CPU Multiprocess Solver v2

Overview

Version 2 of `quad8_cpu_multiprocess.py` fixes a critical issue that caused the solver to fail on Windows when processing large meshes (500K+ nodes).

The Problem

When running benchmarks with large meshes (e.g., Y-Shaped with 772,069 nodes), the original implementation failed with:

```
OSError: [WinError 1450] Insufficient system resources exist to complete the requ
```

Root Cause

The original code passed the entire mesh arrays with every batch sent to worker processes:

```
# Original (v1) - each batch included full mesh data
batches.append((start, end, self.x, self.y, self.quad8, xp, wp))
```

For a 772K node mesh:

- `self.x` and `self.y` : ~6 MB each
- `self.quad8` : ~25 MB
- **Total per batch: ~37 MB**

With `batch_size=1000` and ~250K elements, this created ~250 batches, each attempting to serialize ~37 MB through Windows IPC pipes. Windows `multiprocessing` uses `spawn` (not `fork`), requiring all data to be pickled and sent through pipes, which exhausted system resources.

The Solution

Version 2 uses the **Pool initializer pattern** to broadcast large arrays to workers once at pool creation:

```
# v2 - data sent once via initializer, batches only contain indices
_WORKER_STATE = {}

def _init_assembly_worker(x, y, quad8, xp, wp):
    _WORKER_STATE['x'] = x
    _WORKER_STATE['y'] = y
    _WORKER_STATE['quad8'] = quad8
    _WORKER_STATE['xp'] = xp
    _WORKER_STATE['wp'] = wp

# Pool creation with initializer
with Pool(
    processes=self.num_workers,
    initializer=_init_assembly_worker,
    initargs=(self.x, self.y, self.quad8, xp, wp)
) as pool:
    batches = [(start, end) for start in range(0, self.Nels, self.batch_size)]
    results = pool.map(_process_assembly_batch, batches)
```

Data Transfer Comparison

Version	Data per batch	Total for 250 batches
v1	~37 MB	~9.25 GB through IPC
v2	~16 bytes (two integers)	~4 KB through IPC

Changes Summary

1. Added `_WORKER_STATE` module-level dict for worker process state
2. Added `_init_assembly_worker()` and `_init_velocity_worker()` initializer functions
3. Modified `assemble_system()` to use Pool with `initializer=` argument
4. Modified `compute_derived_fields()` to use Pool with `initializer=` argument
5. Batch tuples now contain only `(start_idx, end_idx)` instead of full arrays
6. Legacy batch functions preserved for backward compatibility

API Compatibility

The public API is **100% unchanged**:

- Same class name: `Quad8FEMSolverMultiprocess`
- Same constructor parameters
- Same `run()` method signature
- Same return dictionary structure
- Same progress callback interface
- Same timing metrics

Usage

Update the import in `solver_wrapper.py` :

```
# Before
from quad8_cpu_multiprocess import Quad8FEMSolverMultiprocess as MultiprocessSolv

# After
from quad8_cpu_multiprocess_v2 import Quad8FEMSolverMultiprocess as MultiprocessS
```

Testing

Verified working on:

- Windows 10/11 with Python 3.10
- Mesh sizes up to 772K+ nodes
- 28-core CPU with full worker utilization