```
import time
from os import cpu_count
from typing import Any, Callable, List, Optional

from loguru import logger
from pathos.multiprocessing import ProcessingPool as Pool


from typing import Tuple


def execute_parallel_optimized(
    callables_with_args: List[
        Tuple[Callable[..., Any], Tuple[Any, ...]]
    ],
    max_workers: Optional[int] = None,
    chunk_size: Optional[int] = None,
    retries: int = 3,
    **kwargs,
) -> List[Any]:
    """
    Executes a list of callables in parallel, leveraging all available CPU cores.

    This function is optimized for high performance and reliability.

    Args:
        callables_with_args (List[Tuple[Callable[..., Any], Tuple[Any, ...]]]):
            A list of tuples, where each tuple contains a callable and a tuple of its arguments.
        max_workers (Optional[int]): The maximum number of workers to use. Defaults to the number of available cores.
        chunk_size (Optional[int]): The size of chunks to split the tasks into for balanced execution. Defaults to automatic chunking.
        retries (int): Number of retries for a failed task. Default is 3.

    Returns:
        List[Any]: A list of results from each callable. The order corresponds to the order of the input list.

    Raises:
        Exception: Any exception raised by the callable will be logged and re-raised after retries are exhausted.
    """
    max_workers = cpu_count() if max_workers is None else max_workers
    results = []
    logger.info(
        f"Starting optimized parallel execution of {len(callables_with_args)} tasks."
    )

    pool = Pool(
        nodes=max_workers, **kwargs
    )  # Initialize the pool once
```

```
def _execute_with_retry(callable_, args, retries):
    attempt = 0
    while attempt < retries:
        try:
            result = callable_(*args)
            logger.info(
                f"Task {callable_} with args {args} completed successfully."
            )
            return result
        except Exception as e:
            attempt += 1
            logger.warning(
                f"Task {callable_} with args {args} failed on attempt {attempt}: {e}"
            )
            time.sleep(1)  # Small delay before retrying
            if attempt >= retries:
                logger.error(
                    f"Task {callable_} with args {args} failed after {retries} retries."
                )
                raise


try:
    if chunk_size is None:
        chunk_size = (
            len(callables_with_args)
            // (max_workers or pool.ncpus)
```

```
                or 1
            )

        # Use chunking and mapping for efficient execution
        results = pool.map(
            lambda item: _execute_with_retry(
                item[0], item[1], retries
            ),
            callables_with_args,
            chunksize=chunk_size,
        )

        pool.close()
        pool.join()

        return results

    except Exception as e:
        logger.critical(
            f"Parallel execution failed due to an error: {e}"
        )
        raise



#    return results
# def add(a, b):
#     return a + b


# def multiply(a, b):
#     return a * b


# def power(a, b):
#     return a**b


# if __name__ == "__main__":
#     # List of callables with their respective arguments
#     callables_with_args = [
#         (add, (2, 3)),
#         (multiply, (5, 4)),
#         (power, (2, 10)),
#     ]

#     # Execute the callables in parallel
#     results = execute_parallel_optimized(callables_with_args)

#     # Print the results
#     print("Results:", results)
```