Publications of the Astronomical Society of the Pacific, 130:024501 (5pp), 2018 February https://doi.org/10.1088/1538-3873/aa9dff

MPI_XSTAR: MPI-based Parallelization of the XSTAR Photoionization Program

Ashkbiz Danehkar[1], Michael A. Nowak[2], Julia C. Lee[3], and Randall K. Smith[1]
[1] Harvard-Smithsonian Center for Astrophysics, 60 Garden Street, Cambridge, MA 02138, USA; ashkbiz.danehkar@cfa.harvard.edu
[2] Massachusetts Institute of Technology, Kavli Institute for Astrophysics, Cambridge, MA 02139, USA
[3] Harvard John A. Paulson School of Engineering & Applied Science, 29 Oxford Street, Cambridge, MA 02138 USA
Received 2017 October 31; accepted 2017 November 21; published 2017 December 29

Abstract

We describe a program for the parallel implementation of multiple runs of XSTAR, a photoionization code that is used to predict the physical properties of an ionized gas from its emission and/or absorption lines. The parallelization program, called MPI_XSTAR, has been developed and implemented in the C++ language by using the Message Passing Interface (MPI) protocol, a conventional standard of parallel computing. We have benchmarked parallel multiprocessing executions of XSTAR, using MPI_XSTAR, against a serial execution of XSTAR, in terms of the parallelization speedup and the computing resource efficiency. Our experience indicates that the parallel execution runs significantly faster than the serial execution, however, the efficiency in terms of the computing resource usage decreases with increasing the number of processors used in the parallel computing.

Key words: methods: miscellaneous – techniques: miscellaneous – quasars: absorption lines – X-rays: binaries – X-rays: galaxies – radiative transfer

Online material: color figure

1. Introduction

The Fortran 77 code XSTAR (Kallman et al. 1996; Kallman & Bautista 2001; Kallman et al. 2004, 2009) is a photoionization program used to model the X-ray spectra from astrophysical plasmas. XSTAR was developed to calculate ionization structures, thermal structures, and emissivities of a spherical gaseous cloud with given physical conditions and elemental abundances being ionized by a central source of ionizing radiation that is specified by its luminosity and spectral continuum. It uses an atomic database (Bautista & Kallman 2001) that includes energy levels, recombination rate coefficients, transition probabilities, and cross sections of most of the lines seen in the UV and X-ray bands. To facilitate model fitting for various physical parameters, a Perl script, called XSTAR2XSPEC, was developed and is included in XSTAR that executes multiple serial runs of XSTAR, and utilizes the results of these XSTAR computations to generate multiplicative tabulated models for spectroscopic analysis tools. These tabulated models are the absorption spectrum table (xout_mtable.fits), the reflected emission spectrum table (xout_ain.fits), and the transmitted emission spectrum table in the absorption direction (xout_aout.fits), which are used for the photoionization model fitting in X-ray spectral modeling tools such as XSPEC (Arnaud 1996) and ISIS (Houck et al. 2000).

XSTAR is widely used to obtain physical conditions of X-ray ionized emitters and/or absorbers in active galactic nuclei (AGNs; e.g., Brenneman et al. 2011; Tombesi et al. 2011a, 2011b, 2014; Lee et al. 2013; Laha et al. 2014) and non-equilibrium ionized gaseous clouds surrounding X-ray binaries (e.g., Neilsen et al. 2009, 2011; Miller et al. 2012, 2015). Fundamental physical parameters in typical models are the gas number density n, the hydrogen column density $N_{\rm H} = n_{\rm H} V_f \triangle r$, and the ionization parameter $\xi = L_{\rm ion}/n_{\rm H} r^2$ (Tarter et al. 1969), where $L_{\rm ion}$ is the luminosity (1–1000 Ryd) of the ionizing source, V_f is the volume filling factor of the gaseous cloud, r is the distance from the central source of ionizing radiation and $\triangle r$ is the thickness of the gaseous cloud. In most cases, the gas number density and chemical abundances are fixed, while the variations of the hydrogen column density ($N_{\rm H}$) and the ionization parameter (ξ) are used to constrain the physical properties of an ionized gas. Hence, multiplicative tabulated models are generated on the two-dimensional $N_{\rm H}$–ξ plane, sampling the column density with n intervals and the ionization parameter with m intervals, which require $n \times m$ times XSTAR runs. Taking our typical one-hour XSTAR run (for our specified model settings), XSTAR2XSPEC takes 600 hr (25 days) for $n = 20$ and $m = 30$ to produce tabulated models.

To facilitate parallel executions of multiple XSTAR runs, a Unix shell script together with some S-Lang codes, so-called PVM_XSTAR[4] (Noble et al. 2009), were developed based on the

[4] http://space.mit.edu/cxc/pvm_xstar/

Parallel Virtual Machine (PVM) library[5] (Geist et al. 1994) and the S-Lang PVM module[6] (Davis et al. 2005; Noble et al. 2006). PVM_XSTAR loads the S-Lang PVM module, spawns multiple slave processes for XSTAR runs using the job list created by XSTINITABLE, and invokes XSTAR2TABLE on the results of these XSTAR runs to create table models. The program XSTINITABLE is a code written in the C language, and is included in the FTOOLS package[7] which is used in the initialization step of the Perl script XSTAR2XSPEC to generate the job list file (xstinitable.lis), containing XSTAR calling commands for the variation of the physical parameters used in photoionization modeling, and to create an initial FITS file (xstinitable.fits) required for producing multiplicative tabulated models. The program XSTAR2TABLE is also written in the C language, and included in the FTOOLS package which is invoked by the Perl script XSTAR2XSPEC to generate tabulated models from multiple serial runs of XSTAR. PVM_XSTAR, similar to XSTAR2XSPEC, invokes XSTINITABLE and XSTAR2TABLE from the FTOOLS package, but executes those XSTAR calling commands in parallel. However, the PVM software package is not employed by modern supercomputers. The two most common protocols of parallel computing used by recent supercomputers are Message Passing Interface (MPI) and OpenMP. The MPI protocol is for clusters with distributed memory, while OpenMP supports shared memory systems. Since it may not be possible to use PVM_XSTAR on a given computer cluster, it was necessary to develop a code that permits parallel executions of multiple XSTAR runs using either MPI or OpenMP. As the MPI protocol is used by all modern clusters with distributed memory, we attempted to implement an MPI-based interface for parallelizing the XSTAR program.

This paper presents a parallelization implementation for multiple runs of the XSTAR program, so called MPI_XSTAR, developed using the MPI library in the C++ language. Section 2 describes the design of the mpi_xstar program. Section 3 evaluates the parallelization speedup and the computing resource efficiency of mpi_xstar, followed by a conclusion in Section 4.

2. Parallel MPI Implementation

MPI_XSTAR, a parallel-manager program, has been written in C++ using the high-performance computing industry standard MPI library (e.g., Gropp et al. 1999). The MPI_XSTAR source code is freely available under GPLv3 license on GitHub.[8] MPI_XSTAR is designed to be used on a cluster or a multi-core machine composed of multiple CPUs (central processor units) that enables an easy generation of multiplicative tabulated models for spectral modeling tools. The code has initially been developed and examined on the ODYSSEY cluster at Harvard University. MPI_XSTAR utilizes XSTINITABLE and XSTAR2TABLE, similar to XSTAR2XSPEC and PVM_XSTAR. It uses the MPI library to run the XSTAR calling commands on a number of CPUs (e.g., specified by the np parameter value of mpirun).

An outline of the implementation of an MPI-based parallelization of XSTAR, MPI_XSTAR, is summarized in a pseudo-code in Algorithm 1. The program begins with the generation of XSTAR calling commands for the variation of the input parameters and an initial FITS file using XSTINITABLE in the master processor (rank zero). It passes the command argument [returned by an `MPI_Init()` call] to XSTINITABLE to create the XSTAR calling commands file (xstinitable.lis) and the initial FITS file (xstinitable.fits). It makes three copies of the initial FITS file, which will be used by XSTAR2TABLE at the end to create three table files (xout_mtable.fits, xout_ain.fits, and xout_aout.fits). While the master processor performs the initialization for a parallel setup of XSTAR, other processors are blocked [by using an `MPI_Barrier()` call] and waited. After the master processor completes its initialization, it broadcasts the number of XSTAR calling commands to other processors [by an `MPI_Bcast()` call], that allows each processor to know which XSTAR calling command should be executed.

Algorithm 1. Pseudo-code structure of the MPI_XSTAR program

```
call MPI_Init(argument)
call MPI_Comm_rank(processor_rank)
call MPI_Comm_size(total_processor_number)

if proc_rank = 0 then
    call xstinitable to initialize the job list

call MPI_Barrier()

call MPI_Bcast(total_job_number)

job_number = processor_rank + 1

loop job_number <=total_job_number
    read the xstar command from the job list
    call xstar on the command
    job_number = job_number + 1
    call MPI_Bcast(job_number)

call MPI_Barrier()

if proc_rank = 0 then
    call xstar2table to make the FITS table models

call MPI_Barrier()
call MPI_Finalize()
```

After each processor is allocated to an initial calling order based on the rank value [returned by an `MPI_Comm_rank()` call], the

[5] http://www.csm.ornl.gov/pvm/
[6] http://space.mit.edu/cxc/modules/pvm/
[7] https://heasarc.gsfc.nasa.gov/ftools/
[8] https://github.com/xstarkit/MPI_XSTAR

program begins to read the XSTAR calling command from the initialization file (xstinitable.lis) and execute it by calling the XSTAR program. The calling order of XSTAR commands, which is being executed by a processor, is sent to other processors [by an `MPI_Bcast()` call], so it will not be run by them. Each processor is being blocked until its XSTAR computation is done. After a processor completes its current XSTAR command, it then reads the next calling command and executes a new XSTAR command. When there is no calling command for execution by a processor, those processors, which finish their task, are blocked [by an `MPI_Barrier()` call] and waited for those processors whose XSTAR commands are still running.

After all XSTAR calling commands are executed, the master processor invokes XSTAR2TABLE upon the results of these XSTAR computations in order to generate multiplicative tabulated models, while other processors are blocked [by an `MPI_Barrier()` call]. The FTOOLS program XSTAR2TABLE uses the XSTAR outputs in each folder, and adds them to table files (xout_mtable.fits, xout_ain.fits, and xout_aout.fits) created from the initial FITS file in the initialization step. For a job list containing n XSTAR calling commands, it is required to execute n runs of XSTAR2TABLE upon the results in each folder to generate multiplicative tabulated models. As this procedure is very quick, it is done by the master processor in a serial mode (on a single CPU). After tabulated model FITS files are produced by the master processor, all processors are unblocked and are terminated [by an `MPI_Finalize()` call]. MPI_XSTAR outputs include the standard output log file (XSTAR2XSPEC.LOG; similar to the script XSTAR2XSPEC) and the error report file for certain failure conditions such as the absence of the xstar commands file (xstinitable.lis) and the initial FITS file (xstinitable.fits).

3. Computing Performance

To evaluate the MPI_XSTAR computing performance, we calculated a grid of 9×6 XSTAR models for the two-dimensional $N_{\rm H}$–ξ parameter space, sampling the column density with 9 logarithmic intervals and an interval size of 0.5 from $\log N_{\rm H} = 20$ to $24\,{\rm cm}^{-2}$, and the ionization parameter with 6 logarithmic intervals and an interval size of 1 from $\log \xi = 0$ to $5\,{\rm erg\,cm\,s}^{-1}$, assuming a gas density of $\log n = 12\,{\rm cm}^{-3}$ and a turbulent velocity of $v_{\rm turb} = 100\,{\rm km\,s}^{-1}$. We assumed a spherical geometry with a covering fraction of $C_f = \Omega/4\pi = 0.4$. The chemical composition is assumed to be solar elemental abundances ($A_{\rm Fe} = 1$; Grevesse et al. 1996). The initial gas temperature of $T_{\rm init} = 10^6$ K used here is typical for AGNs (Nicastro et al. 1999; Bianchi et al. 2005). The parameters used for MPI_XSTAR benchmarks are listed in Table 1. We also employed a spectral energy distribution described in Danehkar et al. (2017) as the central source of ionizing radiation with a typical luminosity of $L_{\rm ion} = 10^{44}\,{\rm erg\,s}^{-1}$ (between 1 and 1000 Ryd). We run MPI_XSTAR on the Harvard ODYSSEY cluster, consisting of 60,000 cores and 4 GB of memory per core on average, running the CentOS v6.5 implementation of the Linux operating system and scheduling the jobs using the SLURM v16.05 resource manager. To calculate the speedup and efficiency, we have submitted different MPI_XSTAR jobs with a single CPU and multiple CPUs (2 to 54).

Table 1
Parameters Used for MPI_XSTAR Benchmark Model

Parameter	Value	Interval Size
$\log N_{\rm H}$ (cm^{-3})	20 … 24	0.5
$\log \xi$ (erg cm s^{-1})	0 … 5	1.0
$\log n$ (cm^{-3})	12	…
$v_{\rm turb}$ (km s^{-1})	100	…
$C_f = \Omega/4\pi$	0.4	…
$A_{\rm Fe}$	1.0	…
$T_{\rm init}$ (10^4 K)	100	…
$L_{\rm ion}$ (10^{38} erg s^{-1})	1.0×10^6	…

Note. Logarithmic interval sizes are chosen for the total hydrogen column density ($N_{\rm H}$) and the ionization parameter ($\xi = L_{\rm ion}/n_{\rm H} r^2$).

The speedup $\mathcal{S}(N)$ of a parallel computation with N processors is defined as follows:

$$\mathcal{S}(N) \equiv \frac{\mathcal{T}(1)}{\mathcal{T}(N)}, \tag{1}$$

where $\mathcal{T}(i)$ is the running time for a parallel execution with i processors, so $\mathcal{T}(1)$ corresponds to a serial execution. The speedup for a single processor ($N = 1$) is defined to be $\mathcal{S}(1) = 1$. Ideally, an excellent speedup is achieved when $\mathcal{S}(N) \approx N$.

The efficiency $\mathcal{E}(N)$ in using the computing resources for a parallel computation with N processors is defined as follows:

$$\mathcal{E}(N) \equiv \frac{\mathcal{S}(N)}{N} = \frac{\mathcal{T}(1)}{N \times \mathcal{T}(N)}. \tag{2}$$

The efficiency is typically between zero and one. An efficiency of more than one describes the so-called superlinear speedup. As the speedup $\mathcal{S}(1) = 1$, the efficiency is $\mathcal{E}(1) = 1$ for a single processor ($N = 1$).

Table 2 lists the running time, the speedup, the efficiency of MPI_XSTAR with 1 to 54 CPUs. It can been seen that the running time $\mathcal{T}(N)$ of the parallel executions is significantly shorter than the serial execution. It took around 18 hr to make XSTAR grid models with 32 and 54 CPUs, whereas about 10 days using a single CPU ($N = 1$). Although the speedup $\mathcal{S}(N)$ increases with the number of processors (N), it does not demonstrate an ideal speedup ($\mathcal{S}(N) \approx N$). We also notice that

Table 2
Running Time, Speedup, and Efficiency of MPI_XSTAR

N	$\mathcal{T}(N)$	$\mathcal{S}(N)$	$\mathcal{E}(N)$
1	254:07:25	1.00	1.00
2	127:23:58	1.99	0.99
4	87:09:47	2.92	0.73
8	41:35:41	6.11	0.76
16	35:18:21	7.20	0.45
32	17:42:15	14.35	0.45
54	18:13:30	13.93	0.26

Note. The running time $\mathcal{T}(N)$ is in hours, minutes, and seconds (hh:mm:ss).

the efficiency $\mathcal{E}(N)$ decreases with increasing the number of processors (N).

The performance results for MPI_XSTAR versus N processors are shown in Figure 1, including the running time $\mathcal{T}(N)$, the speedup $\mathcal{S}(N)$ and the efficiency $\mathcal{E}(N)$. As seen in the figure, the speedup and efficiency are not linearly correlated with the number of processors. This is due to the fact that the running time of each XSTAR process greatly varies according to the physical conditions ($\log N_{\rm H}$ and $\log \xi$), so they are not identical to each other. We notice that the running time of a parallel execution is limited by the maximum running time of the XSTAR program for given physical parameters. For our benchmark example, it took between 25 s and 17.5 hr for each XSTAR run, depending on the column density $\log N_{\rm H}$ and the ionization parameter $\log \xi$ used as input parameters. Parallel running times of multiple XSTAR runs do not exceed the maximum running time of a single XSTAR. There should not be much difference between the parallel executions with $N = 32$ and 54. However, as seen in Table 2, the parallel computing with $N = 54$ is roughly a half hour longer than that with $N = 32$. This is due to the fact that each cluster node (of the Harvard ODYSSEY) used in our benchmark consists of 32 cores, so we had to use 2 nodes for the parallel computing with $N = 54$. The inter-node communication time slightly makes two-node parallel-computing (more than 32 CPUs) slower than single-node parallel-computing (only with 32 CPUs).

As the execution time of each single XSTAR restricts the parallel running time of MPI_XSTAR, it prevents us from achieving a prefect speedup ($\mathcal{S}(N) \approx N$). If the internal Fortran 77 routines of the program XSTAR were extended according to one of convention protocols of parallel computing (MPI or OpenMP), an ideal speedup might be achievable. Nevertheless, despite the low computing efficiency of MPI_XSTAR, it provides a major improvement for constructing photoionization grid models for spectroscopic fitting tools such as XSPEC and ISIS. For example, the photoionization table model with the settings listed in Table 1 can be now produced in 18 hr using a parallel execution with 32 CPUs rather than 10 days using a serial execution.

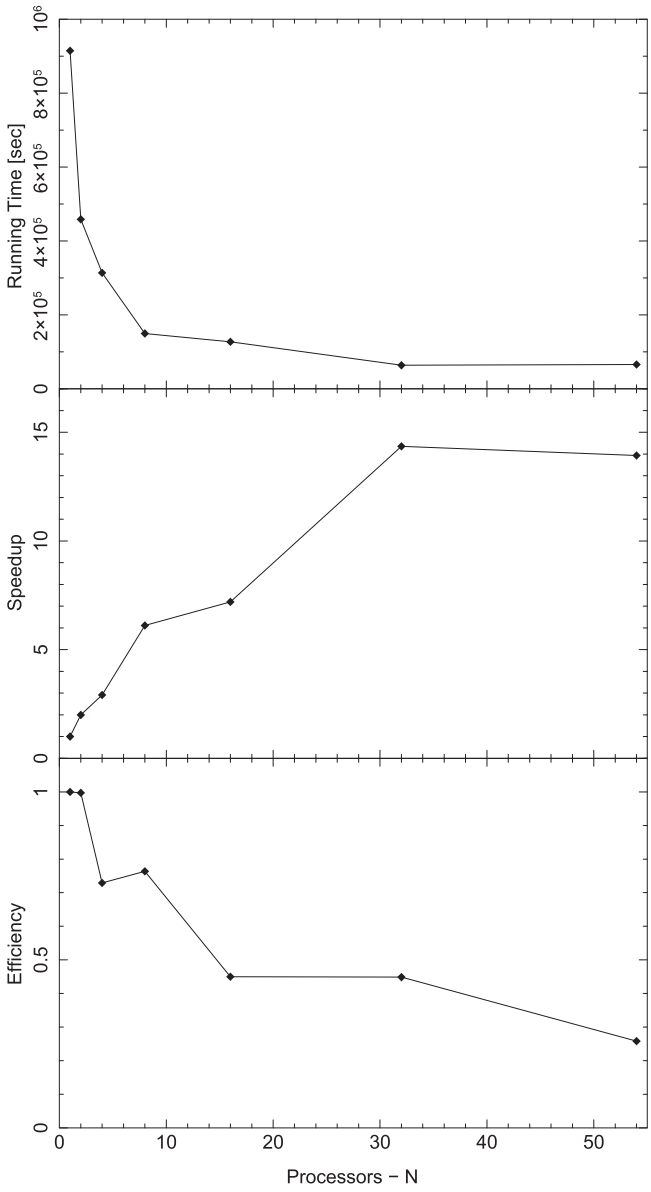

Figure 1. From top to bottom, the running time $\mathcal{T}(N)$, the speedup $\mathcal{S}(N)$, the efficiency $\mathcal{E}(N)$ and as a function of the number of processes (N) for a benchmark photoionization model with the parameters listed in Table 1. The running time $\mathcal{T}(N)$ is in seconds.

4. Conclusion

This paper presents MPI_XSTAR, which is a parallel implementation of multiple XSTAR runs using the MPI protocol (e.g., Gropp et al. 1999) for clusters with distributed memory. MPI_XSTAR expedites the tabulated model generation on the

high-performance computing environments. MPI_XSTAR, similar to XSTAR2XSPEC and PVM_XSTAR, invokes the FTOOLS programs XSTINITABLE and XSTAR2TABLE. The table models take an extremely long time to be produced by XSTAR2XSPEC. Moreover, PVM_XSTAR relies on on the PVM technology (Geist et al. 1994), which is no longer supported by modern supercomputers. Hence, an MPI-based manager for parallelizing XSTAR can overcome the current difficulties in producing the multiplicative tabulated models.

The MPI_XSTAR code that we have developed is available via GitHub (github.com/xstarkit/mpi_xstar). Note that it makes use of the locally installed XSTAR and its associated tools, and will run regardless of XSTAR version so long as the XSTAR parameter inputs and calling sequence do not change. However, should newer versions of XSTAR arise requiring such changes, updates to the MPI_XSTAR code will be made and documented on the GitHub site.

The code was evaluated for the generation of the XSTAR table models with a grid of 9×6 on the $N_{\rm H}$–ξ parameter space. The parallel multiprocessing execution is significantly faster than the serial execution, as the computation, which previously took 10 days, requires only about 18 hr using 32 CPUs. However, our benchmarking study with 1 to 54 CPUs indicates that the parallel efficiency decreases with increasing the number of processors. Moreover, we did not find any linear correlation between the speedup and the number of processors, as shown in Figure 1. Although we did not achieve an ideal speedup ($\mathcal{S}(N) \approx N$), the running times (see Table 2) of parallel execution with 32 and 54 CPUs are enormously shorter than the time of a serial execution. We notice that the performance of MPI_XSTAR is restricted by the maximum running time of a single XSTAR run (about 17.5 hr for our benchmark model results listed in Table 2). However, MPI_XSTAR provides a faster way for the generation of photoionization grid models for spectral model fitting tools such as XSPEC (Arnaud 1996) and ISIS (Houck et al. 2000).

In summary, the new code MPI_XSTAR is able to speed up the photoionization modeling procedure. An important application is a fast generation of photoionization table models of X-ray warm absorbers in AGNs (e.g., Danehkar et al. 2017), whose computation, depending on the number of CPUs requested for the parallel execution, is shorter than a serial execution using XSTAR2XSPEC. The parallelization of XSTAR might be implementable on the graphical processing units (GPU) using the CUDA library. Moreover. it might be possible to parallelize the internal routines (currently in Fortran 77) of the program XSTAR, which will significantly expedite photoionization simulations of ionized gaseous clouds. An MPI-CUDA GPU-based parallelization and rewriting the XSTAR internal routines based on the MPI library for the high-performance computing environments deserve further investigations in the future.

This work was supported by NASA through the CXO grant GO5-16108X. We thank the referee, Tim Kallman, for his careful reading and valuable suggestions. The computations in this paper were run on the Odyssey cluster supported by the FAS Division of Science, Research Computing Group at Harvard University, the Smithsonian Institution High Performance Cluster (Hydra), and the Stampede cluster at the Texas Advanced Computing Center supported by NSF grant ACI-1134872.

ORCID iDs

Ashkbiz Danehkar https://orcid.org/0000-0003-4552-5997

References

Arnaud, K. A. 1996, in ASP Conf. Ser. 101, Astronomical Data Analysis Software and Systems V, ed. G. H. Jacoby & J. Barnes (San Francisco, CA: ASP), 17
Bautista, M. A., & Kallman, T. R. 2001, ApJS, 134, 139
Bianchi, S., Matt, G., Nicastro, F., et al. 2005, MNRAS, 357, 599
Brenneman, L. W., Reynolds, C. S., Nowak, M. A., et al. 2011, ApJ, 736, 103
Danehkar, A., Nowak, M. A., Lee, J. C., et al. 2017, ApJ, submitted
Davis, J. E., Houck, J. C., Allen, G. E., et al. 2005, in ASP Conf. Ser. 347, Astronomical Data Analysis Software and Systems XIV, ed. P. Shopbell, M. Britton, & R. Ebert (San Francisco, CA: ASP), 444
Geist, A., Beguelin, A., Dongarra, J., et al. 1994, in PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing, ed. J. Kowalik (Cambridge, MA: MIT Press)
Grevesse, N., Noels, A., & Sauval, A. J. 1996, in ASP Conf. Ser. 99, Cosmic Abundances, ed. S. S. Holt & G. Sonneborn (San Francisco, CA: ASP), 117
Gropp, W., Lusk, E., & Skjellum, A. 1999, Using MPI: Portable Programming with the Message-Passing Interface (2nd ed.; Cambridge, MA: MIT Press)
Houck, J. C., & Denicola, L. A. 2000, in ASP Conf. Ser. 216, Astronomical Data Analysis Software and Systems IX, ed. N. Manset, C. Veillet, & D. Crabtree (San Francisco, CA: ASP), 591
Kallman, T., & Bautista, M. 2001, ApJS, 133, 221
Kallman, T. R., Bautista, M. A., Goriely, S., et al. 2009, ApJ, 701, 865
Kallman, T. R., Liedahl, D., Osterheld, A., et al. 1996, ApJ, 465, 994
Kallman, T. R., Palmeri, P., Bautista, M. A., et al. 2004, ApJS, 155, 675
Laha, S., Guainazzi, M., Dewangan, G. C., et al. 2014, MNRAS, 441, 2613
Lee, J. C., Kriss, G. A., Chakravorty, S., et al. 2013, MNRAS, 430, 2650
Miller, J. M., Fabian, A. C., Kaastra, J., et al. 2015, ApJ, 814, 87
Miller, J. M., Pooley, G. G., Fabian, A. C., et al. 2012, ApJ, 757, 11
Neilsen, J., Lee, J. C., Nowak, M. A., et al. 2009, ApJ, 696, 182
Neilsen, J., Remillard, R. A., & Lee, J. C. 2011, ApJ, 737, 69
Nicastro, F., Fiore, F., & Matt, G. 1999, ApJ, 517, 108
Noble, M. S., Houck, J. C., Davis, J. E., et al. 2006, in ASP Conf. Ser. 351, Astronomical Data Analysis Software and Systems XV, ed. C. Gabriel et al. (San Francisco, CA: ASP), 481
Noble, M. S., Ji, L., Young, A., et al. 2009, in ASP Conf. Ser. 411, Astronomical Data Analysis Software and Systems XVIII, ed. D. A. Bohlender, D. Durand, & P. Dowler (San Francisco, CA: ASP), 301
Tarter, C. B., Tucker, W. H., & Salpeter, E. E. 1969, ApJ, 156, 943
Tombesi, F., Cappi, M., Reeves, J. N., et al. 2011a, ApJ, 742, 44
Tombesi, F., Sambruna, R. M., Reeves, J. N., et al. 2011b, MNRAS, 418, L89
Tombesi, F., Tazaki, F., Mushotzky, R. F., et al. 2014, MNRAS, 443, 2154