MPMD & MPI Runtime Infrastructure in NOAA Global Workflow

Comprehensive Technical Specification

Version 1.0

MCP/RAG Analysis System
NOAA Environmental Modeling Center
National Centers for Environmental Prediction

global-workflow@noaa.gov

January 30, 2026

Abstract

The Global Workflow, maintained by NOAA's Environmental Modeling Center (EMC), implements production weather forecasting systems including the Global Forecast System (GFS), Global Ensemble Forecast System (GEFS), Seasonal Forecast System (SFS), and Global Composition/Chemistry Aerosol Forecast System (GCAFS). Central to this infrastructure is a sophisticated **Multi-Program Multiple-Data (MPMD)** execution framework that enables heterogeneous parallel task execution across diverse high-performance computing (HPC) platforms.

This technical specification provides a comprehensive analysis of the MPMD architecture, MPI runtime configurations, and platform-specific optimizations that enable the Global Workflow to execute efficiently on eleven distinct HPC systems—from NOAA's operational WCOSS2 production environment to research platforms including Hera, Hercules, Orion, and Gaea, as well as cloud-based ParallelWorks deployments on AWS, Azure, and Google Cloud.

The document covers the core `run_mpmd.sh` orchestration script, the three-level job resource configuration chain, platform-specific MPI tuning parameters, network fabric optimizations, and the critical **Coupled Framework Parallelism (CFP)** module used exclusively on WCOSS2. We present the complete environment variable taxonomy, command file format transformations, and troubleshooting procedures essential for operational reliability.

Keywords: MPI, MPMD, HPC, Global Workflow, NOAA, GFS, GEFS, WCOSS2, Slurm, PBS, Intel MPI, Cray MPICH, CFP, Slingshot, InfiniBand

Contents

List of Figures

List of Tables

1 Introduction

The National Oceanic and Atmospheric Administration (NOAA) operates some of the world's most sophisticated numerical weather prediction systems. At the core of these systems lies the **Global Workflow**—an integrated software framework that orchestrates the execution of complex forecast cycles across multiple high-performance computing (HPC) platforms.

1.1 Problem Statement

Modern weather forecasting requires executing thousands of heterogeneous tasks with varying computational characteristics:

- **Embarrassingly parallel tasks**: Observation processing, post-processing, product generation
- **Tightly coupled tasks**: Model forecast integration requiring MPI collective operations
- **I/O intensive tasks**: Data staging, archive operations, file transformations

The challenge is executing these diverse workloads efficiently across platforms with fundamentally different:

- Job schedulers (Slurm vs. PBS Pro)
- MPI implementations (Intel MPI, Cray MPICH, PMI2)
- Network fabrics (InfiniBand, Slingshot, EFA)
- File systems (Lustre, GPFS)

1.2 Solution Architecture

The Global Workflow implements a **platform abstraction layer** that provides:

1. **Unified MPMD Interface**: The `run_mpmd.sh` script accepts a generic command file and transforms it to platform-specific format
2. **Three-Level Configuration**: Base resources, platform overrides, and environment binding
3. **Launcher Abstraction**: Platform-specific `$launcher` and `$mpmd_opt` variables
4. **Module Environment**: Lua modulefiles for runtime library loading

1.3 Document Scope

This specification covers:

- Complete MPMD execution flow from job script to MPI ranks
- All eleven supported HPC platforms and their configurations
- MPI tuning parameters for Intel MPI and Cray MPICH
- Network fabric optimizations (InfiniBand, Slingshot, EFA)
- CFP (Coupled Framework Parallelism) on WCOSS2
- Troubleshooting procedures and diagnostic techniques

2 Architecture Overview

2.1 System Component Hierarchy

Figure 1 illustrates the complete MPMD execution architecture.

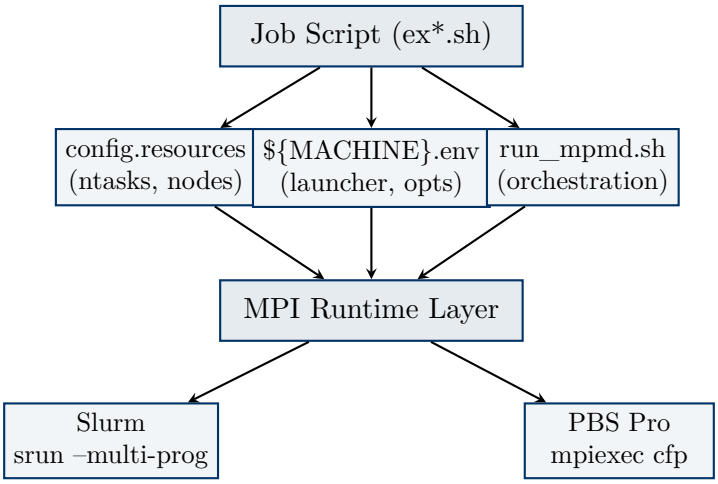

Figure 1: Global Workflow MPMD Execution Architecture

2.2 Directory Structure

The MPMD infrastructure is distributed across several key directories:

```
global-workflow/
|-- ush/
|   `-- run_mpmd.sh              # Core MPMD orchestration
|-- env/
|   |-- HERA.env                 # Hera configuration
|   |-- WCOSS2.env               # WCOSS2 configuration
|   |-- GAEAC6.env               # Gaea C6 configuration
|   `-- ...                      # 11 total platforms
|-- dev/
|   |-- parm/config/gfs/
|   |   |-- config.resources     # Base definitions
|   |   `-- config.resources.*   # Platform overrides
|   `-- workflow/hosts/
|       |-- hera.yaml            # Host configuration
|       `-- wcoss2.yaml
`-- modulefiles/
    |-- gw_run.hera.lua          # Hera modules
    `-- gw_run.wcoss2.lua        # WCOSS2 modules
```

Listing 1: MPMD Infrastructure Directory Layout

3 Core MPMD Script Analysis

3.1 Script Overview

The `ush/run_mpmd.sh` script, authored by Rahul Mahajan (NCEP/EMC), serves as the central orchestrator for MPMD execution. It provides a unified interface that adapts to the underlying MPI implementation.

3.2 Algorithm Description

Algorithm 1 presents the complete MPMD execution logic.

Algorithm 1 MPMD Execution Flow

Require: Command file path `cmdfile`
Require: Environment variables: `USE_CFP`, `launcher`, `mpmd_opt`, `DATA`

`cmdfile` ← argument 1
if `USE_CFP` ≠ "YES" **then**
  Execute `cmdfile` serially via `bash`
  return exit status
end if
`OMP_NUM_THREADS` ← 1 {Force single-threaded for MPMD}
`nprocs` ← line count of `cmdfile`
`mpmd_cmdfile` ← `DATA`/cmdfile.`RANDOM`
if `launcher` matches `"srun.*"` **then**
  {Slurm multi-prog format}
  for each line in `cmdfile` **do**
    Write "`rank command`" to `mpmd_cmdfile`
  end for
  Execute: `launcher mpmd_opt` -n `nprocs mpmd_cmdfile`
else if `launcher` matches `"mpiexec.*"` **then**
  {PBS cfp format}
  Write shebang to `mpmd_cmdfile`
  for each line in `cmdfile` **do**
    Write "`command` > mpmd.`rank`.out 2> &1" to `mpmd_cmdfile`
  end for
  `chmod 755 mpmd_cmdfile`
  Execute: `launcher` -np `nprocs mpmd_opt mpmd_cmdfile`
else
  Execute `cmdfile` serially via `bash`
end if
return exit status

3.3 Key Implementation Details

3.3.1 Thread Control

MPMD execution forces single-threaded mode:

```
export OMP_NUM_THREADS=1
```

Listing 2: Thread Control in MPMD

This ensures each MPI rank executes its assigned serial command without OpenMP parallelism, which would oversubscribe compute resources.

3.3.2 Launcher Detection

Pattern matching determines the execution strategy:

```
if [[ "${launcher:-}" =~ ^srun.* ]]; then
    # Slurm multi-prog execution path
elif [[ "${launcher:-}" =~ ^mpiexec.* ]]; then
    # PBS/CFP execution path
else
```

```
    # Serial fallback
fi
```

Listing 3: Launcher Pattern Detection

4 HPC Platform Configurations

4.1 Platform Classification

Table 1 presents the complete platform matrix with scheduler, MPI implementation, and network fabric details.

Table 1: Global Workflow Supported HPC Platforms

Platform	Scheduler	MPI Library	Network	CFP
WCOSS2	PBS Pro	Cray MPICH	Slingshot	✓
Hera	Slurm	Intel MPI	InfiniBand HDR	–
Hercules	Slurm	Intel MPI	InfiniBand	–
Orion	Slurm	Intel MPI	InfiniBand	–
Ursa	Slurm	Intel MPI	InfiniBand	–
Gaea C5	Slurm	Cray MPICH	Slingshot	–
Gaea C6	Slurm	Cray MPICH	Slingshot 11	–
AWS PW	Slurm	PMI2	EFA	–
Azure PW	Slurm	PMI2	InfiniBand	–
Google PW	Slurm	PMI2	VPC Fabric	–
Container	–	–	–	–

4.2 Tier 1: Production Systems

4.2.1 WCOSS2 Configuration

WCOSS2 (Weather and Climate Operational Supercomputer System, Phase 2) represents NOAA's operational production environment.

```
# MPI Launcher
export launcher="mpiexec -l"
export mpmd_opt="--cpu-bind verbose,core cfp"

# Thread Configuration
export OMP_PLACES=cores
export OMP_STACKSIZE=1G

# libfabric Tuning
export FI_OFI_RXM_SAR_LIMIT=3145728
export FI_OFI_RXM_RX_SIZE=40000
export FI_OFI_RXM_TX_SIZE=40000

# Collective I/O
export MPICH_MPIIO_HINTS="*:romio_cb_write=enable"
```

Listing 4: WCOSS2 Launcher Configuration (env/WCOSS2.env)

The WCOSS2 environment uniquely requires the **CFP module**:

Listing 5: WCOSS2 Runtime Modules (modulefiles/gw_run.wcoss2.lua)

```
load("PrgEnv-intel")
load("craype")
load("intel")
load("cray-mpich")
load("cray-pals")
load("cfp")                -- Coupled Framework Parallelism
setenv("USE_CFP", "YES")   -- Enable CFP by default
```

4.3 Tier 2: Research & Development Systems

4.3.1 Hera Configuration

Hera serves as NOAA's primary R&D platform.

```
# MPI Launcher
export launcher="srun -l --export=ALL --hint=nomultithread"
export mpmd_opt="--multi-prog --output=mpmd.%j.%t.out"

# Intel MPI Tuning
export MPI_BUFS_PER_PROC=2048
export MPI_BUFS_PER_HOST=2048
export MPI_GROUP_MAX=256
export MPI_MEMMAP_OFF=1
export MP_STDOUTMODE="ORDERED"

# OpenMP Configuration
export KMP_AFFINITY=scatter
export OMP_STACKSIZE=2048000

# Lustre Optimization
export I_MPI_EXTRA_FILESYSTEM=1
export I_MPI_EXTRA_FILESYSTEM_LIST=lustre
```

Listing 6: Hera Launcher Configuration (env/HERA.env)

4.3.2 Gaea C6 Configuration

Gaea C6 represents the newest NOAA R&D platform with Slingshot 11 interconnect.

```
# MPI Launcher with Slingshot optimizations
export launcher="srun -l --export=ALL --distribution=block:block"
export mpmd_opt="--multi-prog --output=mpmd.%j.%t.out"

# Cray Slingshot CXI tuning
export FI_CXI_RX_MATCH_MODE=hybrid
# For large collective-heavy jobs:
# export FI_CXI_RX_MATCH_MODE=software
```

Listing 7: Gaea C6 Launcher Configuration (env/GAEAC6.env)

4.4 Tier 3: Cloud Platforms

4.4.1 AWS ParallelWorks Configuration

```
# MPI Launcher with PMI2
export launcher="srun -l --export=ALL --hint=nomultithread --mpi=pmi2"
export mpmd_opt="--multi-prog --output=mpmd.%j.%t.out"
```

Listing 8: AWS ParallelWorks Configuration (env/AWSPW.env)

The `-mpi=pmi2` flag enables PMI2 process management interface, required for MPI on cloud Slurm clusters.

5 MPI Tuning Parameters

5.1 Intel MPI Parameters

Table 2 documents tuning parameters used on Intel MPI platforms.

Table 2: Intel MPI Tuning Parameters

Variable	Value	Purpose
MPI_BUFS_PER_PROC	2048	Per-process eager message buffers
MPI_BUFS_PER_HOST	2048	Per-host shared buffers
MPI_GROUP_MAX	256	Maximum MPI groups
MPI_MEMMAP_OFF	1	Disable memory mapping for IB
I_MPI_ADJUST_ALLREDUCE	5	GSI-specific allreduce algorithm
I_MPI_EXTRA_FILESYSTEM	1	Enable parallel filesystem support
I_MPI_EXTRA_FILESYSTEM_LIST	lustre	Optimize for Lustre

5.2 Cray MPICH Parameters

Table 3 documents parameters for Cray MPICH systems.

Table 3: Cray MPICH Tuning Parameters

Variable	Value	Purpose
FI_OFI_RXM_SAR_LIMIT	3145728	Segment-and-reassemble limit (3MB)
FI_OFI_RXM_RX_SIZE	40000	Receive queue size
FI_OFI_RXM_TX_SIZE	40000	Transmit queue size
FI_CXI_RX_MATCH_MODE	hybrid	Slingshot CXI matching mode
MPICH_MPIIO_HINTS	romio_cb_write	Collective buffering for writes

5.3 OpenMP Stack Configuration

Stack size configuration varies by platform:

Table 4: OpenMP Stack Configuration by Platform

Platform	OMP_STACKSIZE	NTHSTACK
WCOSS2	1G	–
Hera/Hercules	2048000	1024000000
Gaea C5/C6	2048M	–
Forecast jobs	2048M	–

6 Job Resource Configuration

6.1 Three-Level Configuration Chain

The Global Workflow employs a three-level configuration hierarchy:

1. **Base Resources** (`config.resources`): Step-agnostic defaults
2. **Platform Overrides** (`config.resources.$MACHINE`): Machine-specific adjustments
3. **Environment Binding** (`$MACHINE.env`): Runtime MPI/thread settings

6.2 Resource Variables

Table 5 defines the standard resource variables.

Table 5: Job Resource Variables

Variable	Description
`ntasks`	Total MPI tasks requested
`tasks_per_node`	MPI tasks per compute node
`max_tasks_per_node`	Hardware maximum (platform-specific)
`threads_per_task`	OpenMP threads per MPI task
`memory`	Memory requirement (e.g., “96GB”)

6.3 APRUN Variable Mapping

Each job step has a dedicated APRUN variable:

Table 6: APRUN Variables by Job Step

Step	Variable	Configuration
fcst	APRUN_UFS	`$launcher -n $ufs_ntasks`
anal	APRUN_GSI	`$APRUN_default -cpus-per-task=$NTHREADS_GSI`
eupd	APRUN_ENKF	`$launcher -n $ntasks_enkf -cpus-per-task=$N...`
upp	APRUN_UPP	`$APRUN_default -cpus-per-task=$NTHREADS_UPP`
CFP jobs	APRUNCFP	`$launcher -n $ncmd $mpmd_opt`

7 Command File Transformations

7.1 Generic Input Format

Users provide a simple list of commands:

```
/path/to/command1 arg1 arg2
/path/to/command2 arg1 arg2
/path/to/command3 arg1 arg2
```

Listing 9: Generic MPMD Command File

7.2 Slurm Multi-Prog Transformation

For Slurm systems, `run_mpmd.sh` prepends MPI rank numbers:

```
0 /path/to/command1 arg1 arg2
1 /path/to/command2 arg1 arg2
2 /path/to/command3 arg1 arg2
```

Listing 10: Slurm Multi-Prog Format

Execution: `srun -multi-prog -n 3 cmdfile`

7.3 PBS CFP Transformation

For WCOSS2, commands become a shell script with output redirection:

```
#!/bin/bash
/path/to/command1 arg1 arg2 > mpmd.0.out 2>&1
/path/to/command2 arg1 arg2 > mpmd.1.out 2>&1
/path/to/command3 arg1 arg2 > mpmd.2.out 2>&1
```

Listing 11: PBS CFP Format

Execution: `mpiexec -np 3 -cpu-bind verbose,core cfp cmdfile`

8 Network Fabric Details

8.1 Fabric Comparison

Figure **??** illustrates the network fabric architecture across platforms.

Table 7: Network Fabric Specifications

Platform	Fabric	Bandwidth	Latency	Provider
WCOSS2	HPE Slingshot	200 Gb/s	<1 μs	CXI
Hera	IB HDR	200 Gb/s	~1 μs	Verbs
Hercules/Orion	IB FDR/EDR	100 Gb/s	~1 μs	Verbs
Gaea C6	Slingshot 11	200 Gb/s	<1 μs	CXI
AWS PW	EFA	100-400 Gb/s	~2 μs	EFA

8.2 InfiniBand Configuration

For InfiniBand systems (Hera, Hercules, Orion):

```
# Disable memory mapping (required for IB RDMA)
export MPI_MEMMAP_OFF=1

# Enable Lustre-aware collective I/O
export I_MPI_EXTRA_FILESYSTEM=1
export I_MPI_EXTRA_FILESYSTEM_LIST=lustre
```

Listing 12: InfiniBand MPI Configuration

8.3 Slingshot/CXI Configuration

For Cray Slingshot systems (WCOSS2, Gaea):

```
# Default: Hardware-assisted matching (fast, limited scalability)
export FI_CXI_RX_MATCH_MODE=hybrid

# Large jobs: Software matching (slower, unlimited scalability)
# export FI_CXI_RX_MATCH_MODE=software

# Buffer sizing
export FI_OFI_RXM_SAR_LIMIT=3145728
export FI_OFI_RXM_RX_SIZE=40000
export FI_OFI_RXM_TX_SIZE=40000
```

Listing 13: Slingshot CXI Configuration

9 Coupled Framework Parallelism (CFP)

9.1 CFP Overview

CFP (Coupled Framework Parallelism) is a WCOSS2-specific module that enables MPMD execution under PBS Pro. Unlike Slurm's native `-multi-prog` option, PBS Pro requires an external tool for heterogeneous task execution.

9.2 CFP Module Loading

Listing 14: CFP Module in WCOSS2 Runtime

```
-- From modulefiles/gw_run.wcoss2.lua
load("cfp")
setenv("USE_CFP", "YES")
```

9.3 CFP Execution Model

1. CFP reads the shell script command file
2. Each line spawns as a separate MPI task
3. Output redirection captures per-rank stdout/stderr
4. CFP synchronizes task completion before returning

9.4 CFP vs. Slurm Multi-Prog

Table 8: CFP vs. Slurm Multi-Prog Comparison

Feature	CFP (PBS)	–multi-prog (Slurm)
Format	Shell script	Rank-prefixed file
Output	Explicit redirect	Slurm %j.%t pattern
Module	cfp required	Built-in
Exit handling	Per-script	Per-rank

10 MPMD Use Cases

10.1 Observation Processing

GSI assimilation jobs use MPMD for parallel observation type processing:

```
# exglobal_atmos_analysis.sh
for obstype in conv amsua_n19 atms_npp iasi_metop-b; do
    echo "${USHgfs}/process_obs.sh ${obstype}" >> cmdfile
done

"${USHgfs}/run_mpmd.sh" "${DATA}/cmdfile"
```

Listing 15: GSI Observation Processing MPMD

10.2 Wave Grid Initialization

Wave model initialization processes multiple grids in parallel:

```
# exgfs_wave_init.sh
for grdID in ${wavegrids}; do
    echo "${USHgfs}/wave_grid_moddef.sh ${grdID}" >> mpmd_script
done

"${USHgfs}/run_mpmd.sh" "${DATA}/mpmd_script"
```

Listing 16: Wave Grid MPMD

10.3 Product Generation

Atmospheric products use MPMD for parallel GRIB2 generation:

```
# exglobal_atmos_products.sh
export USE_CFP="YES"

# Split work into processor-specific chunks
split -n "l/${nprocs}" -d "${tmpfile}" "${DATA}/cmdfile_"

"${USHgfs}/run_mpmd.sh" "${DATA}/cmdfile"
```

Listing 17: Product Generation MPMD

11 Troubleshooting Guide

11.1 Common Issues

11.1.1 Issue: Tasks Execute Serially

Symptom: MPMD job runs but tasks execute sequentially.

Diagnosis:

```
echo "USE_CFP=${USE_CFP}"
echo "launcher=${launcher}"
```

Listing 18: Serial Execution Diagnosis

Resolution: Ensure `USE_CFP=YES` and `launcher` contains valid MPI launcher command.

11.1.2 Issue: Task Failures with No Output

Symptom: MPMD job fails but no error messages visible.

Diagnosis:

```
ls -la mpmd.*.out
cat mpmd.0.out
```

Listing 19: Output File Inspection

Resolution: Per-rank output files contain individual task stderr/stdout.

11.1.3 Issue: CFP Not Found

Symptom: `cfp: command not found` on WCOSS2.

Resolution:

```
module load cfp
```

Listing 20: CFP Module Loading

11.2 Debug Mode

Enable comprehensive debugging:

```
export MPMD_DEBUG=1
set -x
"${USHgfs}/run_mpmd.sh" "${DATA}/cmdfile"
```

Listing 21: MPMD Debug Configuration

12 Performance Considerations

12.1 Task Granularity

MPMD efficiency depends on task homogeneity:

- **Optimal:** Tasks with similar execution times
- **Suboptimal:** High variance in task duration (stragglers)

12.2 Network Bandwidth

For I/O-intensive MPMD tasks:

1. Use file-per-rank patterns to avoid contention
2. Enable collective buffering for shared output
3. Consider staged I/O with local SSD scratch

12.3 Memory Scaling

Per-task memory calculation:

$$\text{Memory per task} = \frac{\text{Node memory}}{\text{tasks\_per\_node}} \tag{1}$$

13 Conclusion

The Global Workflow's MPMD infrastructure represents a mature, production-hardened solution for heterogeneous parallel execution across diverse HPC environments. The platform abstraction provided by `run_mpmd.sh`, combined with the three-level configuration chain and comprehensive MPI tuning, enables identical workflow code to execute efficiently on eleven distinct HPC systems.

Key architectural achievements:

1. **Portability**: Single codebase supports Slurm, PBS Pro, and cloud platforms
2. **Efficiency**: Platform-specific MPI tuning maximizes hardware utilization
3. **Maintainability**: Centralized configuration reduces duplication
4. **Debuggability**: Per-rank output capture enables rapid failure diagnosis

Future enhancements may include:

- Dynamic task scheduling based on runtime estimates
- Integration with workflow managers for automatic retry
- GPU-aware MPMD for accelerated post-processing

References

[1] SchedMD LLC. Slurm Workload Manager Documentation. https://slurm.schedmd.com/documentation.html, 2025.

[2] Hewlett Packard Enterprise. Cray MPICH Programming Guide. HPE Cray Programming Environment Documentation, 2025.

[3] Intel Corporation. Intel MPI Library Developer Reference. https://www.intel.com/content/www/us/en/docs/mpi-library/, 2025.

[4] NOAA RDHPCS. NOAA Research and Development High Performance Computing Systems User Guide. https://rdhpcs-common-docs.rdhpcs.noaa.gov/, 2026.

[5] NOAA-EMC. Global Workflow Repository. https://github.com/NOAA-EMC/global-workflow, 2026.

A Complete Environment File Listing

A.1 HERA.env

```
export launcher="srun -l --export=ALL --hint=nomultithread"
export mpmd_opt="--multi-prog --output=mpmd.%j.%t.out"
export OMP_STACKSIZE=2048000
export NTHSTACK=1024000000
export MPI_BUFS_PER_PROC=2048
export MPI_BUFS_PER_HOST=2048
export MPI_GROUP_MAX=256
export MPI_MEMMAP_OFF=1
export MP_STDOUTMODE="ORDERED"
export KMP_AFFINITY=scatter
export I_MPI_EXTRA_FILESYSTEM=1
export I_MPI_EXTRA_FILESYSTEM_LIST=lustre
```

A.2 WCOSS2.env

```
export launcher="mpiexec -l"
export mpmd_opt="--cpu-bind verbose,core cfp"
export OMP_PLACES=cores
export OMP_STACKSIZE=1G
export FI_OFI_RXM_SAR_LIMIT=3145728
export FI_OFI_RXM_RX_SIZE=40000
export FI_OFI_RXM_TX_SIZE=40000
export MPICH_MPIIO_HINTS="*:romio_cb_write=enable"
```

A.3 GAEAC6.env

```
export launcher="srun -l --export=ALL --distribution=block:block"
export mpmd_opt="--multi-prog --output=mpmd.%j.%t.out"
export FI_CXI_RX_MATCH_MODE=hybrid
```

B Host YAML Configuration Schema

```
# Platform: example.yaml
SCHEDULER: slurm|pbspro
QUEUE: batch
QUEUE_SERVICE: service
PARTITION_BATCH: partition_name
SUPPORTED_RESOLUTIONS:
  - C1152
  - C768
  - C384
  - C192
  - C96
  - C48
HPSS_HOST: hpss.example.com
```

```
14 HPSS_PROJECT: project_id
```

Listing 22: Host YAML Template