follow the implementation in ATHENA.

The RTVD scheme is dimensionally split and Riemann-solver-free, and the other three schemes are dimensionally unsplit and Riemann-solver-based. GAMER-2 supports four Riemann solvers for hydrodynamics, namely, exact solver based on Toro (2009), Roe’s solver (Roe 1981), HLLC solver (Toro 2009), and HLLC solver (Einfeldt et al. 1991), and three Riemann solvers for magnetohydrodynamic (MHD), namely, Roe’s solver, HLLD solver (Miyoshi & Kusano 2005), and HLLC solver. For the data reconstruction schemes, we have implemented the piecewise linear method (PLM; van Leer 1979) and piecewise parabolic method (PPM; Woodward & Colella 1984), both of which can be applied to either primitive or characteristic variables. We have also implemented a variety of slope limiters, including the generalized minmod limiter, van Leer-type limiter, van Albada-type limiter, and a hybrid limiter combining the generalized minmod and van Leer-type limiters (for an introduction, see Toro 2009).

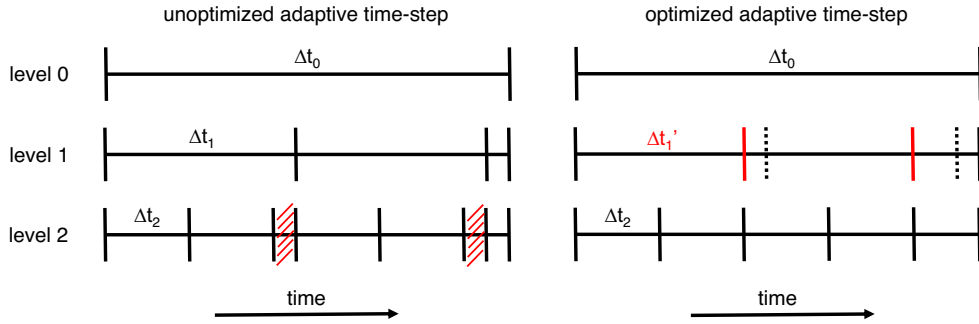
All Riemann-solver-based schemes support multispecies hydrodynamics by additionally solving the continuity equations of an arbitrary number of chemical species, which can be renormalized after every update to ensure that the sum of their mass fractions equals unity. The non-equilibrium chemical reaction network is updated by the library GRACKLE (Smith et al. 2017, see Section 2.5).

There are four standard fluid boundary conditions available in GAMER-2, namely, periodic, outflow, inflow (i.e. user-defined), and reflecting boundaries. See, for example, Bryan et al. (2014), for an introduction. The user-defined boundaries can be time-dependent. The boundary conditions on different faces of the computational domain can be set independently.

It is a well-known problem that the pressure of flows with very high Mach number suffers from large truncation errors, which can be a serious issue when the gas temperature is required (e.g. for calculating chemistry and radiative cooling). The common remedy is to adopt the so-called dual energy formalism, for which one evolves an additional auxiliary variable, either thermal energy (Bryan et al. 1995, 2014) or entropy (Ryu et al. 1993; Springel 2010), and uses that to calculate pressure or temperature when required. GAMER-2 adopts the gas entropy per unit volume,  $s \equiv P/\rho^\gamma - 1$ , where  $P$  is gas pressure,  $\rho$  is gas mass density, and  $\gamma$  is adiabatic index, as the auxiliary variable, since it is a conserved quantity outside

<sup>2</sup>The MHD extension has not been made publicly available yet, but will be released soon.





**Figure 1.** Illustration of the adaptive time-step integration. The left-hand panel shows the unoptimized scheme where the diagonal hatching highlights the extra small time-steps on level 2 required to synchronize with level 1. The right-hand panel shows the optimized scheme where the time-steps on level 1,  $\Delta t_1'$ , are reduced by a small fraction in order to eliminate the extra synchronization time-steps on level 2. The vertical dotted lines represent the original time-steps on level 1. Note that the optimized scheme reduces the total number of updates on level 2 from 7 and 5, while that on levels 0 and 1 remain unchanged.

the shocks and therefore can be easily incorporated by regarding it as one of the passively advected scalars. In the end of the fluid solver, we check the ratio between the gas thermal energy  $E_{\text{the}}$  and kinetic energy  $E_{\text{kin}}$  on each cell, where  $E_{\text{the}}$  is estimated from the original total energy formulation (i.e.  $E_{\text{the}} = E_{\text{tot}} - E_{\text{kin}}$  where  $E_{\text{tot}}$  is the gas total energy density). If this ratio is below a given threshold,  $E_{\text{the}}/E_{\text{kin}} < \xi$ , where  $\xi \sim 10^{-2}$  typically, we correct  $E_{\text{tot}}$  using the entropy calculated from the dual energy formalism. Otherwise, the auxiliary entropy information on this cell is simply disregarded.

Unphysical results such as negative density and pressure may still arise even with the help of the dual energy formalism, especially when adopting a less diffusive hydrodynamic scheme (e.g. CTU scheme and PPM reconstruction) in poorly resolved cold flows (e.g. when the radiative cooling is very effective). If an unphysical result is detected in a cell, GAMER-2 adopts the following procedure to try to remedy the problem.

(i) If unphysical results occur in the intermediate region of the Roe’s solver, we follow the approach adopted in ATHENA and switch to either HLLE, HLLC, or exact solver to recalculate the fluxes on the failed cell interfaces.

(ii) If unphysical results occur in the final update of the fluid solver, we recalculate the solutions with only first-order accuracy in space and time and in a dimensionally unsplit fashion.

(iii) If step (ii) still fails, we repeat step (ii) but with a dimensionally split update.

(iv) If step (iii) still fails, we reduce the time-step by a fixed ratio (0.8 by default) and recalculate all patches on the currently targeted level with this reduced time-step.

(v) Repeat step (iv) until no unphysical results are detected or reaching a given minimum time-step threshold, for which the program is aborted.

Note that steps (i)–(iii) are applied only to the cells with unphysical results, and steps (iv) and (v) are applied to all patches on the currently targeted level. The latter is feasible thanks to the adaptive time-step integration described in Section 2.1. Moreover, the time-step in the next update is automatically restored to the original value, and thus a maximum time-step can be applied whenever possible to reach optimal performance.

When a coarse level is synchronized with its child level, there are two ‘fix-up’ operations required to correct the coarse-grid data in order to ensure the consistency between different levels. First, for non-leaf coarse patches (i.e. patches with child patches), their

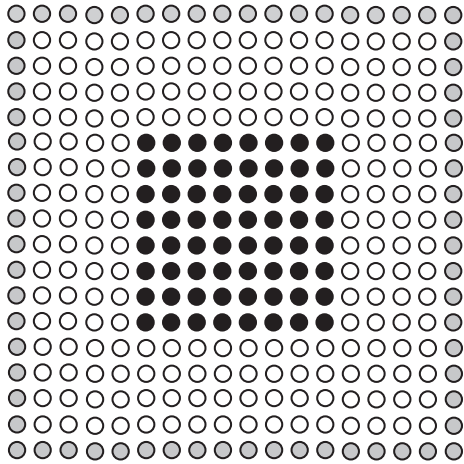
conserved quantities are replaced by the volume-weighted average of the child patch data. This operation is sometimes referred to as ‘restriction’. Second, for leaf coarse patches (i.e. patches without child patches) near coarse-fine interfaces, the conserved quantities on the coarse cells adjacent to the interfaces are corrected by the differences between the coarse-grid and fine-grid fluxes across the interfaces. These corrections ensure conservation of conserved quantities to machine precision in pure hydrodynamic simulations (see the next section for the case with self-gravity).

### 2.3 Gravity

Gravitational potential is evaluated by solving the discretized Poisson equation subject to given boundary conditions. GAMER-2 supports both periodic and isolated boundary conditions for gravity. On the root level, we use the standard fast Fourier transform (FFT) method with the FFTW package (Frigo & Johnson 2005) to convolve mass density with a proper Green’s function in  $k$ -space and then transform back to get real-space potential defined at the cell centres. For periodic boundary conditions, the Green’s function corresponds to a second-order finite-difference representation of the Laplacian operator (Hockney & Eastwood 1988). For isolated boundary conditions, the Green’s function takes into account proper zero-padding in the real space (Eastwood & Brownrigg 1979), which is equivalent to regarding all cells as point masses and solving their pairwise potential.

The gravitational potential in refined regions is calculated by solving the discretized Poisson equation with the Laplacian operator replaced by its second-order finite-difference equivalent. We set the potential boundary conditions by interpolating from the coarse patches using quadratic interpolation in space (and linear interpolation in time when adopting the adaptive time-step integration), and then use relaxation methods to solve the discretized Poisson equation to machine precision. GAMER-2 by default uses the successive over-relaxation (SOR; Press et al. 2007) method. See Section 3.1 for more discussions.

We calculate the gravitational potential of different patches on the same refinement level independently in the sense that we do not exchange solutions between nearby patches during the relaxation. This approach leads to a significantly more efficient parallelization compared to the multilevel Poisson solver adopted in GAMER-1, mainly because all patches on the same level can be updated simultaneously. Moreover, it can be applied straightforwardly to the adaptive time-step integration since no correction from fine



**Figure 2.** 2D example of the active and buffer zones adopted in the Poisson solver used for refined regions. For each patch with  $8^2$  active zones (central black-filled circles), we add five additional buffer zones (open and grey-filled circles) around it to make the potential smoother across the patch boundaries. The outermost cells (grey-filled circles) provide the boundary conditions which are fixed during the SOR iterations. Buffer zones are only temporarily allocated for the patches being updated and are discarded after being used for evaluating both the potential and acceleration in the active zones.

to coarse grids is required. However, it also results in numerical errors accumulated on patch boundaries, including the interfaces between patches both on the same level and on different levels (e.g. Huang & Greengard 1999; Martin & Colella 2000; Ricker 2008). To partially alleviate this problem, we add five additional buffer zones around each patch (see Fig. 2) and perform relaxation for the entire  $(8 + 2 \times 5)^3 = 18^3$  grid, with the outermost cells being fixed during the iterations. These buffer zones are only temporarily allocated for the patches being updated and are discarded after used for evaluating both potential and acceleration in the active zones. This approach is found to substantially reduce errors on patch boundaries, though not completely eliminate them. See Appendix A for details. The comparisons with other codes based on realistic astrophysical applications show satisfactory results (see Section 4). More complicated schemes dedicated to AMR simulations (e.g. Ricker 2008) will be investigated in the future. Also note that although solving a large  $18^3$  grid for each  $8^3$  patch might, at first glance, seem to have a great impact on the performance, experiments show that our GPU Poisson and hydrodynamic solvers achieve similar performance (see Section 4.1).

Gravity is coupled to hydrodynamics in an operator-unsplit predictor-corrector approach identical to that implemented in AREPO (Springel 2010) and described by Müller & Steinmetz (1995), which is second-order accurate. First, we invoke the original hydrodynamic solver with the half-step prediction of velocity incorporating gravity defined at the beginning of the time-step. This gives us the proper mass density at the end of the time-step, from which we can update gravity. With the gravity both at the beginning and end of the time-step in hand, we can correct momentum and then use the corrected momentum to further correct energy, which completes the second-order accurate updates of all fluid variables. Note that this procedure requires only one invocation of Poisson solver per time-step.

For a given potential field, we evaluate the gravitational acceleration at the cell centres via either two- or four-point central differencing along each direction. It can incorporate a user-defined,

time-dependent external acceleration. Note that the conservation of total momentum and energy are prone to truncation errors when self-gravity is included since it is not implemented in a conservative form (see e.g. Jiang et al. 2013).

## 2.4 Particles

GAMER-2 supports evolving active particles. We adopt the standard ‘kick-drift-kick’ (KDK) instead of ‘drift-kick-drift’ (DKD) integrator, since the former requires the gravitational acceleration at the beginning and end of each time-step that is more consistent with the way we incorporate gravity into hydrodynamic solvers (see Section 2.3). Particles are associated with leaf patches and updated together with these patches. They can be dynamically created (e.g. for star formation) or removed (e.g. for those leaving the computational domain) at runtime. The built-in particle attributes include mass, positions, velocities, accelerations, and time. We store the accelerations of each particle to avoid redundant computations associated with the two kick operators in the KDK scheme. We also record the physical time of each particle since, for the adaptive time-step integration, particles are not guaranteed to be synchronized with the associated leaf patches, which will be discussed shortly. Particles can also carry an arbitrary number of additional attributes (e.g. metallicity fraction).

When adopting the adaptive time-step integration, particles moving across coarse-fine interfaces (after the ‘drift’ operator) need to be treated with special care, since the coarse level is updated before the fine level and with a larger time-step. First, for particles moving from coarse to fine patches, we always use the *coarse*-grid gravity for the next kick operator to avoid temporal extrapolation on the potential. Second, particles may not be synchronized with their new associated leaf patches after moving across coarse-fine interfaces. For particles moving from coarse to fine patches, their physical time is always ahead of the time associated with the new host patches. On the other hand, for particles moving from fine to coarse patches, their physical time may be behind the time of the new host patches. To solve this issue, we record the physical time of all particles so that we can synchronize each particle and its host patch in time by taking into account their small time interval differences during the next particle update. It also allows us to ‘predict’ particle positions at any given time when required (e.g. for assigning particle mass onto grids as discussed below).

Particles and fluid share the same gravitational force. Since GAMER-2 adopts a grid-based gravity solver, one needs to not only deposit particles onto grids to get the total mass density for the Poisson solver, but also interpolate the cell-centred accelerations to particle positions. We have implemented the standard nearest-grid-point, cloud-in-cell (CIC), and triangular-shape cloud schemes (Hockney & Eastwood 1988) for this purpose. Special care needs to be taken when assigning the buffer-zone densities (i.e. the open circles in Fig. 2) outside the coarse-fine interfaces of fine patches. Instead of depositing particle mass onto coarse grids and then performing spatial interpolation, we deposit particles onto the buffer zones of fine patches directly. This approach avoids the possibility of double counting particles adjacent to the coarse-fine interfaces. In addition, for the adaptive time-step scheme, particles collected from both higher and lower levels need to be synchronized with the targeted level before mass deposition.

There are several limitations in the current implementation of particles. (i) Tracer particles following the motion of fluid are not supported. (ii) Comoving coordinates for cosmological simulations are not supported. (iii) There can only be a single particle type in

the sense that the number of particle attributes must be the same for all particles. These restrictions will be removed in the near future.

## 2.5 Chemistry and radiative processes

GAMER-2 supports multispecies hydrodynamics and has incorporated the publicly available GRACKLE<sup>3</sup> chemistry and radiative cooling/heating library (Smith et al. 2017). GRACKLE supports both a non-equilibrium solver with 6–12 species and a tabulated cooling function assuming ionization equilibrium for the primordial chemistry and cooling. It also includes other features such as tabulated metal cooling, photoelectric heating from dust, effective cosmic microwave background (CMB) temperature floor, and various ultraviolet (UV) background models. See Smith et al. (2017) for details.

When updating a given AMR level, we transfer the species density and internal energy of a fixed number of patches to GRACKLE at a single time. It then evolves the chemical network with a semi-implicit method (for the non-equilibrium solver), computes the radiative cooling and heating rates, and updates the gas internal energy on a cell-by-cell basis. The integrations are subcycled since the characteristic time-scales of chemistry and radiative processes could be much smaller than those of hydrodynamics and particles. We adopt an operator-split approach to couple the radiative cooling and heating as energy source terms to a hydrodynamic scheme.

## 2.6 Time-step

GAMER-2 adopts various time-step criteria listed below:

$$\Delta t_{\text{CFL1}} \leq C_{\text{CFL}} \frac{\Delta h}{\max(|v_x|, |v_y|, |v_z|) + c_s}, \quad (1)$$

$$\Delta t_{\text{CFL2}} \leq C_{\text{CFL}} \frac{\Delta h}{|v_x| + |v_y| + |v_z| + 3c_s}, \quad (2)$$

$$\Delta t_{\text{acc}} \leq C_{\text{acc}} \left( \frac{\Delta h}{\max(|a_x|, |a_y|, |a_z|)} \right)^{1/2}, \quad (3)$$

$$\Delta t_{\text{par}} \leq C_{\text{par}} \frac{\Delta h}{\max(|v_{\text{par},x}|, |v_{\text{par},y}|, |v_{\text{par},z}|)}, \quad (4)$$

where  $\Delta h$  is cell spacing. For simplicity, here we assume that all cells are cubic.  $C_{\text{CFL}}$ ,  $C_{\text{acc}}$ , and  $C_{\text{par}}$  are the safety factors with typical values of  $\sim 0.5$ .

Equations (1) and (2) specify the Courant–Friedrichs–Lewy (CFL) condition of hydrodynamic schemes, where  $v_x$ ,  $v_y$ , and  $v_z$  are the fluid velocities and  $c_s$  is the sound speed. Equation (1) applies to the RTVD and CTU schemes and equation (2) applies to the MHM and VL schemes. Note that the CTU scheme in GAMER-2, which invokes six Riemann solvers per cell per time-step, requires  $C_{\text{CFL}} \leq 0.5$ , while the other three schemes support  $C_{\text{CFL}} \leq 1.0$ . Equation (3) takes into account the accelerations resulting from both self-gravity and external forces, where  $a_x$ ,  $a_y$ , and  $a_z$  are the accelerations of both fluid and particles. Equation (4) prevents particles from travelling more than one cell width in a single time-step, which is important for both improving accuracy and simplifying particle manipulation.

For the shared time-step integration, we evaluate the time-step constraints of equations (1)–(4) on all levels and take the minimum value. For the adaptive time-step integration, we only need to evaluate these constraints for the patches and particles on a given AMR level, with an additional constraint that the physical time on a child level cannot be ahead of that on its parent level. Moreover, as mentioned in Section 2.1, we allow the minimum time-step determined from equations (1)–(4) to vary by a small fraction (typically  $\sim 10$  per cent) to help synchronize adjacent levels. Finally, as mentioned in Section 2.2, we further reduce the time-step by a fixed ratio (0.8 by default) if the previously adopted time-step led to unphysical results in the hydrodynamic solver.

## 2.7 Miscellaneous features

### 2.7.1 Bitwise reproducibility

GAMER-2 supports bitwise reproducibility, in the sense that the round-off errors can be guaranteed to be the same when (i) running simulations with different numbers of message passing interface (MPI) processes and OpenMP threads (see Section 3.2), and (ii) restarting simulations from checkpoint files. This is a non-trivial task for any parallel AMR code, especially with particles, since the order of all floating-point operations needs to be carefully designed to be deterministic. Specifically, GAMER-2 performs the following additional calculations when bitwise reproducibility is demanded:

(i) Recalculate gravitational potential before writing snapshots if the potential data are not stored on disc. This ensures that the potential and total mass density are fully consistent with the adopted Poisson solver, which otherwise is not strictly guaranteed, for example in non-leaf patches after applying the fine-to-coarse data averaging (i.e. the ‘restriction’ operation; see Section 2.2) and in newly allocated patches whose potential data are initialized by interpolation instead of Poisson solver.

(ii) Ensure that the differences between the coarse- and fine-grid fluxes across the coarse-fine interfaces (which are used in the flux correction; see Section 2.2) are computed in a deterministic order even when the coarse- and fine-grid fluxes are calculated separately by different MPI processes.

(iii) Ensure that the average total mass density in the entire computational domain, which is required when solving the Poisson equation in the comoving coordinates, is calculated in a deterministic order.

(iv) Sort particles in the same patches by their spatial coordinates before mass deposition.

(v) Use the `static` OpenMP scheduling when necessary.

Note that, in general, GAMER-2 does not support bitwise reproducibility for simulations requiring random numbers. However, it is achievable, for example, by having random number seeds be a function of time and patch coordinates. We have implemented it in the stochastic star formation model adopted in the isolated disc galaxy simulations described in Section 4.4.

Bitwise reproducibility is important for scientific reproducibility and very helpful for debugging. However, it can also deteriorate performance, especially for simulations requiring a large number of particles due to the extra particle sorting. Therefore, we implement this feature as a compile-time option, and disable it throughout Section 4. Also note that the bitwise reproducibility addressed here does not apply to the results using different processors (e.g. different

<sup>3</sup><https://grackle.readthedocs.io>

CPUs or GPUs), different compilers, or even different compilation flags.

### 2.7.2 Data analysis

GAMER-2 supports two file formats for writing simulation snapshots: a simple binary format and HDF5.<sup>4</sup> HDF5 is our preferred format as it is more extensible, portable, and easier for post-processing. We currently store each snapshot into a single file and have different MPI processes dump data serially. Writing data in parallel and into multiple files will be investigated in the future. Snapshots in both formats can be used for restarting simulations and can be loaded in parallel.

The HDF5 snapshots of GAMER-2 can be loaded by YT<sup>5</sup> (Turk et al. 2011), a powerful, publicly available, PYTHON-based package for analysing and visualizing volumetric data. YT supports multiple simulation codes (e.g. FLASH and ENZO), which is indispensable for this work as it provides direct and fair comparisons between the simulation results of different codes. It also allows one to share the data analysis scripts to the community straightforwardly, which greatly improves the scientific reproducibility. In addition, for conducting inline analysis, we are experimenting with libyt,<sup>6</sup> a runtime interface for passing in-memory GAMER-2 data structures to YT.

To help monitor the simulation status, GAMER-2 supports recording a rich set of simulation information after each root-level update (i.e. after all levels are synchronized), which can be used as diagnostics during as well as after the simulations:

- (i) Detailed timing analysis of all major routines (such as hydrodynamics, gravity, particles, MPI, etc.).
- (ii) Maximum and average memory consumption per MPI process.
- (iii) Overall performance in terms of total cell updates and particle updates per second.
- (iv) Number of patches and particles on each level.
- (v) Estimation of load imbalance.
- (vi) Errors in the conserved quantities (e.g. gas mass and momentum).
- (vii) Message size and achieved bandwidth in various MPI calls.
- (viii) Evolution time-steps on each level estimated from various constraints (see Section 2.6).

### 2.7.3 Test problem infrastructure

The test problem infrastructure in GAMER-2 is designed in a way that each problem is self-contained and largely decoupled from the complicated AMR structure and parallelization. To add a new simulation, in most cases, one does not have to touch the existing source code (except for defining a new problem identification). Instead, one creates a new problem directory and specifies the gas and particle initial conditions therein. The gas quantities can either be set by an analytical function of space and time or loaded from a binary file with a uniform-resolution array. The particle attributes can also either be set dynamically or loaded from a binary file. In addition, one can specify various problem-specific runtime parameters and functionalities, for instance, refinement

criteria, time-step constraints, boundary conditions, and external accelerations.

### 2.7.4 AMR + GPUs framework

Although GAMER-2 is mainly designed for astrophysical hydrodynamic simulations, it can also be adopted as a high-performance, multiscale AMR framework powered by multi-GPU acceleration (Shukla et al. 2011). It is because the AMR structure, parallelization, and performance optimizations (see Section 3) are implemented carefully to be largely independent of the partial differential equations (PDE) being solved. As a successful example, the code has been extensively used for the wave dark matter ( $\psi$ DM) simulations (Schive, Chiueh & Broadhurst 2014a; Schive et al. 2014b; De Martino et al. 2017), where we solve the Schrödinger–Poisson equation by replacing the hydrodynamic solver with a quantum-mechanical kinematic energy solver and by reusing the same Poisson solver. The detailed implementation of this work will be described elsewhere (Schive et al., in preparation).

## 3 PERFORMANCE OPTIMIZATIONS

In this section, we focus on various performance optimization strategies in GAMER-2, including the GPU implementation, hybrid MPI/OpenMP/GPU parallelization, load balancing, and memory management.

### 3.1 GPU implementation

We use CPUs to manipulate the AMR data structure and only port the time-consuming routines to GPUs, currently including the hydrodynamic solvers, Poisson solvers, and time-step calculations. This approach takes advantage of both CPUs and GPUs, allowing a substantial performance improvement compared to a calculation using only CPUs without sacrificing the flexibility and extensibility of the code. In addition, it allows us to store all the data in the CPU memory and only temporarily transfer a small portion of data to the relatively small GPU memory. We use CUDA (NVIDIA 2017) as the GPU programming interface.

We have implemented all the hydrodynamic solvers mentioned in Section 2.2 on GPUs, namely, the RTVD, MHM, VL, and CTU schemes. See Schive et al. (2010) and Schive et al. (2012) for the detailed implementation. Note that the RTVD scheme takes advantage of the fast GPU shared memory, while the others only use the GPU global memory, since the latter schemes are dimensionally unsplit which makes using the small shared memory less straightforward. However, we still find that the MHM, VL, and CTU schemes achieve significantly larger performance speedups compared to the RTVD scheme, conceivably because the Riemann-solver-based schemes have much higher arithmetic intensity and thus are more GPU-friendly. See Section 4.1 for the performance benchmarks of various CPU and GPU solvers.

The SOR solver mentioned in Section 2.3 has been ported to GPUs. We have abandoned the complicated scheme implemented in GAMER-1 that utilizes *both* the fast GPU shared memory and the per-thread registers to reduce the shared memory usage. Modern GPUs have a significantly larger shared memory (at least 48 KB per multiprocessor), and thus we can simply store the gravitational potential of the entire  $18^3$  grid into the shared memory to boost the performance. To reduce the communication between CPUs and GPUs, we transfer the coarse-grid potential to GPUs, and then

<sup>4</sup><https://support.hdfgroup.org/HDF5>

<sup>5</sup><http://yt-project.org>

<sup>6</sup><https://bitbucket.org/data-exp-lab/libyt>



perform spatial interpolation on the GPU to set both the boundary conditions and initial guess of the potential solution for subsequent iterations. Obtaining an initial guess of the potential on the entire  $18^3$  grid, although unnecessary, accelerates convergence. Also note that we calculate the cell-centred gravitational acceleration and use that to update fluid variables on GPUs immediately after solving the potential, which helps reduce the amount of data transferred between CPUs and GPUs.

We also use GPUs to compute the time-step constraints on grids, i.e. equations (1)–(3), which otherwise would take a surprisingly large fraction of simulation time (e.g. see the timing results of various operations shown in the end of Sections 4.3.3 and 4.4.3). However, currently we have not ported any particle routines to GPUs, which will be investigated in the future.

GAMER-2 stores all the data on the CPU’s main memory, including both the AMR structure and physical data, and only sends the data of patches being updated to GPUs. Moreover, when updating a given level, it is usually unnecessary to transfer all patches on this level to GPUs as long as the performance has saturated. Therefore, the GPU memory consumption can be greatly reduced, and, more importantly, is largely independent of the simulation scale. However, one drawback of this approach is that we need to transfer data between CPUs and GPUs frequently, which can be expensive. To mitigate this issue, we utilize CUDA streams (NVIDIA 2017) to overlap CPU–GPU communication by both CPU and GPU computations (see fig. 5 in Schive et al. 2010 for an illustration), which can lead to a factor of 2 speedup in the GPU solvers (see Section 4.1). In addition, as a result of the octree data structure, we can always group eight sibling patches that share the same parent into a larger grid (referred to as a ‘patch group’). Manipulating on patch groups instead of individual patches reduces both the computation and communication overhead associated with the ghost zones of each patch.

All GPU solvers in GAMER-2 support single and double precision, as a compile-time option. On high-end GPUs (e.g. P100 and V100), single-precision performance is about 2 times faster than double-precision performance, but this ratio can be noticeably higher on older GPUs. We typically find that single precision provides a satisfactory accuracy, as demonstrated in the comparison simulations shown in Sections 4.3.3 and 4.4.3, except for applications requiring either an extremely large dynamic range or resolving extremely small perturbations. Therefore, we adopt single precision throughout this paper unless otherwise specified.

### 3.2 Hybrid MPI/OpenMP/GPU

In GAMER-2, only the most time-consuming routines are ported to GPUs, and the code still uses CPUs extensively for various tasks, including, for example, manipulating the AMR structure, checking the grid refinement criteria, depositing particle mass onto grids, and updating particle attributes. Moreover, even for those GPU-accelerated routines, we still need to use CPUs to collect data from different patches and fill the ghost zones of each patch (or patch group) by either copying directly from sibling patches or interpolating from coarse patches. Therefore, it is essential to efficiently exploit both the CPU and GPU computing power in order to achieve optimal overall performance.

To this end, we have implemented a hybrid MPI/OpenMP/GPU parallelization model. In addition to the GPU acceleration described in the previous section, we further adopt OpenMP for intranode parallelization of all time-consuming CPU routines and MPI for internode communication. Fig. 3 shows an illustration. This ap-

proach allows the code to fully exploit the computing power in heterogeneous CPU/GPU supercomputers. Moreover, the hybrid MPI/OpenMP implementation can significantly improve the parallel scalability by reducing the amount of MPI communication, especially when using a large number of nodes. It can also reduce the CPU memory overhead associated with MPI buffers.

Note that the GRACKLE library used for solving chemistry and radiative processes also supports OpenMP (Smith et al. 2017), and therefore can be easily incorporated into the hybrid MPI/OpenMP parallelization model adopted here. A GPU-accelerated version of GRACKLE is under development.

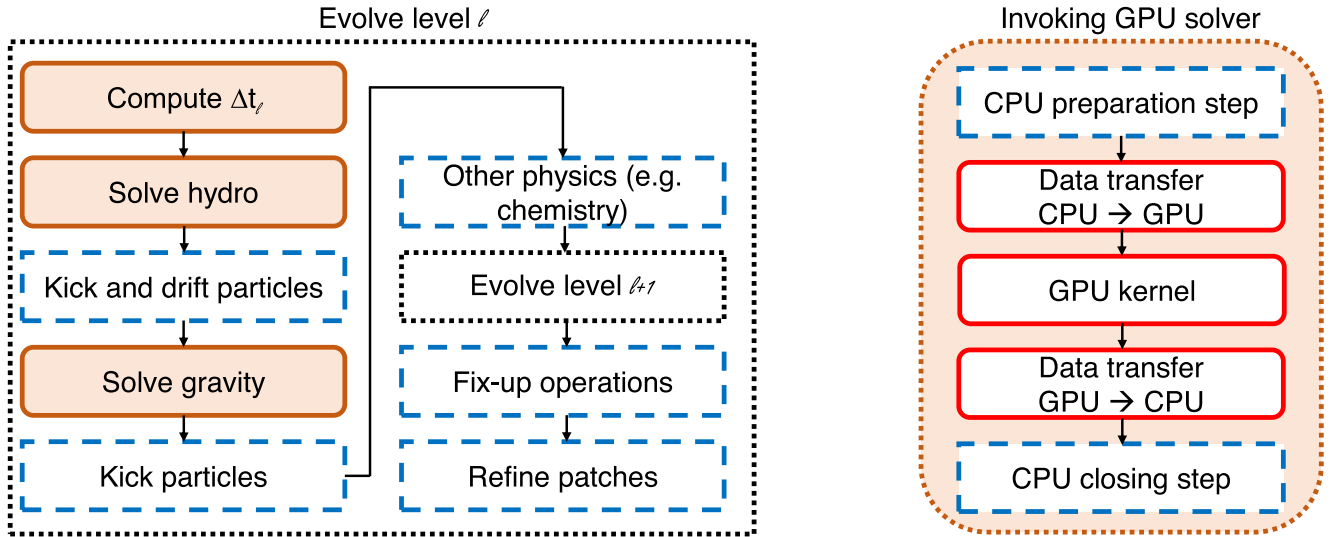
To boost the performance further, we have utilized the asynchronous feature of GPU kernels to let CPUs and GPUs work concurrently. Specifically, we divide all patches on a given level to several subsets, and use CPUs to prepare the input data of one subset and GPUs to update a different subset simultaneously. See fig. 9 in Schive et al. (2010) for an illustration. The number of patches in a single subset is chosen to saturate the GPU performance (see Section 4.1) and is set by default to the product of the number of CUDA streams and the number of multiprocessors in the adopted GPU. We find that substantial performance improvement up to a factor of 2 can be achieved by this approach, especially for simulations where the CPU and GPU computation times are comparable.

There is no restriction on the number of MPI processes per node and the number of OpenMP threads associated with each MPI process. Typically, we set the number of MPI processes equal to the total number of GPUs, and then determine the number of OpenMP threads from the ratio between CPU cores and GPUs. However, empirically we have found that launching multiple MPI processes to access the same GPU using the CUDA Multi-Process Service (MPS) can improve the performance. It is also important to take into account thread affinity and non-uniform memory access (NUMA). Generally, it is recommended to have OpenMP threads running in the same NUMA domain to improve memory affinity. But one needs to experiment with different configurations to fine-tune the overall performance.

Note that GAMER-2 can also run in a ‘CPU-only’ mode, since for all GPU solvers we have implemented their CPU counterparts. These ‘CPU’ solvers are parallelized with OpenMP, with different threads calculating different patches (or patch groups). This parallelization method is found to be very efficient since the computational workload associated with each patch is not only balanced (when disregarding particles) but also generally much larger than the OpenMP overhead. Moreover, the same MPI implementation can be applied to both GPU-accelerated and CPU-only simulations. Therefore, GAMER-2 is also suitable for CPU-only supercomputers, particularly for those with a larger number of cores per node (e.g. Intel Xeon Phi Knights Landing; KNL), for which hybrid MPI/OpenMP is essential to get optimal performance. We will further investigate and optimize this promising feature in the future.

### 3.3 Load balancing

Load balancing is crucial for parallel scalability. For a given level, we use a Hilbert space-filling curve to map the three-dimensional (3D) coordinates of all patches on this level onto a 1D curve. We assign a weight to each patch, which estimates its computational workload, and then cut the curve into  $N_{\text{process}}$  segments with approximately equal weights, where  $N_{\text{process}}$  is the total number of MPI processes. Load balancing can then be achieved by having different MPI processes calculate patches on different curve segments.



**Figure 3.** Illustration of the hybrid OpenMP/GPU parallelization in a single MPI process. Left-hand panel: flowchart of the main loop for evolving a given AMR level  $l$ , where dashed boxes are CPU-only operations parallelized with OpenMP and solid boxes are GPU-accelerated solvers. Right-hand panel: procedure for invoking a GPU solver, such as computing  $\Delta t$  and solving hydrodynamics and gravity as indicated in the left-hand panel. Dashed boxes are CPU-only operations parallelized with OpenMP (see Section 3.1 of Schive et al. 2010 for the definitions of CPU preparation and closing steps) and solid boxes are GPU-related operations. Note that both the GPU kernel execution and CPU–GPU communication are asynchronous and can be overlapped with CPU computation as long as they are targeting different patches. See also figs 5 and 9 in Schive et al. (2010).

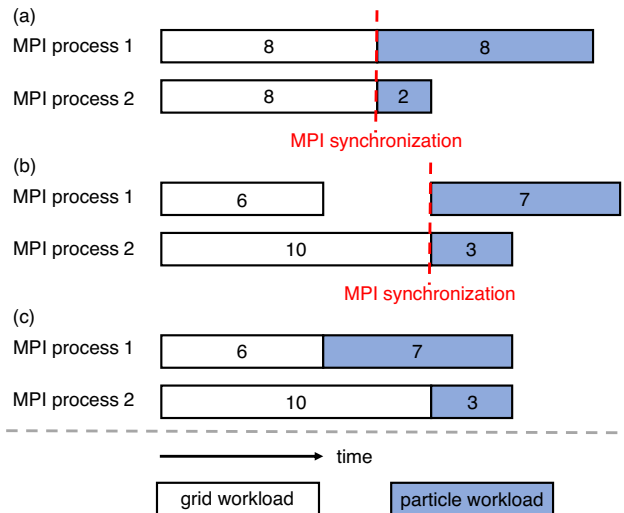
Since some PDE solvers use patch groups (a patch group is defined as the set of eight sibling patches that have the same parent in the octree data structure) instead of patches as the basic unit (e.g. hydrodynamic solvers), we require the eight patches belonging to the same patch group to be located in the same MPI process. However, we do not require parent and child patches to be located in the same process, which allows patches on different levels to be distributed on different MPI processes in a completely independent fashion. Accordingly, we can adopt different and independent Hilbert curves on different levels and then use the Hilbert curve of a given AMR level to assign unique 1D indices for all patches on that level. By doing so, the load balancing can be achieved on a level-by-level basis. This feature is particularly important for the adaptive time-step integration, since in that case patches on different levels in general cannot be evolved simultaneously. In addition, the Hilbert curve mapping between 3D and 1D space preserves locality, meaning that, in general, patches close along a 1D curve are also close in 3D space. This property reduces MPI communication and thus is very important for achieving higher parallel scalability.

We do not duplicate the entire AMR hierarchy on each MPI process. Instead, for each process, we only allocate its ‘real’ patches that store the physical data (i.e. fluid variables and particles) to be updated by this process, and the ‘buffer’ patches necessary for filling the ghost zones of these real patches by either direct copy or interpolation. Specifically, for each real patch, we examine its sibling, parent, and parent–sibling patches (i.e. sibling patches of a parent patch), and allocate the corresponding buffer patches locally if any of these patches exist in the complete AMR hierarchy but do not live on the same MPI process as the targeted real patch. These buffer patches allow each process to correctly identify the nearby AMR structure (e.g. finding the coarse-fine interfaces) and to retain a copy of physical data received from other MPI processes for filling the ghost zones of real patches. By avoiding duplication of the entire AMR hierarchy, we can reduce CPU memory consumption and

improve parallel scalability, particularly when running extremely large parallel simulations.

For the simulations without particles, it is relatively straightforward to achieve good load balancing since the computational workloads of different patches are roughly the same. However, it becomes much more challenging when particles are involved, not only because different patches now may have different workloads due to the different numbers of particles associated with them, but because there may be unavoidable MPI synchronization between grid and particle routines. See Fig. 4 for an illustration. To partially alleviate this problem, we assign a relative weight to each particle,  $W_{\text{par}}$ , which is normalized to the weight per cell, when estimating the total weight of each patch for load balancing. Moreover, we arrange the order of various grid routines (e.g. hydrodynamic and gravity solvers), particle routines (e.g. mass deposition and particle update), and MPI communication (e.g. filling the grid data of buffer patches and transferring particles between neighbouring patches) carefully so as to minimize the MPI synchronization between grid and particle routines. We typically adopt  $W_{\text{par}} = 1.0\text{--}2.0$ . The optimal value depends on the adopted physics, for example, whether or not the radiative library GRACKLE is included. See Sections 4.3.3 and 4.4.3 for some comparisons between the simulation performance with and without applying these optimizations. Also note that chemistry and radiative cooling/heating (e.g. when using GRACKLE) may also lead to widely different costs from cell to cell and deteriorate load balancing.

Note that, when estimating the load-balancing weight of a non-leaf patch, it is necessary to take into account all particles occupying the same space as this targeted patch, even though these particles live on leaf patches. It is because we still need to temporarily transfer the masses and positions of particles from leaf to non-leaf patches when calculating the total mass density on a lower level. Ignoring the weights of higher level particles may result in an undesirable situation where a large fraction of particles are temporarily transferred



**Figure 4.** Illustration of load balancing in the simulations with particles. Open and filled rectangles represent the computational workloads associated with grids and particles, respectively. Case (a) does not take into account particle weights, and thus the total workloads of two MPI processes are imbalanced. Case (b) considers both grid and particle weights. However, there is an undesirable MPI synchronization between grid and particle routines, leading to even worse performance compared to case (a). Case (c) further eliminates this MPI synchronization and gives the optimal performance.

to a small fraction of MPI processes when depositing particles onto lower level grids, which could lead to severe load imbalance and memory exhaustion. See also Section 5 for a possible alternative solution.

Also note that it may be impossible to completely eliminate the MPI communication between grid and particle routines. For example, after updating particle positions, one must transfer the particles moving across patch boundaries to neighboring patches before invoking the Poisson solver. We will investigate other more advanced approaches, for example, using MPI non-blocking communication (e.g. ENZO; Bryan et al. 2014) or task-based parallelism model (e.g. ATHENA+ + <sup>7</sup>) in the future.

As a side note, we reuse large MPI send and receive buffers when applicable. In other words, we do not deallocate these buffers after performing an MPI communication. Instead, we reuse them for the next communication provided that their sizes are large enough. We find that this approach improves the achieved bandwidth in some circumstances.

### 3.4 Memory management

AMR simulations require allocating and deallocating grids and rebuilding the data structure frequently, which can easily lead to memory fragmentation that deteriorates performance and exhausts memory. GAMER-2 supports using a local memory pool for each MPI process to solve this problem. However, unlike conventional methods, it does not require pre-allocating this memory pool. Instead, it relies on the *reuse* of allocated memory. Specifically, when removing patches from the AMR hierarchy after the derefinement operation, we simply mark these patches as ‘inactive’ but do not actually free the memory associated with them. These inactive patches can then be ‘reactivated’ later to serve as new patches, which can be

either real or buffer patches, after the refinement operation. In comparison to the method of pre-allocating a large memory pool (which is also supported in GAMER-2), this approach is more flexible since it eliminates the need for users to guess in advance the maximum number of patches required. On the contrary, the size of the memory pool will be adjusted automatically to fit the requirement.

We also allocate a separate memory pool for each MPI process to store all particle attributes, and have leaf patches only record the particle indices associated with them. By doing so, when particles moving between patches living on the same MPI process, we only have to update the particle index list of relevant leaf patches and do not have to touch the particle memory pool. In addition, for particles travelling to different host processes, we simply mark these particles as ‘inactive’ in their original host processes but do not actually free the memory associated with them, which is similar to the method adopted in the grid memory pool. These inactive particles can then be reactivated later to represent, for example, particles migrating from other processes or new particles triggered by star formation. Moreover, to further minimize memory management due to particle movement, we pre-allocate small memory buffers (10 per cent of the total number of particles tracked by a given MPI process by default) for both the particle memory pool and the particle index list of each leaf patch. For example, for a process owning 1000 particles, we can pre-allocate a particle memory pool with the size of 1100 particles to reduce frequent memory reallocation caused by the migration of a small number of particles. Also note that the size of the particle memory pool will be adjusted automatically based on the number of particles currently hosted by each process, and thus users do not have to guess in advance the maximum number of particles per process during the entire simulation.

The ghost zones associated with each patch can lead to severe memory overhead. For example, the CTU scheme with PPM reconstruction needs three ghost zones. Accordingly, for a patch consisting of  $8^3$  cells, the total memory consumption including the ghost zones is  $\sim 5.4$  times larger than that without the ghost zones. To solve this problem, GAMER-2 does not permanently allocate the ghost zones of all patches. Instead, it only temporarily allocates the ghost zones for the patches being transferred to GPUs, the number of which can be fixed (typically a few thousand) and is independent of the simulation scale. This approach also solves the issue of the relatively small GPU memory since the code does not need to send all patches to GPUs at a single time.

## 4 CODE TESTS

In this section, we conduct various tests to demonstrate the performance and accuracy of GAMER-2. Since the numerical algorithms of GAMER-2 (see Section 2) have been tested extensively by many astrophysical codes (e.g. Fryxell et al. 2000; Stone et al. 2008; Bryan et al. 2014), we do not repeat the analysis of these standard numerical tests here (see however, Schive et al. 2010; Schive et al. 2012 for the standard tests conducted previously). Instead, we directly compare GAMER-2 with two widely adopted codes, namely, FLASH (Fryxell et al. 2000) and ENZO (Bryan et al. 2014), based on more complicated and realistic astrophysical applications, which arguably provides much more direct and convincing results.

This section is organized as follows. We first measure the performance of individual GPU solvers (Section 4.1) and the weak scaling with and without AMR in a 3D Kelvin–Helmholtz (KH) instability test (Section 4.2). We then compare the accuracy and strong scaling performance of GAMER-2 with FLASH in binary cluster merger

<sup>7</sup><http://princetonuniversity.github.io/athena>

simulations (Section 4.3), and with ENZO in isolated disc galaxy simulations (Section 4.4).

#### 4.1 Performance of GPU solvers

The key feature of GAMER-2 is GPU acceleration. So we first measure the performance of individual GPU solvers, which is largely independent of the adopted test problems. This performance includes the time of transferring data between CPU and GPU but excludes the time of all other CPU operations related to AMR and MPI communication. It thus represents the optimal performance of GAMER-2, which can only be approached in certain particular cases (e.g. large uniform-grid simulations) according to Amdahl's law (Amdahl 1967). This information can be useful for assessing the performance deterioration in more complicated simulations.

Fig. 5 shows the performance in cell updates per second for individual GPU solvers and their CPU counterparts using exactly the same numerical schemes. The performance is measured on a Blue Waters XK node with an NVIDIA Tesla K20X GPU and a 16-core AMD Opteron 6276 CPU. We also measure the performance on an NVIDIA Tesla P100-PCIe GPU. The key findings can be summarized as follows.

(i) The K20X GPU is measured to be 27 and 8 times faster than the 16-core CPU for the hydrodynamic and Poisson solvers, respectively. Furthermore, the P100 GPU is measured to be 2.3–3.0 times faster than the K20X GPU, achieving  $\sim 2 \times 10^8$  cells  $s^{-1}$  in all three solvers.

(ii) The GPU performance already begins to saturate when updating only  $\sim 10^6$  cells at a time. It is typically much smaller than the total number of cells computed by each MPI process in a real astrophysical application, suggesting that we only need to transfer a small fraction of cells to GPU at a time to fully exploit the GPU acceleration. This important property allows for (1) efficient overlapping between CPU and GPU computations (see Section 3.2) and (2) efficient overlapping between CPU–GPU communication and CPU/GPU computations.

(iii) The GPU performance increases by a factor of 2 by taking advantage of the asynchronous data transfer between CPU and GPU with CUDA streams. The performance saturates when using more than  $\sim 10$ –20 streams.

(iv) The GPU hydrodynamic and Poisson solvers exhibit comparable performance ( $\sim 5 \times 10^7$ – $2 \times 10^8$  cell updates per second).

(v) The performance of CPU hydrodynamic solvers seems to be relatively low compared to other codes, for example, ATHENA and RAMSES, partially because of the different CPU and the smaller patch size ( $8^3$  cells) adopted in this work. More quantitative comparisons and further optimizations will be investigated.

The small patch size ( $8^3$  cells) adopted throughout this paper results in a considerable computational overhead associated with the ghost zones of each patch (although it allows for more flexible grid refinement and efficient load balancing). Increasing the patch size to  $16^3$  cells is found to improve the performance of the hydrodynamic GPU solvers by  $\sim 20$  per cent. In addition, compared to the single-precision performance, adopting double precision on the P100 GPU is measured to be 2.1 and 2.6 times slower for the hydrodynamic and Poisson solvers, respectively. The Poisson solver shows greater performance degradation due to the larger number of iterations required to converge to the machine precision and the limited amount of GPU shared memory.

#### 4.2 Weak scaling

We now measure the overall performance of GAMER-2, starting by showing the weak scaling from a 3D KH instability test. The weak scaling is useful for demonstrating algorithmic scalability and is particularly important for uniform-grid simulations, for example, in the study of non-gravitating turbulence.

The simulation setup is as follows. Each node computes a periodic domain of unit length on a side. We set the gas density  $\rho = 2$  and velocity  $v_x = 0.5$  in the regions  $z < 0.25$  and  $0.5 < z < 0.75$  and have  $\rho = 1$  and  $v_x = -0.5$  otherwise, leading to four surfaces of contact discontinuities per node. Velocity perturbations with an amplitude of  $10^{-2}$  and a white noise spectrum are added along all three directions to trigger the instabilities and make it a 3D test. The gas has a uniform pressure  $P = 2.5$  and an adiabatic index  $\gamma = 1.4$ . All simulations are conducted from  $t = 0$  to 0.5. We adopt the CTU scheme with PPM reconstruction and Roe's solver.

We measure the performance of both uniform-grid and AMR simulations using 1–4096 nodes on Blue Waters, where we use one MPI process and 16 OpenMP threads per node. For the uniform-grid test, each node computes a  $640^3$  grid, resulting in an overall resolution as high as  $10\,240^3$  with 4096 nodes. For the AMR test, each node computes a  $128^3$  root grid with three refinement levels, where we adopt flow vorticity as the refinement criterion and enable the adaptive time-stepping. Fig. 6 shows a density slice perpendicular to the shear-flow plane at  $t = 0.5$  in a single node, with the grid patches overlaid.

Fig. 7 shows the weak scaling of the KH instability test, and Fig. 8 records the corresponding performance metrics, including the number of cell updates per second per node, total number of cells, parallel efficiency, and the fraction of time spent on MPI communication as a function of the number of nodes. The parallel efficiency of weak scaling is defined as

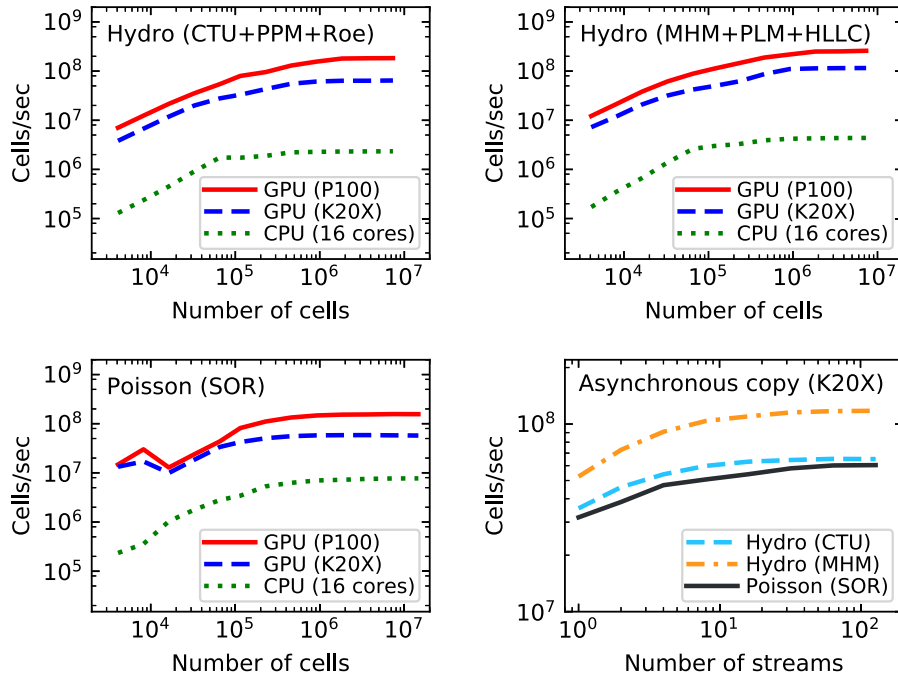
$$P_{\text{weak}}(N_{\text{node}}) = \frac{T(1)}{T(N_{\text{node}})}, \quad (5)$$

where  $T(N_{\text{node}})$  is the simulation wall time using  $N_{\text{node}}$  nodes. Both the uniform-grid and AMR tests exhibit reasonably good scalability for  $N_{\text{node}} = 1$ –4096. Note that, thanks to the hybrid MPI/OpenMP/GPU parallelization, we are able to fully exploit both 4096 GPUs and 65–536 CPU cores simultaneously, and achieve a peak performance of  $8.3 \times 10^{10}$  cells  $s^{-1}$  and  $P_{\text{weak}}(4096) = 74$  per cent in the uniform-grid test and  $4.6 \times 10^{10}$  cells  $s^{-1}$  and  $P_{\text{weak}}(4096) = 58$  per cent in the AMR test. A noticeably higher fraction of time in MPI is found in the case with AMR (see the lower right panel of Fig. 8), thus partially explaining the relatively lower parallel efficiency achieved. The total CPU memory consumption for  $N_{\text{node}} = 4096$  is  $\sim 53$  and  $\sim 74$  TB for the uniform-grid and AMR tests, respectively.

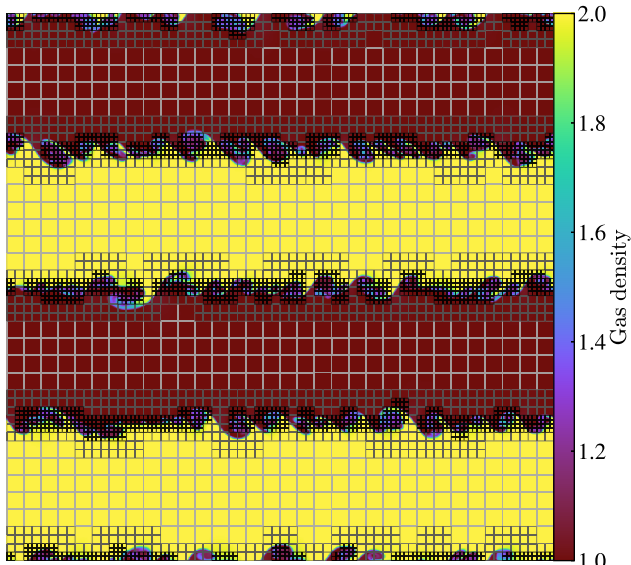
#### 4.3 Galaxy cluster merger: GAMER-2 versus FLASH

Mergers play an important role in the formation of galaxy clusters, driving shocks, and turbulence that heat up the intracluster medium (e.g. Gaspari & Churazov 2013; Banerjee & Sharma 2014; Lau et al. 2017), provide additional support against gravity (e.g. Nagai, Vikhlinin & Kravtsov 2007; Khatri & Gaspari 2016), and accelerate relativistic particles emitting radio waves (e.g. Brunetti & Lazarian 2007; Eckert et al. 2017). Numerical simulations of the galaxy cluster merger are challenging partially due to the large dynamic range required to both capture the large-scale effects of the cluster merger and to resolve the properties of turbulence down to at least



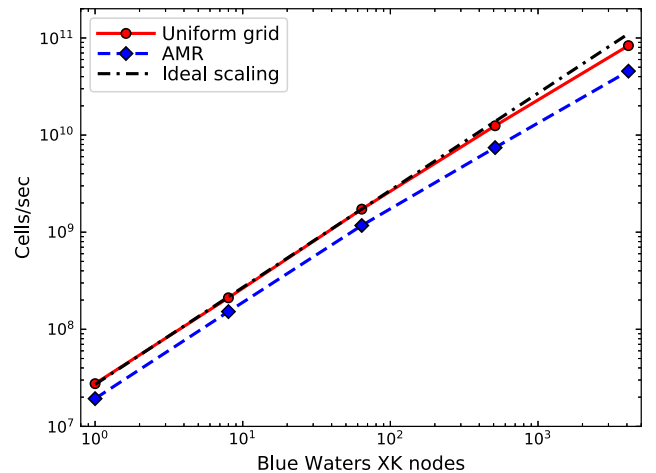


**Figure 5.** Performance in cell updates per second for individual GPU solvers and their CPU counterparts. The first three panels show the performance of the hydrodynamic solver with the CTU scheme, PPM reconstruction, and Roe’s solver (upper left), the hydrodynamic solver with the MHM scheme, PLM reconstruction, and HLLC solver (upper right), and the Poisson solver with the SOR scheme (lower left) as a function of the number of cells updated at a time. The GPU performance is measured on both NVIDIA Tesla K20X and P100-PCIe GPUs, and the CPU performance is measured on a 16-core AMD Opteron 6276 processor (using all 16 cores). The P100 GPU achieves a performance of  $\sim 2 \times 10^8$  cells  $s^{-1}$  in all three solvers. We also measure the K20X GPU performance as a function of the number of CUDA streams (lower right), demonstrating a factor of 2 speedup when utilizing the asynchronous data transfer between CPU and GPU.

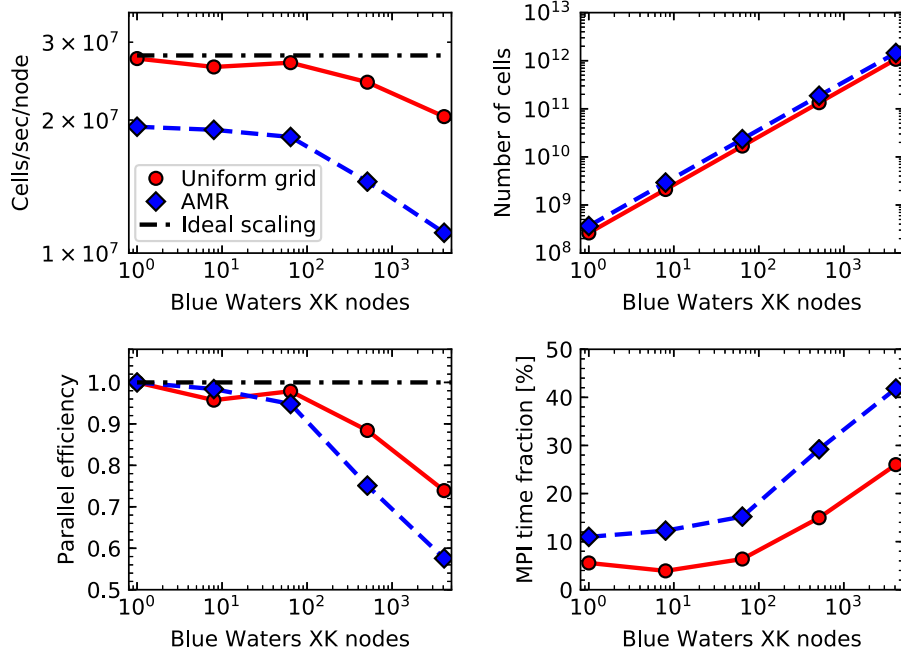


**Figure 6.** Density slice and grid patches in a single node in the KH instability weak scaling test at  $t = 0.5$ .

the kpc scale. The lack of sufficient resolution will produce non-negligible numerical viscosity that alters the turbulence cascade and biases the mass estimates of clusters.



**Figure 7.** Weak scaling for the KH instability test, showing the total number of cell updates per second as a function of the number of XK nodes on Blue Waters. Each XK node is composed of one NVIDIA Tesla K20X GPU and one 16-core AMD Opteron 6276 CPU. We measure the performance of both uniform-grid ( $640^3$  cells per node; solid line) and AMR ( $128^3$  root grid per node with three refinement levels; dashed line) simulations. Note that for the uniform-grid test with 4096 nodes, we achieve a resolution of  $10\,240^3$  cells and an overall performance of  $8.3 \times 10^{10}$  cells  $s^{-1}$ . The dashed–dotted line represents the ideal scaling. See Fig. 8 for the detailed performance metrics of this test.



**Figure 8.** Performance metrics of the KH instability weak scaling test. This is complementary to and uses the same symbols as Fig. 7. Different panels show the number of cell updates per second per node (upper left), total number of cells (upper right), parallel efficiency (lower left), and fraction of time spent on MPI communication (lower right).

In this section, we conduct cluster merger simulations with both GAMER-2 and FLASH, and demonstrate that the physical results produced by the two codes agree very well with each other, and GAMER-2 outperforms FLASH by almost two orders of magnitude. We first describe the simulation setup, with particular emphasis on the similarities and differences of the two codes. We then check the consistency of the physical results, and finally compare the strong scaling performance.

#### 4.3.1 Simulation setup

We simulate a head-on merger of two equal-mass clusters for 10 Gyr. The simulation setup is identical to that of ZuHone (2011). The two clusters have initial separation of 3.1 Mpc and initial relative velocity of  $1352 \text{ km s}^{-1}$ . The simulation domain is cubic with a length  $L = 14.26 \text{ Mpc}$ . Each cluster has a virial mass  $M_{200} = 6 \times 10^{14} M_{\odot}$  and a gas mass fraction  $f_g = 0.1056$ . We adopt an Navarro–Frenk–White (NFW; Navarro, Frenk & White 1996) profile with a concentration parameter  $c = 4.5$  as the total density profile, and calculate the gas density profile under the assumptions of spherical symmetry, hydrostatic equilibrium, and a power-law entropy profile. The DM particle velocities are realized by sampling the velocity distribution function directly via solving the Eddington formula (Eddington 1916) instead of assuming a Maxwellian distribution. See ZuHone (2011) for the detailed implementation.

Table 1 summarizes the similarities and differences of the numerical setup adopted by GAMER-2 and FLASH in these comparison simulations. Here, we elaborate on the major differences.

(i) GAMER-2 adopts an adaptive time-step integration where higher levels can have smaller time-steps. Moreover, the time-step ratio between two adjacent levels does not need to be a constant (see Section 2.1). By contrast, FLASH adopts a shared time-step integration where all levels share the same time-step.

(ii) GAMER-2 adopts the original CTU scheme (Colella 1990; Stone et al. 2008) requiring six Riemann solvers per cell per time-step and  $C_{\text{CFL}} \leq 0.5$ . In comparison, FLASH adopts a revised CTU scheme (Lee 2013) requiring only three Riemann solvers per cell per time-step and allowing for a larger CFL number  $C_{\text{CFL}} \leq 1.0$ , which in principle should be significantly faster than GAMER-2. We, however, adopt  $C_{\text{CFL}} = 0.6$  for FLASH since it crashed with  $C_{\text{CFL}} > 0.6$ .

(iii) For the Poisson solver, GAMER-2 uses the SOR scheme and adds five additional buffer zones around each patch to make the potential smoother across the patch boundaries (see Fig. 2). In comparison, FLASH adopts a finite-volume multigrid scheme that aims to minimize the global residual (Ricker 2008), which, in general, should be more accurate but also more computationally expensive.

(iv) Unlike FLASH, GAMER-2 does not implement explicit grid derefinement criteria (see Section 2.1). However, we have verified that the grid distribution of the two codes are very similar in this test (the difference in the numbers of maximum level cells is less than  $\sim 20$  per cent, e.g. see Table 2).

(v) For the floating-point accuracy, GAMER-2 uses single precision while FLASH uses double precision, because single precision is not officially supported in FLASH. This discrepancy makes our performance comparison in favor of GAMER-2, which could be unfair in this sense. However, we also demonstrate in the next section that the physical results obtained by the two codes are very consistent, suggesting that double precision may not be necessary for this test.

#### 4.3.2 Accuracy comparison

Fig. 9 shows the slices of gas temperature through the cluster centre at  $t = 0.0, 3.3, 6.6$ , and  $10.0 \text{ Gyr}$  obtained by GAMER-2 and FLASH. The first core passage occurs at  $t \sim 1.5 \text{ Gyr}$ , after which the oscillating DM cores continue driving shocks and turbulence that heat up the intracluster medium, eventually forming a high temperature

**Table 1.** Comparison of the numerical setup between GAMER-2 and FLASH in the galaxy cluster merger simulations.

	GAMER-2	FLASH
AMR implementation	Fixed patch size of $8^3$ cells, no permanent allocation for patch ghost zones	Fixed patch size of $8^3$ cells, patch ghost zones are allocated permanently <sup>a</sup>
Grid resolution	Root grid $128^3$ , four refinement levels, maximum resolution $\Delta h = 7.0$ kpc	Same as GAMER-2
Particle resolution	Number of particles $N_p = 5 \times 10^6$ (for each cluster) and mass resolution $m_p = 1.4 \times 10^8 M_\odot$	Same as GAMER-2
Fluid solver	CTU scheme with six Riemann solvers per cell, PPM reconstruction, Roe's solver, and van Leer slope limiter	Same as GAMER-2 except for a revised CTU scheme requiring only three Riemann solvers per cell <sup>b</sup>
Poisson solver	SOR	multigrid <sup>c</sup>
Particle solver	CIC interpolation and KDK particle update	CIC interpolation and variable time-step Leapfrog particle update
Boundary condition	Fluid solver: outflow Poisson solver: isolated	Same as GAMER-2
Refinement	(1) Löhner's error estimator on gas density, pressure, and temperature with a refinement threshold of 0.8 and a minimum gas density threshold of $10^{-28} \text{ g cm}^{-3}$ (2) Maximum number of particles in a patch: 100	Same as GAMER-2
Derefinement	No explicit derefinement criteria	(1) Löhner's error estimator with a derefinement threshold of 0.2 (2) Minimum number of particles in a patch: 12
Time-step	$C_{\text{par}} = 0.8$ , $C_{\text{CFL}} = 0.5$	$C_{\text{par}} = 0.8$ , $C_{\text{CFL}} = 0.6$
Parallelization	Hybrid MPI/OpenMP/GPU	MPI and CPU-only
Load balancing	Hilbert space-filling curve	Morton space-filling curve
Time integration	Adaptive time-step	Shared time-step
Floating-point format	Single precision	Double precision

Notes. <sup>a</sup>FLASH supports the 'NO-PERMANENT-GUARDCELLS' (npg) mode, which however does not work well with active particles. We have therefore disabled this functionality.

<sup>b</sup>Lee (2013).

<sup>c</sup>Ricker (2008).

<sup>d</sup>Single precision is not officially supported in FLASH. But we have demonstrated that this discrepancy does not affect the physical results here. See discussions in Sections 4.3.2 and 4.3.3.

**Table 2.** Comparison of the volume-filling fractions on the refinement levels between GAMER-2 and FLASH in the lower resolution galaxy cluster merger simulations at  $t = 5$  Gyr.

Level	$\Delta h$ /kpc	Filling fraction		Number of cells	
		GAMER-2 (per cent)	FLASH (per cent)	GAMER-2	FLASH
1	55.70	56.45	38.65	$9.5 \times 10^6$	$6.5 \times 10^6$
2	27.85	31.62	23.08	$4.2 \times 10^7$	$3.1 \times 10^7$
3	13.93	7.13	6.48	$7.7 \times 10^7$	$7.0 \times 10^7$
4	6.96	0.86	0.75	$7.4 \times 10^7$	$6.4 \times 10^7$

and constant entropy core. The results of GAMER-2 (upper panels) and FLASH (lower panels) are verified to be very consistent with each other.

Fig. 10 shows the radial profiles of the electron number density  $n_e$ , gas entropy  $S$ , gas temperature  $T$ , and DM mass density  $\rho_{\text{DM}}$  at  $t = 10$  Gyr, where gas is assumed to be fully ionized and the gas specific entropy is defined as  $S \equiv k_B T n_e^{-2/3}$  with  $k_B$  the Boltzmann constant. A constant entropy core can be clearly identified within  $\sim 300$  kpc. Most strikingly, the results obtained by GAMER-2 and FLASH are found to be literally indistinguishable. It demonstrates the consistent numerical setup we adopt for this code comparison experiment, including, for example, the initial condition, boundary conditions, spatial and temporal resolution, AMR implementation, and grid refinement criteria. It also indicates that the different numerical schemes between the two codes described in Section 4.3.1, for example, the time integration, fluid and Poisson solvers, and floating-point accuracy, do not have a significant impact here.

#### 4.3.3 Performance comparison

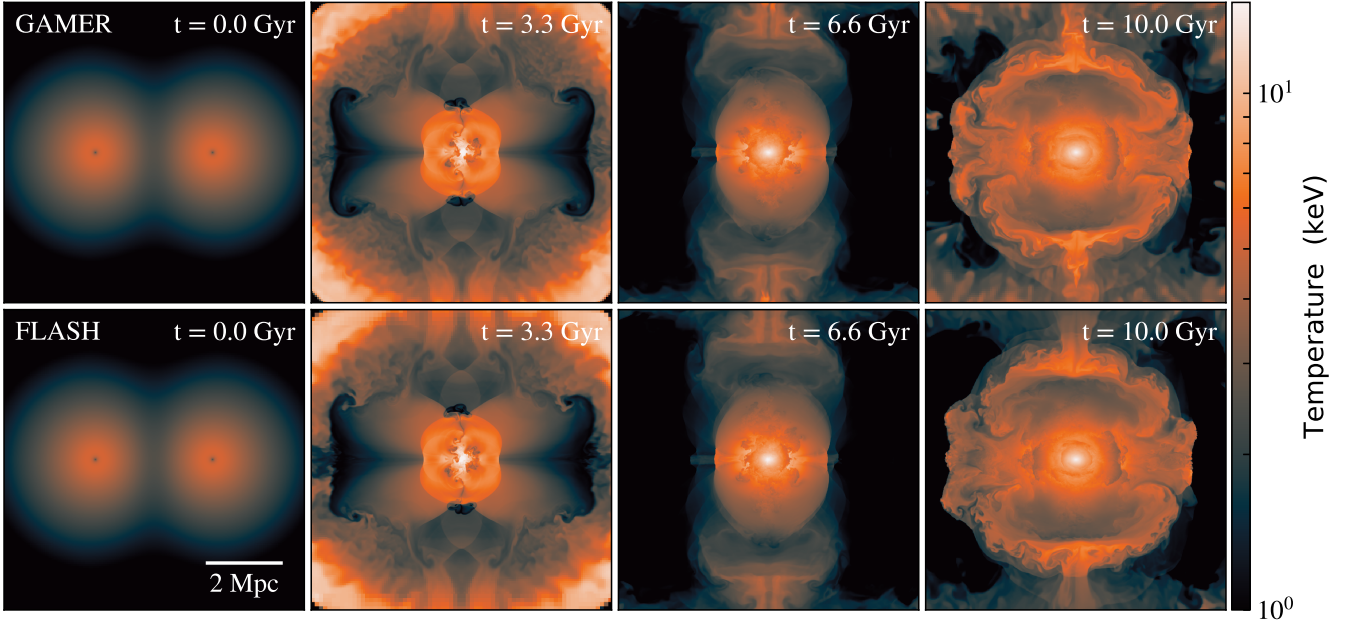
Based on the very consistent physical results between GAMER-2 and FLASH, as shown in Figs 9 and 10, here we compare their strong scaling performance on Blue Waters. In order to have a fair comparison between the codes with and without GPU acceleration, we run GAMER-2 on the XK nodes while FLASH on the XE nodes: each XK node is composed of one GPU (NVIDIA Tesla K20X) and one 16-core CPU (AMD Opteron 6276), and each XE node is composed of two 16-core CPUs. In addition, since there are two NUMA domains per XK node, each of which shares an eight MB

L3 cache, for GAMER-2, we launch two MPI processes per node and  $7^8$  OpenMP threads per MPI process in order to improve memory affinity by having all threads running in the same NUMA domain. The two MPI processes running on the same node shares the same GPU by taking advantage of the CUDA MPS. For FLASH, we launch 32 MPI processes per XE node.

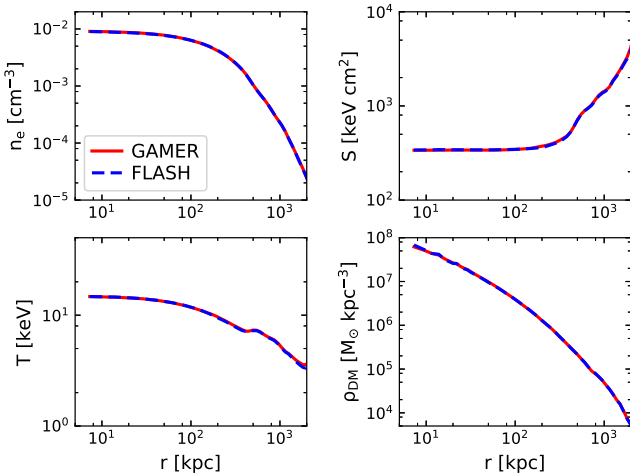
Fig. 11 shows the strong scaling of the cluster merger simulations. We first conduct the lower resolution tests ( $\Delta h \sim 7.0$  kpc, the same as that adopted in Section 4.3.2) for both GAMER-2 and FLASH using up to 256 nodes. Note that the minimum number of nodes adopted in FLASH is 16 instead of 1 due to its much larger memory consumption, which will be discussed at the end of this section. We measure the performance in  $t = 4\text{--}6$  Gyr, during which there are  $\sim 2.1 \times 10^8$  cells in total and  $\sim 7.1 \times 10^7$  cells on the maximum refinement level. When using the same number of nodes, the speedup of GAMER-2 over FLASH is measured to be 78–101 and 64–83 in terms of total wall time and cell updates per second, respectively. For example, for  $N_{\text{node}} = 16$ , GAMER-2 achieves  $8.3 \times 10^6$  cell updates per second per XK node, and FLASH achieves  $1.0 \times 10^5$  cell updates per second per XE node (corresponding to  $3.1 \times 10^3$  cell updates per second per CPU core). Moreover, this speedup ratio only drops by  $\sim 23$  per cent when increasing  $N_{\text{node}}$  from 16 to 128, and the performance of both codes starts to decline when  $N_{\text{node}} > 128$ . It suggests that GAMER-2 and FLASH exhibit similar parallel scalability, despite the fact that the computational time of GAMER-2 has been greatly reduced with GPUs.

The different speedups measured from the total wall time and cell updates per second are due to several factors. The cell updates per second depends mostly on the performance of individual PDE solvers, which itself is related to the CPU/GPU performance, the adopted floating-point accuracy, and the numerical schemes adopted in the PDE solvers. In comparison, the speedup in term of total wall time is arguably more comprehensive, since it takes into account not only the performance of PDE solvers but also many other factors, such as the time integration scheme, the evolution time-step, and the number of cells on each level. See Section 4.3.1, especially Table 1, for the summary of different numerical setup between GAMER-2 and FLASH.

<sup>8</sup>We use seven instead of eight OpenMP threads per MPI process since using eight threads somehow binds two threads to the same CPU core.



**Figure 9.** Slices of gas temperature through the cluster centre at four different epochs in the galaxy cluster merger simulations. Each panel is 8 Mpc on a side. The results obtained by GAMER-2 (upper panels) and FLASH (lower panels) are found to be in very good agreement with each other. See Fig. 10 for more quantitative comparisons of the radial profiles.



**Figure 10.** Radial profiles at  $t = 10$  Gyr in the galaxy cluster merger simulations. Different panels show the electron number density (upper left), gas entropy (upper right), gas temperature (lower left), and DM mass density (lower right). A remarkable agreement is observed between the results of GAMER-2 (solid lines) and FLASH (dashed lines).

In short, FLASH uses a more efficient fluid solver and a more accurate but also more computationally expensive Poisson solver. The CFL safety factor adopted for FLASH ( $C_{\text{CFL}} = 0.6$ ) is larger than GAMER-2 ( $C_{\text{CFL}} = 0.5$ ), but the average time-steps on the maximum level are found to be very similar due to the same time-step criterion for updating particles. More precisely, the minimum time-steps in both codes are set by the fastest-moving particles on the maximum refinement level, which always move faster than the characteristic hydrodynamic speed which sets the hydrodynamic time-step. GAMER-2 is found to allocate  $\sim 30$ – $50$  per cent more cells on lower levels and  $\sim 10$ – $20$  per cent more cells on higher levels than FLASH (e.g. see Table 2), mainly because GAMER-2 tends to pre-allocate

patches earlier than FLASH due to the implementation of large flag buffers (see Section 2.1). This issue of overallocation in GAMER-2 is relatively more serious on lower levels, since there are fewer patches on these levels. However, it is not a serious problem since GAMER-2 adopts an adaptive time-step integration that allows patches on lower levels to have larger time-steps. It is also the main reason why the speedup in terms of total wall time is  $\sim 20$  per cent higher than that in terms of cell updates per second. Last but not least, we remind the reader that we adopt single precision for GAMER-2 and double precision for FLASH in this test. In principle, using single precision for FLASH could improve performance, however single-precision calculations are not currently supported by FLASH.

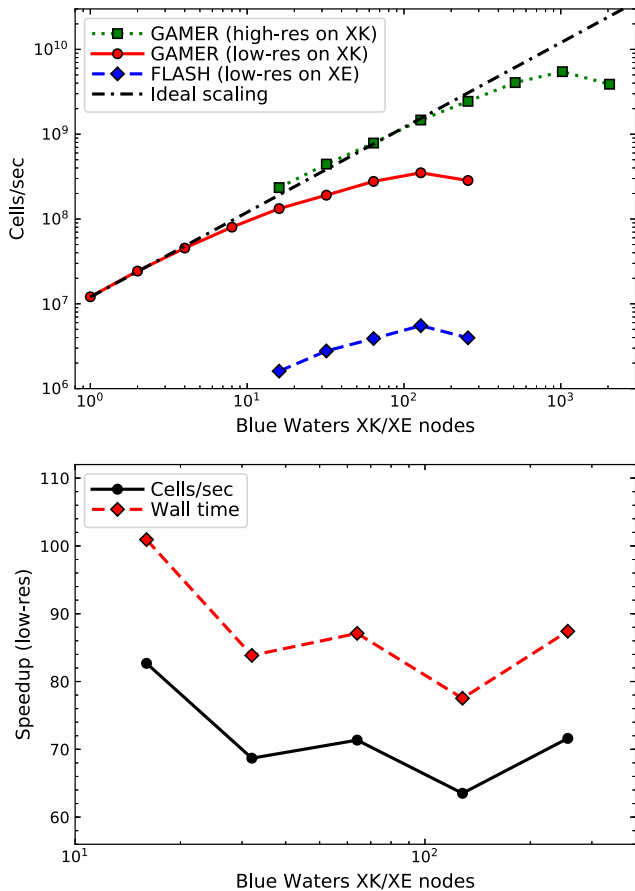
To test the scalability of GAMER-2 further, we also conduct higher resolution runs ( $\Delta h \sim 0.87$  kpc, eight times higher than the lower resolution counterpart) using 16–2048 XK nodes. We measure the performance from  $t = 4.0$  to  $4.5$  Gyr, during which there are  $\sim 8.8 \times 10^9$  cells in total and  $\sim 7.1 \times 10^9$  cells on the maximum refinement level. The total number of particles is fixed to  $10^7$ , the same as the lower resolution test. The total memory consumption for  $N_{\text{node}} = 2048$  at  $t = 4.5$  Gyr is  $\sim 2.2$  TB. Fig. 11 (upper panel, dotted line) shows the strong scaling of this higher resolution test, exhibiting a much better scaling than its lower resolution counterpart. It demonstrates that GAMER-2 can scale well to thousands of GPUs in a realistic astrophysical application. It also indicates that the parallel scalability may be sensitive to the load imbalance resulting from particles, which will be investigated shortly in this section.

Before giving a more detailed analysis of strong scaling, we first introduce two quantities useful for quantifying the parallel scalability. The ‘parallel efficiency’ of strong scaling is defined as

$$P_{\text{strong}}(N_{\text{node}}) = \frac{T(N_{\text{node,ref}})N_{\text{node,ref}}}{T(N_{\text{node}})N_{\text{node}}}, \quad (6)$$

where  $T(N_{\text{node}})$  is the simulation wall time using  $N_{\text{node}}$  nodes. Generally speaking, the parallel efficiency also depends on the refer-





**Figure 11.** Strong scaling of the galaxy cluster merger simulations run with GAMER-2 and FLASH on Blue Waters. GAMER-2 runs on the XK nodes composed of one GPU (NVIDIA Tesla K20X) and one 16-core CPU (AMD Opteron 6276) per node, while FLASH runs on the XE nodes composed of two 16-core CPUs per node. The upper panel shows the total number of cell updates per second for (i) the lower resolution tests ( $\Delta h \sim 7.0$  kpc) of GAMER-2 (solid line) and FLASH (dashed line) using up to 256 nodes, and (ii) the higher resolution test of GAMER-2 ( $\Delta h \sim 0.87$  kpc, dotted line) using 16–2048 nodes. Note that the minimum number of nodes adopted in FLASH is 16 instead of 1 due to its much larger memory consumption (see the text for details). The dashed–dotted line represents the ideal scaling. The lower panel shows the speedup of GAMER-2 over FLASH in the lower resolution tests in terms of either cell updates per second (solid line) or total wall time (dashed line). Both cases reveal nearly two orders of magnitude speedup. See Fig. 12 for the detailed performance metrics of this test. Note that GAMER-2 uses single precision but FLASH uses double-precision arithmetic.

ence performance  $T(N_{\text{node, ref}})$ . For a proper comparison, we adopt  $N_{\text{node, ref}} = 16$  for the lower resolution tests of both codes, even though the minimum  $N_{\text{node}}$  in the GAMER-2 runs is 1 instead of 16. For the higher resolution test, we adopt  $N_{\text{node}} = 16$ . We also introduce another quantity to quantify the scalability, namely, the ‘doubling efficiency’, which is defined as

$$D_{\text{strong}}(N_{\text{node}}) = \frac{T(N_{\text{node}}/2)}{T(N_{\text{node}})} - 1, \quad (7)$$

This quantity corresponds to the performance gain when doubling the computational resource from  $N_{\text{node}}/2$  to  $N_{\text{node}}$ , which is arguably more intuitive than the conventional parallel efficiency and has the advantage of being independent of the minimum  $N_{\text{node}}$  adopted. For example, one may have  $D_{\text{strong}}(N_{\text{node}}) = 0.6$  for  $N_{\text{node}} = 2$ –2048, which suggests reasonable scalability since a perfor-

mance speedup of 1.6 is always obtained when doubling the number of nodes. However, the corresponding parallel efficiency is as low as  $P_{\text{strong}}(2048) = (1.6/2)^{11} \sim 9$  per cent, which could be misleading.

Fig. 12 shows the performance metrics of the cluster merger simulations, including the total wall time, maximum CPU memory consumption per MPI process, parallel efficiency, and doubling efficiency as a function of the number of XK and XE nodes for GAMER-2 and FLASH, respectively. Most importantly, as demonstrated by both the parallel and doubling efficiencies, the two codes exhibit similar parallel scalability, especially for  $N_{\text{node}} > 32$ . It is consistent with the finding of an almost constant speedup of GAMER-2 over FLASH for  $N_{\text{node}} > 32$ , as shown in the lower panel of Fig. 11.

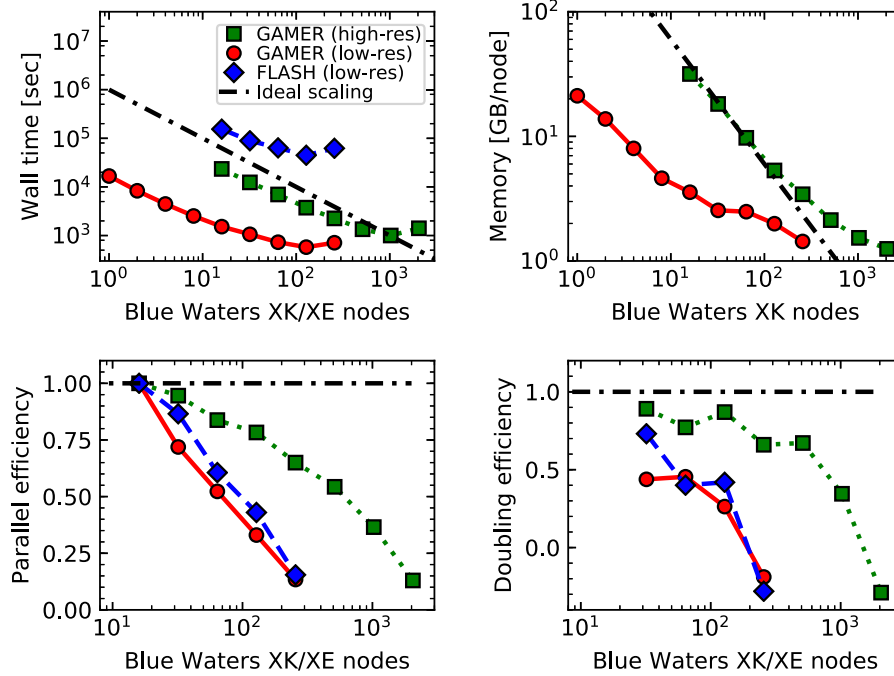
In addition, the higher resolution test of GAMER-2 shows a better scaling than its lower resolution counterpart. The scalability is reasonably good in a large range of  $N_{\text{node}}$  from 16 to 1024. The parallel efficiency is measured to be 78 per cent for  $N_{\text{node}} = 128$  and 37 per cent for  $N_{\text{node}} = 1024$ , and the doubling efficiency is measured to be 87 per cent for  $N_{\text{node}} = 128$  and 35 per cent for  $N_{\text{node}} = 1024$ .

Also, note that the CPU memory consumption per MPI process in GAMER-2 deviates from the ideal scaling, especially when increasing the number of nodes. It is most likely due to the allocation of buffer patches and MPI buffers.

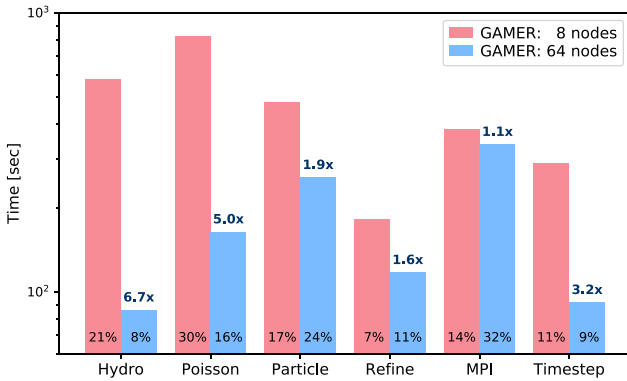
To determine the performance bottleneck in GAMER-2, especially for large  $N_{\text{node}}$ , we show in Fig. 13 the wall time of various operations in the lower resolution runs using 8 and 64 nodes. It is found that the grid PDE solvers, namely, the hydrodynamic and Poisson solvers, are the major performance bottlenecks for  $N_{\text{node}} = 8$ , while for  $N_{\text{node}} = 64$  the bottlenecks shift to the particle routines and MPI communication. From  $N_{\text{node}} = 8$  to 64, the fraction of time spent on the hydrodynamic and Poisson solvers decrease from 21 per cent and 30 per cent to 8 per cent and 16 per cent, respectively; in comparison, the fraction of time spent on the particle routines and MPI communication increase from 17 per cent and 14 per cent to 24 per cent and 32 per cent, respectively.

There are several things to note about these results. First of all, the average MPI message size per process for  $N_{\text{node}} = 64$  is found to be as small as  $\sim 1$  MB, which thus suffers from a relatively larger latency. Second, for better clarification, the performance shown in Fig. 13 is measured from a separate set of simulations that (i) adds an explicit MPI synchronization between grid and particle routines in order for a proper timing analysis, and (ii) does not consider the particle load-balancing weights when estimating the workload of each patch (i.e.  $W_{\text{par}} = 0.0$ ). It partially explains the poor scalability observed in the particle routines and MPI communication, the latter of which also includes transferring particles. For  $N_{\text{node}} = 64$ , it is found that by removing that extra MPI synchronization between grid and particle routines, the overall performance is improved by  $\sim 37$  per cent. Having  $W_{\text{par}} = 2.0$  further improves the performance by  $\sim 10$  per cent. The performance shown in Figs 11 and 12 has incorporated these optimizations. These findings reveal the importance of balancing the workload of both grids and particles simultaneously, as discussed in Section 3.3.

Finally, we compare the CPU memory consumption between GAMER-2 and FLASH. We find that FLASH consumes about an order of magnitude more memory than GAMER-2 when using the same number of nodes, which is why the minimum  $N_{\text{node}}$  adopted for FLASH is 16 instead of 1 as for GAMER-2. This is mainly because FLASH allocates the ghost zones of all patches permanently (see footnote *a* in Table 1) but GAMER-2 does not (see Section 3.4). For a patch with  $8^3$  interior cells and four ghost zones on each side (which is the number of ghost zones adopted in



**Figure 12.** Performance metrics of the strong scaling of the cluster merger simulations. This is complementary to and uses the same symbols as Fig. 11. Different panels show the total wall time (upper left), maximum CPU memory consumption per MPI process (upper right), parallel efficiency (lower left), and doubling efficiency (lower right). See equations (6) and (7) for the definitions of parallel and doubling efficiencies in strong scaling. Note that the minimum number of nodes adopted in FLASH is 16 instead of 1 due to its much larger memory consumption. Therefore, for a proper comparison, we adopt  $N_{\text{node, ref}} = 16$  in equation (6) for both codes when calculating the parallel efficiency. GAMER-2 and FLASH exhibit similar parallel scalability, even though GAMER-2 is about two orders of magnitude faster. We do not show the maximum per-process memory consumption of FLASH because it is mainly determined by the size of the pre-allocated memory pool set manually.



**Figure 13.** Wall time of various operations measured from the lower resolution cluster merger simulations with GAMER-2 using 8 and 64 nodes. For each operation, we also show the fraction of time in the two runs and the speedup gained from increasing  $N_{\text{node}} = 8$  to 64. The ‘MPI’ time includes transferring both grids and particles for the ‘Hydro’, ‘Poisson’, and ‘Particle’ routines but excludes the MPI communication during the grid refinement, which is included in the ‘Refine’ time. The ‘Poisson’ time includes depositing particle mass onto grids with the CIC interpolation. It shows that the grid PDE solvers (i.e. the hydrodynamic and Poisson solvers) are the major performance bottlenecks for  $N_{\text{node}} = 8$ , while the particle routines and MPI communication become the bottlenecks for  $N_{\text{node}} = 64$  as these operations exhibit poor scalability. Note that, for better clarification, the performance shown here does not consider particle weights for load balancing, which is therefore different from the optimized performance shown in Figs 11 and 12. See the text for details.

FLASH), the total memory consumption including the ghost zones is eight times larger than that without the ghost zones. In addition, FLASH uses double precision arithmetic which doubles the memory consumption. On the other hand, GAMER-2 adopts the adaptive time-step integration requiring storing all the grid data at two different physical times for the temporal interpolation, which also roughly doubles the memory consumption. Last but not least, unlike FLASH, GAMER-2 does not pre-allocate a memory pool for all blocks that will be used during the simulation (see Section 3.4).

#### 4.4 AGORA isolated disc galaxy: GAMER-2 versus ENZO

Simulations of the gas, stars, and DM in an idealized isolated disc galaxy present a unique numerical challenge for astrophysical simulation codes. These simulations combine self-gravity, gas dynamics, particle dynamics with particles existing at a range of masses, radiative cooling, and star formation. In addition, gas temperatures may reach as low as 10 K, but have a velocity relative to the simulation box of hundreds of kilometres per second, requiring the use of a dual energy formalism to avoid spurious temperature fluctuations and negative temperatures due to truncation errors. On top of the bulk circular velocity, the gas also exhibits supersonic turbulence driven by gravitational instability (Goldbaum, Krumholz & Forbes 2015). Despite these challenges, isolated disc galaxy simulations are commonly used to understand more complicated zoom-in sim-

ulations (Kim et al. 2016), galaxy merger simulations (Robertson et al. 2006), and as a testbed for physics modules that will be used in more realistic simulations. These simulations also allow *ab initio* exploration of the dynamics of the interstellar medium (ISM) of a galaxy much like the Milky Way (Goldbaum, Krumholz & Forbes 2016), enabling direct comparison with observations of the ISM of our own Galaxy.

In this section, we simulate a Milky Way-sized isolated disc galaxy with star formation using both GAMER-2 and ENZO, from which we demonstrate that, not only the physical results obtained by the two codes are in good agreement, but GAMER-2 outperforms ENZO by almost one order of magnitude. Similar to the previous section, we first describe the simulation setup, with particular emphasis on the similarities and differences of the two codes. We then check the consistency of the physical results, and finally compare the strong scaling performance.

#### 4.4.1 Simulation setup

The simulation setup of our isolated disc galaxy simulations closely follow Goldbaum et al. (2015) and the AGORA High-resolution Galaxy Simulations Comparison Project (Kim et al. 2016). So we only provide a short summary here.

The initial condition is composed of three components: a DM halo, a galactic disc consists of both gas and stars, and a stellar bulge. (i) The DM halo follows an NFW profile with a virial mass  $M_{200} = 1.1 \times 10^{12} M_{\odot}$ , a concentration parameter  $c = 10$ , and a spin parameter  $\lambda = 0.04$ . (ii) The disc follows an exponential profile with a total disc mass  $M_d = 4.3 \times 10^{10} M_{\odot}$ , a scale length  $r_d = 3.4$  kpc, a scale height  $z_d = 0.34$  kpc, and a gas mass fraction  $f_d = M_{d, \text{gas}}/M_d = 0.2$ . The gas disc has an initial metal mass fraction  $Z_d = M_{d, \text{metal}}/M_{d, \text{gas}} = 1.3 \times 10^{-2}$  and an initial temperature  $T_d = 10^4$  K. The circular velocity is set such that the disc is in centrifugal equilibrium. (iii) The stellar bulge is modelled as a Hernquist profile (Hernquist 1990) with a mass  $M_b = 4.3 \times 10^9 M_{\odot}$ . The initial conditions of both DM and stellar particles as well as the gas rotation curve can be downloaded from the AGORA Project.<sup>9</sup> The simulation has a cubic domain with a length  $L = 1.31$  Mpc and is evolved for 500 Myr.

The GRACKLE library is used to solve the chemistry and radiative processes in this work. We adopt the equilibrium solver in GRACKLE using tabulated cooling and heating rates, which are functions of gas density and temperature. These tables incorporate both primordial and metal cooling as well as a UV background. We also include the photoelectric heating from dust and an effective CMB temperature floor. The metal field is treated as a passive scalar advected along with the gas, and thus the metal fraction can vary in space and time. To avoid artificial fragmentation, we follow Goldbaum et al. (2015) and Kim et al. (2016) and employ a pressure floor in the hydrodynamic solver to ensure that the local Jeans length is resolved by at least four cells on the maximum refinement level.

The full details of the subgrid model for stochastic star formation can be found in Section 2.4 of Goldbaum et al. (2015). Specifically, we adopt a gas density threshold  $n_{\text{H,thres}} = 20 \text{ cm}^{-3}$  and a star formation efficiency  $f_{\star} = 1$  per cent. We also impose a minimum star particle mass  $m_{\star} = 2 \times 10^3 M_{\odot}$ . No star formation feedback is included in this work.

Table 3 summarizes the similarities and differences of the numerical setup adopted by GAMER-2 and ENZO in these comparison simulations. Here, we elaborate on the major differences.

(i) GAMER-2 restricts all patches to have exactly the same size ( $8^3$  cells in this work), which has several advantages. For example, the AMR hierarchy can be manipulated efficiently with an octree data structure. The memory allocation is more predictable, which eases the issue of memory fragmentation and maximizes memory reuse (see Section 3.4). The smaller patch size conceivably leads to a more efficient use of both CPU and GPU caches. In addition, it is more straightforward to optimize load balancing. In comparison, ENZO allows all patches to have different sizes. It reduces the number of cells that are unnecessarily refined compared to GAMER-2. The relatively larger patches also reduce both the computation and communication overhead associated with the ghost zones of each patch. However, arguably, reconstructing the AMR hierarchy becomes more expensive in this approach due to the more complicated grid structure, which might deteriorate the parallel scalability, especially in massively parallel simulations.

(ii) For the Poisson solver, GAMER-2 uses the SOR scheme suitable for smaller grids (see Section 3.1), while ENZO adopts the multigrid scheme suitable for larger grids due to its higher convergence rate. In addition, GAMER-2 has a fixed patch size of  $8^3$  cells and adds five additional buffer zones around each patch to make the potential smoother across the patch boundaries (see Fig. 2). In comparison, ENZO has grid patches generally larger than  $8^3$  and also allocates a slightly larger buffer zones of six around each patch. Besides, it applies an iterative procedure to exchange potential between sibling grids to improve the accuracy further, which has not been implemented into GAMER-2. In this work, we adopt 10 such iterations in the ENZO simulations, which is measured to increase the simulation time by  $\sim 10$  per cent.

#### 4.4.2 Accuracy comparison

Fig. 14 shows the face-on projection of gas density in the central 30 kpc region at  $t = 0, 100, 300$ , and 500 Myr obtained by GAMER-2 and ENZO. The filamentary structures form quickly due to self-gravity, radiative cooling, and shear flow. These filaments then continuously collapse into gravitationally bound clouds and trigger star formation. We notice that, at later epochs of the simulations, a significant fraction of gas has collapsed and merged into large clouds and formed unrealistically massive star clusters, and there is no prominent spiral structure. These results are inconsistent with the smooth disc and prominent spiral arms observed in disc galaxies, and are mainly due to the lack of star formation feedback in this work that leads to overcooling of gas (see fig. 2 in Goldbaum et al. 2016). Active galactic nucleus feedback is also expected to strongly change the thermodynamics and kinematics of the multi-phase gas (e.g. Gaspari et al. 2018). It is, however, not a concern here since we focus on the comparison between different codes. Fig. 14 shows that the gross morphological features obtained by GAMER-2 and ENZO agree well with each other. Subtle differences are expected to some extent because of the stochastic star formation and the different numerical implementations (see Table 3). More quantitative comparisons are provided below.

Fig. 15 shows the azimuthally averaged profiles of various gas properties at  $t = 500$  Myr, including the surface density, temperature, rotation velocity, and velocity dispersion. Following Kim et al. (2016), we set the galactic centre to the location of peak gas density within 1 kpc from the centre of gas mass. All profiles exhibit clear

<sup>9</sup><http://goo.gl/8JzbJJ>. Note that we use the high-resolution files, while Kim et al. (2016) use the low-resolution files.



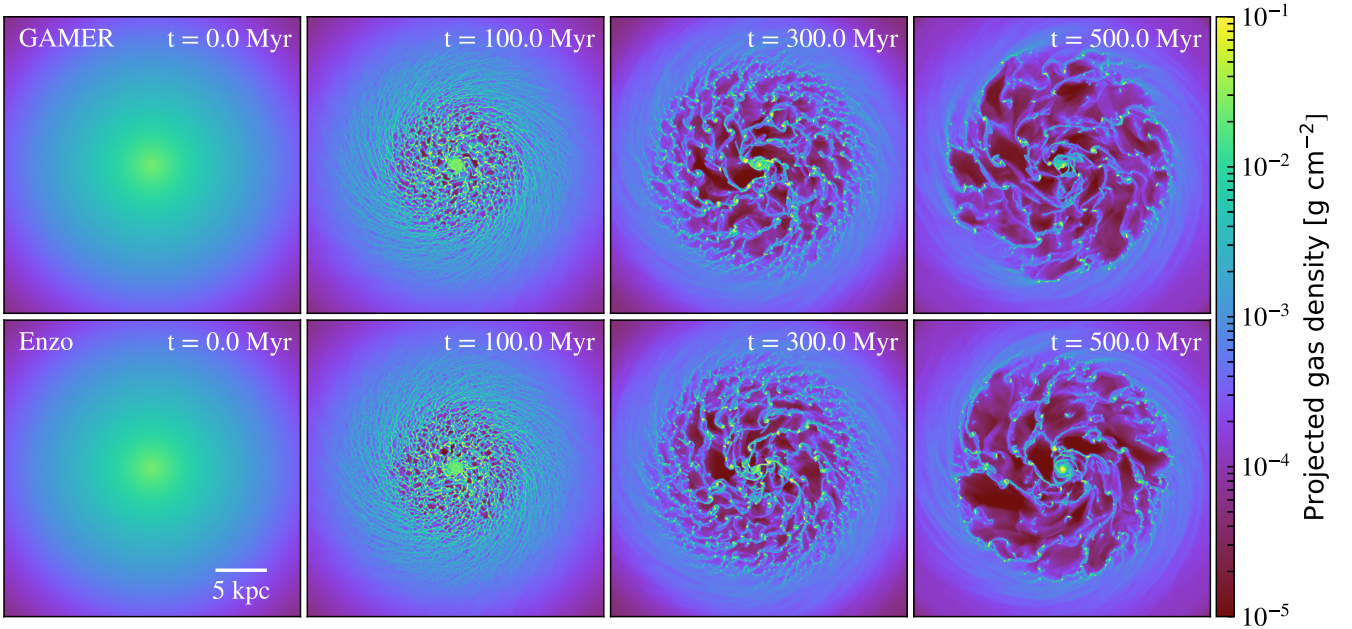
**Table 3.** Comparison of the numerical setup between GAMER-2 and ENZO in the isolated disc galaxy simulations.

	GAMER-2	ENZO
AMR implementation	Fixed patch size of $8^3$ cells, no permanent allocation of patch ghost zones	Patch size is not fixed, patch ghost zones are allocated permanently
Grid resolution	Root grid $64^3$ , 10 refinement levels, maximum resolution $\Delta h = 20$ pc	Same as GAMER-2
Particle resolution	Halo: $N_p = 1 \times 10^7$ and $m_p = 1.3 \times 10^5 M_\odot$ stellar disc: $N_p = 1 \times 10^7$ and $m_p = 3.4 \times 10^3 M_\odot$ bulge: $N_p = 1.25 \times 10^6$ and $m_p = 3.4 \times 10^3 M_\odot$	Same as GAMER-2
Fluid solver	Dimensionally unsplit MUSCL-Hancock scheme with PPM reconstruction (requiring three Riemann solvers per cell), HLLC solver, hybrid van Leer and generalized minmod slope limiter, dual energy formalism solving the entropy equation	Dimensionally split direct Eulerian approach with PPM reconstruction (requiring three Riemann solvers per cell), HLLC solver, hybrid van Leer and generalized minmod slope limiter, dual energy formalism solving the internal energy equation
Poisson solver	SOR	Multigrid
Particle solver	CIC interpolation and KDK particle update	CIC interpolation and DKD particle update
Boundary condition	Fluid solver: outflow Poisson solver: isolated	Fluid solver: periodic <sup>a</sup> Poisson solver: isolated
Refinement	(1) Maximum particle mass in a cell: $1.0 \times 10^6 M_\odot$ (2) Maximum gas mass in a cell: $3.4 \times 10^2 M_\odot$ (3) Resolving Jeans length by at least 64 cells (4) Five additional levels of statically refined regions above the root grid, enclosing volumes that are successively smaller by a factor of 8	Same as GAMER-2
Derefinement	No explicit derefinement criteria	Same as GAMER-2
Time-step	$C_{\text{par}} = 0.5$ and $C_{\text{CFL}} = 0.5$	Same as GAMER-2
Parallelization	Hybrid MPI/OpenMP/GPU	MPI and GPU acceleration <sup>b</sup>
Load balancing	Hilbert space-filling curve	Same as GAMER-2
Time integration	Adaptive time-step <sup>c</sup>	Same as GAMER-2
Floating-point format	Single precision	Same as GAMER-2

Notes. <sup>a</sup>The difference in the boundary conditions of the fluid solver is found to have negligible effect in this work.

<sup>b</sup>In ENZO, currently only the fluid and MHD solvers have been ported to GPUs.

<sup>c</sup>Both GAMER-2 and ENZO do not restrict the time-step ratio between two adjacent levels to be a constant.

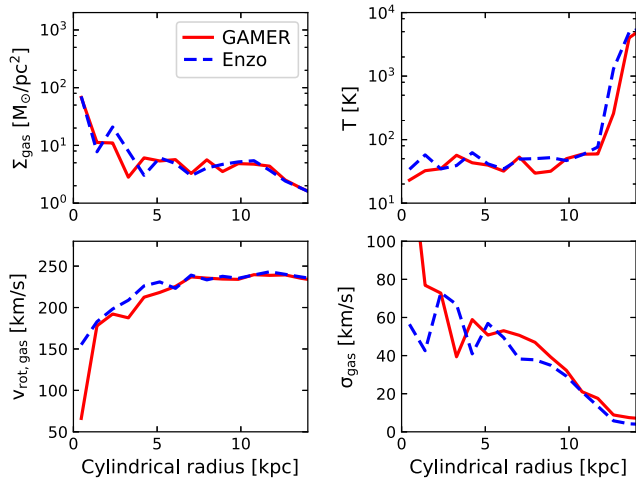


**Figure 14.** Face-on projection of gas density at four different epochs in the isolated disc galaxy simulations. Each panel is 30 kpc on a side. The simulations with GAMER-2 (upper panels) and ENZO (lower panels) show very similar filamentary structures. Subtle differences are expected to some extent because of the stochastic star formation and the different numerical implementations (see Table 3). See Figs 15–17 for more quantitative comparisons between the two codes. At late times, a significant fraction of gas has collapsed and merged into large gravitationally bound clouds and there are no prominent spiral arms, mainly because we do not include star formation feedback in this work (see fig. 2 in Goldbaum et al. 2016).

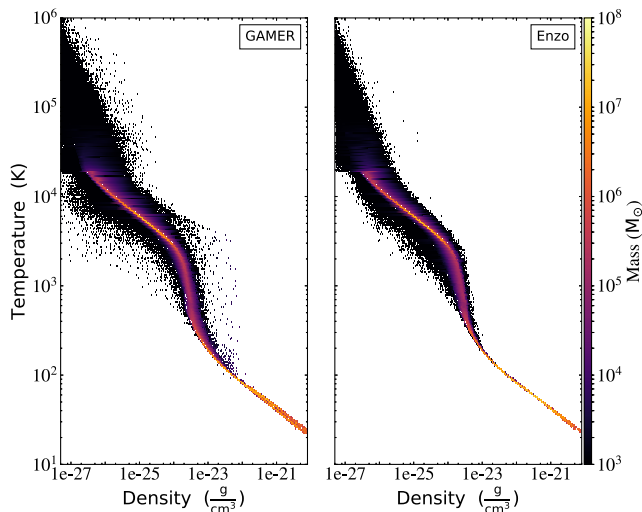
oscillation, which become more prominent in time as an increasing fraction of gas collapses into massive clouds. The temperature

within  $\sim 12$  kpc drops significantly from the initial temperature of  $10^4$  K to below  $\sim 100$  K due to the balance between efficient metal





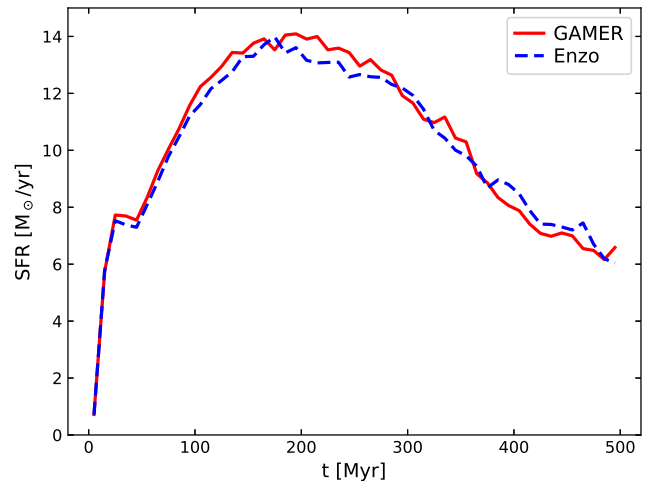
**Figure 15.** Azimuthally averaged profiles at  $t = 500$  Myr in the isolated disc galaxy simulations. Different panels show the gas surface density (upper left), gas temperature (upper right), gas rotation velocity (lower left), and gas velocity dispersion (lower right). The results of GAMER-2 (red lines) and ENZO (blue lines) are in good agreement with each other. A relatively large discrepancy appears in the innermost region, which is somewhat expected since the central region is sensitive to both the determination of the galactic centre and local mergers.



**Figure 16.** Probability distribution function of gas in the density–temperature plane of the isolated disc galaxy simulations at  $t = 500$  Myr. Colour bar represents the gas mass in each bin. The results obtained by GAMER-2 (left-hand panel) and ENZO (right-hand panel) are in very good agreement thanks to the common library GRACKLE adopted for calculating the chemistry and radiative processes.

line cooling and UV heating. Substantial turbulent velocity dispersion, primarily driven by gravitational instability, develops quickly and increases toward the galactic centre. Importantly, we find very good agreement between GAMER-2 and ENZO in all four profiles. A relatively large discrepancy appears in the innermost region, which is somewhat expected since the central region is sensitive to both the determination of the galactic centre and local mergers.

Fig. 16 shows the probability distribution function of gas in the density–temperature plane at  $t = 500$  Myr. A clear branch toward the high-density, low-temperature corner can be easily identified, resulting from the balance between the various cooling and heating



**Figure 17.** SFR as a function of time in the isolated disc galaxy simulations. We find very good agreement between GAMER-2 (red line) and ENZO (blue line).

mechanisms adopted in this work (see Section 4.4.1). The low-density, high-temperature component in the upper left corner corresponds to the gaseous halo included in the consideration of numerical stability (Kim et al. 2016). This figure further validates the consistency of the gaseous thermodynamic properties between our GAMER-2 and ENZO simulations, thanks to the common chemistry and radiative library GRACKLE. A relatively large scatter is found in the GAMER-2 simulation, which however constitutes a negligible fraction of the total gas mass.

Fig. 17 shows the star formation rate (SFR) as a function of time, for which again GAMER-2 and ENZO agree very well with each other. The SFR reaches  $\sim 10 M_{\odot} \text{ yr}^{-1}$  after  $t \gtrsim 100$  Myr, consistent with the result of Goldbaum et al. (2015) which also does not include star formation feedback, and is about a factor of 5 higher than the SFR obtained in the simulations with feedback (Kim et al. 2016; Goldbaum et al. 2016). Interestingly, the consistency between GAMER-2 and ENZO shown in this figure seems to be better than that found in the comparison of grid codes in the AGORA comparison project (see fig. 26 in Kim et al. 2016). It remains to be investigated whether this level of consistency could be achieved after including feedback.

The agreement between the simulations results of GAMER-2 and ENZO, as verified in Figs 14–17, demonstrates the consistent numerical setup adopted for this code comparison experiment, including, for example, the initial condition, spatial and temporal resolution, and grid refinement criteria. It also suggests that the differences between the two codes described in Section 4.4.1, for example, the AMR implementation, fluid and Poisson solvers, and particle integration, do not have a serious impact here. These facts strengthen the results of performance comparison shown in the next section.

#### 4.4.3 Performance comparison

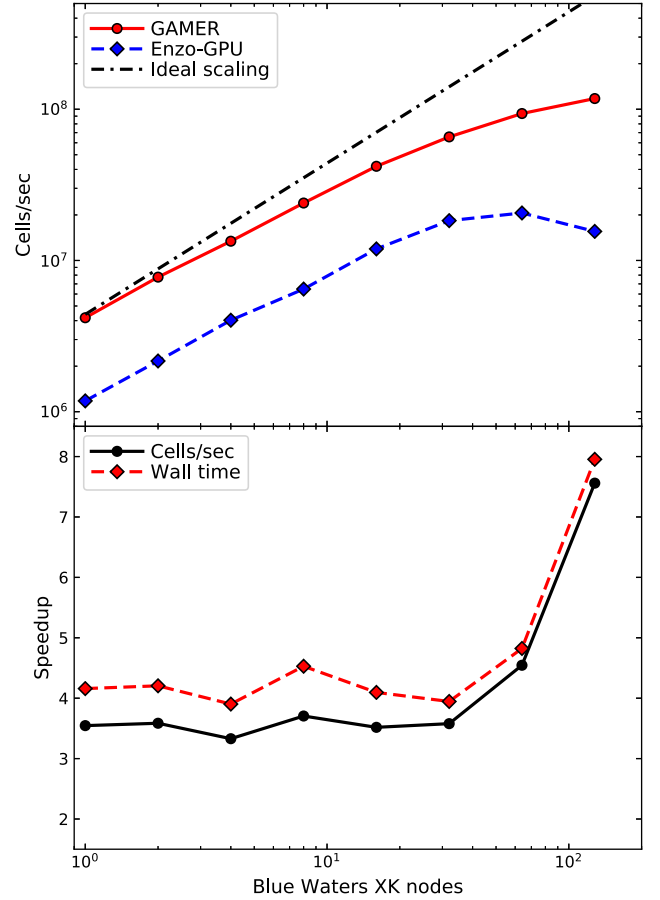
Here, we compare strong scaling performance between GAMER-2 and ENZO measured on Blue Waters. Since ENZO also supports GPU acceleration, although only for the hydrodynamic and MHD solvers, we run both codes on the XK nodes, each of which is composed of one GPU (NVIDIA Tesla K20X) and one 16-core CPU (AMD Opteron 6276). Similar to the cluster merger simulations described in Section 4.3.3, for GAMER-2, we launch two

MPI processes per node and seven OpenMP threads per MPI process to improve memory affinity, and use CUDA MPS to allow these two processes to share the same GPU. For ENZO, we launch 16 MPI processes per node since it does not support hybrid MPI/OpenMP parallelization. In addition, for  $N_{\text{node}} = 1-4$ , we disable `SubgridSizeAutoAdjust`<sup>10</sup> and set `MaximumSubgridSize = 8192` to avoid exhausting the GPU memory due to the too large grid size. Changing `MaximumSubgridSize` by a factor of 2 reduces the performance of  $N_{\text{node}} = 4$  by  $\sim 10-20$  per cent. For  $N_{\text{node}} = 8-128$ , we enable `SubgridSizeAutoAdjust` and have `OptimalSubgridsPerProcessor = 16`, which are found to generally achieve the best performance in our tests. The Hilbert space-filling curve is adopted for load balancing in both GAMER-2 and ENZO.<sup>11</sup> We measure the performance in a relatively short period of  $t = 300-305$  Myr, which is representative enough since both the total number of cells and evolution time-step are found to be quite stable after  $t \gtrsim 100$  Myr. At  $t = 300$  Myr, there are  $\sim 2.2 \times 10^8$  cells in total,  $\sim 20$  per cent of which are on the maximum refinement level.

Fig. 18 shows the strong scaling of the isolated disc galaxy simulations. The speedup of GAMER-2 over ENZO is measured to be about 4–8 in terms of both total wall time and cell updates per second. For example, for  $N_{\text{node}} = 64$ , GAMER-2 and ENZO achieve  $1.5 \times 10^6$  and  $3.2 \times 10^5$  cell updates per second per node, respectively. This result is encouraging, especially because both codes take advantage of GPU acceleration. More importantly, this speedup ratio is approximately a constant for  $N_{\text{node}} \leq 32$  and increases for  $N_{\text{node}} > 32$ . The overall performance of ENZO starts to drop for  $N_{\text{node}} > 64$ , while that of GAMER-2 starts to drop for  $N_{\text{node}} > 128$ . These results suggest that GAMER-2 not only runs faster but also scales better than ENZO.

In Fig. 18, we show the performance speedups in terms of both total wall time and cell updates per second. The former arguably provides a more comprehensive comparison because it considers not only the performance of all PDE solvers but also many other factors such as the AMR implementation and evolution time-step. In this test, the speedup in terms of total wall time is measured to be  $\sim 5-20$  per cent higher than that in terms of cell updates per second, partially because GAMER-2 generally requires less extra updates for synchronizing nearby AMR levels (see Section 2.1, especially Fig. 1). On the other hand, although GAMER-2 and ENZO allocate roughly the same number of cells on the maximum refinement level  $l = 10$  (the difference is less than 3 per cent), we find that GAMER-2 typically allocates  $\sim 50-150$  per cent more cells than ENZO on other high levels (e.g.  $l = 7-9$ , see Table 4). It is mainly because GAMER-2 restricts all patches to have the same size which results in over-refinement, especially along the direction perpendicular to the galactic disc and on the levels with cell sizes much larger than the disc scale height. This issue, however, does not pose a serious problem here since lower levels are updated much less frequently thanks to the adaptive time-step integration.

Fig. 19 shows the performance metrics of the isolated disc galaxy simulations, including the total wall time, maximum CPU memory consumption per MPI process, parallel efficiency, and doubling efficiency as a function of the number of XK nodes. Most importantly,



**Figure 18.** Strong scaling of the isolated disc galaxy simulations run with GAMER-2 (solid line) and ENZO (dashed line) using 1–128 nodes. Both codes support GPU acceleration and run on the Blue Waters XK nodes, each of which is composed of one GPU (NVIDIA Tesla K20X) and one 16-core CPU (AMD Opteron 6276). The upper panel shows the total number of cell updates per second, where the dashed-dotted line represents the ideal scaling. The lower panel shows the speedup of GAMER-2 over ENZO in terms of both cell updates per second (solid line) and total wall time (dashed line). GAMER-2 is measured to be  $\sim 4-8$  times faster than ENZO. More importantly, this speedup ratio is approximately a constant for  $N_{\text{node}} \leq 32$  and increases for  $N_{\text{node}} > 32$ , suggesting that GAMER-2 exhibits better parallel scalability than ENZO. See Fig. 19 for the detailed performance metrics of this test.

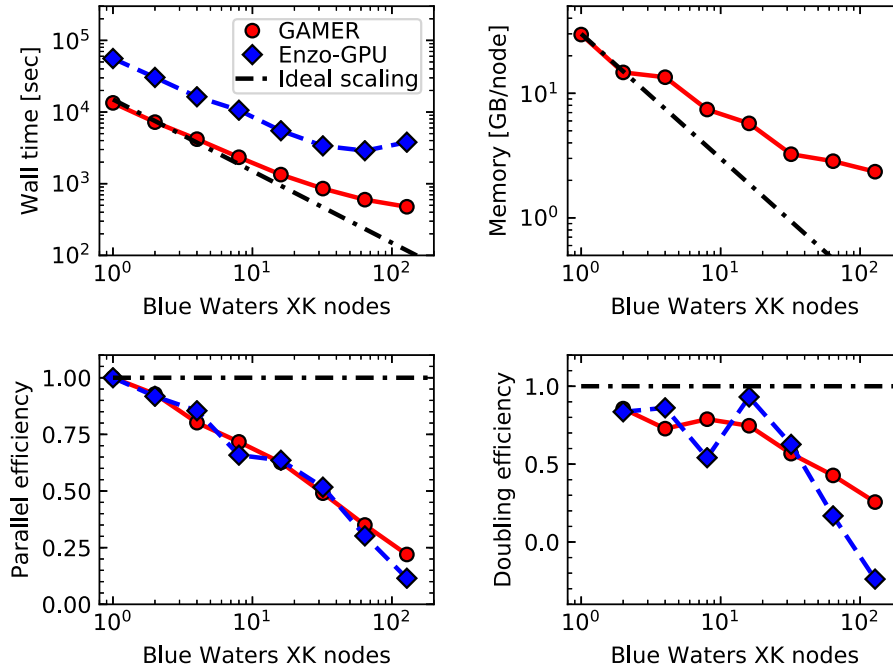
**Table 4.** Comparison of the volume-filling fractions on higher refinement levels between GAMER-2 and ENZO in the isolated disc galaxy simulations at  $t = 300$  Myr.

Level	$\Delta h$ /pc	Filling fraction		Number of cells	
		GAMER-2 (per cent)	ENZO (per cent)	GAMER-2	ENZO
7	160.0	$2.4 \times 10^{-3}$	$1.0 \times 10^{-3}$	$1.3 \times 10^7$	$5.7 \times 10^6$
8	80.0	$4.3 \times 10^{-4}$	$2.0 \times 10^{-4}$	$1.9 \times 10^7$	$8.8 \times 10^6$
9	40.0	$1.2 \times 10^{-4}$	$6.3 \times 10^{-5}$	$4.2 \times 10^7$	$2.2 \times 10^7$
10	20.0	$1.5 \times 10^{-5}$	$1.5 \times 10^{-5}$	$4.2 \times 10^7$	$4.1 \times 10^7$

both the parallel and doubling efficiencies demonstrate that the two codes exhibit very similar parallel scalability for  $N_{\text{node}} \leq 32$ , and, furthermore, GAMER-2 scales noticeably better than ENZO for  $N_{\text{node}} > 32$ , consistent with Fig. 18.

<sup>10</sup>See <https://enzo.readthedocs.io/en/latest/index.html>

<sup>11</sup>It corresponds to `LoadBalancing = 4`, which is not officially supported but works well in our tests.

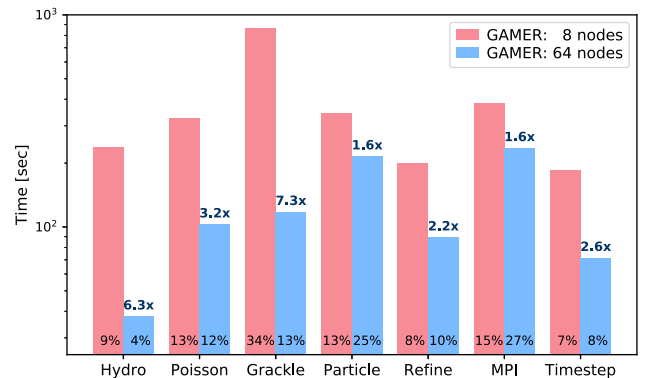


**Figure 19.** Performance metrics of the strong scaling of the isolated disc galaxy simulations. This is complementary to and uses the same symbols as Fig. 18. Different panels show the total wall time (upper left), maximum CPU memory consumption per MPI process (upper right), parallel efficiency (lower left), and doubling efficiency (lower right). See equations (6) and (7) for the definitions of parallel and doubling efficiencies in strong scaling. GAMER-2 and ENZO exhibit very similar parallel scalability for  $N_{\text{node}} \leq 32$ , and GAMER-2 scales noticeably better than ENZO for  $N_{\text{node}} > 32$ . The maximum per-process memory consumption of ENZO is not shown since the data are not available.

We notice that the doubling efficiency of ENZO oscillates for  $N_{\text{node}} = 4$ –16, likely because, as described in the beginning of this section, we enable `SubgridSizeAutoAdjust` for  $N_{\text{node}} > 4$  to improve performance and scalability. Also note that the CPU memory consumption per MPI process in GAMER-2 deviates from the ideal scaling, especially when increasing the number of nodes, most likely due to the allocation of buffer patches and MPI buffers.

To determine the performance bottleneck of GAMER-2 in the isolated disc galaxy simulations, especially for large  $N_{\text{node}}$ , we compare in Fig. 20 the wall time of various operations in the  $N_{\text{node}} = 8$  and 64 runs. We find that, for  $N_{\text{node}} = 8$ , the chemistry and cooling library GRACKLE is the major performance bottleneck. In contrast, for  $N_{\text{node}} = 64$ , the bottlenecks move to the particle routines and MPI communication, which is similar to what we find in the cluster merger simulations (see Fig. 13). It is mainly because, currently, the load balancing in GAMER-2 is better optimized for grid solvers like GRACKLE than for particle-related routines. From  $N_{\text{node}} = 8$  to 64, the fraction of time spent on GRACKLE decreases from 34 per cent to 13 per cent, while that on the particle routines and MPI communication increase from 13 per cent and 15 per cent to 25 per cent and 27 per cent, respectively. Also note that a GPU-supported GRACKLE is currently under development, which would likely lead to larger speedups.

There are several things to note about these results, which are similar to the discussions given in the cluster merger simulations. First of all, the average MPI message size per process for  $N_{\text{node}} = 64$  is found to be as small as  $\sim 1$  MB, which is smaller than that for  $N_{\text{node}} = 8$  by a factor of few. The  $N_{\text{node}} = 64$  run thereby suffers



**Figure 20.** Wall time of various operations measured from the isolated disc galaxy simulations with GAMER-2 using 8 and 64 nodes. For each operation, we also show the fraction of time in the two runs and the speedup gained from increasing  $N_{\text{node}} = 8$ –64. The ‘MPI’ time includes transferring both grids and particles for the ‘Hydro’, ‘Poisson’, and ‘Particle’ routines but excludes the MPI communication during the grid refinement, which is included in the ‘Refine’ time. The ‘Poisson’ time includes depositing particle mass onto grids with the CIC interpolation. It shows that the chemistry and cooling library GRACKLE is the major performance bottleneck for  $N_{\text{node}} = 8$ , while the particle routines and MPI communication become the bottlenecks for  $N_{\text{node}} = 64$  because these operations exhibit poor scalability. Note that, for better clarification, the performance shown here does not consider particle weights for load balancing, which is therefore different from the optimized performance shown in Figs 18 and 19. See the text for details. Also note that a GPU-supported GRACKLE is currently under development, which would likely lead to larger speedups.

from a relatively larger communication latency. Second, for better clarification, the performance shown in Fig. 20 is measured from a separate set of simulations that (i) adds an explicit MPI synchronization between grid and particle routines in order for a proper timing analysis, and (ii) disregards particle weights in load balancing (i.e.  $W_{\text{par}} = 0.0$ ), both of which deteriorate the scalability. For  $N_{\text{node}} = 64$ , we find that the overall performance is improved by  $\sim 22$  per cent after removing that extra MPI synchronization between grid and particle routines, and by another  $\sim 23$  per cent after adopting  $W_{\text{par}} = 1.0$ . These optimizations have been incorporated into the strong scaling shown in Figs 18 and 19. Third, note that the time fraction of the Poisson solver shown in Fig. 20 includes the time for depositing particle mass onto grids, which is why it scales worse than the other two grid solvers (i.e. hydrodynamic and GRACKLE solvers).

## 5 SUMMARY AND FUTURE WORK

In this paper, we have presented GAMER-2, a significant revision of the GPU-accelerated AMR code GAMER-1 (Schive et al. 2010). It includes much richer functionality and incorporates significant improvements in accuracy, stability, performance, and scalability. Table 5 summarizes the major differences between GAMER-2 and GAMER-1.

To reveal the optimal performance of GAMER-2, we first measure the performance of individual GPU solvers and show that both the hydrodynamic and Poisson solvers achieve a single-precision performance of  $\sim 2 \times 10^8$  cells  $\text{s}^{-1}$  on an NVIDIA Tesla P100 GPU. We also measure the weak scaling performance with and without AMR in a 3D KH instability test on the Blue Waters supercomputer using 1–4096 XK nodes, each of which is composed of one NVIDIA Tesla K20X GPU and one 16-core AMD Opteron 6276 CPU. By taking advantage of the hybrid MPI/OpenMP/GPU parallelization, we are able to fully exploit both 4096 GPUs and 65 536 CPU cores simultaneously, and achieve a peak performance of  $8.3 \times 10^{10}$  and  $4.6 \times 10^{10}$  cells  $\text{s}^{-1}$  and a parallel efficiency of 74 per cent and 58 per cent in the uniform-grid and AMR tests, respectively. Note that the simulation reaches an overall resolution as high as  $10\,240^3$  cells with 4096 nodes in the uniform-grid test.

To further provide clear and convincing demonstrations of the accuracy and performance of GAMER-2, we directly compare GAMER-2 with two widely adopted AMR codes, FLASH and ENZO, based on realistic astrophysical simulations running on Blue Waters. First, we compare GAMER-2 with FLASH in binary cluster merger simulations, which closely follow the numerical setup of ZuHone (2011) and involve hydrodynamics, self-gravity, and DM. We show that the physical results obtained by the two codes are in excellent agreement, and GAMER-2 is measured to be 78 – 101 times faster than FLASH in strong scaling tests using 1–256 nodes. More importantly, both codes exhibit similar parallel scalability, despite the fact that the computational time of GAMER-2 has been greatly reduced by exploiting GPUs. We also measure the strong scaling of GAMER-2 from 16 to 2048 nodes using a set of higher resolution simulations, and obtain a parallel efficiency of 78 per cent with 128 nodes and 37 per cent with 1024 nodes.

Second, we compare GAMER-2 with ENZO in isolated disc galaxy simulations, which closely follow the numerical setup of Goldbaum et al. (2015) and the AGORA High-resolution Galaxy Simulations Comparison Project (Kim et al. 2016) but with a spatial resolution of 20 pc. These simulations involve a richer set of physical modules, including hydrodynamics, self-gravity, DM, advection of

metals, radiative cooling and heating, and stochastic star formation. Again, we find very good agreement between the physical results obtained by GAMER-2 and ENZO. To compare the performance, we also enable GPU acceleration in ENZO for the hydrodynamic solver. Even so, GAMER-2 is still measured to be  $\sim 4$ –8 times faster than ENZO in strong scaling tests using 1–128 nodes. It may be partially due to the fact that ENZO currently does not support asynchronous GPU kernel execution and CPU–GPU communication. Further investigation will be conducted in the future. More importantly, this speedup ratio is approximately a constant of 4–5 with 1–32 nodes and increases to 5–8 when using more than 32 nodes, suggesting that GAMER-2 not only runs faster but also scales noticeably better than ENZO.

GAMER-2 has supported the following features to improve the parallel scalability:

- (i) *Hybrid OpenMP/MPI parallelization* (see Section 3.2). It reduces internode communication and therefore improves the parallel scalability, especially when using a large number of nodes.
- (ii) *Fixed patch size*. It greatly simplifies the parallel manipulation of AMR hierarchy and load balancing, especially in massively parallel simulations. Moreover, we do not require duplicating the entire AMR hierarchy on each MPI process (see Section 3.3).
- (iii) *Level-by-level load balancing with Hilbert space-filling curves* (see Section 3.3). Especially, we take into account the particle weights in load balancing (see Fig. 4), and further minimize the MPI synchronization between grid and particle routines.

GAMER-2 allocates memory pools for both grid and particle data to alleviate the issue of memory fragmentation and to maximize memory reuse (see Section 3.4). Moreover, the code minimizes the GPU memory requirement by storing all the data on the CPU’s main memory and transferring only a small and fixed amount of patch data to GPUs (typically several hundreds of MB to a few GB per GPU) at a time.

We have identified several performance bottlenecks from the detailed timing analysis conducted in this work (e.g. see Figs 13 and 20), including load imbalance due to particles, GRACKLE library, MPI communication, and CPU performance when preparing the input data for GPU solvers. To improve performance further, we are currently investigating the following optimizations:

- (i) Transferring the deposited particle mass density on grids instead of individual particles when calculating the total mass density on levels other than the maximum level. This will greatly reduce the MPI communication for particles and also improve load balancing.
- (ii) Porting some of the particle routines to GPUs.
- (iii) MPI non-blocking communication. It will allow overlapping MPI communication by both CPU and GPU computations.
- (iv) GPU-accelerated GRACKLE. GRACKLE computes the chemistry and radiative processes on a cell-by-cell basis, which should be very GPU-friendly because no synchronization and data exchange between different cells are required. We have obtained an order of magnitude speedup in preliminary tests.
- (v) Optimization of CPU routines. One important optimization in GAMER-2 is to allow CPUs and GPUs to work concurrently (see Section 3.2). Accordingly, depending on the CPU and GPU specifications, we find that the performance bottleneck may occur in CPUs when invoking a GPU kernel (since we still rely on CPUs to prepare the input data for GPUs). It is therefore essential to optimize the CPU performance further by, for example, improving the OpenMP parallel efficiency, porting more operations to GPUs, and optimizing memory access.



**Table 5.** Summary of the major differences between GAMER-2 and GAMER-1.

Features	GAMER-2	GAMER-1	References
Adaptive time-step	Fully supported without requiring the time-step ratio between two adjacent levels to be a constant	Only supported in pure hydrodynamic simulations and the time-step ratio between two adjacent levels must be 2	Section 2.1
Fluid solvers	Both dimensional split, Riemann-solver-free scheme (RTVD) and dimensional unsplit, Riemann-solver-based schemes (MHM, VL, and CTU) with PLM/PPM data reconstructions and various Riemann solvers	RTVD only	Section 2.2
Dual energy formalism	Supported	Unsupported	Section 2.2
MHD	Supported using the CTU + CT scheme	Not supported	Zhang et al. (2018)
Poisson solver	Adding five buffer zones around each refinement-level patch	Multilevel solver that eliminates the pseudo-mass sheets on the patch interfaces	Section 2.3
Gravity in hydro	Operator-unsplit approach	operator-split approach	Section 2.3
Boundary conditions	Fluid: periodic, outflow, reflecting, inflow (i.e. user-defined) Gravity: periodic, isolated	Fluid: periodic Gravity: periodic	Section 2.2, Section 2.3
Particles	Supported	Unsupported	Section 2.4
GRACKLE	Supported	Unsupported	Section 2.5
Bitwise reproducibility	Supported	Unsupported	Section 2.7.1
HDF5 output	Supported	Unsupported	Section 2.7.2
YT data analysis	Supported	Unsupported	Section 2.7.2
Test problem infrastructure	Supported	Unsupported	Section 2.7.3
AMR + GPUs framework	Supported (e.g. $\psi$ DM simulations)	Unsupported	Section 2.7.4
Hybrid MPI/OpenMP	Supported	Unsupported	Section 3.2
Parallelization	Hilbert curve for load balancing	Rectangular domain decomposition	Section 3.3
Memory pool	Supported	Unsupported	Section 3.4

Given the excellent performance reported here, it is then essential to extend the functionality of GAMER-2 so that it can be applied to a broader range of applications. The following new features are being or will be investigated in the near future. In addition, since GAMER-2 is an open-source code, we are also looking forward to contributions from the community.

(i) New particle features including tracer particles, comoving coordinates, and multiple species. We also plan to store ‘relative’ instead of absolute particle positions, which can be very helpful for simulations demanding an extremely large dynamic range.

(ii) Non-Cartesian coordinates.

(iii) Non-ideal hydrodynamics and non-ideal MHD.

(iv) Radiative transfer.

(v) Parallel I/O.

(vi) Testing framework for ensuring the correctness of the code.

Note that GAMER-2 can also run in a ‘CPU-only’ mode using a hybrid MPI/OpenMP parallelization, for which we simply replace all GPU solvers by their CPU counterparts parallelized with OpenMP and use the same MPI implementation as in the GPU-accelerated code. Therefore, GAMER-2 is also suitable for CPU-only supercomputers, especially for those with a larger number of cores per node like Intel Xeon Phi. Hybrid MPI/OpenMP is essential to achieve optimal performance in such systems, and it may require further optimization about, for example, thread affinity, thread load balancing, and OpenMP nested parallelism.

Finally, we emphasize that the great performance and scalability of GAMER-2 demonstrated here in both binary cluster merger and isolated disc galaxy simulations allow one to study various astrophysical phenomena requiring resolutions that are not realistically attainable previously. For example, for the cluster merger simulations, we have obtained preliminary results from simulations with sub-kpc resolution, which will enable us to reduce the numerical viscosity significantly and to investigate the properties of the turbulent cascade down to a scale where the effects of a physical

viscosity are expected to become relevant. It is also possible to increase the spatial resolution of isolated disc galaxy simulations to  $\sim 5$  pc, which will produce a dynamically evolving ISM, undergoing repeated cycles of collapse, star formation, feedback, rarefaction, and recollapse which have been extremely difficult to fully resolve in a global galactic-scale simulation over galactic dynamical times.

## ACKNOWLEDGEMENTS

HS would like to thank Edward Seidel, Gabrielle Allen, and Tzi-hong Chiueh for their great support on this project, and Roland Haas and Brian O’Shea for stimulating discussions. HS would also like to thank Sandor Molnar for helping implement the galaxy cluster merger simulations and Hsiang-Chih Hwang for helping implement particles into GAMER-2. HS and MG are grateful to James Stone for insightful discussions. The authors are also grateful to Britton Smith for helping incorporate GRACKLE into GAMER-2. Finally, we want to thank the referee, Michael Norman, for a constructive report that helped to improve the paper. This publication is supported in part by the Gordon and Betty Moore Foundation’s Data-Driven Discovery Initiative through Grant GBMF4561 to Matthew Turk, and is based upon work supported by the National Science Foundation under Grant No. ACI-1535651. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications (NCSA). This work also used computational resources provided by the Innovative Systems Lab (ISL) at NCSA. MG is supported by NASA through Einstein Postdoctoral Fellowship Award Number PF5-160137 issued by the Chandra X-ray Observatory Center, which is operated by the SAO for and on behalf of NASA under contract NAS8-03060. Support for this work was also provided by Chandra grant GO7-18121X.

## REFERENCES

- Almgren A. S. et al., 2010, *ApJ*, 715, 1221
- Almgren A. S., Bell J. B., Lijewski M. J., Lukić Z., Van Andel E., 2013, *ApJ*, 765, 39
- Amdahl G. M., 1967, Proceedings of the Spring Joint Computer Conference (AFIPS '67, Spring). ACM, New York, NY, USA, p. 483
- Banerjee N., Sharma P., 2014, *MNRAS*, 443, 687
- Berger M. J., Colella P., 1989, *J. Comput. Phys.*, 82, 64
- Berger M. J., Oliger J., 1984, *J. Comput. Phys.*, 53, 484
- Brunetti G., Lazarian A., 2007, *MNRAS*, 378, 245
- Bryan G. L., Norman M. L., Stone J. M., Cen R., Ostriker J. P., 1995, *Comput. Phys. Commun.*, 89, 149
- Bryan G. L. et al., 2014, *ApJS*, 211, 19
- Colella P., 1990, *J. Comput. Phys.*, 87, 171
- Cunningham A. J., Frank A., Varnière P., Mitran S., Jones T. W., 2009, *ApJS*, 182, 519
- De Martino I., Broadhurst T., Tye S.-H. H., Chiueh T., Schive H.-Y., Lazkoz R., 2017, *Phys. Rev. Lett.*, 119, 221103
- Eastwood J., Brownrigg D., 1979, *J. Comput. Phys.*, 32, 24
- Eckert D., Gaspari M., Vazza F., Gastaldello F., Tramacere A., Zimmer S., Etori S., Paltani S., 2017, *ApJ*, 843, L29
- Eddington A. S., 1916, *MNRAS*, 76, 572
- Einfeldt B., Munz C., Roe P., Sjögren B., 1991, *Journal of Computational Physics*, 92, 273
- Evans C. R., Hawley J. F., 1988, *ApJ*, 332, 659
- Falle S. A. E. G., 1991, *MNRAS*, 250, 581
- Frigo M., Johnson S. G., 2005, *Proc. IEEE*, 93, 216
- Fryxell B. et al., 2000, *ApJS*, 131, 273
- Gaspari M., Churazov E., 2013, *A&A*, 559, A78
- Gaspari M. et al., 2018, *ApJ*, 854, 167
- Goldbaum N. J., Krumholz M. R., Forbes J. C., 2015, *ApJ*, 814, 131
- Goldbaum N. J., Krumholz M. R., Forbes J. C., 2016, *ApJ*, 827, 28
- Hernquist L., 1990, *ApJ*, 356, 359
- Hockney R., Eastwood J., 1988, *Computer Simulation Using Particles*. Taylor & Francis, New York
- Hopkins P. F., 2015, *MNRAS*, 450, 53
- Huang J., Greengard L., 1999, *SIAM J. Sci. Comput.*, 21, 1551
- Jiang Y.-F., Belyaev M., Goodman J., Stone J. M., 2013, *New Astron.*, 19, 48
- Jin S., Xin Z., 1995, *Commun. Pure Appl. Math.*, 48, 235
- Kestener P., Château F., Teyssier R., 2010, *Algorithms and Architectures for Parallel Processing (ICA3PP'10)*. Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, p. 281
- Khatri R., Gaspari M., 2016, *MNRAS*, 463, 655
- Kim J.-h. et al., 2016, *ApJ*, 833, 202
- Kravtsov A. V., Klypin A. A., Khokhlov A. M., 1997, *ApJS*, 111, 73
- Lau E. T., Gaspari M., Nagai D., Coppi B., 2017, *ApJ*, 849, 54
- Lee D., 2013, *J. Comput. Phys.*, 243, 269
- Löhner R., Morgan K., Peraire J., Vahdati M., 1987, *Int. J. Numer. Methods Fluids*, 7, 1093
- Lukat G., Banerjee R., 2016, *New Astron.*, 45, 14
- Martin D. F., Colella P., 2000, *J. Comput. Phys.*, 163, 271
- Mignone A., Zanni C., Tzeferacos P., van Straalen B., Colella P., Bodo G., 2012, *ApJS*, 198, 7
- Miyoshi T., Kusano K., 2005, *J. Comput. Phys.*, 208, 315
- Müller E., Steinmetz M., 1995, *Comput. Phys. Commun.*, 89, 45
- Nagai D., Vikhlinin A., Kravtsov A. V., 2007, *ApJ*, 655, 98
- Navarro J. F., Frenk C. S., White S. D. M., 1996, *ApJ*, 462, 563
- NVIDIA, 2017, *CUDA C Programming Guide*, 8.0 edn. NVIDIA Corporation. Available at: <http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>
- Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P., 2007, *Numerical Recipes. The Art of Scientific Computing*, 3rd edn. Cambridge Univ. Press, Cambridge
- Ricker P. M., 2008, *ApJS*, 176, 293
- Robertson B., Bullock J. S., Cox T. J., Di Matteo T., Hernquist L., Springel V., Yoshida N., 2006, *ApJ*, 645, 986
- Roe P., 1981, *J. Comput. Phys.*, 43, 357
- Ryu D., Ostriker J. P., Kang H., Cen R., 1993, *ApJ*, 414, 1
- Schive H.-Y., Tsai Y.-C., Chiueh T., 2010, *ApJS*, 186, 457
- Schive H.-Y., Zhang U.-H., Chiueh T., 2012, *Int. J. High Perform. Comput. Appl.*, 26, 367
- Schive H.-Y., Chiueh T., Broadhurst T., 2014a, *Nat. Phys.*, 10, 496
- Schive H.-Y., Liao M.-H., Woo T.-P., Wong S.-K., Chiueh T., Broadhurst T., Hwang W.-Y. P., 2014b, *Phys. Rev. Lett.*, 113, 261302
- Shukla H., Schive H.-Y., Woo T.-P., Chiueh T., 2011, Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11). ACM, New York, NY, USA, p. 37:1
- Smith B. D. et al., 2017, *MNRAS*, 466, 2217
- Springel V., 2005, *MNRAS*, 364, 1105
- Springel V., 2010, *MNRAS*, 401, 791
- Stone J. M., Gardiner T., 2009, *New Astron.*, 14, 139
- Stone J. M., Gardiner T. A., Teuben P., Hawley J. F., Simon J. B., 2008, *ApJS*, 178, 137
- Teyssier R., 2002, *A&A*, 385, 337
- Toro E. F., 2009, *Riemann Solvers and Numerical Methods for Fluid Dynamics. A Practical Introduction*, 3rd edn. Springer-Verlag, Berlin
- Trac H., Pen U.-L., 2003, *PASP*, 115, 303
- Turk M. J., Smith B. D., Oishi J. S., Skory S., Skillman S. W., Abel T., Norman M. L., 2011, *ApJS*, 192, 9
- van Leer B., 1979, *J. Comput. Phys.*, 32, 101
- van Leer B., 2006, *Commun. Comput. Phys.*, 1, 192
- Wang P., Abel T., Kaehler R., 2010, *New Astron.*, 15, 581
- White C. J., Stone J. M., Gammie C. F., 2016, *ApJS*, 225, 22
- Woodward P., Colella P., 1984, *J. Comput. Phys.*, 54, 115
- Zhang U.-H., Schive H.-Y., Chiueh T., 2018, *ApJS*, 236, 50
- ZuHone J. A., 2011, *ApJ*, 728, 54

## APPENDIX A: POISSON SOLVER

The Poisson solver of GAMER-2 on the refined patches is substantially different from that of GAMER-1. To smooth out the gravitational potential across patch boundaries, GAMER-2 adds several buffer zones around each patch (see Fig. 2), while GAMER-1 adopts a multilevel relaxation scheme to reduce the pseudo-mass sheets on the patch boundaries (Schive et al. 2010). To compare their accuracy, we calculate the potential of a Hernquist profile (Hernquist 1990):

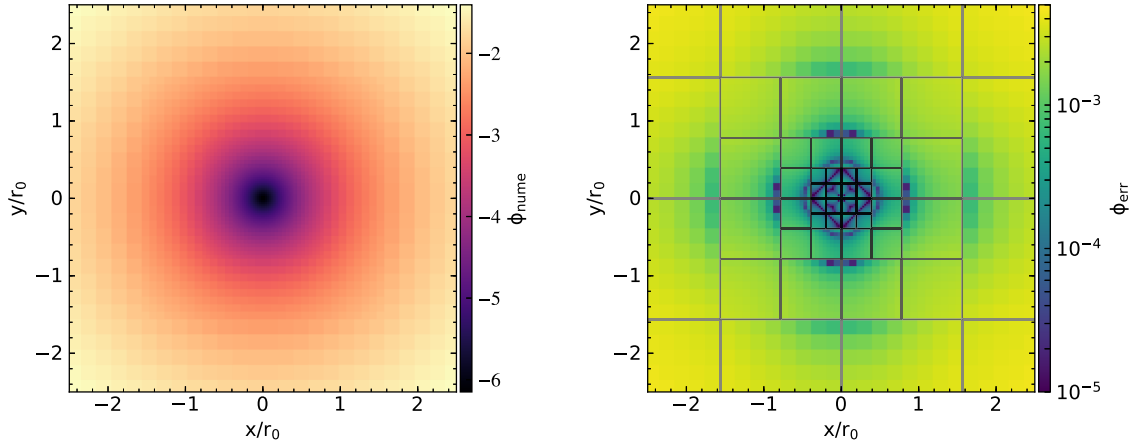
$$\rho(r) = \frac{\rho_0}{r/r_0(1 + r/r_0)^3}, \quad (\text{A1})$$

where  $r_0$  and  $\rho_0$  are the characteristic radius and density, respectively. This profile has a finite mass  $M = 2\pi r_0^3 \rho_0$  and an analytical form of potential:

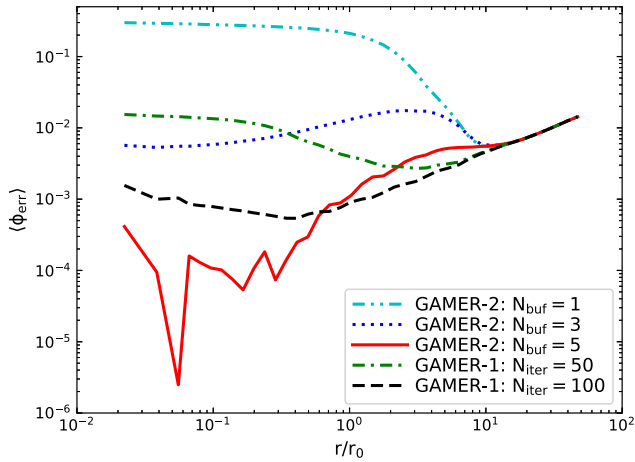
$$\phi_{\text{anal}}(r) = -\frac{GM}{r + r_0}, \quad (\text{A2})$$

where  $G$  is the gravitational constant. We adopt  $G = r_0 = \rho_0 = 1$ . The computational domain is cubic with a length  $L = 100$  and a  $64^3$  root grid. A cell on level  $l$  is flagged for refinement if its density exceeds  $10^{-2} \times 4^l$ , and we enable six refinement levels to well resolve  $r_0$  by a maximum resolution of  $\sim 2.4 \times 10^{-2}$ . Isolated boundary conditions for gravity are adopted.

Fig. A1 shows the gravitational potential on a central  $5r_0$  slice evaluated by GAMER-2 using five buffer zones. The left- and right-hand panels show the numerical results  $\phi_{\text{nume}}$  and the corresponding relative errors,  $\phi_{\text{err}} \equiv |(\phi_{\text{nume}} - \phi_{\text{anal}})/\phi_{\text{anal}}|$ , respectively. The relative errors within  $r \lesssim r_0$  are found to be as low as on the order of  $10^{-3}$ – $10^{-5}$ , although numerical artefacts introduced by the patch interfaces are still present.



**Figure A1.** Central slice of the gravitational potential of a Hernquist profile. The left-hand panel shows the numerical results evaluated by GAMER-2 using five buffer zones and six refinement levels. The right-hand panel shows the corresponding relative errors by comparing with the analytical solution, overlaid with AMR patch outlines.



**Figure A2.** Volume-weighted numerical errors as a function of radius for computing the potential of a Hernquist profile. We compare the errors of different schemes: GAMER-2 with 1 (dashed-double-dotted line), 3 (dotted line), and 5 (solid line) buffer zones, and GAMER-1 with 50 (dashed-dotted line) and 100 (dashed line) V-cycle iterations and sibling relaxation steps (see Schive et al. 2010 for details). GAMER-2 with five buffer zones is found to provide the most accurate solution within  $r \lesssim r_0$ .

Fig. A2 shows the volume-weighted radial profile of the numerical errors of different schemes. We compare GAMER-2 with  $N_{\text{buf}} = 1, 3$ , and 5 buffer zones and GAMER-1 with  $N_{\text{iter}} = 50$  and 100 V-cycle iterations and sibling relaxation steps (see Schive et al. 2010 for details). It shows that in these cases the numerical accuracy improves with a larger number of  $N_{\text{buf}}$  or  $N_{\text{iter}}$ , and GAMER-2 with  $N_{\text{buf}} = 5$  provides the most accurate solution within  $r \lesssim r_0$ . We also find that  $N_{\text{buf}} > 5$  does not improve accuracy further. In addition, the gravitational potential of a patch of  $8^3$  cells and five ghost zones on each side consumes  $\sim 46$  KB memory per patch in double precision, which can just fit into the small but fast shared memory of modern GPUs. Therefore, we adopt  $N_{\text{buf}} = 5$  by default.

This paper has been typeset from a  $\text{\LaTeX}$  file prepared by the author.