Supplementary information for

Meteorological and Land-Cover Controls on Grassland Fire Behaviour by WRF v4.4-SFIRE v0.1 with the computational optimization

Mengjie Lou[1], Qizhong Wu[1,2,*], Yongli Wang[3], Baogang Zhang[1,4], Jinhua Tao[5], Pu Gan[1], Huaqiong Cheng[1], Jiating Zhang[1], Jiahuan He[6], Meng Fan[5]

[1] Institute of Earth System Science, Faculty of Geographical Science, Beijing Normal University, Beijing, 100875, China
[2] College of Global and Earth System Science, Beijing Normal University, Beijing 100875, China
[3] National Institute of Natural Hazards, Ministry of Emergency Management of China, Beijing, 100085, China
[4] State Key Laboratory of Remote Sensing and Digital Earth, Beijing Normal University, Beijing, 100875, China
[5] The Aerospace Information Research Institute, Chinese Academy of Sciences, Beijing, 100094, China
[6] Hulun Buir Meteorological Bureau, Hulun Buir, 021008, China

Correspondence: Qizhong Wu (wqizhong@bnu.edu.cn)

Contents of this file:

Anderson 13 Fuel Characteristics and Fuel Mapping Rules

China currently lacks a systematic and regionally adapted fuel classification database, which prevents the direct provision of complete fuel parameters, such as fuel load, fuel moisture, and fuel depth, for fire behavior modeling. Therefore, land-cover types were used to infer the corresponding Anderson 13 fuel categories. As shown in Table S1, this fuel classification system includes several key parameters controlling fire spread, including 1 h, 10 h, and 100 h timelag dead fuel loads, live fuel load, fuel depth, and dead fuel moisture of extinction (Hu et al., 2024). Together, these parameters characterize the heat release potential and combustion efficiency of different vegetation types.

Table S1 Characteristics of the Anderson 13 fuel categories.

Fuel model	Typical fuel complex	Fuel loading (tonnes/hectare)				Fuel bed depth (m)	Moisture of extinction dead fuels(%)
		1h	10 h	100 h	Live		
1	Short grass	1.658	0	0	0	0.305	12
2	Timber (grass and understory)	4.480	2.240	1.120	1.120	0.305	15
3	Tall grass	6.742	0	0	0	0.762	25
4	Chaparral	11.222	8.982	4.480	11.222	1.829	20
5	Brush	2.240	1.120	0	4.480	0.610	20
6	Dormant brush, hardwood slash	3.360	5.600	4.480	0	0.762	25
7	Southern rough	2.531	4.189	3.360	0.829	0.762	40
8	Closed timber litter	3.360	2.240	5.600	0	0.061	30
9	Hardwood litter	6.541	0.918	0.336	0	0.061	25
10	Timber (litter and understory)	6.742	4.480	11.222	4.480	0.305	25
11	Light logging slash	3.360	10.102	12.342	0	0.305	15
12	Medium logging slash	8.982	31.427	35.907	0	0.701	20
13	Heavy logging slash	15.702	51.610	62.832	0	0.915	25

Accordingly, three fuel-mapping schemes were developed based on the selected high-resolution land-cover datasets by linking the vegetation descriptions in each dataset with the Anderson 13 fuel models in terms of vegetation structure, fuel continuity, and fuel characteristics. Through this procedure, land-cover classes were converted into fuel categories recognizable by the model. The main burnable land-cover types were mapped to the Anderson 13 fuel models, whereas non-burnable classes were assigned separately to Fuel Model 14 as a model-recognized non-burnable category

rather than as part of the original Anderson 13 set. The main mapping relationships were as follows. In FROM-GLC (2017), cropland and grassland were assigned to Fuel Model 1 (Short grass), shrubland to Fuel Model 4 (Chaparral), and forest to Fuel Model 8 (Closed timber litter). In GlobeLand30 (2020), grassland was assigned to Fuel Model 1 (Short grass), cropland to Fuel Model 3 (Tall grass), and forest to Fuel Model 8 (Closed timber litter). In GLC_FCS30D (2022), herbaceous vegetation, grassland, and sparse herbaceous vegetation were assigned to Fuel Model 1 (Short grass); orchards, sparse vegetation, and sparse shrubland to Fuel Model 2 [Timber (grass and understory)]; rainfed cropland and irrigated cropland to Fuel Model 3 (Tall grass); evergreen shrubland and shrubland to Fuel Model 5 (Brush); deciduous shrubland to Fuel Model 6 (Dormant brush, hardwood slash); closed evergreen broadleaf forest, closed deciduous broadleaf forest, and closed mixed forest to Fuel Model 8 (Closed timber litter); deciduous broadleaf forest and lichen/moss vegetation to Fuel Model 9 (Hardwood litter); and open evergreen forest, open deciduous forest, and open mixed forest to Fuel Model 10 [Timber (litter and understory)]. In addition, all non-burnable classes were uniformly assigned to Fuel Model 14 (Non-burnable) to avoid unrealistic burning in the simulations.

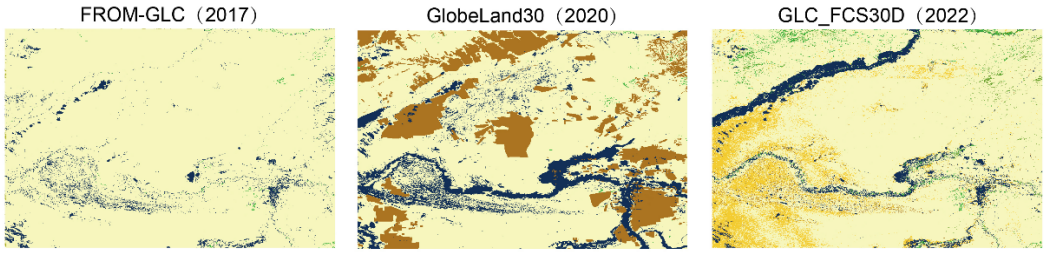

Figure S1 Mapped fuel-type raster datasets.

The resulting fuel-type raster datasets are shown in Figure S1. These datasets were subsequently converted into the Geogrid format readable by the model, thereby replacing the default land-cover information in the WRF static geographical dataset. By constructing three independent fuel-mapping schemes, this study was able to evaluate the effects of different land-cover products on fuel spatial distribution and fire behavior simulations, thereby providing a basis for fuel parameter localization and model optimization.

Compilation configuration parameters of WRF-SFIRE for different Intel compiler versions

In numerical simulations, compiler selection and configuration are critical to computational efficiency, as different compilers differ in optimization strategies, automatic vectorization, memory layout, and support for parallel instruction sets. As a computationally intensive model, WRF-SFIRE is particularly sensitive to compiler optimization, especially in large-scale floating-point operations, data dependencies, and memory access. In this study, numerical simulations were conducted on an Intel Xeon processor platform. Owing to their deep optimization for Intel processor architectures, integration of high-performance mathematical libraries, and strong support for hybrid parallelization, Intel compilers have been widely adopted and officially recommended for the WRF model family. Therefore, under the CentOS 7 operating system, and considering its compatibility with different compiler versions, three Intel compiler versions listed in Table S2 were selected and systematically evaluated in terms of their parallel performance. By independently configuring the corresponding environment variables for each version, both the WRF-SFIRE source code and its key dependent libraries were consistently built with the same compiler version, thereby avoiding performance interference or runtime errors caused by inconsistencies in the compilation environment.

Table S2 Compilation configuration parameters of WRF-SFIRE with different Intel compiler versions.

Version	Parameter	Flags
v18	CFLAGS_LOCAL	-O3 -xCORE-AVX512 -fp-model fast=2 -qopenmp
	FCOPTIM	-O3 -xCORE-AVX512 -fp-model fast=2 -heap-arrays 8192
v21	CFLAGS_LOCAL	-O3 -xHost -fp-model fast=2 -qopenmp -qopt-zmm-usage=high
	FCOPTIM	-O3 -xHost -fp-model fast=2 -heap-arrays 8192
v22	CFLAGS_LOCAL	-O3 -xHost -fp-model precise -qopenmp -qopenmp-simd
	FCOPTIM	-O3 -xHost -fp-model precise -heap-arrays 8192 -qopenmp-simd

As shown in the compilation settings, the optimization strategies were continuously adjusted with compiler version updates. The v18 version explicitly specified -xCORE-AVX512 to optimize for the AVX-512 instruction set and adopted -fp-model fast=2 to enable aggressive floating-point optimization. In contrast, the v21 version replaced this option with -xHost, allowing the compiler to automatically detect and optimize for the highest instruction set supported by the current platform, and additionally introduced -qopt-zmm-usage=high to encourage more extensive use of 512-bit ZMM registers. Notably, the v22 version changed the floating-point model from fast=2 to precise. While still pursuing computational performance, this adjustment strengthened control

over numerical consistency and reproducibility. In addition, -qopenmp-simd was introduced to support OpenMP SIMD vectorization. These changes reflect the trade-off between maximizing performance and maintaining numerical robustness and precision.

Theory and Technical Implementation of MPI and Hybrid Parallel Computing

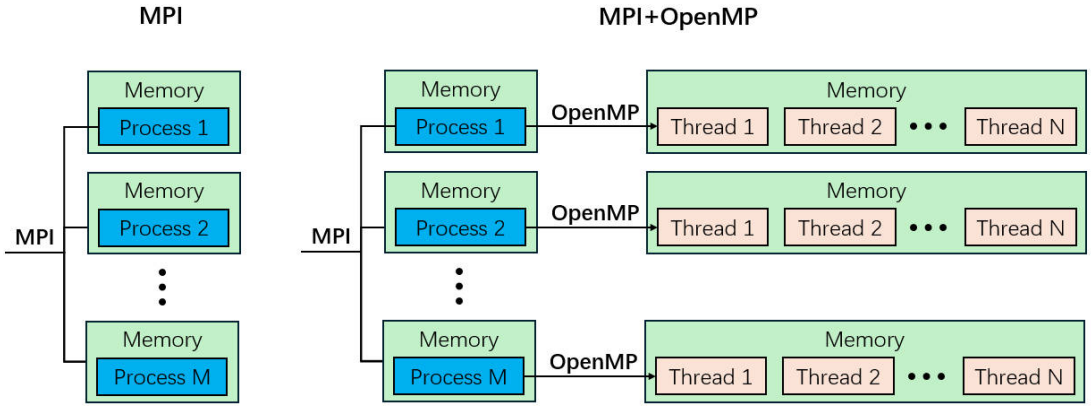

Figure S2 Schematic framework of MPI and MPI+OpenMP hybrid parallelization.

As shown in Figure S2, in the conventional pure-MPI parallel mode, each process occupies an independent memory space. As the number of nodes increases, the communication overhead among processes grows substantially, while memory resources remain relatively isolated, which is not conducive to fine-grained collaboration among multiple cores within a node. To address this limitation, the MPI+OpenMP hybrid parallel mode was introduced in this study. By enabling multithreading within each MPI process, this mode achieves fine-grained parallelism in shared-memory space, helping to reduce inter-node communication overhead, improve cache utilization, and further alleviate performance degradation caused by contention in intra-node memory access.

From the implementation perspective, the hybrid parallel mode requires systematic configuration at both the compilation and runtime stages. First, during compilation, the (dm+sm) option together with INTEL (ifort/icc) and Intel MPI was selected to support both distributed-memory and shared-memory parallelism. In addition, -fopenmp was added to the Fortran compiler options and -qopenmp to the C/C++ compiler options in the configure.wrf file. During job submission, #SBATCH --ntasks-per-node was used in the Slurm script to control the number of MPI processes per node, and #SBATCH --cpus-per-task was specified to define the number of CPU cores allocated to each MPI process. The environment variable export OMP_NUM_THREADS was then set consistently with this configuration to ensure proper process-thread resource binding.

Theory of Grid Partitioning

As illustrated by the theoretical schematic of grid decomposition in Figure S3, the mechanisms by which two extreme decomposition patterns affect model performance, as well as the corresponding optimization principles, are clearly demonstrated. During parallel execution of the two-way coupled WRF-SFIRE model, the entire simulation domain is dynamically partitioned into multiple subdomains (patches/tiles), each of which is assigned to a single processor core for computation. To enable data synchronization and physical coupling among adjacent subdomains, the system automatically adds a halo region around each subdomain to cache boundary information from neighboring subdomains. Although this boundary mechanism ensures physical consistency in the calculations, it also introduces additional communication overhead and memory consumption.

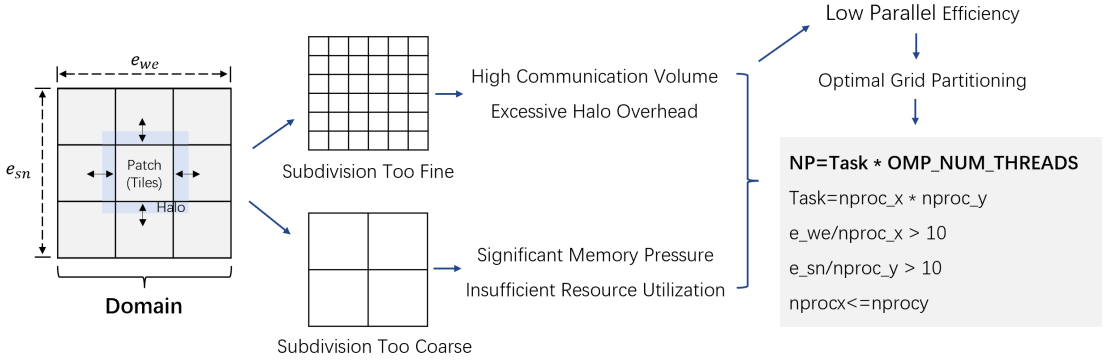

Figure S3 Schematic illustration of grid decomposition.

When the grid is decomposed too finely, that is, when the number of subdomains is excessively large and thus too many cores are used, as shown in the upper part of Figure S3, the actual computational grid size of each subdomain decreases substantially, while the relative proportion of boundary regions increases markedly. This leads to two major problems: (1) a sharp increase in communication overhead, as processes must frequently exchange a large proportion of halo data; and (2) a reduction in the proportion of effective computational area, because substantial computational resources are consumed by boundary data synchronization rather than core physical calculations. Together, these effects ultimately result in a significant decline in parallel efficiency. In contrast, when the grid is decomposed too coarsely, that is, when the subdomains are excessively large and too few cores are used, as shown in the lower part of Figure S3, communication overhead is reduced, but the grid size that each process must compute and store becomes too large. This can substantially increase memory pressure within a node and may even interrupt the computation because of insufficient memory.

Meanwhile, under a fixed total number of available cores, too few subdomains imply that a considerable amount of computational resources remains idle, resulting in insufficient overall resource utilization and preventing full exploitation of the advantages of large-scale parallel computing. Achieving optimal parallel performance therefore requires finding a balance between excessively fine and excessively coarse decomposition, that is, an appropriate grid decomposition.

Theoretical Computational Processes of Different I/O Strategies

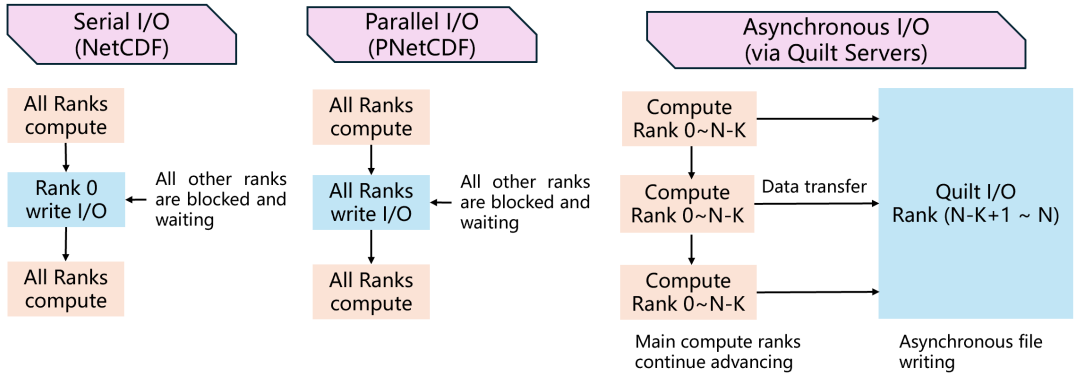

Figure S4 Schematic illustration of the computational workflows of different I/O strategies.

As shown in Figure S4, serial NetCDF I/O adopts a centralized output scheme, in which all compute processes must first gather their data to the master process (Rank 0) for unified writing. As a result, all other processes remain idle during the output phase, making I/O a major performance bottleneck. Parallel I/O, such as PNetCDF and Parallel NetCDF4, improves output efficiency through collaborative writing by multiple processes; however, the writing phase still usually requires synchronous participation of all processes, making it difficult to overlap computation with I/O. In contrast, asynchronous I/O separates output operations from the main compute processes by introducing dedicated I/O server groups (Quilt servers), allowing the compute processes to continue advancing the simulation while file writing is performed in the background. This effectively reduces waiting time and improves overall runtime efficiency. This mode requires the parameters nio_tasks_per_group and nio_groups to be specified in namelist.input, with the total number of I/O processes given by the product of the two. Under fixed total computational resources, this configuration essentially reflects a trade-off between computational parallelism and I/O throughput.

Reference

Hu, H., Deng, X., Zhang, G., Feng, L., Long, J., Li, Z., Zhu, Y., Wang, Y.: Fire behavior simulation of xintian forest fire in 2022 using WRF-fire model. Front. For. Glob. Change 7, https://doi.org/10.3389/ffgc.2024.1336716, 2024.