Master in High Performance Computing

Improve investigating gravitational-wave sources with the help of MPI and OpenMP interface

Supervisor(s):

Mario Spera and ,

Ivan Girotto

Candidate:

Natalia Nazarova

8th edition

2021–2022

Contents

Abstract

The most recent catalog of gravitational waves compiled by the LIGO-Virgo-KAGRA collaboration contains over 90 compact-binary coalescences, mostly binary black holes. The astrophysical interpretations of the detected sources are still uncertain, although the number is expected to rise significantly over the next few years.

From a theoretical point of view, one of the possible explanations for the formation of merging compact binaries is the isolated binary scenario. In this instance, two stars are gravitationally bound from the moment they are formed. During stellar evolution, stars move closer to one another, and at the end of their lives, they turn into compact remnants. As a result, a formed compact binary system has the potential to merge during the lifetime of the Universe.

Binary population-synthesis codes are tools that can evolve massive populations of single or binary stars from the formation moment to the compact remnants stage. They play an essential role in the investigation of this scenario. The evolution of one binary system does not require significant computational resources. However, to obtain sufficient statistics on compact object mergers, we require simulations of billions of binary systems with different initial conditions, stellar masses, evolutionary prescriptions, and metallicities.

In this project, I will implement a novel parallelization method for the SEVN code, a state-of-the-art population-synthesis code developed in SISSA and at the University of Padova. The final goal is to make the SEVN code run effectively on multi-node supercomputers. To accomplish that, I will use the Message Passing Interface (MPI) for inter-node

parallelization and the Open Multi-Processing (OpenMP) interface for intra-node parallelization. Furthermore, I will also implement an automatic and adaptive data loading algorithm to load input binaries in chunks. Finally, I will investigate the weak and strong scaling of the code on various computing machines.

The new code is expected to significantly speed up the evolution of binary systems, giving us the chance to investigate the formation of gravitational-wave sources in different stellar environments, possibly up to the regime of galaxies (i.e., billions of binaries).

Chapter 1

Introduction

1.1 Merging compact object binaries

On September 14, 2015, the LIGO-Virgo collaboration discovered the first direct evidence of merging compact-object binaries. The two ground-based interferometers of the LIGO [1] measured the effect of a passing GW, identified as GW150914. The signal was associated with the merger of two BHs with masses $M_1 = 36^{+5}_{-4}\, M_\odot$ and $M_2 = 29^{+4}_{-4}\, M_\odot$ [1, 2]. The event had numerous scientific implications and laid the groundwork for a new way to investigate the Universe. GW150914 detection confirmed the existence of BHs binaries that can merge within the Hubble time and revealed stellar BHs with masses $\gtrsim 30 M_\odot$.

GW150914 marked the beginning of a new chapter in astrophysics. It gave an unprecedented boost to the development of new theoretical models to investigate the formation and evolution of compact-object binaries and their progenitor stars, with the new objective of providing an astrophysical interpretation of GW sources.

The first run, O1[2], reported the first 3 detections, all BBH mergers. The second run, O2[3], detected 7 BBH mergers and the first BNS merger [3]. The third run, O3, splitted into

[1] At the time of the event, the Virgo detector was offline and undergoing a major upgrade.
[2] O1 run from 12 September 2015 to 19 January 2016
[3] O2 run from 30 November 2016 to 25 August 2017

O3a[4] and O3b[5] made the first detection of the merger of a NS with a BH. The most recent catalog of GWs that was compiled by the LIGO-Virgo-KAGRA collaboration contains 93 compact-binary coalescences, the vast majority of which are BBH [4, 5, 6, 7]. The catalog already contains numerous merging compact-object binaries that challenge even the most recent theoretical models. For example, GW190814 is an event with very asymmetric masses, a merger that most theoretical models find difficult to explain [8]. Furthermore, the lightest member is a compact mystery object with an uncertain nature: it can be the heaviest NS or the lightest BH ever observed, and its mass falls right into the lower mass gap. GW190521 is the event with the heaviest BHs, with at least one of the two falling in the upper mass gap [9, 10]. Its merger product, a BH with mass $148^{+28}_{-16}\,M_{\odot}$, is the first confirmation of the existence of intermediate-mass BHs. Almost all merger events obtained by the LIGO-Virgo-KAGRA collaboration are consistent with eccentricity equal to zero. The main subject of the debate is GW190521. Some work show that the binary system that produced GW190521 might be consistent with a non-zero eccentricity merger [11, 12, 13].

The scientific insights uncovered by these detections have already revolutionized multiple areas of physics and astrophysics. For example, GW170817 is an event associated with a merger of two NSs. It is the only event observed through GWs and throughout the electromagnetic spectrum, a crucial milestone for multi-messenger astronomy [14].

From a theoretical perspective, astrophysical interpretations of the detected GW sources are highly uncertain, although the number of GWs will increase significantly over the following years due to upcoming next-generation GW detectors (for example, the LIGO-India [15, 16], LIGO Voyager [17], the Einstein Telescope [18], Cosmic Explorer [19], the Laser Interferometer Space Antenna [20, 21, 22, 23, 24, 25, 26]).

The processes that can shrink a binary system so that it can merge within Hubble time via GWs are highly uncertain, as well as the mass spectrum of stellar BHs.

The time required for a system to merge is proportional to the total integrated GW

[4]O3a run from 1 April to 30 September 2019

[5]O3b run from 1 November 2019 until it was suspended on 27 March 2020 due to COVID-19

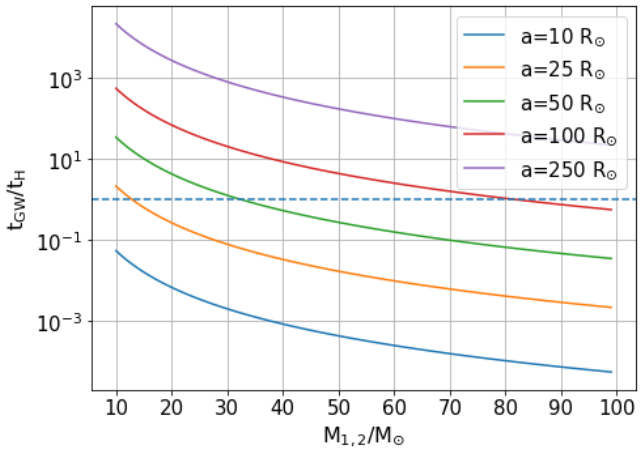

Figure 1.1: The time needed for a binary system to merge via GW emission as a function of a star's masses ($M_1 = M_2$). Time is shown in the units of Hubble time ($t_{\rm H}$) for systems with different semi-major axes a. The time is calculated using formula 1.1 for binary systems with circular orbits ($e = 0$).

luminosity during the coalescence phase:

$$\tau_{GW} = \frac{1}{4}\frac{a}{|\dot{a}|} = \frac{5}{256}\frac{c^5}{G^3}\frac{a^4}{M_1\,M_2\,(M_1+M_2)}(1-e^2)^{7/2} \tag{1.1}$$

Equation 1.1 demonstrates that the time required for a binary system to reach coalescence is highly dependent on the semi-major axis, the eccentricity of the orbit, and the masses of the two compact objects:

$$\tau_{GW} \sim a^4 M^{-3}\,(1-e^2)^{7/2} \tag{1.2}$$

According to formula 1.1, the time required for a binary system to merge depends on the orbit's eccentricity. Suppose that the binary's eccentricity equals zero, i.e., the binary has a circular orbit. In that case, we can estimate the merger time of binaries with different initial masses and the semi-major axis.

Figure 1.1 shows the time required for a binary system to merge via GW emission in units of the Hubble time ($t_{\rm H}$) for different initial masses and semi-major axes, in the case of circular orbits ($e = 0$). From figure 1.1 it is apparent that, to merge in a Hubble time and produce detectable sources of GWs, compact objects must have quite small initial semi-

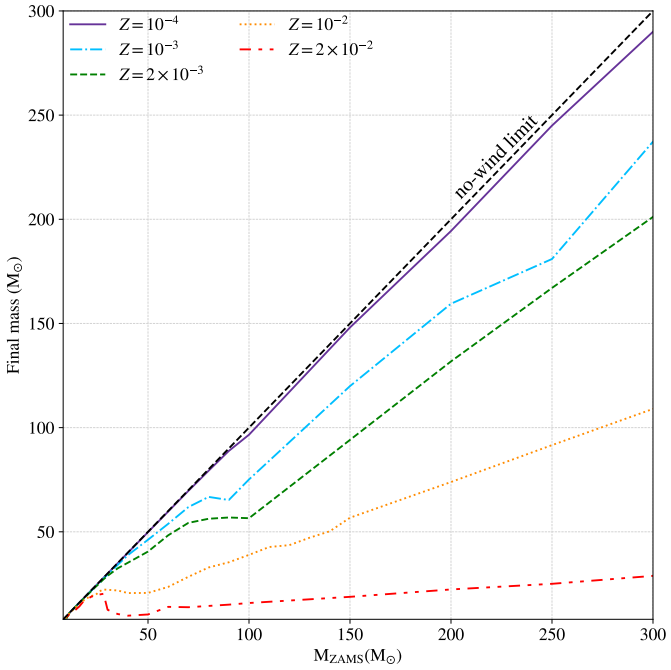

Figure 1.2: Final mass of the stars as a function of their initial mass, for different values of metallicity. The dashed line at 45 degrees corresponds to the no-wind limit (i.e., final mass = initial mass). The plot has been obtained using the SEVN code [34, 35, 36] couples with the PARSEC [37], [38], [39], [40], [41], [42] tracks and it is presented in detail in [33].

major axes. For example, two objects with mass 5 $M_\odot$ merge within $t_{\rm H}$ provided that the semi-major axis is below 10 $R_\odot$. Two objects with masses of 80 $M_\odot$ merge within the Hubble time if $a \lesssim 100 R_\odot$. The processes that can shrink a binary system so that it can merge within Hubble time via GWs are highly uncertain.

The final mass of a stellar remnant are also quite uncertain and depend on rotation, chemical composition, convection, dredge-up, wind mass loss, and nuclear reaction rates [27, 28, 29, 30, 31, 32]. Stellar winds have a central role in this concept since they drive mass loss over the lifetime of a star. Stellar winds, especially for massive stars, are uncertain, but their strength depends crucially on metallicity (e.g., [33]). Figure 1.2 shows the typical impact of different metallicity values on the stars' final mass, prior to the SN explosion. Stars at low Z retain significantly more mass than stars at higher Z. Thus the former can collapse and form significantly heavier BHs. Stars with the same initial mass but different amounts of metals will form a remnant whose mass differs by one order of magnitude.

The SN process, and so the link between progenitor stars and BHs, is also complex and very uncertain. The SN explosion starts with the collapse of the stellar structure, which is not sustained anymore by either the core's nuclear reactions or electron degeneracy pressure. The mechanism that triggers the explosion is still a matter of debate, but neutrinos and convection instability are thought to play a crucial role in the explodability of stars. State-of-the-art, three-dimensional hydrodynamical simulations of neutrino-driven SNe predict booming explosions for stars up to $25 - 30\,M_{\odot}$. However, such sophisticated multi-dimensional simulations are subject to significant uncertainties, and they are computationally intensive (e.g., [43, 44, 45])

Electromagnetic observations and modeling of systems containing BHs have led to speculation about potential "gaps" in the BH mass spectrum. The observations of X-ray binaries combined with Bayesian population modeling [46, 47, 48] suggest a dearth of compact objects with masses between 2.5 $M_{\odot}$ and 5 $M_{\odot}$ [49, 50, 51, 52]. The existence and nature of this gap is still to be determined [53]. GW observations can either limit the size of the lower mass gap or disprove its existence [54, 55, 56, 57]. On the high end of the BH mass spectrum, models of late stellar evolution phases predict the presence of another gap, generally referred to as the upper mass gap. Very massive stars reach a stage where either pair-instability (PISNe) or pulsational pair-instability (PPISNe) occur [58, 59, 60, 36, 61]. Both gaps may be probed using data from current ground-based GW interferometers and have been the target of a number of studies. The possible detection of BHs in the mass gaps might be explained by the merger of NS or BH binaries and not by the collapse of stars into a BH [62].

While there are many uncertainties about the evolution of isolated stars and their remnants, the number of uncertainties increases even more when we try to investigate the formation and evolution of binaries and how the latter evolution may end up forming merging compact-object systems.

So far, two main formation channels have been proposed to explain the formation of

merging compact-object binaries. The first of these is the dynamical scenario. In this channel, two compact objects approach after a single or a series of gravitational interactions with other stars or compact objects in dense stellar environments, such as globular and young dense star clusters [63, 64, 65, 66, 67], nuclear star clusters [68, 69], or disks of active galactic nuclei [70, 71, 72].Dynamical interactions in triple [73] or quadruple [74] stellar systems can also contribute to the shrinkage of orbital distances and facilitate mergers of compact objects.

The second channel for the formation of merging compact objects is the isolated channel. In the isolated binary scenario, two progenitor stars are bound since their formation. They evolve, become compact objects, and merge without experiencing any external perturbations [75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88]. The processes of single and binary stellar evolution drive this scenario.

In reality, the two formation pathways might have a strong interplay in star clusters, because single, binary evolution, and stellar dynamics are all active at the same time, and binaries might form in dense environments and merge outside them because of dynamical kicks. Such kind of hybrid scenarios blur the line between the dynamical and the isolated binary channel, and they have already been investigated by various authors [89, 90, 91, 92].

In the context of the isolated channel, chemical homogeneous evolution (CHE) might be one of the key processes that can bring two compact objects very close to each other so they can merge via GWs within the Hubble time. [93, 94, 95]. CHE assumes that the stellar evolution in a very close stellar binary differs from that of a single star due to strong tidal forces. The burning of chemical elements occurs homogeneously, thus the star's outer layers do not expand significantly and they can stay very close to each other without merging during their life. Therefore, the close star systems can survive the phase of stellar expansion and form a merging compact-object binary.

The common envelope (CE) phase is another process that can play a crucial role in the evolution of merging binary systems [96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106,

107, 108, 109, 110, 111]. In a tight binary system, the CE phase happens when one star overfills its Roche lobe and initiates a process of dynamically unstable mass transfer. In this scenario, the mass transfer rate increases with time, the secondary star cannot accrete all the incoming material, and the latter surrounds the entire binary. The gas surrounding the binary star is known as CE.

During the CE phase, the binary system rotates at a different rate than the CE. The orbital energy decreases due to the friction between the binary system and the CE. More massive and bound envelopes result in stronger friction forces and enhanced loss of orbital energy. Due to orbital energy loss, the core of the donor and the companion star spiral toward one another within the CE (spiral-in phase). The orbital semi-major axis of binary systems can shrink by orders of magnitude during the spiral-in phase.

The lost fraction of orbital energy is transferred to the envelope, which heats up and expands. The CE phase can end with two different outcomes. In the first scenario, the envelope is ejected, leaving the binary system with quite small semi-major axes. In the other scenario, during the spiral-in phase, the two stars merge and become an (evolved) massive star.

If the envelope is completely ejected we have

$$E_{\rm bind} \leq \Delta E_{\rm orb} \tag{1.3}$$

otherwise, if the stars merge before the envelope is ejected,

$$E_{\rm bind} \geq \Delta E_{\rm orb} \tag{1.4}$$

where $E_{\rm bind}$ is the total binding energy of the envelope and $\Delta E_{\rm orb}$ is the change in orbital energy that has been transferred from the orbit to the envelope.

In particular, the amount of energy removed from the orbit and transferred to the envelope (efficiency of the CE evolution) and the envelope's binding energy, which roughly

corresponds to the maximum amount of energy that can be removed from the orbit, determine the occurence of the two scenarios. The details of the overall energy balance during the CE phase are highly uncertain from both the observational and theoretical points of view. This aspect hampers us from having detailed constraints on the outcomes of the CE evolution.

CE evolution is very uncertain from both the observational and theoretical points of view. This happens because (i) the CE phase is supposed to be very short ($\sim 10^4$ years), so it is very difficult to observe, (ii) the details of energy balancing considerations are difficult to constrain, and (iii) we have major uncertainties in stellar evolution calculations especially when rapid mass losses are involved.

Accurate three-dimensional hydrodynamic simulations might provide a comprehensive view of the processes within the overall envelope, however such simulations are very complex because the involve a very wide range of time and spatial scales [112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 112, 125, 126, 127, 128, 126, 129].

For fast population-synthesis calculations, the CE process is modeled using approximate approaches. The (α,λ)-formalism is one of the most used among the scientific community. This is based on simplified considerations on energy exchanges between the orbit and the envelope where α parameterizes the fraction of orbital energy transferred to the envelope, and λ is the envelope's binding-energy parameter.

The $\alpha_{\rm CE}$ parameter parametrizes all sinks or sources of energy, such as radiative losses and, possibly, recombination energy. In this formalism, the condition for envelope ejection becomes

$$E_{\rm bind} \leq \alpha_{\rm CE}\, \Delta E_{\rm orb} \tag{1.5}$$

and, respectively, the condition for a merger is

$$E_{\rm bind} \geq \alpha_{\rm CE}\, \Delta E_{\rm orb} \tag{1.6}$$

.

The λ parameter is generally referred to as the envelope shape factor and it constraints the envelope energy reservoir for a specific binary star system.

A parameter λ was introduced by de Kool [130] as a numerical factor to simplify the computationally expensive calculation of the binding energy of the stellar envelope:

$$E_{\rm bind} = -G\,\frac{M_{\rm d}\,M_{\rm env}}{\lambda\,a_i\,r_L} \tag{1.7}$$

where $M_{\rm env}$ is the mass of the donor's envelope.

A proper binding energy estimation is required to accurate prediction of a binary system's fate.

For many years, the λ parameter was used as a constant for all stars, independently of their initial mass $M_{\rm ZAMS}$[6], chemical composition, metallicity Z, and, more importantly, stellar evolution phase. This is a rough approximation since the main stellar evolution parameters and evolutionary stage all have a crucial impact on the stellar envelopes' binding energies.

Using the (α,λ)-formalism, assuming that envelope is completely ejected, we can write the change of orbital energy as

$$\Delta E_{\rm orb} = G\left(\frac{M_{\rm core}\,M_{\rm a}}{2a_{\rm f}} - \frac{M_{\rm d}\,M_{\rm a}}{2a_{\rm i}}\right) \tag{1.8}$$

By combining equation 1.8 with the energy balance equation $E_{\rm bind} = \alpha_{\rm CE}\,\Delta E_{\rm orb}$, we can find the ratio between the final and the initial orbital semi-major axis:

$$\frac{a_{\rm f}}{a_{\rm i}} = \frac{M_{\rm core}\,M_{\rm a}}{M_{\rm d}}\left(M_{\rm a} + \frac{2\,M_{\rm env}}{\alpha_{\rm CE}\,\lambda\,r_L}\right)^{-1} \tag{1.9}$$

[6]A star's initial mass is the star's mass at the beginning of the main sequence. Such mass is called the Zero-Age Main Sequence (ZAMS) mass.

By substituting equation 1.9 into equation 1.1, we obtain the following:

$$\tau_{GW+CE} = \frac{5}{256}\frac{c^5}{G^3}\frac{a_{\rm i}^4\, M_{\rm core}^3\, M_{\rm a}^3}{M_{\rm d}^4\,(M_{\rm core}+M_{\rm a})}\left(M_{\rm a}+\frac{2\,M_{\rm env}}{\alpha_{\rm CE}\,\lambda\,r_L}\right)^{-4}(1-e^2)^{7/2} \qquad (1.10)$$

The time required for the coalescence of a binary system that evolves through a CE phase scales as

$$\tau_{GW+CE} \;\propto\; \alpha_{\rm CE}^4\,\lambda^4 \qquad (1.11)$$

The time required for a binary system to merge is highly dependent on the α and λ parameters, so their values have a crucial impact on the interpretation of many astrophysical systems, including merging compact-object binaries. While constraining the α parameter is challenging [131, 132, 133, 134], we can calculate the λ values and consider λ as a physical quantity instead of a parameter.

Despite the existence of self-consistent λ parameter calculations, a comprehensive and detailed analysis of λ parameters for all stellar stages at all possible metallicities and for a broad range of stellar masses is still missing, as well as studying the implications of self-consistent calculations of λ on large populations of binary stars and how up-to-date λ values can affect the formation of loud GW sources.

The main result is that differences in the details of stellar evolution calculations, such as stellar rotation, stellar winds models, criteria of core definition, and overshooting, can have a crucial impact on the calculation of the λ parameters for both non-naked and naked-helium stars. Thus, having self-consistent calculations for the envelopes' binding energies is a crucial step that must be performed before studying populations of merging compact-object binaries via population-synthesis codes.

We performed self-consistent calculations of the λ parameter for a large set of stars, at different metallicities, through an up-to-date version of the PARSEC stellar evolution code. We calculated the values of λ for non-rotating hydrogen stars with metallicities Z =0.0001, 0.0005, 0.001, 0.002, 0.004, 0.006, 0.008, 0.014, 0.017, 0.02 and 48 values of initial masses

in the range between 2.0 $M_\odot$ and 600.0 $M_\odot$ (equally spaced in logarithmic scale). Furthermore, we also considered pure-helium stars as potential donors in the CE phase, and we performed self-consistent λ-parameter calculations for a large set of helium stars, at metallicity Z =0.0001, 0.0002, 0.0005, 0.001, 0.002, 0.004, 0.006, 0.008, 0.01, 0.02, 0.03, 0.05, and for 78 log-spaced values of initial masses in the range between 2.0 $M_\odot$ and 350.0 $M_\odot$.

We have provided all the obtained binding energy parameters for hydrogen and helium stars for public use in the form of look-up tables. Tables containing the following data: the mass ($M_{\rm i}$, in grams), radius ($R_{\rm i}$, in centimeters), effective temperature ($T_{\rm eff}$, in Kelvin), luminosity (L, in watts), the $\lambda_{\rm g}$, $\lambda_{\rm b}$ and $\lambda_{\rm h}$ parameter of the star during stellar evolution.

We have also provided a detailed description of the fitting functions and their coefficients for public use. For each function, we pointed out the value of mean squared errors. The data available on the website github.com/NataNazar/Binding_energy_parameters

This thesis aims to study how new prescriptions of CEE affect the formation of loud GW sources with the use of the up-to-date population-synthesis code - SEVN.

1.2 Binary population synthesis codes

Binary population synthesis (BPS) codes can provide valuable insights into the expected rate and distribution of the target population's properties, the different evolutionary pathways that lead to the formation of merging compact object binaries, and the effect that different physical processes and parameters have on their evolution.

In general, there are essentially three different approaches to implement stellar evolution in BPS codes:

- the most common method to quickly evolve the fundamental stellar parameters (e.g., luminosity, radius, mass, temperature, and chemical composition) far a large number of either single or binary stars as a function of time, is using polynomial fitting formulas. Population-synthesis codes based on fitting formulas are computationally very fast,

but updating fitting formulas with new prescriptions might be challenging. Some examples of BPSs based on fitting formulas are the "Single Star Evolution" (SSE, [135]), the "Binary Stellar Evolution" (BSE, [82]), the SEBA [136], the BRUSSELS CODE [137], the BINARYC [138], the STARTRACK [139], the "Compact Object Mergers: Population Astrophysics Statistics" (COMPAS, [140, 86]), the "Massive Objects in Binary Stellar Evolution" (MOBSE, [141, 142]), and the "Compact Object Synthesis and Monte Carlo Investigation Code" (COSMIC, [143, 144]).

- The second option is to utilize look-up tables instead of fitting formulas [145]. These tables include grids of pre-evolved stellar evolution models for single stars that are dynamically read and interpolated on the fly by BPS codes, for example by "Stellar Evolution for N-body" (SEVN, [34, 35, 36]) and COMBINE [146]. For binary stellar evolution such codes also use analytical prescriptions. This method is both computationally efficient and flexible. The main advantage is also that we can update stellar models by simply changing the look-up tables, without the need of modifying directly the code.

- Hybrid methods typically use population synthesis codes combined with detailed simulations of stellar and binary evolution [147, 148, 149, 150, 151, 152, 153, 154, 155] These studies show that including more detailed modeling of binary interactions may reveal details that are missed using simpler approaches. The "Binary Population and Spectral Synthesis code" (BPASS, [107]) and POSYDON [156, 157] use a hybrid approach to study the effect of detailed modeling of stellar and binary physics on compact object mergers.

1.3 The SEVN code

In this thesis, we focus on the SEVN population-synthesis code, which is based on the look-up tables approach. This will allow us to easily implement the new λ descriptions

developed in this thesis in the form of a new look-up table that SEVN can interpolate on the fly. SEVN interpolates stellar evolution from look-up tables (the default tables being derived from PARSEC, [158, 159], includes five different models for core-collapse SNe, contains prescriptions for PPISNe and PISNe and has been updated to implement also binary evolution processes (wind mass transfer, Roche lobe overflow, CE, stellar mergers, tidal evolution, GW decay and magnetic braking).

Since SEVN is already based on PARSEC tracks, the obtained binding energy parameters will be fully self-consistent with the code.

The attributes for each object that we want to evolve can be set in the SEVN input file. If the object is a "single star", the input file needs to contain the following initial attributes:

- initial mass,
- metallicity,
- spin,
- stage of stellar evolution of each star.

If the type of the object to evolve is a "binary star", we need to specify the following initial physical quantities:

- initial mass of each star,
- metallicity of each star,
- spin of each star,
- stage of stellar evolution of each star,
- initial orbital distance,
- eccentricity of binary system.

1.4 Time and memory constrains of the SEVN code

Currently, the SEVN code is not parallel and sequentially evolves a population of binary systems. Figure 1.3 shows the calculation time and the memory consumption of the serial

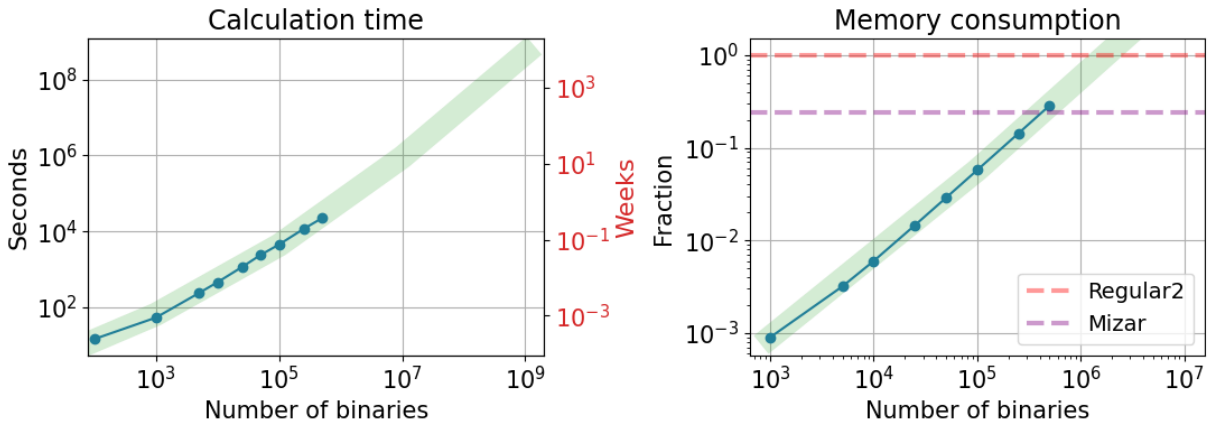

Figure 1.3: Left panel shows the computational time of SEVN for different numbers of binaries. The left vertical axis displays the total computational time for SEVN in seconds, while the right vertical axis displays the time in weeks. Right panel is the memory usage of SEVN for different numbers of binaries. The vertical axes represents the percentage of total available memory on the `regular2` partition during the execution of SEVN code. The dashed purple horizontal line represents the maximum memory available on the Mizar workstation.

SEVN code for different numbers of input binaries. Consumed memory is shown in units of available memory on each Ulysses `regular2` nodes, i.e. 64.0 GB. From Fig. 1.3 it is apparent that the time needed to evolve 1 million binaries is approximately a week, which is a very large amount of time considering that, to have a statistically significant sample of merging compact-object binaries at all metallicities, we will need to evolve at least tens of thousands of binaries.

Also, it is apparent that the serial version of SEVN is a high memory-consumption code. Due to memory constraints, we can evolve up to one million binaries on Ulysses's `regular2` partition while only 500 thousand on Mizar's workstation. The memory consumption of SEVN is surely one of the aspects that must be improved, thus just speeding up the code's execution is not enough to guarantee sustainable performance on multiple architectures. Specifically, the serial version of the SEVN code will complete the evolution of one set of billion objects over 1000 weeks or 19 years (see Figure 1.3). In addition, such calculations require an enormous amount of memory (more than 50 thousand GB)

Thus, we need to modify the SEVN code to both (i) speed up the calculations, and (ii)

avoid the high pressure on memory. To evaluate the performance of our parallelized code, we will evaluate the speed-up as:

$$S_{\rm p} = \frac{\text{execution time using one process}}{\text{execution time using } p \text{ processes}} = \frac{t_1}{t_{\rm p}} \tag{1.12}$$

1.5 Optimization of the SEVN code

Before implementing the new λ prescriptions, the SEVN code must be first optimized, especially in terms of performance, so that we can quickly simulate large population of binary stars (tens of millions, at least), that is an enough number to get statistically significant result on the population of merging compact-object binaries.

Parallel computing can accomplish this goal. The strategy to enable an effective parallelization of the SEVN code would be to divide the total number of objects to evolve into smaller groups and run their simulations simultaneously using different processing units (central processing units - CPUs or even graphics processing units - GPUs).

Supercomputers, workstations, and even personal laptops with multiple CPUs can execute parallel tasks. Workstations offer higher performance than mainstream personal computers, especially in CPU, graphics, memory, and multitasking. Distributed systems with many nodes, such as supercomputers, allow for massive parallelism.

We will achieve our main goal to speed-up the SEVN code by completing the following tasks:

- implementing the parallelization on a single computing node,
- doing the same for multi-node supercomputers,
- evaluating potential bottlenecks, including input-output operations and/or memory pressure, to understand how to further improve performance in the next future.

Chapter 2

Methodology

2.1 Computer Architectures

We develop and test the parallel version of the SEVN code on the "Mizar" workstation (single node) and on the Ulysses Compute Cluster in SISSA (multi node). Mizar and Ulysses are both Linux-based systems. Mizar is a workstation, which contains 6 cores - 12 threads. In contrast, each node on the `regular2` partition of Ulysses contains:

- 2 sockets,
- 16 cores per socket.
- 2 CPUs per core,
- The maximum number of threads is 2 per core or 64 per node.
- Max available memory is 63500 MB.

The detailed information about Mizar and Ulysses (`regular2` partition) is presented in table 2.1. We chose the `regular2` partition of Ulysses because all nodes have the same configuration and available memory.

.

Ulysses has then been extended and a new infrastructure was made available during late 2019. This new infrastructure consists of additional nodes, an upgraded software stack and

Computer/property	**Mizar**	**Ulysses, regular2**
Architecture:	x86_64	x86_64
CPU op-mode(s):	32-bit, 64-bit	32-bit, 64-bit
Byte Order:	Little Endian	Little Endian
Address sizes:	46 bits physical, 48 bits virtual	
CPU(s):	12	64
On-line CPU(s) list:	0-11	0-63
Thread(s) per core:	2	2
Max Threads per Node		64
Core(s) per socket:	6	16
Socket(s):	1	2
NUMA node(s):	1	2
Max Memory per Node (MB)	15500 MB	63500 MB
Vendor ID:	GenuineIntel	GenuineIntel
CPU family:	6	6
Model:	62	79
Model name:	Intel(R) Core(TM) i7-4930K CPU @ 3.40GHz	Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz
Stepping:	4	1
CPU MHz:	1253.196	1200.000
CPU max MHz:	3900.0000	2100.0000
CPU min MHz:	1200.0000	1200.0000
BogoMIPS:	6800.40	4190.39
Virtualization:	VT-x	VT-x
L1d cache:	192 KiB	32K
L1i cache:	192 KiB	32K
L2 cache:	1,5 MiB	256K
L3 cache:	12 MiB	40960K
NUMA node0 CPU(s):	0-11	0-15,32-47
NUMA node1 CPU(s):		16-31,48-63

Table 2.1: The table displays information about the architecture of Mizar workstation and Ulysses, regular2 partition.

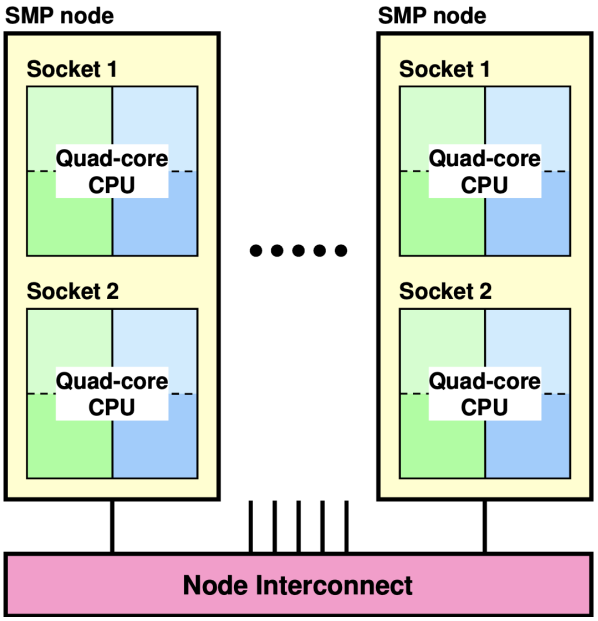

Figure 2.1: Symmetric multiprocessing or shared-memory multiprocessing design.

a new job scheduler, SLURM (Simple Linux Utility for Resource Management). After the upgrade the old nodes are used in partitions `regular1, long1, wide1` and `gpu1`, while the new nodes are available as `regular2, long2, wide2` and `gpu2` (datailed information on the Ulysses webcite).

Computer Architecture is very important for parallel computing. Total amount of CPUs, their type and amount of available memory constrain the total speed up and scaling of paralellization.

On Ulysses we used the following modules and compilers:

- `cmake/3.15.4` CMake is a cross-platform build system generator. Projects specify their build process with platform-independent CMake listfiles included in each directory of a source tree with the name `CMakeLists.txt`. Users build a project by using CMake to generate a build system for a native tool on their platform.

- `gnu8/8.3.0` GNU module provides C++ compilers. The variant 3.15.4 supporting OpenMP and OpenACC offload is available. GCC 8.3 February 22, 2019 https://gcc.gnu.org/gcc-8/changes.html

- `openmpi3/3.1.4`
- `C++14`

2.2 Adaptive data loading

Device capabilities and size of the Random-Access Memory (RAM) vary a lot. Occasionally, the amount of data loaded during calculations may exceed the device's capabilities. To mitigate the issue of memory consumption of the SEVN code when evolving a very large number of binaries, we implement the following automatic strategy. The SEVN code splits the set of stars into chunks that fit into the available memory of the actual device. We call this method adaptive data loading.

We use some pre-load strategies to find the ideal adaptive chunk size for the device the SEVN code is running on. To find what is the maximum possible amount of binaries the SEVN code can load without filling the hardware memory resources, we use the following proportional formula.

$$\text{Adaptive chunk} = \text{Memory Target Max} \times \frac{\text{Test Chunk}}{\text{Memory Consumed}} \tag{2.1}$$

where "Test Chunk" is a test-reference sample of any size which we run with the SEVN code (for example, 1000 binary stars), and "Memory Consumed" is the memory consumption associated with the evolution of the test chunk. "Maximum Target Memory" is the maximum fraction of total Available Memory that we want to SEVN code to allocate during the simulation:

$$\text{Memory Target Max} = \text{Avail Memory} \times \text{Mem Perc Max} \tag{2.2}$$

The SEVN code reads the total available memory from file `/proc/meminfo` on the fly.

To estimate the memory consumed by the test chunk during the run, we use the difference between the amount of allocated virtual memory before and after the code lines that run

the test sample. The file `/proc/self/status` contains the information about the current SEVN run, including information about the allocated virtual memory, i.e., Virtual Memory Resident Set Size (`VmRSS`). The SEVN code reads `VmRSS` from file `/proc/self/status` on-the-fly during its run and we can estimate

$$\text{Memory consumed } = \Delta\text{VmRSS} = \texttt{VmRSS}_2 - \texttt{VmRSS}_1 \tag{2.3}$$

where $\texttt{VmRSS}_2$ and and $\texttt{VmRSS}_1$ are the amount of allocated virtual memory after and before the code run the test sample.

Users need to manually set the maximum percentage of memory "Mem Perc Max" the code can use during the run. The user will use this percentage of all available memory at the start of the simulation for calculations. However, we must limit this percentage from above since the code uses memory for calculations and storing calculation results. During the calculation, the SEVN code inserts information regarding the simulation outcomes of each binary system into the output files. At the same time, we should take into account that such output files can take up a significant amount of memory. Figure 2.2 shows the size on input and output files of the SEVN code for a different amount of binaries. The results of 10^8 binaries will use around 100 GB of memory, which is more than the total memory of `regular2` one node (62 GB) and a bit less than that of `regular1` nodes with the maximum memory (312.5 GB). The size of the initial data impacts the percentage of memory used only indirectly since it reduces the value of available memory before the simulations start. Therefore, the percentage must be re-calculated on the fly by the SEVN code and may be less than the value manually selected by the user:

$$\text{Mem Perc Max} = \min\left(\text{Mem Perc Max},\ \frac{\text{Avail Memory}}{\text{Total Memory}} - X \times \frac{\text{Number of Binaries}}{\text{Avail Memory}}\right) \tag{2.4}$$

where X is memory of output of a single binary system. Approximately the size of a output for a single binary is $X = 1.6$ KB.

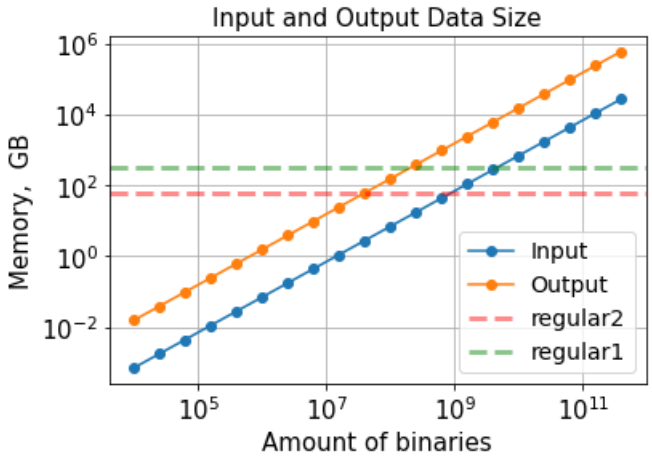

Figure 2.2: The dependence of the size of files with initial data and output files depending on the number of binary systems. The solid blue line shows the amount of memory needed for a single MPI process, while the orange solid line shows the amount of memory needed for 16 MPI processes. The horizontal dashed green line shows the maximum available memory on the `regular1` partition (or `long1`, etc.). The horizontal dashed red line shows the maximum available memory on the `regular2`.

In all our calculations, we use 80 % of the available memory on the devices. It is crucial to avoid the usage of all the available memory so to avoid filling up the resources of any system without control.

2.3 Parallelization with OpenMP

After implementing the adaptive loading method, our next goal is to parallelize the SEVN code on a single computing node. We can accomplish the goal through the Open Multi-Processing (OpenMP) directives.

OpenMP is an Application Programming Interface (API) that supports multi-platform shared-memory multiprocessing programming. Multithreading, in which a single thread (a sequence of instructions executed in order) splits off into a number of parallel "sub-threads," is implemented in OpenMP. The threads are then scheduled to execute simultaneously, with the runtime environment distributing them across the available CPU cores.

The beginning of a parallel section in the code is identified through a compiler directive

that initiates thread creation before the code is executed. Once the parallelized code has completed running, all created threads will merge back into the main thread, which continues the execution until the end of the run.

Each thread operates autonomously on its own copy of the parallelized code, by default. Using work-sharing constructs, it is possible to delegate specific portions of a task's execution to individual threads. OpenMP allows to accomplish task parallelism and data parallelism in this way.

Threads are distributed to processors by the runtime environment based on factors such as usage, machine load, and other conditions. The number of threads can be determined either by the runtime environment using environment variables or by the code using thread assignment functions. In C/C++, the "omp.h" header file contains the OpenMP directives.

In the SEVN code, all the objects to evolve can be either single or binary stars and their initial conditions are stored into a C++ standard-template-library vector. In the following, we will only refer to binaries, though the parallelization and the presented results are the same for single stars. In our parallelization strategy, each OpenMP thread takes care of the evolution of only its part of binaries (Bin Per Thread), simultaneously to other threads:

$$\text{Bin Per Thread} = \text{std::ceil} \left(\frac{\text{Total Number of Binaries}}{\text{Total Number of Threads}} \right) \tag{2.5}$$

The adaptive chunk loading modulates the total number of binaries:

$$\text{Total Number of Binaries} = \min(\ \text{Total Number of Binaries},\ \text{Adaptive chunk}\ ) \tag{2.6}$$

Function std::ceil(x) returns the smallest integer that is greater than or equal to x, so only the last OpenMP thread will evolve less binaries in case the total amount of binaries and adopted number of threads are not exact multiple of each other. Then each set of binaries evolves simultaneously within the `parallel` loop construct. The initial (Omp Start) and

final (Omp End) binary that each thread must evolve within the parallel loop are then

$$\text{Omp Start} = \text{ Bin Per Thread } \times \text{ Thread ID} \tag{2.7}$$

and

$$\text{Omp End} = \min(\text{ Bin Per Thread} \times (\text{Thread ID} + 1),\ \text{Total Number of Binaries })) \tag{2.8}$$

The main part of the code with OpenMP implementation is shown in the Appendix A.4. The results of our OpenMP implementation are presented in the following chapter (Sec. 3.2).

2.4 Parallelization with MPI

Message Passing Interface (MPI) is a standardized and portable message-passing standard designed to function on parallel computing architectures. MPI does not depend on its underlying programming language.

MPI and OpenMP enable parallel programming, but they have an important difference. MPI is used in distributed memory architectures. Unlike shared memory described above, distributed memory uses a collection of independent memory units that synchronize using a network, primarily found in supercomputers. It means that each core or node has a memory space of its own and does not require locks like shared memory.

However, synchronization is still required to distribute the computation and collect results, and that is done through message passing. OpenMPI provides API calls such as MPI_Send and MPI_Recv to allow communication between computation nodes. Unlike OpenMP, each computational unit has to send its results to a master and manually compile and aggregate the final result.

Global communication primitives are carried out on all processes belonging to the same

communication group. By default, once MPI got initialized, all processes belong to the same group of communication called `MPI_COMM_WORLD`.

Typically, for maximum performance, to each CPU core will be assigned just a single MPI process.

To perform MPI parallelization, we divided the input binaries in the SEVN code into partitions, similarly to what we did for OpenMP, but now considering the total number of MPI processes:

$$\text{Bin Per MPI} = \frac{\text{Total number of binaries}}{\text{Number of MPI procs}} \tag{2.9}$$

Then the number of binaries is compared with the size of the adaptive chunk.

$$\text{Bin to do } = \min\left(\text{Bin Per MPI} - \text{Bin Done}, \text{Adaptive Chunk}\right) \tag{2.10}$$

An important aspect that it is worth noting is that here the adaptive chunk method must constrain the binaries to evolve per MPI process ("Bin per MPI") because, as already mentioned, each MPI process has its own memory copy of the binaries. Thus, if the number of binaries per MPI process requires more memory than is available on the device, then the evolution of objects by each MPI process happens in a loop. Specifically, depending on the available memory, increasing the number of MPI processes results in a decreased chunk size and, consequently, in a decreased speed-up (e.g., too few binaries per MPI process, see figure 3.1).

The main part of the code with MPI implementation is in the Appendix A.5. The results of our MPI implementation are presented in the following chapter (Sec. 3.2).

Chapter 3

Results

3.1 Adaptive data loading

The formula 2.1 for calculating the adaptive chunk size assumes that the consumed memory scales linearly with the number of binaries. Figure 1.3 shows that this is a reasonable assumption.

Figure 3.1 shows the relative size of a chunk and the percentage of memory consumed in the process of calculations as the function of the maximum available memory. The figure shows that as the number of available memory increases, the adaptive chunk size grows linearly for 2.5 and 5 million binaries. For 1 million binaries, the size of the chunk saturates when we use 60 percent of available memory. Saturation happens because the node's memory has sufficient capacity to simultaneously store the entire set of binary systems (see figure 1.3). For 1 million of binaries, the size of an adaptive chunk for 4 MPI processes is exactly four times smaller than that for 1 MPI process. It happens because 4 MPI processes use in four times more available memory to store in the memory vector with objects.

The right side of figure 3.1 shows that the amount of physical RAM used for calculations differs from the maximum value of memory specified. For threshold values below 60%, the actual memory consumption is roughly 10% higher. This occurs because the SEVN

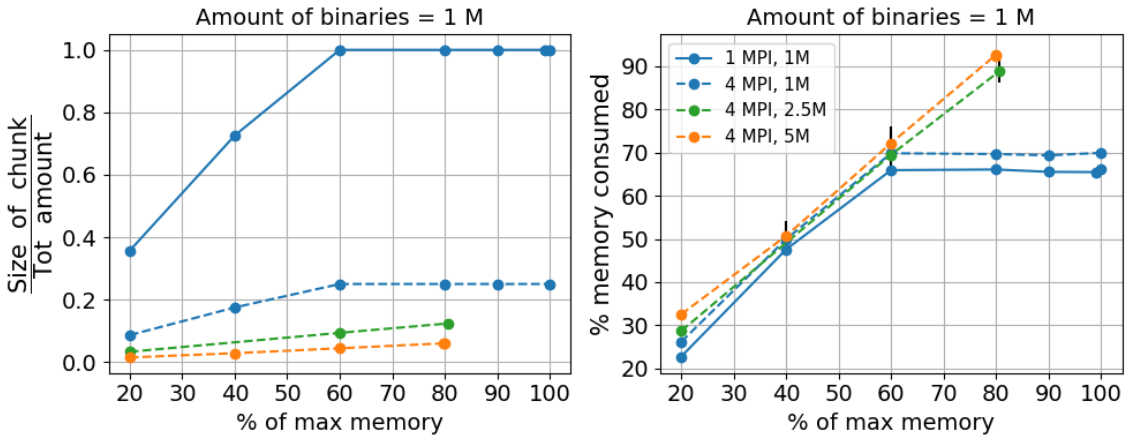

Figure 3.1: The relative size of a chunk (on the left) and the percentage of memory consumed during the run of SEVN (on the right) as the function of the maximum available memory (62 GB). We obtained the dependencies for 1 (solid lines) and 4 (dashed lines) MPI processes and for 1 (blue lines), 2.5 (green lines), and 5 (orange lines) million binaries.

code, initial data, and libraries take up about 10% of total RAM. The simulation's memory consumption stabilizes at approximately 70% at a threshold value greater than 60% for 1 million binaries because such simulations cannot use more memory.

Figure 3.1 shows that the larger the set of binary systems to simulate, the smaller the maximum percentage of memory we can use for simulations themselves. It happens because a more extensive set of binaries require more memory to hold the results in the RAM (see formula 2.4). The figure also shows that consumed memory is a few percent less than total memory. The available memory is always less than the total memory due to the need to keep the operating system CentOS and the Slurm Workload Manager in the memory.

3.2 OpenMP Scalability - Single Node

Figure 3.2 shows the calculation time, and speed-up of the parallelized version of the SEVN code on a single node of the Ulysses computing cluster at SISSA (see Table 2.1), as a function of the adopted CPUs and for different total numbers of evolved binaries. We show the results of both the OpenMP and the MPI parallelizations.

We see that the code calculation time decreases by one order of magnitude when using

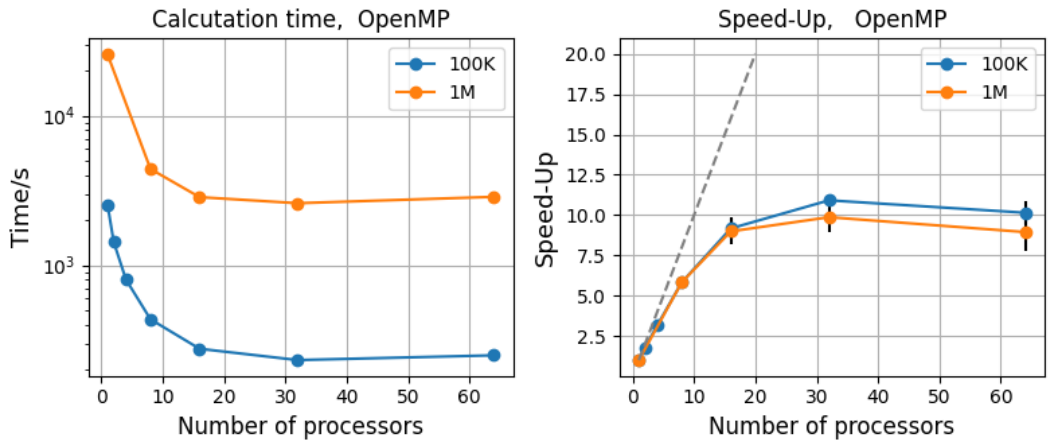

Figure 3.2: The SEVN code calculation time and speed-up for the different number of OpenMP threads (or used CPUs). Calculations performed on Ulysses, `regular2` partition. The solid orange lines show the results for 100 thousand binaries, and the solid blue line for 1 million binaries. The dashed grey line is perfect speed-up.

only the OpenMP interface. The speed-up of the SEVN code when using MPI is quite close to the values of perfect speed-up. When using OpenMP, the speed-up saturates at 16 used CPUs and it gives a maximum speed-up of 10x on a single node.

The reason why the OpenMP version of the SEVN code stops scaling for a number of threads larger than 16 might be ascribed to false-sharing. Most high-performance processors insert a cache buffer between slow memory and the high-speed registers of the CPU. Accessing a memory location causes a slice of actual memory (a cache line) containing the memory location requested to be copied into the cache. Each update of an individual element of a cache line coming from different threads marks the line as invalid, and threads are forced to fetch a more recent copy of the line from memory, even though the element accessed has not been modified. As a result, there will be an increase in interconnect traffic and overhead. Also, while the cache-line update is in progress, access to the elements in the line is inhibited. If this occurs frequently, the performance and scalability of an OpenMP application will suffer significantly. This probably happens in our code since all the OpenMP threads access simultaneously the same C++ vector, which is shared among the threads.

If the issue is false-sharing, then the code needs to be restructured, especially when it

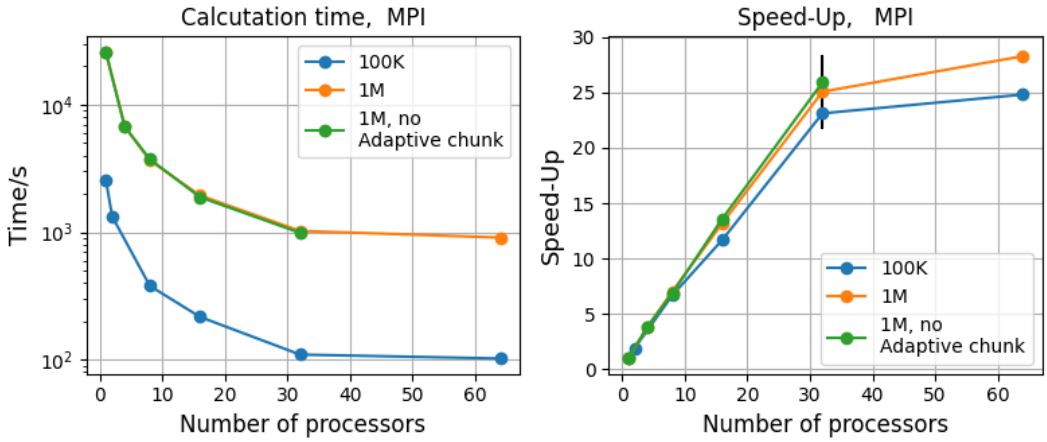

Figure 3.3: The SEVN code calculation time and speed-up for the different number of MPI processes (or used CPUs). Calculations performed on Ulysses, `regular2` partition. The solid orange lines show the results for 100 thousand binaries, and the solid blue line for 1 million binaries. The dashed grey line is perfect speed-up.

comes to storing and accessing data. As a follow-up project, we will investigate whether it is worth restructuring the code for better OpenMP performance, or simply do a complete porting on, for example, GPUs, though the latter is beyond the scope of this thesis.

3.3 MPI Scalability - Single Node

Figure 3.3 shows the calculation time, and speed-up of the parallelized version of the SEVN code on a single node of the Ulysses computing cluster at SISSA (see Table 2.1), as a function of the adopted CPUs and for different total numbers of evolved binaries. We show the results of the MPI parallelizations on a single node.

The SEVN code calculation time decreases by a factor 25 when using only the MPI version of the code. The speed-up of the SEVN code when using MPI is quite close to the values of perfect speed-up. When using OpenMP, the speed-up saturates at 16 used CPUs and it gives a maximum speed-up of 10x on a single node.

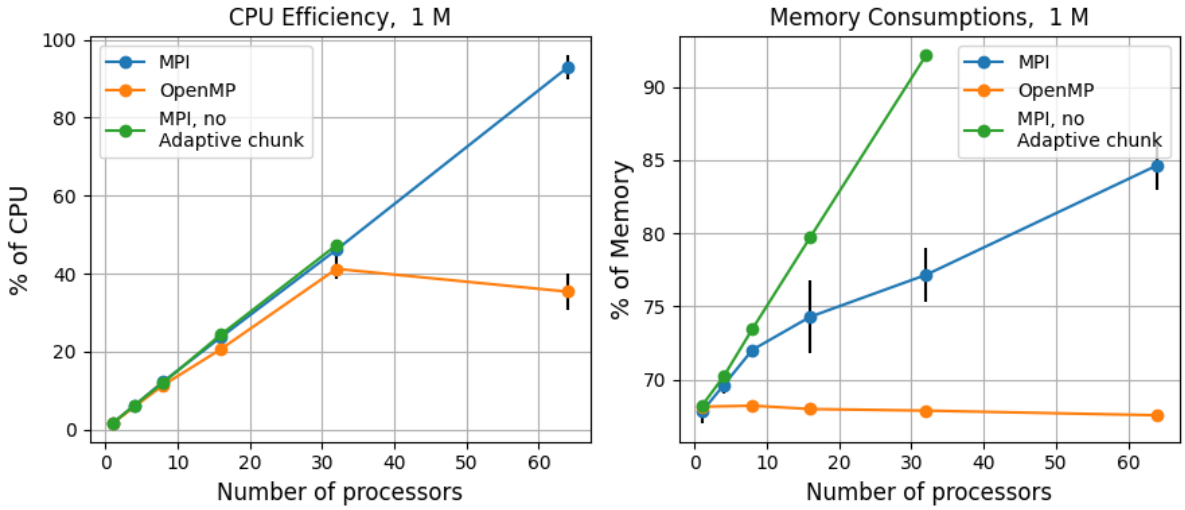

Figure 3.4: CPU efficiency (on the left) and memory consumption (on the right) for 1 million of input binaries for different amount of processors units (CPUs), used during the run.

3.4 CPU Efficiency and Memory Consumption

Successful parallelization of code requires efficient use of both processing power (CPUs) and memory. Figure 1.3 shows that simulations of a million binary systems with serial SEVN code use all the available node memory on the Ulysses, `regular2` partition. We performed calculations using the parallelized SEVN code to check whether we had reached our mission. In Figure 3.4, we show the CPU efficiency and the memory consumption for one million input binaries. We can see that CPU efficiency grows almost linear with the number of threads or MPI processes. This outcome is consistent with our expectations, as we utilize more available CPUs while increasing the number of MPI processes or threads.

Memory consumption increases when using only the MPI interface, while for OpenMP, it remains constant. It happens for obvious reasons: With the OpenMP interfaces, processes share memory and access an array of binary systems. With the MPI interface, all processes copy and allocate memory for the entire array of binary systems. Memory usage grows linearly without the adaptive chunk implementation. For a larger number of systems or employed CPUs, all available memory will be used for calculations.

3.5 Hybrid Parallelization - MPI + OpenMP

We have many possible implementations of hybrid parallelization. To choose the best combination, we need to consider all the advantages and pitfalls of using OpenMP and MPI interfaces.

OpenMP: The advantage of using only the OpenMP interface is the ease of use, and the data is not replicated as the number of threads increases. The latter is a major benefit for the SEVN code, which has high memory pressure. The major disadvantages are the reduced speed-up on a single node and the inability to perform parallelization on multiple nodes.

MPI: The main advantage of using MPI is increased performance, if a single process is assigned to a single CPU core, and the possibility to scale on multi-node computing clusters. Also, MPI is a good candidate for perfectly load-balanced applications. The main problem of MPI is memory consumption because each MPI process replicates the data. Specifically, if we start increasing the number of MPI processes, at some point, the adaptive data loading mechanism will force the chunk of binaries to a very small value, resulting in a compromised speed-up (see also Fig. 3.1)

Thus, with a hybrid parallelization strategy we might try to take full advantage of both OpenMP and MPI implementations. The hybrid approach will take the main advantage of MPI in terms of scaling performance across multiple nodes and the main advantage of OpenMP in terms of reducing memory pressure.

The optimal implementation of hybrid parallelization on a single node is an outer loop with an MPI interface and an inner loop with OpenMP. It is necessary to understand how to choose the optimal number of OpenMP threads per MPI process and the number of MPI processes per node.

So we tried the following combinations:

- 1 MPI process per socket, i.e. 2 MPI processes per node on Ulysses,
- 2 MPI process per socket, i.e. 4 MPI processes per node on Ulysses,

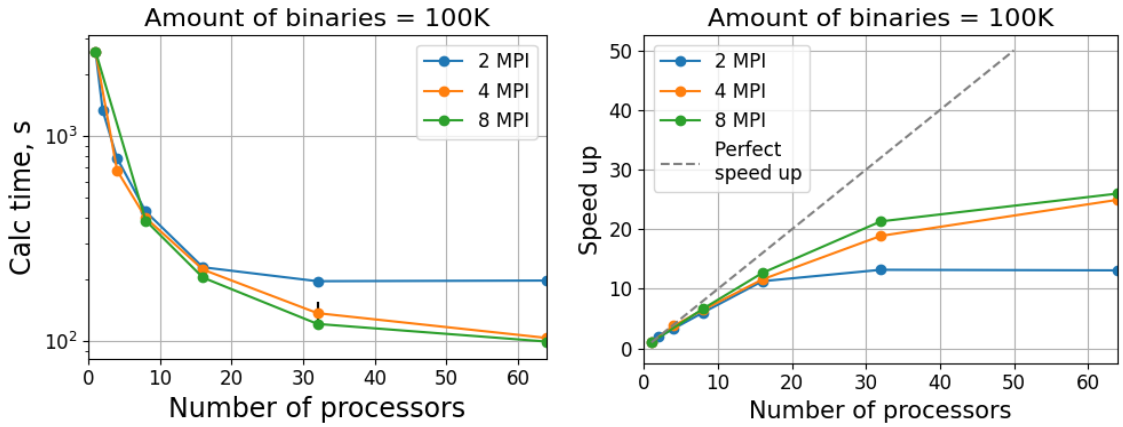

Figure 3.5: The calculation time (on the left) and speed-up (on the right) depending on the number of processors (CPUs) used. We obtained the results for 100 thousand binary star systems. During the calculation, we used 2, 4, and 8 MPI processes and a different number of threads. The dashed grey line is perfect speed up.

- 4 MPI processes per socket, i.e. 8 MPI processes per node on Ulysses.

We varied the number of threads for each combination from one to the maximum possible number. Figure 3.5 shows the calculation time and speed up depending on the number of CPUs used on a single node of the Ulysses cluster using 100 thousand binary star systems. We get the slowest calculations when using 2 MPI processes per node. When using 4 and 8 MPI processes per node, we get approximately the same calculation time/speed-up. Given that 4 MPI processes use half as much memory as 8 MPI, we get the following optimal configuration of processes for the current version of the SEVN code on Ulysses: 2 MPI process per socket (i.e., 4 per node) and 16 OpenMP threads, i.e. total of 64 "processors", as in the x-axis of Fig. 3.5. Such a combination of MPI and OpenMP processes is favorable for large statistical sample simulations. As we stated above, a smaller number of MPI processes significantly softens the memory requirements and increases the size of the adaptive chunk.

3.6 Multi-node parallelization

Figure 3.6 shows the calculation time and speed-up of the SEVN code when using multiple nodes on the Ulysses cluster, `regular2` partition. We run the parallelized SEVN code on

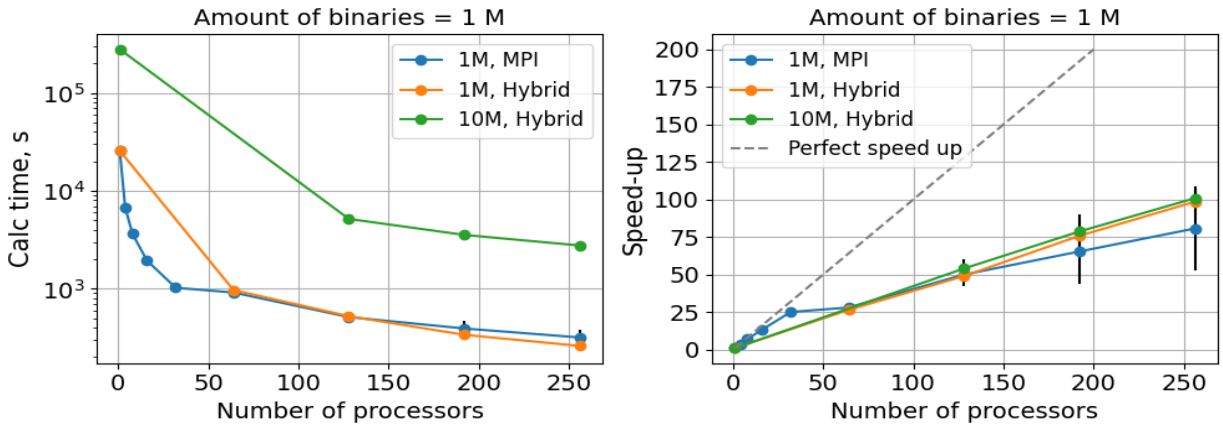

Figure 3.6: The calculation time (on the left) and speed-up (on the right) depending on the number of processors (CPUs) used. The solid blue lines show the results using the MPI interface on multiple nodes. The solid orange lines show the results using hybrid parallelization (both the OpenMP and MPI interface). We used 16 OpenMP threads and 4 MPI processes per node for hybrid parallelization. The dashed grey line is perfect speed up.

up to four compute nodes for 1 and 10 million binaries. The speed-up of the SEVN code with hybrid approach on one node (up to 64 CPUs) is about 25x for 1 and 10 million binaries. On four Ulysses compute nodes, with hybrid parallelization, we get a speed-up of about 100x. The speed-up on multi-nodes is about 40 percent of a perfect speed-up:

$$\text{Speed-up}_p \sim 0.4 \times \text{Speed-up}_1 \times p \tag{3.1}$$

where p is a number of the node.

The speed-up of the SEVN code with pure MPI (up to 64 CPUs) is also about 25x on one node and about 75x on four nodes for 1 million binaries. For 10 million binary systems, we could not perform calculations with a sufficiently large number of MPI processes per `regular2` node. Figure 3.7 shows the dependence of the percentage of memory used only for simulations (excluding used memory for storing input data and results) as a function of the number of MPI processes per node. Due to the extensive memory usage for storing input data by each MPI process, even with 10 MPI processes per `regular2` node, no memory is left for simulations. At the same time, hybrid parallelization uses the same amount of

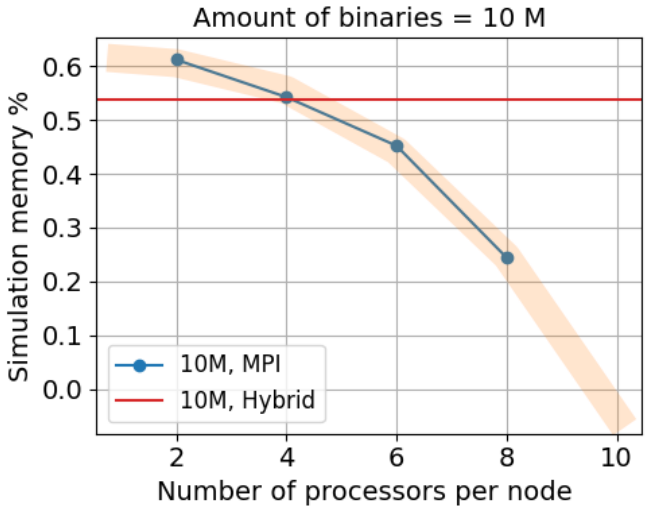

Figure 3.7: The dependence of the percentage of memory used only for simulations as a function of the number of MPI processes per node.

memory for simulations when scaling calculations across multiple nodes. We can optimize hybrid parallelization for a bigger sample of binary systems by reducing the number of MPI processes per node. For example, we can use 1 MPI process and 64 threads instead of 4 MPI and 16 threads for a hundred million binary systems.

Hybrid parallelization gives approximately the same computation time and acceleration as pure MPI, but it uses significantly less memory. Therefore, hybrid parallelization has a more promising potential for very large statistical population samples.

Chapter 4

Conclusions

In this thesis, we have focused on a preliminary optimization of the SEVN code, preparatory for implementing our new CE prescriptions. We found that the SEVN code requires significant optimization to evolve large samples of objects with different initial parameters since such simulations require a significant amount of memory and time (see figure 1.3).

Thus, we optimized SEVN to run very large samples of binary systems in a reasonable amount of time and with low memory consumption. To do that, we parallelized the SEVN code on both single computing nodes (through OpenMP) and multi-node supercomputers (through MPI).

In order to mitigate the memory requirements of the computing device, we implemented a new method which we refer to as adaptive data loading. Adaptive loading divides the entire initial set of objects into chunks of sufficient size to avoid filling the system's available memory.

We tested our parallel implementations on the Ulysses computing cluster at SISSA. We used the OpenMP interface to speed up and optimize simulations on a single node and achieved a maximum speed-up of 10x. We used the MPI interface to scale computations across multiple compute nodes and improve the speed-up. On a single node, we got a speed-up of 25x with only MPI, while on four compute nodes, we get a speed-up of about

75x.

However, a large number of MPI processes on one node significantly increase memory requirements. Therefore, we used hybrid parallelization to take full advantage of OpenMP and MPI parallelization. As a result, we found that, on the Ulysses computing cluster, the best hybrid combination for one node is 2 MPI processes per socket, and the number of threads is equal to the number of cores on one socket. This combination provides approximately 25x speed-up per compute node and a speed-up of about 100x on four compute nodes.Hybrid parallelization gives a slightly better computation time and speed-up than pure MPI, but more importantly, it uses significantly less memory.

Our main task of parallelization and code optimization is to evolve with SEVN large statistical samples of binaries, which may contain up to 10^9 objects. However, in this case, we meet another issue related to loading the input files containing all the initial conditions of all stars and storing the obtained results. The problem is that such files will be heavy and require a large amount of memory. From figure 2.2 it is apparent that the `regular2` partition on Ulysses will not have enough memory to simulate all binaries, even using just one MPI process. Thus, in the future, we will need to modify the SEVN data-loading method to take into account this aspect.

A future optimization of the SEVN code will also be to port the code, or part of it, on (multiple) graphics processing units (GPUs). The strategy might be very advantageous when SEVN is used in combination of massively GPU-parallel stellar dynamics codes (e.g., ISTEDDAS [160]), but it requires a profound restructuring of the SEVN code and the GPU-porting must be carefully investigated.

List of abbreviations

GR – General Relativity

GW - Gravitational Waves

BH – Black Hole

NS – Neutron Star

SN – Supernova

WR – Wolf-Rayet

WDs – White Dwarfs

BBHs – Binary Black Holes

BNSs – Binaries of Neutron Stars

sGRBs – short Gamma-Ray-Bursts

CO – carbon-oxygen

MS – Main Sequence

TAMS – The Terminal-Age Main Sequence

ZAMS – Zero-Age Main Sequence

HG – Hertzsprung Gap

GB – Giant Branch CHeB – Core-Helium Burning

AGB – Asymptotic Giant Branch

LIGO – Laser Interferometer Gravitational-wave Observatory

KAGRA – Kamioka Gravitational Wave Detector

ET – Einstein Telescope

CE – Cosmic Explorer

LISA – Laser Interferometer Space Antenna

SNR – Signal-to-Noise ratio

PISNe – Pair Instability Supernova

PPISNe – Pulsational Pair-Instability Supernova

EOS – Equation Of State

CHE - Chemical Homogeneous Evolution

CE – Common Envelope

CEE – Common Envelope Evolution

RLO – Roche-lobe overflow

HR diagram – Hertzsprung-Russell Diagramm

MSE – Mean Squared Error

BEC – the Bonn Evolutionary Code

MESA – the Modules for Experiments in Stellar Astrophysics

BPSs – Binary Population-Synthesis

SEVN – Stellar Evolution N-body

PARSEC – PAdova TRieste Stellar Evolution Code

SSE – Single Star Evolution

BSE – Binary-Star Evolution

MOBSE – Massive Objects in Binary Stellar Evolution

COSMIC – Compact Object Synthesis and Monte Carlo Investigation Code

BPASS – Binary Population and Spectral Synthesis code

RAM – Random-Access Memory

OpenMP – Open Multi-Processing

MPI – Message Passing Interface

API – Application Programming Interface

RAM – Random Access Memory

CPU – Central Processing Units

GPU – Graphics Processing Unit

NUMA – Non-Uniform Memory Access

VmRSS – Virtual Memory Resident Set Size

SLURM – Simple Linux Utility for Resource Management

List of Figures

List of Tables

Appendix A

Modifications of SEVN code

A.1 Checking the memory status

```
#ifndef SEVN_SEVNMEM_H
#define SEVN_SEVNMEM_H

#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
#include <lookup_and_phases.h>
#include <utilities.h>
#include <mpi.h>

using namespace Lookup;

class SevnMemory{
```

```
class memory_element{


public:
    memory_element(){
        _ncalls = 0;
        _mem_cumulative = 0.0;
    }


    void begin(const double value){
        _start = value;
        _hasstarted = true;
        _hascompleted = false;
    }
    void end(const double value){
        _end = value;
        if(!_hasstarted) {exit(1);} //TODO please return an error here....
            section is closing but has not started
        //_consumed = (selfproc) ? _end - _start : _start - _end; //depending
            if I am checking memory available or memory allocated by the
            process
        _consumed = _end - _start;
        _hasstarted = false;


        _ncalls++;


        _mem_cumulative += _consumed;
        _mem_avg = _mem_cumulative/_ncalls;
```

```
            _hascompleted = true;
        }

        double perstep(){return _consumed;}
        double average(){return _mem_avg;}
        bool hascompleted(){return _hascompleted;}

    private:
        double _start, _end, _consumed;
        double _mem_cumulative, _mem_avg;
        bool _hasstarted, _hascompleted;
        long _ncalls;

    };
public:
    SevnMemory(Lookup::MemoryConversion _output = _megabytes) : output(_output){}

    void begin(std::string section_id){

            auto it = memorymap.find(section_id);

            if (it != memorymap.end()) {
                memorymap[section_id]->begin(read_entry());
                return;
            }

            memorymap.insert(std::make_pair(section_id, new memory_element()));

            memorymap[section_id]->begin(read_entry());
```

```
        return;

}

void end(std::string section_id){


        auto it = memorymap.find(section_id);
        if (it == memorymap.end()) { exit(1); } //TODO please return an error
            here: not in map

        memorymap[section_id]->end(read_entry());

        return;


}

double mem_available(){
    return read_entry("MemAvailable", "/proc/meminfo");
}

    double sevn_memory(){
    return read_entry();
}

double get(std::string section_id){
```

```
        auto it = memorymap.find(section_id);
        if (it == memorymap.end()) { exit(1); } //TODO please return an error
            here: not in map

        return memorymap[section_id]->perstep();
}

void dump(){
        std::string unit;
        if (output == _megabytes) unit = "MB";
        else if (output == _gigabytes) unit = "GB";
        else if (output == _bytes) unit = "B";
        else if (output == _kilobytes) unit = "kB";
        else if (output == _terabytes) unit = "TB";
        else exit(1); //TODO please return a proper error here.... unknown
            output string

        std::cout << std::endl;
        std::cout << "*********************************" << std::endl;
        std::cout << " SEVN total allocated memory = " << sevn_memory() << "
            " << unit << std::endl;
        std::cout << " System remaining memory = " << mem_available() << " "
            << unit << std::endl;

        for (auto const &x : memorymap) {
            if (x.second->hascompleted()) {
                std::cout << " Section [" << x.first << "]" << std::endl;
                std::cout << std::scientific << std::setprecision(4) << "
                    -------------- Memory addition (step) = ["
```

```
                        << x.second->perstep() << "] " << unit
                        << std::endl;
                std::cout << "*********************************" << std::endl;
            }
        }
        return;
    }

private:

    double read_entry(std::string input_string = "VmRSS", std::string filename =
        "/proc/self/status"){

        in.close();
        in.open(filename); //TODO this works only on Linux... for MacOS please
            use the vm_stat command

        for (;;) {

            std::string line, value, token;
            std::vector<std::string> tmp;

            stream.clear();
            getline(in, line);
            stream.str(line);
            if (in.eof()) {
                in.close();
                exit(1);
```

```
        }//TODO return a proper error here... critical... entry_string not
            found


        while (stream >> value) {
            tmp.push_back(value);
            stream.clear();
        }

        std::string delimiter = ":"; //remove the delimiter from the meminfo
            entry
        size_t pos = 0;
        token = ((pos = tmp[0].find(delimiter)) != std::string::npos) ?
            tmp[0].substr(0, pos) : tmp[0];

        if (token == input_string) {

            get_units(tmp);

            in.close();
            int exponent = units - output;
            long pow2 = ((long) 1 << std::abs(exponent));
            double memory_value = utilities::s2n<double>(tmp[1], __FILE__,
                __LINE__);
            return (exponent < 0) ? memory_value / pow2 : memory_value * pow2;
        }
    }
}
void get_units(std::vector<std::string> &tmp){
```

```
    if (tmp[2].find("kB") != std::string::npos || tmp[2].find("KB") !=
        std::string::npos ||
        tmp[2].find("kb") != std::string::npos)
        units = _kilobytes;
    else if (tmp[2].find("MB") != std::string::npos || tmp[2].find("mB") !=
        std::string::npos ||
             tmp[2].find("mb") != std::string::npos)
        units = _megabytes;
    else if (tmp[2].find("GB") != std::string::npos || tmp[2].find("gB") !=
        std::string::npos ||
             tmp[2].find("gb") != std::string::npos)
        units = _gigabytes;
    else if (tmp[2].find("B") != std::string::npos || tmp[2].find("b") !=
        std::string::npos) units = _bytes;
    else {
        in.close();
        std::cout << "Unit not recognized in proc meminfo [" << tmp[2] << "]"
            << std::endl;
        exit(1);
    } //TODO please return a critical error here.... unit not recognized in
        /proc/meminfo

}



std::istringstream stream;
std::ifstream in;
Lookup::MemoryConversion units, output;
```

```
    typedef std::map<std::string, memory_element*> MEMPROFID;
    MEMPROFID memorymap;

};

#endif //SEVN_SEVNMEM_H
```

A.2 Adaptive chunk implementation

```
template<typename T> long adapt_evolve_chunk(IO *sevnio) {

        if(sevnio->svpar.get_num("ev_base_Nchunk") >=
            sevnio->STARS_MATRIX.size()) return sevnio->STARS_MATRIX.size();

        long base_chunk = (long)sevnio->svpar.get_num("ev_base_Nchunk");
        double mem_perc_max = 1; //sevnio->svpar.get_num("ev_max_mem_perc");
        std::cout << sevnio->svpar.get_num("ev_max_mem_perc") << " max perc " <<
            mem_perc_max ;
        double available = sevnio->svmem.mem_available();
        double memory_target_max = mem_perc_max * available;

        std::vector<std::unique_ptr<System>> systems;

        size_t index = 0;

        sevnio->svmem.begin("Load CHUNK stars");
```

```
        for (long i = 0; i < base_chunk ; i++) {
            systems.emplace_back(new T(sevnio, sevnio->STARS_MATRIX[index],
                index));
            index++;
        }
        sevnio->svmem.end("Load CHUNK stars");

        double memory_current = sevnio->svmem.get("Load CHUNK stars");

        return (base_chunk * memory_target_max / memory_current);
    }
```

A.3 SEVNomp class

```
#ifndef SEVN_SEVNOMP_H
#define SEVN_SEVNOMP_H

class SEVNomp{

public:
    SEVNomp(int num_threads) : nthreads(num_threads){
        evolving = new long int [nthreads];
        previous = new long int [nthreads];
        msg30 = new bool [nthreads];
        timer = new double [nthreads];
        start = new struct timespec [nthreads];
        stop = new struct timespec [nthreads];
```

```
}

~SEVNomp(){
    delete [] evolving;
    delete [] previous;
    delete [] msg30;
    delete [] timer;
    delete [] start;
    delete [] stop;
}

inline long int get_start(int toevolve){
    long int bin_per_thread = (long int)
        (std::ceil(toevolve/(double)nthreads));
    return(omp_get_thread_num()*bin_per_thread);
}

inline long int get_end(int toevolve){
    long int bin_per_thread = (long int)
        (std::ceil(toevolve/(double)nthreads));
    return(std::min(bin_per_thread*(omp_get_thread_num() + 1), (long
        int)toevolve));
}

    inline void set_evolving(long int i){
    evolving[omp_get_thread_num()] = i + start_system;
}

void set_completed() {
```

```
        evolving[omp_get_thread_num()] = -1; //not evolving anymore

#pragma omp single
        {

            for(int i = 0; i < omp_get_num_threads(); i++) {
                timer[i] = 0.0;
                clock_gettime(CLOCK_REALTIME, &start[i]);
            }

            for(int i = 0; i < omp_get_num_threads(); i++) {
                previous[i] = evolving[i];
                msg30[i] = false;
            }

            for (;;) {

                int sum = 0;
                for(int i = 0; i < omp_get_num_threads(); i++)
                    sum = (evolving[i] == -1) ? sum + 1 : sum;

                if (sum == omp_get_num_threads()) break; //all threads have
                    finished their chunk... everything is fine

                for(int i = 0; i < omp_get_num_threads(); i++){

                    if(timer[i] >= 60.0){
                        if(evolving[i] != -1 && previous[i] == evolving[i]){
```

```
            std::cout<<"Thread "<<i<<" is still(?) stalling
                evolving system number "<<evolving[i]<<std::endl;
            std::cout<<"Terminating execution"<<std::endl;
            exit(1); //TODO return exception here with complete
                message
        }
        else {
            previous[i] = evolving[i];
            msg30[i]=false;
            timer[i] = 0.0; //reset timer... the core is just slow,
                but it is not stalling
            clock_gettime(CLOCK_REALTIME, &start[i]);
        }
    }

    if (timer[i] >= 30.0) { ////30-second range to warn the user
        about possible stalling
        if(evolving[i] != -1){
            if(evolving[i] == previous[i]) {
                if(!msg30[i]) {
                    std::cout << "Thread " << i << " is probably
                        stalling evolving system number "
                            << evolving[i] << std::endl;
                    msg30[i] = true;
                }
            }
            else {
                previous[i] = evolving[i];
                msg30[i]=false;
```

```
                        timer[i] = 0.0; //reset timer... the core is just
                            slow, but it is not stalling
                        clock_gettime(CLOCK_REALTIME, &start[i]);
                    }
                }
            }


            clock_gettime(CLOCK_REALTIME, &stop[i]);
            timer[i] = ((stop[i].tv_nsec - start[i].tv_nsec) < 0) ?
                      (stop[i].tv_sec - start[i].tv_sec - 1) + (1e9 +
                         stop[i].tv_nsec - start[i].tv_nsec) / 1.0e9 :
                         (stop[i].tv_sec - start[i].tv_sec) +
                         (stop[i].tv_nsec - start[i].tv_nsec) / 1.0e9;

        }
      }
    }

#pragma omp barrier ////be sure that other threads wait for the master
  }

  void set_offset(long int offset){
    start_system = offset;
    return;
  }

private:
  int nthreads;
```

```
    long int *evolving;
    long int *previous;
    double *timer;
    long int start_system;
    bool *msg30;
    struct timespec* start;
    struct timespec* stop;

};

#endif //SEVN_SEVNOMP_H
```

A.4 OpenMP implementation

```
    int evolve_list(EvolveFunctor* evolve_function,
        std::vector<std::unique_ptr<System>>& systems, _UNUSED IO& sevnio,
        SEVNomp *sevnomp, int Nevolve=-1){
        SevnLogging svlog;

        if (Nevolve==-1) Nevolve=systems.size();
        unsigned Nfailed =0;

        #pragma omp parallel num_threads(sevnio.nthreads) reduction(+: Nfailed)
        {
            long int omp_start = sevnomp->get_start(Nevolve);
            long int omp_end = sevnomp->get_end(Nevolve);

            for (long int i = omp_start; i < omp_end; i++) {
```

```
            sevnomp->set_evolving(i);
            utilities::mtrand.seed(systems[i]->get_rseed());//Set random state
                for riproducibility


            T_BEGIN("Binary")
            /****** EVOLVE ******/
            if((*evolve_function)(systems[i])==EXIT_FAILURE)
                Nfailed++;
            T_END("Binary")

        }

        sevnomp->set_completed();

    }


    return Nfailed;
}
```

A.5 MPI implementation

```
#ifndef SEVN_EVOLVE_H
#define SEVN_EVOLVE_H
```

```
#include <star.h>
#include <binstar.h>
#include <IO.h>
#include <vector>
#include <omp.h>
#include <random>
#include <errhand.h>
#include <sevnlog.h>
#include <sevnomp.h>
#include <sevnmem.h>
#include <mpi.h>
using sevnstd::SevnLogging;


    /**
     * Evolve using chunk version with functor
     * @tparam T It could be Binstar or Star
     * @param evolve_function Pointer to functor derived from base class
        EvolveFunctor
     * @param Nchunk Number of systems to evolve in each chunk
     * @param sevnio Instance of the IO class (the one linked to the binaries)
     * @param systems Vector containing the systems to evolve.
     * @param progress If true print progress information to the standard output
     * @return Number of failed evolutions
     * @Note The vector of systems is passed by reference and it is cleared if
        not empty yet.
     */
    template <typename T>
```

```
inline int chunk_dispatcher(EvolveFunctor* evolve_function, IO& sevnio, bool
    progress=true){

    long Nchunk = adapt_evolve_chunk<T>(&sevnio);

    SevnLogging sevnlog;

    SEVNomp sevnomp(sevnio.nthreads);
    ////////////////////////////////
    int numprocs;
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    //Get the rank of this process in MPI_COMM_WORLD
    int procid;
    MPI_Comm_rank(MPI_COMM_WORLD, &procid);
    ////////////////////////////////
    std::cout<< "  "<<Nchunk << " adaptive chunk   "<<   numprocs << "
        numprocs  "<< Nchunk/numprocs << " ";
    Nchunk = Nchunk/numprocs;
    ////////////////////////////////
    std::vector<std::unique_ptr<System>> systems;

           ///Preliminary assignment
    //Max size
    long Ntot = sevnio.STARS_MATRIX.size();
    long Tot_binaries = Ntot;
    //Reserve space for systems
    systems.reserve(Nchunk);
```

```
///Cycle
if (progress)
    std::cout<<"Evolving systems:"<<std::endl;

unsigned int Ndone=0;
size_t current_idx=0;
int Ntodo=0;
int Nfailed=0;
////////////////////////////////////////////////////////////////////////
unsigned int partition =std::ceil((double) Ntot /(double) numprocs);
////////////////////////////////////////////////////////////////////////

Ntot = partition ;
current_idx = procid*partition;

Ntot = current_idx + Ntot > Tot_binaries ? Tot_binaries - current_idx:
    Ntot;

while (Ndone<Ntot){
    Ntodo = current_idx + Ntodo > Tot_binaries ? Tot_binaries -
        current_idx : Ntot-Ndone;

    //Assign Ntodo
    Ntodo = std::min(Ntot-Ndone, Nchunk);

    //Fill vector
    sevnomp.set_offset(current_idx);
    T_BEGIN("EmplaceBack")
    for (size_t i = 0; i < (size_t) Ntodo; i++) {
```

```
    try{
        systems.emplace_back(new T(&sevnio,
            sevnio.STARS_MATRIX[current_idx], current_idx));
    }
    catch(sevnstd::sevnio_error& e){ //sevnio error contains
        initialisation errors
        sevnio.print_failed_initilisation_summary(current_idx);
        sevnlog.error("Failed initilisation for System with
            ID="+utilities::n2s(current_idx,__FILE__,__LINE__)+
                    " with
                        message:\n"+e.what(),__FILE__,__LINE__,sevnio.svpar.get_bool
    }

    current_idx++;


}
T_END("EmplaceBack")

           //Evolve and update Nfailed

Nfailed+=evolve_list(evolve_function, systems, sevnio, &sevnomp,
    Ntodo);

//Clear
systems.clear();

//Update Ndone
```

```
            Ndone+=Ntodo;

            // sevnmem.adapt_chunk(&Nchunk);

            //TODO 1: Here we can estimate time needed to perform a chunk run and
                estimate the time to the end
            //TODO 2: Maybe Nfailed is not needed because svlog has already
                static counters to warning, error and critical messages
            if (progress){
                std::cout<<"\r"<<Ndone<<"/"<<Ntot<<" (Nfailed:"<<Nfailed<<")";
                std::cout<<std::flush;
            }


        }

        if (progress)
            std::cout<<std::endl;

        return EXIT_SUCCESS;
    }


}


#endif //SEVN_EVOLVE_H
```

Bibliography

[1] B. P. Abbott, R. Abbott et al., "Observation of gravitational waves from a binary black hole merger."

[2] ——, vol. 818, no. 2, p. L22, feb 2016.
[Online]. Available: https://dx.doi.org/10.3847/2041-8205/818/2/L22

[3] B. Abbott, R. Abbott et al., "GWTC-1: A gravitational-wave transient catalog of compact binary mergers observed by LIGO and virgo during the first and second observing runs," Physical Review X, vol. 9, no. 3, sep 2019.
[Online]. Available: https://doi.org/10.1103%2Fphysrevx.9.031040

[4] B. P. Abbott, R. Abbott et al., "Gwtc-1: A gravitational-wave transient catalog of compact binary mergers observed by ligo and virgo during the first and second observing runs," Physical Review X, vol. 9, no. 3, Sep 2019.
[Online]. Available: http://dx.doi.org/10.1103/PhysRevX.9.031040

[5] The LIGO Scientific Collaboration and the Virgo Collaboration, "Gwtc-2: Compact binary coalescences observed by ligo and virgo during the first half of the third observing run," Physical Review X, vol. 11, no. 2, Jun 2021.
[Online]. Available: http://dx.doi.org/10.1103/PhysRevX.11.021053

[6] ——, "Gwtc-2.1: Deep extended catalog of compact binary coalescences observed by ligo and virgo during the first half of the third observing run," 2021.

[7] The LIGO Scientific Collaboration and the Virgo Collaboration and the KAGRA Collaboration, "Gwtc-3: Compact binary coalescences observed by ligo and virgo during the second part of the third observing run," 2021.

[8] R. Abbott, T. D. Abbott et al., "Gw190814: Gravitational waves from the coalescence of a 23 solar mass black hole with a 2.6 solar mass compact object," The Astrophysical Journal Letters, vol. 896, no. 2, p. L44, jun 2020.
[Online]. Available: https://dx.doi.org/10.3847/2041-8213/ab960f

[9] LIGO Scientific Collaboration and Virgo Collaboration, "Gw190521: A binary black hole merger with a total mass of 150 $M_{\odot}$," Phys. Rev. Lett., vol. 125, p. 101102, Sep 2020.
[Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.125.101102

[10] R. Abbott, T. D. Abbott et al., "Properties and astrophysical implications of the 150 m binary black hole merger gw190521," The Astrophysical Journal Letters, vol. 900, no. 1, p. L13, sep 2020.
[Online]. Available: https://dx.doi.org/10.3847/2041-8213/aba493

[11] I. Romero-Shaw, P. D. Lasky et al., "Gw190521: Orbital eccentricity and signatures of dynamical formation in a binary black hole merger signal," The Astrophysical Journal Letters, vol. 903, no. 1, p. L5, oct 2020.
[Online]. Available: https://dx.doi.org/10.3847/2041-8213/abbe26

[12] V. Gayathri, J. Healy et al., "Eccentricity estimate for black hole mergers with numerical relativity simulations," 2020.
[Online]. Available: https://arxiv.org/abs/2009.05461

[13] A. M. Holgado, A. Ortega, and C. L. Rodriguez, "Dynamical formation scenarios for gw190521 and prospects for decihertz gravitational-wave astronomy with

gw190521-like binaries," The Astrophysical Journal Letters, vol. 909, no. 2, p. L24, mar 2021.
[Online]. Available: https://dx.doi.org/10.3847/2041-8213/abe7f5

[14] B. P. Abbott, R. Abbott et al., "Gw170817: Observation of gravitational waves from a binary neutron star inspiral," Phys. Rev. Lett., vol. 119, p. 161101, Oct 2017.
[Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.119.161101

[15] C. S. UNNIKRISHNAN, "IndIGO AND LIGO-INDIA: SCOPE AND PLANS FOR GRAVITATIONAL WAVE RESEARCH AND PRECISION METROLOGY IN INDIA," International Journal of Modern Physics D, vol. 22, no. 01, p. 1341010, jan 2013.
[Online]. Available: https://doi.org/10.1142%2Fs0218271813410101

[16] S. Fairhurst, "Localization of transient gravitational wave sources: beyond triangulation," Classical and Quantum Gravity, vol. 35, no. 10, p. 105002, apr 2018.
[Online]. Available: https://dx.doi.org/10.1088/1361-6382/aab675

[17] R. X. Adhikari, K. Arai et al., "A cryogenic silicon interferometer for gravitational-wave detection," Classical and Quantum Gravity, vol. 37, no. 16, p. 165003, jul 2020.
[Online]. Available: https://doi.org/10.1088%2F1361-6382%2Fab9143

[18] M. Punturo, M. Abernathy et al., "The einstein telescope: a third-generation gravitational wave observatory," Classical and Quantum Gravity, vol. 27, no. 19, p. 194002, sep 2010.
[Online]. Available: https://dx.doi.org/10.1088/0264-9381/27/19/194002

[19] B. P. Abbott, R. Abbott et al., "Exploring the sensitivity of next generation gravitational wave detectors," Classical and Quantum Gravity, vol. 34, no. 4, p. 044001, Feb. 2017.

[20] N. Cornish and T. Robson, "Galactic binary science with the new LISA design," Journal of Physics: Conference Series, vol. 840, p. 012024, may 2017.
[Online]. Available: https://doi.org/10.1088%2F1742-6596%2F840%2F1%2F012024

[21] P. Amaro-Seoane, H. Audley et al., "Laser interferometer space antenna," 2017.

[22] A. Klein, E. Barausse et al., "Science with the space-based interferometer eLISA: Supermassive black hole binaries," Physical Review D, vol. 93, no. 2, jan 2016.
[Online]. Available: https://doi.org/10.1103%2Fphysrevd.93.024003

[23] É . É. Flanagan and S. A. Hughes, "Measuring gravitational waves from binary black hole coalescences. i. signal to noise for inspiral, merger, and ringdown," Physical Review D, vol. 57, no. 8, pp. 4535–4565, apr 1998.
[Online]. Available: https://doi.org/10.1103%2Fphysrevd.57.4535

[24] A. Sesana, "Prospects for Multiband Gravitational-Wave Astronomy after GW150914," , vol. 116, no. 23, p. 231102, Jun. 2016.

[25] P. Amaro-Seoane, J. R. Gair et al., "Intermediate and extreme mass-ratio inspirals—astrophysics, science applications and detection using LISA," Classical and Quantum Gravity, vol. 24, no. 17, pp. R113–R169, aug 2007.
[Online]. Available: https://doi.org/10.1088%2F0264-9381%2F24%2F17%2Fr01

[26] C. P. L. Berry and J. R. Gair, "Expectations for extreme-mass-ratio bursts from the galactic centre," Monthly Notices of the Royal Astronomical Society, vol. 435, no. 4, pp. 3521–3540, sep 2013.
[Online]. Available: https://doi.org/10.1093%2Fmnras%2Fstt1543

[27] S. E. Woosley, A. Heger, and T. A. Weaver, "The evolution and explosion of massive stars," Rev. Mod. Phys., vol. 74, pp. 1015–1071, Nov 2002.
[Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.74.1015

[28] S. J. Smartt, "Progenitors of Core-Collapse Supernovae," , vol. 47, no. 1, pp. 63–106, Sep. 2009.

[29] S. Horiuchi, J. F. Beacom et al., "The cosmic core-collapse supernova rate does not match the massive-star formation rate," The Astrophysical Journal, vol. 738, no. 2, p. 154, aug 2011.
[Online]. Available: https://dx.doi.org/10.1088/0004-637X/738/2/154

[30] S. J. Smartt, "Observational constraints on the progenitors of core-collapse supernovae: The case for missing high-mass stars," Publications of the Astronomical Society of Australia, vol. 32, p. e016, 2015.

[31] V. S. Aguirre, "Stellar evolution and modelling stars," in Asteroseismology and Exoplanets: Listening to the Stars and Searching for New Worlds, T. L. Campante, N. C. Santos, and M. J. P. F. G. Monteiro, Eds. Cham: Springer International Publishing, 2018, pp. 3–25.

[32] J. S. Vink, "Theory and Diagnostics of Hot Star Mass Loss," , vol. 60, pp. 203–246, Aug. 2022.

[33] M. Spera, A. A. Trani, and M. Mencagli, "Compact binary coalescences: Astrophysical processes and lessons learned," Galaxies, vol. 10, no. 4, p. 76, jun 2022.
[Online]. Available: https://doi.org/10.3390%2Fgalaxies10040076

[34] M. Spera, M. Mapelli et al., "Merging black hole binaries with the SEVN code," Monthly Notices of the Royal Astronomical Society, vol. 485, no. 1, pp. 889–907, feb 2019.
[Online]. Available: https://doi.org/10.1093%2Fmnras%2Fstz359

[35] M. Spera, M. Mapelli, and A. Bressan, "The mass spectrum of compact remnants from the parsec stellar evolution tracks," Monthly Notices of the Royal Astronomical Society, vol. 451, no. 4, pp. 4086–4103, 06 2015.

[Online]. Available: https://doi.org/10.1093/mnras/stv1161

[36] M. Spera and M. Mapelli, "Very massive stars, pair-instability supernovae and intermediate-mass black holes with the sevn code," Monthly Notices of the Royal Astronomical Society, vol. 470, no. 4, pp. 4739–4749, 06 2017.
[Online]. Available: https://doi.org/10.1093/mnras/stx1576

[37] A. Bressan, P. Marigo et al., "PARSEC: stellar tracks and isochrones with the PAdova and TRieste Stellar Evolution Code," , vol. 427, no. 1, pp. 127–145, Nov. 2012.

[38] Y. Chen, L. Girardi et al., "Improving PARSEC models for very low mass stars," , vol. 444, no. 3, pp. 2525–2543, Nov. 2014.

[39] J. Tang, A. Bressan et al., "New PARSEC evolutionary tracks of massive stars at low metallicity: testing canonical stellar evolution in nearby star-forming dwarf galaxies," , vol. 445, no. 4, pp. 4287–4305, Dec. 2014.

[40] J. Montalbán, A. Bressan et al., "PARSEC evolutionary tracks and isochrones including seismic properties," in Rediscovering Our Galaxy, C. Chiappini, I. Minchev et al., Eds., vol. 334, Aug. 2018, pp. 343–344.

[41] P. Marigo, L. Girardi et al., "A New Generation of PARSEC-COLIBRI Stellar Isochrones Including the TP-AGB Phase," , vol. 835, no. 1, p. 77, Jan. 2017.

[42] X. Fu, A. Bressan et al., "New PARSEC database of alpha enhanced stellar evolutionary tracks and isochrones for Gaia," IAU Focus Meeting, vol. 29B, pp. 144–146, Jan. 2016.

[43] H.-T. Janka, Neutrino-Driven Explosions. Cham: Springer International Publishing, 2017, pp. 1095–1150.
[Online]. Available: https://doi.org/10.1007/978-3-319-21846-5_109

[44] ——, "Explosion mechanisms of core-collapse supernovae," Annual Review of Nuclear and Particle Science, vol. 62, no. 1, pp. 407–451, 2012.
[Online]. Available: https://doi.org/10.1146/annurev-nucl-102711-094901

[45] A. Mezzacappa, E. Endeve et al., "Physical, numerical, and computational challenges of modeling neutrino transport in core-collapse supernovae," 2020.
[Online]. Available: https://arxiv.org/abs/2010.09013

[46] C. D. Bailyn, R. K. Jain et al., "The mass distribution of stellar black holes," The Astrophysical Journal, vol. 499, no. 1, p. 367, may 1998.
[Online]. Available: https://dx.doi.org/10.1086/305614

[47] F. Özel, D. Psaltis et al., "The black hole mass distribution in the galaxy," The Astrophysical Journal, vol. 725, no. 2, p. 1918, dec 2010.
[Online]. Available: https://dx.doi.org/10.1088/0004-637X/725/2/1918

[48] W. M. Farr, N. Sravan et al., "The mass distribution of stellar-mass black holes," The Astrophysical Journal, vol. 741, no. 2, p. 103, oct 2011.
[Online]. Available: https://dx.doi.org/10.1088/0004-637X/741/2/103

[49] B. P. Abbott, R. Abbott et al., "Binary black hole population properties inferred from the first and second observing runs of advanced LIGO and advanced virgo," The Astrophysical Journal, vol. 882, no. 2, p. L24, sep 2019.
[Online]. Available: https://doi.org/10.3847%2F2041-8213%2Fab3800

[50] F. Özel and P. Freire, "Masses, radii, and the equation of state of neutron stars," Annual Review of Astronomy and Astrophysics, vol. 54, no. 1, pp. 401–440, 2016.
[Online]. Available: https://doi.org/10.1146/annurev-astro-081915-023322

[51] P. C. C. Freire, S. M. Ransom et al., "Eight new millisecond pulsars in ngc 6440 and ngc 6441," The Astrophysical Journal, vol. 675, no. 1, p. 670, mar 2008.
[Online]. Available: https://dx.doi.org/10.1086/526338

[52] B. Margalit and B. D. Metzger, "Constraining the maximum mass of neutron stars from multi-messenger observations of gw170817," The Astrophysical Journal Letters, vol. 850, no. 2, p. L19, nov 2017.
[Online]. Available: https://dx.doi.org/10.3847/2041-8213/aa991c

[53] L. Kreidberg, C. D. Bailyn et al., "Mass measurements of black holes in x-ray transients: Is there a mass gap?" The Astrophysical Journal, vol. 757, no. 1, p. 36, sep 2012.
[Online]. Available: https://dx.doi.org/10.1088/0004-637X/757/1/36

[54] T. B. Littenberg, B. Farr et al., "Neutron stars versus black holes: Probing the mass gap with ligo/virgo," The Astrophysical Journal Letters, vol. 807, no. 2, p. L24, jul 2015.
[Online]. Available: https://dx.doi.org/10.1088/2041-8205/807/2/L24

[55] I. Mandel, C.-J. Haster et al., "Distinguishing types of compact-object binaries using the gravitational-wave signatures of their mergers," Monthly Notices of the Royal Astronomical Society: Letters, vol. 450, no. 1, pp. L85–L89, 04 2015.
[Online]. Available: https://doi.org/10.1093/mnrasl/slv054

[56] E. D. Kovetz, I. Cholis et al., "Black hole mass function from gravitational wave measurements," Phys. Rev. D, vol. 95, p. 103010, May 2017.
[Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.95.103010

[57] I. Mandel, W. M. Farr et al., "Model-independent inference on compact-binary observations," Monthly Notices of the Royal Astronomical Society, vol. 465, no. 3, pp. 3254–3260, 11 2016.
[Online]. Available: https://doi.org/10.1093/mnras/stw2883

[58] A. Heger and S. E. Woosley, "The nucleosynthetic signature of population iii," The Astrophysical Journal, vol. 567, no. 1, p. 532, mar 2002.

[Online]. Available: https://dx.doi.org/10.1086/338487

[59] K. Belczynski, A. Heger et al., "The effect of pair-instability mass loss on black-hole mergers," , vol. 594, p. A97, Oct. 2016.

[60] S. E. Woosley, "Pulsational pair-instability supernovae," The Astrophysical Journal, vol. 836, no. 2, p. 244, feb 2017.
[Online]. Available: https://doi.org/10.3847/1538-4357/836/2/244

[61] P. Marchant, M. Renzo et al., "Pulsational pair-instability supernovae in very close binaries," The Astrophysical Journal, vol. 882, no. 1, p. 36, aug 2019.
[Online]. Available: https://doi.org/10.3847%2F1538-4357%2Fab3426

[62] M. Safarzadeh, A. S. Hamers et al., "Formation and merging of mass gap black holes in gravitational-wave merger events from wide hierarchical quadruple systems," The Astrophysical Journal, vol. 888, no. 1, p. L3, dec 2019.
[Online]. Available: https://doi.org/10.3847%2F2041-8213%2Fab5dc8

[63] C. L. Rodriguez, S. Chatterjee, and F. A. Rasio, "Binary black hole mergers from globular clusters: Masses, merger rates, and the impact of stellar evolution," Physical Review D, vol. 93, no. 8, Apr 2016.
[Online]. Available: http://dx.doi.org/10.1103/PhysRevD.93.084029

[64] A. Askar, M. Szkudlarek et al., "MOCCA-SURVEY Database - I. Coalescing binary black holes originating from globular clusters," , vol. 464, no. 1, pp. L36–L40, Jan. 2017.

[65] J. Samsing, "Eccentric black hole mergers forming in globular clusters," , vol. 97, no. 10, p. 103014, May 2018.

[66] C. L. Rodriguez, M. Zevin et al., "Black holes: The next generation—repeated

mergers in dense star clusters and their gravitational-wave properties," Physical Review D, vol. 100, no. 4, Aug 2019.
[Online]. Available: http://dx.doi.org/10.1103/PhysRevD.100.043027

[67] M. Mapelli, "Astrophysics of stellar black holes," 2018.

[68] M. Arca-Sedda and A. Gualandris, "Gravitational wave sources from inspiralling globular clusters in the Galactic Centre and similar environments," Monthly Notices of the Royal Astronomical Society, vol. 477, no. 4, pp. 4423–4442, 04 2018.
[Online]. Available: https://doi.org/10.1093/mnras/sty922

[69] G. Fragione and B. Kocsis, "Black hole mergers from quadruples," Monthly Notices of the Royal Astronomical Society, vol. 486, no. 4, pp. 4781–4789, 05 2019.
[Online]. Available: https://doi.org/10.1093/mnras/stz1175

[70] F. Antonini and F. A. Rasio, "Merging Black Hole Binaries in Galactic Nuclei: Implications for Advanced-LIGO Detections," , vol. 831, no. 2, p. 187, Nov. 2016.

[71] N. C. Stone, B. D. Metzger, and Z. Haiman, "Assisted inspirals of stellar mass black holes embedded in AGN discs: solving the 'final au problem'," , vol. 464, no. 1, pp. 946–954, Jan. 2017.

[72] B. McKernan, K. S. Ford et al., "Constraining stellar-mass black hole mergers in agn disks detectable with ligo," The Astrophysical Journal, vol. 866, no. 1, p. 66, 2018.

[73] F. Antonini, S. Toonen, and A. S. Hamers, "Binary Black Hole Mergers from Field Triples: Properties, Rates, and the Impact of Stellar Evolution," , vol. 841, no. 2, p. 77, Jun. 2017.

[74] G. Fragione, E. Grishin et al., "Black hole and neutron star mergers in galactic nuclei," Monthly Notices of the Royal Astronomical Society, vol. 488, no. 1, pp. 47–63, 2019.

[75] R. F. Webbink, "Evolution of Helium White Dwarfs in Close Binaries," Monthly Notices of the Royal Astronomical Society, vol. 171, no. 3, pp. 555–568, 06 1975.
[Online]. Available: https://doi.org/10.1093/mnras/171.3.555

[76] B. Paczynski, "Common Envelope Binaries," in Structure and Evolution of Close Binary Systems, P. Eggleton, S. Mitton, and J. Whelan, Eds., vol. 73, Jan. 1976, p. 75.

[77] E. P. J. van den Heuvel, "Late Stages of Close Binary Systems," in Structure and Evolution of Close Binary Systems, P. Eggleton, S. Mitton, and J. Whelan, Eds., vol. 73, Jan. 1976, p. 35.

[78] P. Hut, "Tidal evolution in close binary systems." , vol. 99, pp. 126–140, Jun. 1981.

[79] R. F. Webbink, "Double white dwarfs as progenitors of R Coronae Borealis stars and type I supernovae." , vol. 277, pp. 355–360, Feb. 1984.

[80] H. A. Bethe and G. E. Brown, "Evolution of binary compact objects that merge," The Astrophysical Journal, vol. 506, no. 2, p. 780, oct 1998.
[Online]. Available: https://dx.doi.org/10.1086/306265

[81] K. Belczynski, V. Kalogera, and T. Bulik, "A comprehensive study of binary compact objects as gravitational wave sources: Evolutionary channels, rates, and physical properties," The Astrophysical Journal, vol. 572, no. 1, p. 407, jun 2002.
[Online]. Available: https://dx.doi.org/10.1086/340304

[82] J. R. Hurley, C. A. Tout, and O. R. Pols, "Evolution of binary stars and the effect of tides on binary populations," Monthly Notices of the Royal Astronomical Society, vol. 329, no. 4, pp. 897–928, 02 2002.
[Online]. Available: https://doi.org/10.1046/j.1365-8711.2002.05038.x

[83] S. E. de Mink, M. Cantiello et al., "Rotational mixing in close binaries," Proceedings of the International Astronomical Union, vol. 4, no. S252, p. 365–370, 2008.

[84] I. Mandel and S. E. de Mink, "Merging binary black holes formed through chemically homogeneous evolution in short-period stellar binaries," Monthly Notices of the Royal Astronomical Society, vol. 458, no. 3, pp. 2634–2647, 02 2016.
[Online]. Available: https://doi.org/10.1093/mnras/stw379

[85] P. Marchant, N. Langer et al., "A new route towards merging massive black holes," Astronomy &amp Astrophysics, vol. 588, p. A50, mar 2016.
[Online]. Available: https://doi.org/10.1051%2F0004-6361%2F201628133

[86] S. Stevenson, A. Vigna-Gómez et al., "Formation of the first three gravitational-wave observations through isolated binary evolution," Nature Communications, vol. 8, no. 1, apr 2017.
[Online]. Available: https://doi.org/10.1038%2Fncomms14906

[87] N. Giacobbo and M. Mapelli, "The progenitors of compact-object binaries: impact of metallicity, common envelope and natal kicks," Monthly Notices of the Royal Astronomical Society, vol. 480, no. 2, pp. 2011–2030, 07 2018.
[Online]. Available: https://doi.org/10.1093/mnras/sty1999

[88] M. Zevin, S. S. Bavera et al., "One channel to rule them all? constraining the origins of binary black holes using multiple formation pathways," The Astrophysical Journal, vol. 910, no. 2, p. 152, apr 2021.
[Online]. Available: https://dx.doi.org/10.3847/1538-4357/abe40e

[89] J. Kumamoto, M. S. Fujii, and A. Tanikawa, "Gravitational-wave emission from binary black holes formed in open clusters," Monthly Notices of the Royal Astronomical Society, vol. 486, no. 3, pp. 3942–3950, 04 2019.
[Online]. Available: https://doi.org/10.1093/mnras/stz1068

[90] U. N. Di Carlo, M. Mapelli et al., "Binary black holes in young star clusters: the impact of metallicity," Monthly Notices of the Royal Astronomical Society, vol. 498, no. 1, pp. 495–506, 08 2020.
[Online]. Available: https://doi.org/10.1093/mnras/staa2286

[91] A. A. Trani, A. Tanikawa et al., "Spin misalignment of black hole binaries from young star clusters: implications for the origin of gravitational waves events," Monthly Notices of the Royal Astronomical Society, vol. 504, no. 1, pp. 910–919, 04 2021.
[Online]. Available: https://doi.org/10.1093/mnras/stab967

[92] M. A. Sedda, M. Mapelli et al., "Population synthesis of black hole mergers with b-pop: the impact of dynamics, natal spins, and intermediate-mass black holes on the population of gravitational wave sources," 2021.
[Online]. Available: https://arxiv.org/abs/2109.12119

[93] I. Mandel and S. E. de Mink, "Merging binary black holes formed through chemically homogeneous evolution in short-period stellar binaries," Monthly Notices of the Royal Astronomical Society, vol. 458, no. 3, pp. 2634–2647, 02 2016.
[Online]. Available: https://doi.org/10.1093/mnras/stw379

[94] S. E. de Mink and I. Mandel, "The chemically homogeneous evolutionary channel for binary black hole mergers: rates and properties of gravitational-wave events detectable by advanced LIGO," Monthly Notices of the Royal Astronomical Society, vol. 460, no. 4, pp. 3545–3553, 05 2016.
[Online]. Available: https://doi.org/10.1093/mnras/stw1219

[95] P. Marchant, N. Langer et al., "A new route towards merging massive black holes," , vol. 588, p. A50, Apr. 2016.

[96] N. Ivanova, S. Justham et al., "Common envelope evolution: where we stand and

how we can move forward," The Astronomy and Astrophysics Review, vol. 21, no. 1, Feb 2013.
[Online]. Available: http://dx.doi.org/10.1007/s00159-013-0059-2

[97] N. Ivanova and S. Chaichenets, "Common envelope: Enthalpy consideration," The Astrophysical Journal, vol. 731, no. 2, p. L36, Mar 2011.
[Online]. Available: http://dx.doi.org/10.1088/2041-8205/731/2/L36

[98] J. D. M. Dewi and T. M. Tauris, "On the energy equation and efficiency parameter of the common envelope evolution," , vol. 360, pp. 1043–1051, Aug. 2000.

[99] X.-J. Xu and X.-D. Li, "On the binding energy parameter of common envelope evolution," The Astrophysical Journal, vol. 716, no. 1, p. 114–121, May 2010.
[Online]. Available: http://dx.doi.org/10.1088/0004-637X/716/1/114

[100] C. Wang, K. Jia, and X.-D. Li, "The binding energy parameter for common envelope evolution," Research in Astronomy and Astrophysics, vol. 16, no. 8, p. 009, Aug 2016.
[Online]. Available: http://dx.doi.org/10.1088/1674-4527/16/8/126

[101] J. Klencki, G. Nelemans et al., "It has to be cool: Supergiant progenitors of binary black hole mergers from common-envelope evolution," Astronomy Astrophysics, vol. 645, p. A54, Jan 2021.
[Online]. Available: http://dx.doi.org/10.1051/0004-6361/202038707

[102] M. U. Kruckow, T. M. Tauris et al., "Common-envelope ejection in massive binary stars," Astronomy Astrophysics, vol. 596, p. A58, Nov 2016.
[Online]. Available: http://dx.doi.org/10.1051/0004-6361/201629420

[103] ——, "Progenitors of gravitational wave mergers: binary evolution with the stellar grid-based code combine," Monthly Notices of the Royal Astronomical Society, vol. 481, no. 2, p. 1908–1949, Aug 2018.
[Online]. Available: http://dx.doi.org/10.1093/mnras/sty2190

[104] M. Dominik, K. Belczynski et al., "Double Compact Objects. I. The Significance of the Common Envelope on Merger Rates," , vol. 759, no. 1, p. 52, Nov. 2012.

[105] N. Mennekens and D. Vanbeveren, "Massive double compact object mergers: gravitational wave sources and r-process element production sites," , vol. 564, p. A134, Apr. 2014.

[106] K. Belczynski, D. E. Holz et al., "The first gravitational-wave source from the isolated evolution of two stars in the 40-100 solar mass range," , vol. 534, no. 7608, pp. 512–515, Jun. 2016.

[107] J. J. Eldridge and E. R. Stanway, "BPASS predictions for binary black hole mergers," , vol. 462, no. 3, pp. 3302–3313, Nov. 2016.

[108] J. Klencki, M. Moe et al., "Impact of inter-correlated initial binary parameters on double black hole and neutron star mergers," , vol. 619, p. A77, Nov. 2018.

[109] M. Mapelli and N. Giacobbo, "The cosmic merger rate of neutron stars and black holes," , vol. 479, no. 4, pp. 4391–4398, Oct. 2018.

[110] M. U. Kruckow, T. M. Tauris et al., "Progenitors of gravitational wave mergers: binary evolution with the stellar grid-based code COMBINE," , vol. 481, no. 2, pp. 1908–1949, Dec. 2018.

[111] K. Breivik, S. Coughlin et al., "COSMIC variance in binary population synthesis," The Astrophysical Journal, vol. 898, no. 1, p. 71, jul 2020.
[Online]. Available: https://doi.org/10.3847/1538-4357/ab9d85

[112] P. M. Ricker and R. E. Taam, "AN AMR STUDY OF THE COMMON-ENVELOPE PHASE OF BINARY EVOLUTION," The Astrophysical Journal, vol. 746, no. 1, p. 74, jan 2012.
[Online]. Available: https://doi.org/10.1088%2F0004-637x%2F746%2F1%2F74

[113] J.-C. Passy, F. Herwig, and B. Paxton, "The Response of Giant Stars to Dynamical-timescale Mass Loss," , vol. 760, no. 1, p. 90, Nov. 2012.

[114] J. L. A. Nandez and N. Ivanova, "Common envelope events with low-mass giants: understanding the energy budget," , vol. 460, no. 4, pp. 3992–4002, Aug. 2016.

[115] M. MacLeod, A. Antoni et al., "Common Envelope Wind Tunnel: Coefficients of Drag and Accretion in a Simplified Context for Studying Flows around Objects Embedded within Stellar Envelopes," , vol. 838, no. 1, p. 56, Mar. 2017.

[116] T. Fragos, J. J. Andrews et al., "The Complete Evolution of a Neutron-star Binary through a Common Envelope Phase Using 1D Hydrodynamic Simulations," , vol. 883, no. 2, p. L45, Oct. 2019.

[117] A. Frank, B. Balick et al., "Astrophysical Gasdynamics Confronts Reality: The Shaping of Planetary Nebuale," , vol. 404, p. L25, Feb. 1993.

[118] G. Mellema and A. Frank, "Numerical models and our understanding of aspherical planetary nebulae," 1994.
[Online]. Available: https://arxiv.org/abs/astro-ph/9410057

[119] N. Soker and M. Livio, "Disks and jets in planetary nebulae," , vol. 421, p. 219, Jan. 1994.

[120] A. Frank, Z. Chen et al., "Planetary Nebulae Shaped by Common Envelope Evolution," Galaxies, vol. 6, no. 4, p. 113, Oct. 2018.

[121] Y. Zou, A. Frank et al., "Bipolar planetary nebulae from outflow collimation by common envelope evolution," Monthly Notices of the Royal Astronomical Society, vol. 497, no. 3, pp. 2855–2869, jul 2020.
[Online]. Available: https://doi.org/10.1093%2Fmnras%2Fstaa2145

[122] F. A. Rasio and M. Livio, "On the formation and evolution of common envelope systems," The Astrophysical Journal, vol. 471, no. 1, pp. 366–367, nov 1996.
[Online]. Available: https://doi.org/10.1086/177975

[123] M. Livio and N. Soker, "The Common Envelope Phase in the Evolution of Binary Stars," , vol. 329, p. 764, Jun. 1988.

[124] E. L. Sandquist, R. E. Taam et al., "Double Core Evolution. X. Through the Envelope Ejection Phase," , vol. 500, no. 2, pp. 909–922, jun 1998.

[125] J.-C. Passy, O. D. Marco et al., "SIMULATING THE COMMON ENVELOPE PHASE OF a RED GIANT USING SMOOTHED-PARTICLE HYDRODYNAMICS AND UNIFORM-GRID CODES," The Astrophysical Journal, vol. 744, no. 1, p. 52, dec 2011.
[Online]. Available: https://doi.org/10.1088%2F0004-637x%2F744%2F1%2F52

[126] N. Ivanova and J. L. A. Nandez, "Common envelope events with low-mass giants: understanding the transition to the slow spiral-in," Monthly Notices of the Royal Astronomical Society, vol. 462, no. 1, pp. 362–381, jul 2016.
[Online]. Available: https://doi.org/10.1093%2Fmnras%2Fstw1676

[127] S. T. Ohlmann, F. K. Röpke et al., "Constructing stable 3d hydrodynamical models of giant stars," Astronomy &amp Astrophysics, vol. 599, p. A5, feb 2017.
[Online]. Available: https://doi.org/10.1051%2F0004-6361%2F201629692

[128] R. Iaconi, O. De Marco et al., "The effect of binding energy and resolution in simulations of the common envelope binary interaction," Monthly Notices of the Royal Astronomical Society, vol. 477, no. 2, pp. 2349–2365, 03 2018.
[Online]. Available: https://doi.org/10.1093/mnras/sty794

[129] J. L. A. Nandez and N. Ivanova, "Common envelope events with low-mass giants:

understanding the energy budget," Monthly Notices of the Royal Astronomical Society, vol. 460, no. 4, pp. 3992–4002, 05 2016.
[Online]. Available: https://doi.org/10.1093/mnras/stw1266

[130] M. de Kool, "Common Envelope Evolution and Double Cores of Planetary Nebulae," , vol. 358, p. 189, Jul. 1990.

[131] P. F. L. Maxted, R. Napiwotzki et al., "Survival of a brown dwarf after engulfment by a red giant star," Nature, vol. 442, no. 7102, pp. 543–545, aug 2006.
[Online]. Available: https://doi.org/10.1038%2Fnature04987

[132] M. Afşar and C. Ibanoğlu, "Two-colour photometry of the binary planetary nebula nuclei UU Sagitte and V477 Lyrae: oversized secondaries in post-common-envelope binaries," , vol. 391, no. 2, pp. 802–814, Dec. 2008.

[133] M. Zorotovic, M. R. Schreiber et al., "Post-common-envelope binaries from SDSS. IX: Constraining the common-envelope efficiency," , vol. 520, p. A86, Sep. 2010.

[134] E. L. Sandquist, R. E. Taam et al., "Double core evolution. x. through the envelope ejection phase," The Astrophysical Journal, vol. 500, no. 2, p. 909, jun 1998.
[Online]. Available: https://dx.doi.org/10.1086/305778

[135] J. R. Hurley, O. R. Pols, and C. A. Tout, "Comprehensive analytic formulae for stellar evolution as a function of mass and metallicity," Monthly Notices of the Royal Astronomical Society, vol. 315, no. 3, pp. 543–569, 07 2000.
[Online]. Available: https://doi.org/10.1046/j.1365-8711.2000.03426.x

[136] S. F. Portegies Zwart and F. Verbunt, "Population synthesis of high-mass binaries." , vol. 309, pp. 179–196, May 1996.

[137] E. De Donder and D. Vanbeveren, "The influence of neutron star mergers on the

galactic chemical enrichment of r-process elements," New Astronomy, vol. 9, no. 1, pp. 1–16, 2004.
[Online]. Available: https://www.sciencedirect.com/science/article/pii/S1384107603000708

[138] R. G. Izzard, C. A. Tout et al., "A new synthetic model for asymptotic giant branch stars," Monthly Notices of the Royal Astronomical Society, vol. 350, no. 2, pp. 407–426, 05 2004.
[Online]. Available: https://doi.org/10.1111/j.1365-2966.2004.07446.x

[139] K. Belczynski, V. Kalogera et al., "Compact object modeling with the startrack population synthesis code," The Astrophysical Journal Supplement Series, vol. 174, no. 1, p. 223, jan 2008.
[Online]. Available: https://dx.doi.org/10.1086/521026

[140] J. W. Barrett, I. Mandel et al., "Exploring the parameter space of compact binary population synthesis," Proceedings of the International Astronomical Union, vol. 12, no. S325, p. 46–50, 2016.

[141] M. Mapelli, N. Giacobbo et al., "The cosmic merger rate of stellar black hole binaries from the Illustris simulation," , vol. 472, no. 2, pp. 2422–2435, Dec. 2017.

[142] N. Giacobbo, M. Mapelli, and M. Spera, "Merging black hole binaries: The effects of progenitor's metallicity, mass-loss rate and eddington factor," Monthly Notices of the Royal Astronomical Society, vol. 474, 11 2017.

[143] K. Breivik, S. Coughlin et al., "COSMIC Variance in Binary Population Synthesis," , vol. 898, no. 1, p. 71, Jul. 2020.

[144] J. R. Hurley, C. A. Tout, and O. R. Pols, "Evolution of binary stars and the effect of tides on binary populations," , vol. 329, no. 4, pp. 897–928, Feb. 2002.

[145] R. Voss and T. M. Tauris, "Galactic distribution of merging neutron stars and black holes – prospects for short gamma-ray burst progenitors and LIGO/VIRGO," Monthly Notices of the Royal Astronomical Society, vol. 342, no. 4, pp. 1169–1184, 07 2003.
[Online]. Available: https://doi.org/10.1046/j.1365-8711.2003.06616.x

[146] M. U. Kruckow, T. M. Tauris et al., "Progenitors of gravitational wave mergers: binary evolution with the stellar grid-based code ComBinE," Monthly Notices of the Royal Astronomical Society, vol. 481, no. 2, pp. 1908–1949, 08 2018.
[Online]. Available: https://doi.org/10.1093/mnras/sty2190

[147] L. Nelson, "New Population Synthesis Techniques in the Analysis of Interacting Binaries," in Journal of Physics Conference Series, ser. Journal of Physics Conference Series, vol. 341, Feb. 2012, p. 012008.

[148] H.-L. Chen, T. E. Woods et al., "Next generation population synthesis of accreting white dwarfs - I. Hybrid calculations using BSE+MESA," , vol. 445, no. 2, pp. 1912–1923, Dec. 2014.

[149] Y. Shao, X.-D. Li, and Z.-G. Dai, "A Population of Neutron Star Ultraluminous X-Ray Sources with a Helium Star Companion," , vol. 886, no. 2, p. 118, Dec. 2019.

[150] Y. Shao and X.-D. Li, "Population Synthesis of Black Hole Binaries with Compact Star Companions," , vol. 920, no. 2, p. 81, Oct. 2021.

[151] S. S. Bavera, T. Fragos et al., "The impact of mass-transfer physics on the observable properties of field binary black hole populations," , vol. 647, p. A153, Mar. 2021.

[152] J. Román-Garza, S. S. Bavera et al., "The Role of Core-collapse Physics in the Observability of Black Hole Neutron Star Mergers as Multimessenger Sources," , vol. 912, no. 2, p. L23, May 2021.

[153] E. Zapartas, M. Renzo et al., "Revisiting the explodability of single massive star progenitors of stripped-envelope supernovae," , vol. 656, p. L19, Dec. 2021.

[154] T. Fragos, T. Linden et al., "On the formation of ultraluminous x-ray sources with neutron star accretors: The case of m82 x-2," The Astrophysical Journal Letters, vol. 802, no. 1, p. L5, mar 2015.
[Online]. Available: https://dx.doi.org/10.1088/2041-8205/802/1/L5

[155] Y. Shao and X.-D. Li, "A population of ultraluminous x-ray sources with an accreting neutron star," The Astrophysical Journal, vol. 802, no. 2, p. 131, apr 2015.
[Online]. Available: https://dx.doi.org/10.1088/0004-637X/802/2/131

[156] T. Fragos, J. J. Andrews et al., "Posydon: A general-purpose population synthesis code with detailed binary-evolution simulations," 2022.
[Online]. Available: https://arxiv.org/abs/2202.05892

[157] S. S. Bavera, T. Fragos et al., "The impact of mass-transfer physics on the observable properties of field binary black hole populations," Astronomy &amp Astrophysics, vol. 647, p. A153, mar 2021.
[Online]. Available: https://doi.org/10.1051%2F0004-6361%2F202039804

[158] A. Bressan, P. Marigo et al., "PARSEC: stellar tracks and isochrones with the PAdova and TRieste Stellar Evolution Code," Monthly Notices of the Royal Astronomical Society, vol. 427, no. 1, pp. 127–145, 11 2012.
[Online]. Available: https://doi.org/10.1111/j.1365-2966.2012.21948.x

[159] Y. Chen, A. Bressan et al., "parsec evolutionary tracks of massive stars up to 350 M at metallicities 0.0001 Z 0.04," Monthly Notices of the Royal Astronomical Society, vol. 452, no. 1, pp. 1068–1080, 07 2015.
[Online]. Available: https://doi.org/10.1093/mnras/stv1281

[160] M. Mencagli, N. Nazarova, and M. Spera, "ISTEDDAS: a new direct N-Body code to study merging compact-object binaries," in Journal of Physics Conference Series, ser. Journal of Physics Conference Series, vol. 2207, Mar. 2022, p. 012051.