

NCMAS 2020 Resource Request

Title: Star Formation and Feedback in a Turbulent Interstellar Medium

Lead CI: M. Krumholz (ANU)

Project: jh2

1. Scalability

1.1. General notes on scaling

We plan to use four codes for this project, and we discuss the scalability of each of them in turn. However, before reporting on scalability, we provide two caveats. First, the tests we have performed are all on Raijin or on comparable hardware with fewer CPUs per node than Gadi. We have extrapolated to Gadi assuming that parallel efficiencies will be comparable when using the same number of nodes. Second, the three primary codes we plan to use, **ORION**, **enzo**, and **GIZMO**, are all adaptive, and for this reason scalability is much more problem-dependent than for non-adaptive codes. **ORION** and **enzo** are Eulerian adaptive mesh refinement (AMR) gas dynamics codes that dynamically add higher resolution grids to meet user-defined accuracy goals, and scalability inevitably depends on the number and placement of such grids, which change from problem to problem and even over time within a single calculation. Similarly, **GIZMO** is an arbitrary Lagrangian-Eulerian code that provides adaptivity by organising the fluid elements it follows into a tree-based hierarchy built on the local CFL condition; within the tree, different groups of fluid elements advance on different time steps. Thus scalability depends on the distribution of CFL time steps, and is much better for calculations where most fluid elements advance on similar time steps than for those where the cost is dominated by a small number of elements that require much smaller time steps than the rest. Given these caveats, the best guide to scalability and cost for a particular problem is generally just experience having used that code on similar problems in the past. Despite this caveat, we have performed some general scaling studies as a rough guide, which we report below.

1.2. ORION

Our most recent performance studies for **ORION** are detailed in [Rosen et al. \(2017\)](#), and all the figures included here are reproduced from that paper. We refer to the paper for full details of the testing, and summarise only the most salient parts of the results here. We performed all these studies on the NASA Ames Research Center supercomputer **Pleiades**, using

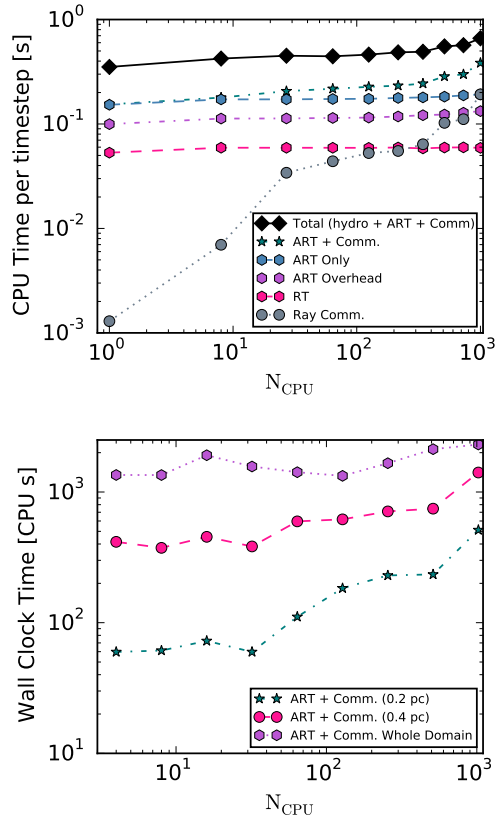


Fig. 1.— Results of a weak scaling test (top) and a strong scaling test (bottom) with **ORION**, from 1 to 1024 cores. The top panel (weak scaling) shows the CPU time required to advance the calculation one time step, while the bottom (strong scaling) shows total CPU-seconds, so in both panels perfect scaling corresponds to a flat line. In the top panel, black diamonds show the total time, and other symbols show the times taken by individual sub-parts of the code (see [Rosen et al. 2017](#) for details). In the bottom panel, the different symbols show the computational cost of the ray-tracing step depending on the distance traversed by the rays; the entire domain is 1 pc in length.

the Sandy Bridge nodes, which have hardware comparable to the Raijin Sandy Bridge nodes (16 cores/node, FDR infiniband interconnects). The total cost of most ORION computations is completely dominated by the radiative transfer step, and that step is also the hardest to parallelise due to the non-local nature of radiative transfer; thus our scaling study focuses on that part of the code. We are in the process of developing a new radiative transfer methodology that will be accelerated by GPU, but since this is in development we cannot report performance for it yet. We therefore report on the code in its current state.

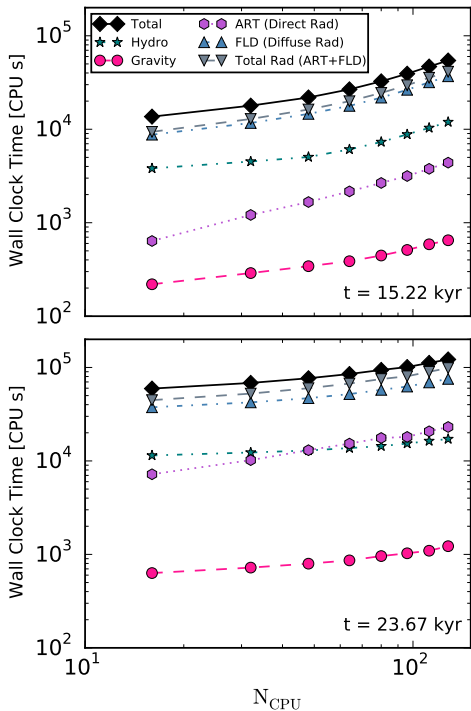


Fig. 2.— Scaling results for ORION is a realistic calculation; a flat line corresponds to perfect scaling. The upper and lower panels correspond to two different times in the simulation, which have different numbers and geometries of adaptive grids.

and flux-limited diffusion radiation). This illustrates the dependence of the scaling on the particular problem. At the later time (lower panel), the efficiency loss in going from 16 to 256 cores (1 to 16 nodes) is moderate, $\approx 50\%$, while the earlier time the calculation is only $\approx 30\%$ efficient. The difference is that the earlier time slice has a smaller number of grids to distribute across processors, producing worse load balancing. Fortunately, our experience in Armillotta et al. (2018) was that we the grid layout was somewhat more forgiving than in the Rosen et al. (2016) problem, so that we retained 50% efficiency out to 32 nodes. One Gadi, this translates to 1536 cores.

1.3. enzo

We tested the strong scalability of `enzo` using a simulation of an M83-like barred spiral galaxy similar to the one we plan to carry out in the coming year (see Project Description). We simulated the evolution of this galaxy on the Raijin Broadwell nodes (28 cores/node) for 1

We performed three studies: weak scaling in an idealised problem, strong scaling in an idealised problem, and scaling in a real example problem. Figure 1 shows the results of the weak and strong scaling tests in idealised problems. For the weak scaling test, we placed a single radiation source in the centre of a $(1 \text{ pc})^3$ region of dust and gas, resolved by 32^3 cells, for each CPU used in the test. For the strong scaling test, we used a fixed grid of 256^3 cells to resolve a $(1 \text{ pc})^3$ region with a single point source at its centre. Neither test used adaptivity. The figure shows that, in this idealised test, ORION has excellent weak scaling; efficiency is $> 50\%$ even at 1024 cores (16 nodes). For strong scaling, we find a similar result if the rays traverse a significant portion of the computational domain; efficiency is better than 50% even at 1024 cores (16 nodes). On the other hand the efficiency is worse if the region over which the ray trace is being done is a small portion of the computational domain, so that some processors are idle.

Figure 2 shows a scaling test conducted in a real, published scientific application (Rosen et al. 2016). This calculation does use adaptivity, and has been evolved for a significant time so that the grid layout is that produced naturally by the system being simulated (in this case collapse of a gas cloud to form a massive star). Black points show the total cost, in CPU-seconds, per time step advance, while other symbols show the cost for individual portions of the code (hydrodynamics, gravity, adaptive ray trace, and flux-limited diffusion radiation). This illustrates the dependence of the scaling on the particular problem. At the later time (lower panel), the efficiency loss in going from 16 to 256 cores (1 to 16 nodes) is moderate, $\approx 50\%$, while the earlier time the calculation is only $\approx 30\%$ efficient. The difference is that the earlier time slice has a smaller number of grids to distribute across processors, producing worse load balancing. Fortunately, our experience in Armillotta et al. (2018) was that we the grid layout was somewhat more forgiving than in the Rosen et al. (2016) problem, so that we retained 50% efficiency out to 32 nodes. One Gadi, this translates to 1536 cores.

Myr of simulation time with 6 AMR levels, using 112 to 896 cores (4 to 32 nodes). In the test simulation the peak resolution was 15 pc, and thus a factor of 2 worse than what we plan to use for our final production runs; going to higher resolution should either leave the scalability unchanged or improve it, since there will be more work to distribute. We show the total CPU time required for this computation (excluding IO, which is negligible in a production calculation) as a function of processor count in [Figure 3](#).

The test shows *better* than perfect scaling out to 448 cores (16 nodes) on Raijin, and perfect from 448 to 896 (32 nodes). Seemingly better than perfect scaling is possible due to the subtleties of AMR: user-specified accuracy goals require adding higher-resolution grids at a minimum set of locations, but in practice the code refines a somewhat larger volume, auto-tuning grid placement and distribution to minimise communication overhead. The efficiency of the auto-tuning process depends on the CPU count. As a result, the higher CPU count calculations shown in [Figure 3](#) are able to achieve the same accuracy goal using somewhat fewer total cell advances, and with less communications overhead due to better grid distribution. This result demonstrates that we can efficiently use up to 32 Raijin nodes for this problem; assuming a similar node count for Gadi, this is 1536 cores.

1.4. GIZMO

We have measured the strong scaling of GIZMO on the Sandy Bridge nodes of Raijin (16 cores/node) by running a simulation with a resolution of 300^3 (27 million zones), using the same setup as that in [Gentry et al. \(2019, see Progress Report\)](#): a series of clustered supernovae exploding in the interstellar medium. For the test we run the problem for the period between the first and second supernovae, when the hot bubble produced by the first supernova is expanding.

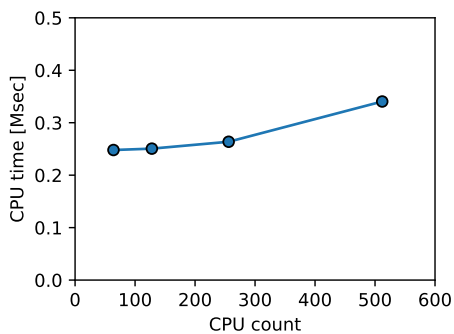


Fig. 4.— Strong scaling results for GIZMO running on Raijin. The y axis shows CPU time required to do a fixed amount of work, so perfect scaling corresponds to a flat line.

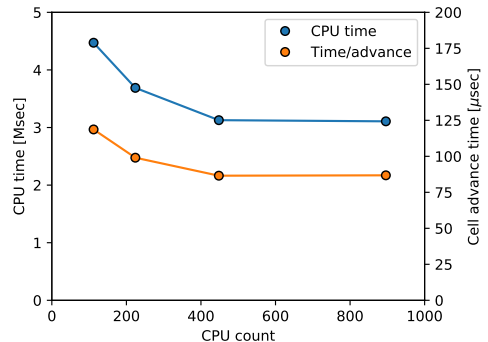


Fig. 3.— Results for our test of strong scaling in `enzo`. The y axis shows the total amount of CPU time (blue, left axis) taken to perform a fixed amount of work and the CPU time required to advance a single cell through one time step (orange, right axis); in both cases perfect scaling corresponds to a flat line, while a decreasing line as we have found corresponds to *better* than ideal scaling; such a result is possible for reasons explained in the main text.

We ran the problem using 64 to 512 CPUs (4 to 32 nodes); for this test we use nodes with 64 GB of memory each because on the 32 GB nodes the problem does not fit in memory at the smallest CPU count. However, this should not affect scalability. We show the results of our strong scaling test in [Figure 4](#). As the plot shows, we have near-perfect scalability to 256 cores (16 nodes), and extremely good scalability out to 512 cores (32 nodes). The efficiency in scaling from 64 to 512 cores is 73%. On Gadi, we therefore expect this level of efficiency up to 1536 cores.

1.5. SLUG

SLUG is unlike all the other codes we plan to use this year in that, as a Monte Carlo code, it is embarrassingly parallel: every realisation of a stellar population, and calculation of that stellar population’s light passing through the ISM, is

Calculation	Code	# Runs	Cores	CPU time [kSU/run]	Wall time [days/run]	SU cost [kSU, total]
Galactic winds	ORION	9	768	290	16	2610
Galactic chemodynamics						
galaxy stage	enzo	2	1536	450	12	900
zoom-in stage	ORION	2	1536	1300	35	2600
Galactic centres	GIZMO	2	1536	780	21	1560
Stellar populations	SLUG	1	3072	2000	27	2000
Subtotal						9700
+10% overhead						970
Total						10640

Table 1: Summary of simulations and costs.

independent and serial. The code is implemented to allow either OpenMP or MPI-based parallelism, but inter-process communication is limited to updating the count of number of Monte Carlo trials completed. For this reason we do not present scalability data for **SLUG**, since the scaling is trivially perfect.

2. Job resources

Our total request for all runs is 10.6 MSU. Costs of individual proposed calculations are listed in [Table 1](#), and explained below. Note that we add a 10% overhead. Roughly half of this is for setup, testing, and contingency. The remainder is because storage limitations on the scratch file system mean that we can only write checkpoints roughly once per hour, so for a calculation running for the 24 hour wall clock limit, we typically lose the last few percent of our run time after our last checkpoint write due to hitting the wall clock limit.

2.1. General notes

In general the CPU cost of an adaptive hydrodynamics or MHD calculation can be expressed as

$$t_{\text{CPU}} = \frac{t_{\text{adv}}}{\epsilon} \left(\frac{t_{\text{run}}}{\Delta t_0} \right) \sum_{i=0}^{L_{\text{max}}} 2^i N_i \quad (1)$$

where t_{CPU} is the total CPU time required, t_{run} is the duration of the simulation in physical time, Δt_0 is the root time step taken at the coarsest resolution level, t_{adv} is the time required to advance a cell / fluid element through a single time step, L_{max} is the maximum refinement level (either AMR level for an AMR code like **ORION** or **enzo**, or level of time step hierarchy for a tree code like **GIZMO**), N_i is the number of resolution elements on level i , and ϵ is the parallel efficiency. The time step Δt_0 is generally set by the CFL condition, which we can express as $\Delta t_0 = C \Delta x_0 / v_{\text{sig}}$, where C is the CFL number of the algorithm, Δx_0 is the spatial resolution on the coarsest level of the hierarchy (cell size for an AMR code, twice the local smoothing length h for an SPH-like code such as **GIZMO**), and v_{sig} is the fastest signal speed. Thus we express the computational costs for all our calculations using **ORION**, **enzo**, and **GIZMO** as

$$t_{\text{CPU}} = \frac{t_{\text{adv}}}{\epsilon C} \left(\frac{t_{\text{run}} v_{\text{sig}}}{\Delta x_0} \right) \sum_{i=0}^{L_{\text{max}}} 2^i N_i. \quad (2)$$

We next discuss these factors for each planned simulation.

2.2. Project 1: Radiatively-driven galactic winds

As discussed in the project proposal, the basic setup of the simulations we plan to perform matches that we previously used in Krumholz & Thompson (2012, 2013) and Wibking et al. (2018) to study the radiation Rayleigh-Taylor instability (RRTI): a rectangular domain with a reflecting boundary at the bottom, an open boundary at the top, and periodic boundaries in the horizontal direction. The size of the domain is naturally expressed in units of the dimensionless scale length $h_* = c_{s,*}^2/g$ for the system, where $c_{s,*}$ is the isothermal sound speed at the top of the atmosphere (dictated by the radiation flux) and g is the gravitational acceleration; times are naturally expressed in units of $t_* = h_*/c_{s,*}$.

In the horizontal directions, the domain must be large enough to contain the largest unstable modes; the exact value of this scale depends on system parameters, but is typically $\approx 50h_*$. In the vertical direction the box must be large enough that gas that is thrown upward by radiatively-driven turbulent motion, but not actually unbound, has room to travel upward and then fall back without encountering the top of the box. Based on our previous experience, this requires a box that is at least twice as large in the vertical direction as the horizontal, and four or eight times is better. For this reason, our fiducial setup will be a box that is $64h_* \times 64h_* \times 512h_*$. Our maximum resolution is dictated by the need to resolve h_* at least marginally, so that we can capture the onset of the RRTI at small scales; this required a finest resolution of $\approx h_*/4$. We therefore plan to use three levels of AMR with a base grid resolution of $\Delta x_0 = h_*$, giving a maximum resolution $\Delta x_2 = h_*/4$. We will refine the entire bottom $64h_*$ of the box in the vertical direction to the maximum AMR level, the next $64h_*$ to AMR level 1 (resolution $\Delta x_1 = h_*/2$), and leave the top $128h_*$ at AMR level 0 ($\Delta x_0 = h_*$). Thus the number of cells on each level are $N_{0,1,2} = (64 \times 64 \times 512, 128 \times 128 \times 256, 256 \times 256 \times 256)$.

Based on our previous experience, the total time required for the simulation to reach statistical steady state is $t_{\text{run}} \approx 100t_*$, and the highest speeds we produce, in low-density gas being driven into highly-supersonic motion by radiation forces, as $v_{\text{sig}} \sim 20c_{s,*}$. We generally run ORION with a CFL number $C = 0.4$; the algorithm is formally stable up to $C = 0.5$, but using 0.4 gives a margin of safety. Based on our testing above, we plan to use 16 Gadi nodes (768 cores) for our standard case, which gives a parallel efficiency $\epsilon \approx 0.75$, though once Gadi becomes available and we can test the efficiency of our code, we may increase or decrease the node count. Finally, our measured cell advance time, again using the test from Rosen et al. (2017) and described above, is $t_{\text{adv}} \approx 1$ msec. Inserting these factors into equation 2, we arrive at a cost of 290 kSU per calculation. In order to map out parameter space, we plan to carry out a total of 9 calculations, at three different values of the Eddington ratio (ratio of laminar radiation force to gravity) times three different values of optical depth, for a total cost of 2.6 MSU.

2.3. Project 2: Galactic chemodynamics

This project uses two computational stages, a galactic one done with `enzo` and a zoom-in star formation stage run with ORION. For the galaxy stage of the simulation, our parallel efficiency $\epsilon \approx 1$ up to 32 nodes on Raijin, and we will run for $t_{\text{run}} = 700$ Myr, roughly 3 galaxy rotation periods at the galactocentric radius of the Sun, which is long enough to achieve statistical steady state in the ISM. For `enzo` we use a CFL number $C \approx 0.3$. Our base grid has cells of size $\Delta x_0 = 1$ kpc, and the highest signal speeds are in supernova-heated gas where the sound speed is very high, and $v_{\text{sig}} \approx 1000$ km s⁻¹. The cell advance time determined from our tests on Raijin is $t_{\text{adv}} \approx 90$ μ sec (see Figure 3). In production calculations we will use $L_{\text{max}} = 7$, corresponding to a maximum resolution of ≈ 7 pc. The most uncertain part of our time estimate is the cell counts, which are determined on the fly as the system evolves; the counts in the tests presented in Section 1.3 provide only a very limited guide, since they are obtained at an early stage in the simulation before it settles to steady state. That said, our

past experience with simulations of this type, as presented in [Fujimoto et al. \(2018, 2019\)](#), is that $N_i \sim 3 \times 10^7$ cells per level are typical on the finer levels (which dominate the total cost) at late times. Assuming this to be the case, inserting the various factors into [equation 2](#) gives a total cost estimate of 450 kSU per computation. We will use 32 Gadi nodes; if our parallel efficiency on Gadi proves worse than expected, we can reduce this to 16 nodes.

For the zoom-in stage using ORION, our measured parallel efficiency is $\epsilon \approx 0.5$ up to 32 nodes; though our efficiency is higher at 16 nodes, we accept a somewhat lower efficiency in order to keep the total wall time of the simulations reasonable. As noted in the pervious section, our measured cell advance times as determined are $t_{\text{adv}} \approx 1$ msec. (ORION’s parallel efficiency is lower than *enzo*’s, and its cell advance times larger, because radiative transfer is much harder to parallelise efficiently than gravity or hydrodynamics.) Following [Armillotta et al. \(2018\)](#), we will run our simulations for $t_{\text{run}} \approx 3$ Myr, zooming in on a cubical region 384 pc on a side with a based grid of 512^3 cells, giving a base grid resolution $\Delta x_0 = 0.75$ pc. Our fastest signal speeds are in photoionised gas, where $v_{\text{sig}} \approx 10$ km s⁻¹, so our coarse time step size is $\Delta t_0 \approx 22$ kyr. We use 6 AMR levels on top of this, reaching a maximum resolution 0.01 pc. We take estimates of cell counts from [our first generation study, Armillotta et al. \(2018\)](#), where we found $N_i \approx 512^3$ on all levels. Inserting these factors into [equation 2](#) (using the exact cell counts from [Armillotta et al.](#)), we obtain a total cost of 1.3 MSU.

We plan to run two simulations, one of a galaxy like the Milky Way, and one using a setup similar to M83, which has a stronger bar than the Milky Way and thus represents an interesting contrast that will let us study the effects of bars. Thus the total cost of both stages for two galaxies is 3.5 MSU.

2.4. Project 3: The life cycle of galaxy centres

Our galactic centre simulations uses GIZMO, and our practical experience with the simulations presented in [Armillotta et al. \(2019\)](#) are that the cost tends to be dominated by the coarsest rather than the finest levels of the hierarchy, the opposite of the ORION collapse calculations. Thus to good approximation we can consider only the coarsest level. In the tests shown in [Figure 4](#), we find a parallel efficiency $\epsilon = 0.73$ for 32 nodes. Our advance time is $t_{\text{adv}} \approx 300$ μ sec, slower than *enzo* due to the extra work of neighbour-finding in a Lagrangian method, but faster than ORION due to the lack of radiative transfer. At our target mass resolution $\Delta M = 300 M_{\odot}$, we will have $N_0 \approx 4 \times 10^6$ Lagrangian fluid elements. We run for a total of 350 Myr, but during the first 300 Myr we turn off self-gravity and simply let the simulation run to reach statistical steady state; because we do not reach high densities during this phase, the smoothing scale h is large, and thus the time steps are large and the cost is negligible. Almost all of the cost is incurred after we turn on self-gravity, and during this phase we run for a time $t_{\text{run}} = 50$ Myr. The peak density we reach is $n \approx 10^8$ H atoms cm⁻³, and the corresponding spatial resolution is $h \approx (3\Delta M/4\pi n m_{\text{H}})^{1/3} \approx 0.03$ pc, corresponding to an effective resolution $\Delta x \approx 0.06$ pc. The peak signal speed is in supernova-heated gas, which due to its high temperature has a sound speed $v_{\text{sig}} \approx 1000$ km s⁻¹. Finally, our GIZMO runs use a CFL number $C = 0.5$. Inserting these factors into [equation 2](#), we obtain a cost per simulation of 780 kSU. As discussed in the Project Description, we plan to carry out two simulations, one with an alternate Milky Way potential, and one to mimic M83. We plan to use 32 Gadi nodes for each run.

2.5. Project 4: Stochasticity-robust stellar population inference

As explained above, our Monte Carlo stellar population synthesis calculations with SLUG are embarrassingly parallel. Each Monte Carlo realisation involves two steps. First, we generate the spectrum of a stellar population, and then we calculate the propagation of the light through the ISM and predicts the observable nebular line emission; this second step

uses the photoionisation and radiative transfer code `cloudy` (Ferland et al. 2013), which SLUG calls as a subroutine. Generating a stellar population requires at most a few seconds, so the cost is dominated by the radiative transfer step. This is variable, since `cloudy` uses an iterative method to solve the simultaneous equations of radiative transfer and statistical and chemical equilibrium for a large set of atoms and ions, and the number of iterations required for convergence depends on problem parameters such as the gas density and the intensity of the ionising radiation. Based on testing on the Avatar cluster at ANU, which has processors similar to those on the Raijin Sandy Bridge nodes, we find an average of ≈ 4 minutes on a single core.

The Monte Carlo library we seek to construct will be used to produce a Gaussian kernel density estimate (KDE) of the joint distribution of physical properties (star cluster age and mass, ISM elemental abundance) and light output. The KDE requires a bandwidth, which must be chosen large enough that the results are not dominated by the shot noise of the Monte Carlo realisation, but which we want to make as small as possible to ensure that any uncertainties in the metal abundances we derive will be dominated by the inherent stochasticity of the stellar population or by measurement errors, rather than by the resolution of the KDE. The required resolution is difficult to estimate *a priori*, since stellar properties are highly-correlated, but our experiments with star clusters showed that 10^7 realisations is sufficient to achieve errors of ≈ 0.1 dex in reconstructed star cluster mass and age, smaller than the typical uncertainties induced by stochastic sampling (Krumholz et al. 2015b). For the present application we anticipate needing a somewhat larger number of realisations, because we are adding an extra dimension (metal abundance), but we can compensate by sampling a much smaller range of stellar population ages, since only stars $\lesssim 5$ Myr old produce observable levels of nebular emission. Based on preliminary testing, we anticipate needing a library of $\approx 3 \times 10^7$ realisations; using the cost estimate above, the total amount of CPU time required is ≈ 2 MSU. In practical terms, we will generally use 64 nodes at a time to keep the total wall clock time reasonable, though of course we can vary this as needed given the embarrassingly parallel nature of the calculation.

3. Storage

3.1. Memory

Our calculations will fit comfortably within the standard 4.0 GB / core memory footprint of a standard node on Raijin or Gadi. We are CPU-bound, not memory-bound.

3.2. Scratch space on /short

We require significantly more than the standard 1 TB quota on the `/short` file system due to the need for checkpoints. All our codes run for longer than the queue wall clock limit, and therefore checkpoint regularly to allow restarts. These checkpoints also allow analysis of intermediate states in the hydrodynamic simulations. While SLUG checkpoints are modest in size and do not necessitate any special treatment, those produced by ORION, `enzo`, and GIZMO are substantially larger, because they must contain the full state of the calculation in order to allow restart. We estimate the storage requirements as follows:


Project 1: galactic winds. For the grid configuration described above, we have 23 million cells. In each cell we store the density, three components of momentum density, total energy, three components of magnetic field, radiation energy density, and (once our new M1 method becomes available) three components of radiation flux, for a total of 12 components. Each is an 8-byte double, so the total size of a checkpoint is 2.2 GB, or 53 GB per 24 hours.

Project 2: galactic chemodynamics. The galactic stage of our calculations, using `enzo`, has $\approx 2 \times 10^8$ cells over all levels, and ≈ 40 quantities per cell: density, momentum density, total energy, and magnetic field, plus ≈ 30 passive scalars representing abundances of various elements. Thus each checkpoint is ≈ 64 GB in size. During the zoom-in phase using `ORION`, this grows slightly due to the addition of four radiation quantities, to ≈ 70 GB per checkpoint, or 1.7 TB per 24 hours.

Project 3: galactic centres. The `GIZMO` simulation snapshots are smaller, since they contain only 4×10^6 elements and no radiation quantities or passive scalars. For 8 quantities per resolution element (5 hydrodynamic variables and 3 positions), each checkpoint is ≈ 0.3 GB in size, or 6 GB per 24 hours.

Our practice is to offload data to `massdata` at the end of each pass through the queue, but this takes time, and we would like to be able to restart simulations immediately, without waiting for data transfer to complete. Our wall clock times are challenging for us to complete within a year, and having to wait for transfers to complete before submitting the continuation of a run would represent a significant bottleneck in our workflow. Consequently, we would like to have space to store several days worth of outputs at any given time, and therefore we request a `/scratch` quota of at least 10 TB, with 20 TB preferred if possible given the new `/scratch` file system. Our current quota is 10 TB, and we have hit it several times in 2019; we are asking for more time in 2020, and thus if possible would prefer a somewhat larger quota as well.

3.3. Long-Term Storage on `massdata`

 We must archive our outputs for the purposes of analysis. While we have some storage at `RSAA`, which we use to host subsets of the data for local analysis, transferring the entire set of outputs over the internet is not feasible. We therefore request space on `massdata` to archive our runs for later analysis. Multiplying the wall time estimates in [Table 1](#) by our estimates of the data production rates from each simulation in [Section 3.2](#), we estimate that our work will generate a total of 300 TB of data. However, we do not need to save every checkpoint permanently, as our checkpointing frequency is driven primarily by desire not to waste CPU cycles by having calculations terminate too long after checkpointing. For the purposes of analysis our experience is that saving every other checkpoint over most of the run is sufficient, so we request 150 TB of long term storage on `massdata`.

4. Algorithms and workflows

4.1. Algorithms and parallelism

`ORION`. `ORION` is a block-structured, sub-cycled AMR code for astrophysical radiation-MHD plus gravity. It is developed primarily by Mark Krumholz at ANU and collaborators at UC Berkeley, UT Austin, and Harvard in the US. The AMR scheme follows the approach of [Berger & Olinger \(1984\)](#) and [Berger & Colella \(1989\)](#): we decompose the domain into a base grid at the coarsest level, and recursively add finer, covering grid where needed. Levels advance hierarchically: we first advance all grids on the coarsest AMR level, denoted level 0, through a time step Δt_0 . Then we advance the next-finest grids (level 1) through two time steps of size $\Delta t_1 = \Delta t_0/2$, and perform a synchronisation step to ensure conservation of mass, momentum, and energy across the level 0-1 boundary. This process is repeated recursively, so each time we update AMR level 1 we update all level 2 grids twice and synchronise across the level 1-2 boundary, and so forth. We obtain ghost cells at the edges of grids on level i that do not have neighbours on level $i+1$ by coarsening cells on level $i+1$ if those exist, or by interpolating cells on level $i-1$ if not. The code uses MPI-based parallelism: grids are distributed across

processors using a knapsack algorithm to achieve optimal load balance, and grids exchange information at their mutual boundaries using point-to-point MPI communication, with a minimum of global synchronisation. For MHD, the exchange consists of passing information about ghost cells to neighbouring grids. For self-gravity and for the diffuse radiation part of the radiation solve, we use an MPI-based multigrid method, implemented in the **hypre** library¹. For the long characteristic radiation solve, we use the **HARM**² algorithm introduced by [Rosen et al. \(2017\)](#), which minimises communications overhead by using a non-blocking, opportunistic MPI method to overlap communication and computation.

enzo. The **enzo** code ([Bryan et al. 2014](#)) is a community-supported AMR fluid dynamics code specialised in cosmological and galaxy simulations. It solves the equations of ideal MHD plus gravity for a system consisting a fluid plus collisionless point masses, which can represent either stars or dark matter. It also includes subgrid prescriptions for radiative cooling, star formation, stellar feedback, including a detailed nucleosynthetic module that our group has added based on the **SLUG** stochastic stellar population synthesis code. The AMR implementation and parallelisation methods in **enzo** are essentially the same as those of **ORION** – [Berger & Olinger \(1984\)](#)-style AMR with MPI parallelism. There are two primary differences. First, rather than using the knapsack algorithm to distribute grids, **enzo** uses a space-filling Peano-Hilbert curve to minimise the placement of adjacent grids on different processors, thereby reducing interprocessor communication costs. Second, rather than using a multigrid method for self-gravity, **enzo** uses a Fourier method.

GIZMO. **GIZMO** is a meshless MHD plus gravity code using a smoothed particle hydrodynamics (SPH) arbitrary Lagrangian-Eulerian (ALE) scheme. It is based on discretisation of the volume using a smoothing kernel, similar to traditional SPH, but then uses a Riemann solver rather than the traditional SPH solver to obtain fluxes between zones. The zone interfaces can either be adjusted so as to maintain constant mass per zone (meshless finite mass) or to maintain constant volume per zone (meshless finite volume). Details of the algorithm are given in [Hopkins \(2015\)](#) and [Hopkins & Raives \(2016\)](#). The hydrodynamic update operates within a hierarchical tree format where particles are grouped based on neighbour and on the time step with which they must be updated. The same tree partitioning is used to update gravity in a tree scheme ([Springel 2010](#)). The tree is distributed in memory using an MPI paradigm.

SLUG. As explained above, since it is a Monte Carlo code, **SLUG** is embarrassingly parallel. It offers both openMP- and MPI-based parallel options, but the sole communication involved is updating the number of Monte Carlo trials completed.

4.2. Workflow

The workflows for Projects 1 – 3 are all the same, and are very simple: starting from an initial state, we run each simulation, writing checkpoints to `/scratch` approximately once per hour as the simulation runs. When the simulation terminates due to reaching the wall clock limit for the queue in which we are running, we restart from the last saved checkpoint and continue until we reach the desired run time. While the restart is running or waiting in queue, we move the checkpoints from the previous job from `/scratch` to `massdata`. We also move selected snapshots to local machines for analysis and visualisation; our primary analysis tool is **yt** ([Turk et al. 2011](#)), a Python-based visualisation and analysis platform for 3D data.

¹<http://computation.llnl.gov/projects/hypr-scalable-linear-solvers-multigrid-methods/software>

Our workflow for Project 4 is equally simple, since this project just requires accumulating a large Monte Carlo library. When we start **SLUG**, a run will go for the maximum allowed wall clock time before terminating. We will then move the output checkpoints containing the accumulated trials off Gadi to local machines after each run period, aggregating them into a single large library as we do so.

References

- Armillotta, L., Krumholz, M. R., Di Teodoro, E. M., & McClure-Griffiths, N. M. 2019, *MNRAS*, submitted, arXiv:1905.01309
- Armillotta, L., Krumholz, M. R., & Fujimoto, Y. 2018, *MNRAS*, 481, 5000
- Berger, M. J., & Colella, P. 1989, *J. Comp. Phys.*, 82, 64
- Berger, M. J., & Olinger, J. 1984, *J. Comp. Phys.*, 53, 484
- Bryan, G. L., Norman, M. L., O’Shea, B. W., et al. 2014, *ApJS*, 211, 19
- da Silva, R. L., Fumagalli, M., & Krumholz, M. 2012, *ApJ*, 745, 145
- Ferland, G. J., Porter, R. L., van Hoof, P. A. M., et al. 2013, *RMxAA*, 49, 137
- Fujimoto, Y., Chevance, M., Haydon, D. T., Krumholz, M. R., & Kruijssen, J. M. D. 2019, *MNRAS*, 487, 1717
- Fujimoto, Y., Krumholz, M. R., & Tachibana, S. 2018, *MNRAS*, 480, 4025
- Gentry, E. S., Krumholz, M. R., Madau, P., & Lupi, A. 2019, *MNRAS*, 483, 3647
- Hopkins, P. F. 2015, *MNRAS*, 450, 53
- Hopkins, P. F., & Raives, M. J. 2016, *MNRAS*, 455, 51
- Krumholz, M. R., Fumagalli, M., da Silva, R. L., Rendahl, T., & Parra, J. 2015a, *MNRAS*, 452, 1447
- Krumholz, M. R., McKee, C. F., & Klein, R. I. 2004, *ApJ*, 611, 399
- Krumholz, M. R., & Thompson, T. A. 2012, *ApJ*, 760, 155
- . 2013, *MNRAS*, 434, 2329
- Krumholz, M. R., Adamo, A., Fumagalli, M., et al. 2015b, *ApJ*, 812, 147
- Li, P. S., Martin, D. F., Klein, R. I., & McKee, C. F. 2012, *ApJ*, 745, 139
- Martin, D. F., Colella, P. C., & Graves, D. 2008, *JCP*, 227, 1863
- Rosen, A. L., Krumholz, M. R., McKee, C. F., & Klein, R. I. 2016, *MNRAS*, 463, 2553
- Rosen, A. L., Krumholz, M. R., Oishi, J. S., Lee, A. T., & Klein, R. I. 2017, *J. Comp. Phys.*, 330, 924
- Springel, V. 2010, *ARA&A*, 48, 391
- Turk, M. J., Smith, B. D., Oishi, J. S., et al. 2011, *ApJS*, 192, 9
- Wibking, B. D., Thompson, T. A., & Krumholz, M. R. 2018, *MNRAS*, 477, 4665