Improving an MPI application-level migration approach through checkpoint file splitting

M. Rodríguez, I. Cores, M.J. Martín and P. González

Universidade da Coruña
Computer Architecture Group
ivan.coresg@udc.es
Introduction

Migration is a necessity in many scientific computing environments as it enables:

- Load balancing
- Data locality
- Fault tolerance
Introduction

Migration is a necessity in many scientific computing environments as it enables:

- Load balancing
- Data locality
- Fault tolerance

The migration of parallel MPI processes is a great challenge:

- Communication objects need to be dynamically reconfigured.
Introduction

Migration is a necessity in many scientific computing environments as it enables:

- Load balancing
- Data locality
- Fault tolerance

The migration of parallel MPI processes is a great challenge:

- Communication objects need to be dynamically reconfigured.

Efficient and scalable mechanism for process migration are a requirement in HPC environments.
# Introduction

Migration phases of MPI processes:

1. Write process state to memory (disk is also allowed).
2. Send process data using the network.
3. Reconfiguration of the relevant communication objects.
4. Reading and recovery.

![Diagram of migration phases](image)

Problem: The I/O operations are the prevalent cause of overhead during the migration.
Introduction

Migration phases of MPI processes:
1. Write process state to memory (disk is also allowed).
2. Send process data using the network.
3. Reconfiguration of the relevant communication objects.
4. Reading and recovery.

Problem: The I/O operations are the prevalent cause of overhead during the migration.
Introduction

Aim of this work:

Reducing the overhead by splitting checkpoint files into multiple smaller files to overlap the different phases of the migration operation.
CPPC is an application-level open-source checkpointing tool focused on the insertion of fault tolerance into long-running message-passing applications.

CPPC appears to the final user as a compiler tool and a runtime library.

http://cppc.des.udc.es
CPPC
CPPC Overview

CPPC Compiler: inserts fault tolerance and flow control code

Parallel Application

Fault-Tolerant
Parallel Application

COMPILE TIME
RUNTIME

CPPC Interface

Facade

Checkpointing Layer

Writing Layer

CPPC Controller

Stable storage

Network Hardware

State file

If Ckpt

If Migra

Registers

Checkpoint data
Main CPPC features:

- CPPC reduces the amount of data to be saved working at the variable level.
- CPPC applies a size reduction technique called zero-blocks exclusion.
- CPPC stores data using portable file formats. Restart is possible on different architectures.

To achieve portability, CPPC uses the Hierarchical Data Format 5 (HDF5) library.
Memory-based migration using CPPC

CPPC Overview

Memory-based migration using CPPC:

1. **Negotiation** to achieve the migration point.

2. **Spawn** new processes that will be in charge of continuing the work.

3. **Checkpoint** is generated and the communication groups re-constructed.

4. **Checkpoint** transferring to new processes.

5. **Read** checkpoint, **restart** and continue execution.
Memory-based migration using CPPC

CPPC Overview

- Ckpt buffer
- Ckpt creation
- Spawn & reconf
- File recv
- & restart
- New P3
- mpi run migration request
- Ckpt buffer
- Spawn & reconf
- Ckpt creation
- Nego
- P0
- P1
- P2
- P3
- Nego
- Nego
- Nego
- Nego
- New P3
- File recv & restart
Problem: Time of the three phases may become unreasonable.

Without pipeline

Write  Transfer  Read & Restart

With pipeline

Write  Transfer  Read & Restart
Overlapping phases in the migration process

Checkpoint-based migration with CPPC: a) starting solution b) splitted solution.
Overlapping phases in the migration process

Checkpoint structure

Splits are made at **HDF5 level**: checkpoint files had to preserve the portability.
Overlapping phases in the migration process

Checkpoint structure

Splits are made at **HDF5 level**: checkpoint files had to preserve the portability.

CPPC checkpoint files are structured hierarchically, including four main HDF5 structures:

1. **Header**. Contains information about the checkpoint file.
2. **FileMap**. Records all the files opened during execution.
3. **Context**. Keeps track of the execution context changes.
4. **MemBlocks**. Includes the value of all register variables. It is the bigger structure ⇒ **SPLIT**
Overlapping phases in the migration process

Checkpoint structure

Splits are made at **HDF5 level**: checkpoint files had to preserve the portability.

CPPC checkpoint files are structured hierarchically, including four main HDF5 structures:

1. **Header**: Contains information about the checkpoint file.
2. **FileMap**: Records all the files opened during execution.
3. **Context**: Keeps track of the execution context changes.
4. **MemBlocks**: Includes the value of all register variables. It is the bigger structure ⇒ SPLIT

Checkpoint chunks had to **store metadata** to reconstruct the original checkpoint file.
Experimental results

The application testbed was comprised of six applications in the MPI version of the NAS Parallel Benchmarks (Class D).

A multicore cluster was used to evaluate the proposal:

- **Compute nodes**: 16 nodes, each one powered by 2 octo-core Intel Xeon CPUs (16 real cores per node) with 64 GB of RAM.

- **Network**: Compute nodes are connected via an Infiniband FDR network.
### Experimental results

#### Testbed

Experiments focus on the evaluation of the *reduction in the migration time with different fragment sizes*. Measured times:

- **Negotiation**. Time to achieve the migration point.
- **WriteCkpt**. Time for checkpoint writing.
- **Spawn**. Time of spawn function and communicators reconfiguration.
- **TransferRestart**. Transfer checkpoint files and restart.
Experimental results

Testbed

Experiments focus on the evaluation of the reduction in the migration time with different fragment sizes. Measured times:

- **Negotiation.** Time to achieve the migration point.
- **WriteCkpt.** Time for checkpoint writing.
- **Spawn.** Time of spawn function and communicators reconfiguration.
- **TransferRestart.** Transfer checkpoint files and restart.

Only results for LU will be shown in this presentation. All benchmarks have similar behaviour.

Checkpoint file size per process in LU benchmark: **1374 MB** (8 procs) and **694 MB** (16 procs).
Experimental results
Impact of the chunk size

Migration time (LU) using 8 procs **migrating 4 simultaneously.**

In the new proposal **WriteCkpt** time is overlapped with the **TransferRestart** time
Experimental results
Impact of the number of processes

Migration time (LU) using 16 procs **migrating 8 simultaneously.**

Migration time decreases when the total number of processes grows.
Conclusions

- We propose splitting checkpoint files as a solution to minimize the overhead in migration, overlapping the writing in the terminating processes, the data transfer through the network, and the state file read in the new processes.
Conclusions

- We propose splitting checkpoint files as a solution to minimize the overhead in migration, overlapping the writing in the terminating processes, the data transfer through the network, and the state file read in the new processes.

- **Main features:**
  - The splitting performed preserves the most important feature of CPPC: *portability.*
Conclusions

- We propose splitting checkpoint files as a solution to minimize the overhead in migration, overlapping the writing in the terminating processes, the data transfer through the network, and the state file read in the new processes.

- Main features:
  1. The splitting performed preserves the most important feature of CPPC: **portability**.
  2. New processes begin with the restart operation without waiting for the whole checkpoint file.
Conclusions

- We propose splitting checkpoint files as a solution to minimize the overhead in migration, overlapping the writing in the terminating processes, the data transfer through the network, and the state file read in the new processes.

- Main features:
  1. The splitting performed preserves the most important feature of CPPC: portability.
  2. New processes begin with the restart operation without waiting for the whole checkpoint file.
  3. The reduction in migration time is critical when migration is used to prevent application failures proactively.
Conclusions

- We propose splitting checkpoint files as a solution to minimize the overhead in migration, overlapping the writing in the terminating processes, the data transfer through the network, and the state file read in the new processes.

- Main features:
  1. The splitting performed preserves the most important feature of CPPC: portability.
  2. New processes begin with the restart operation without waiting for the whole checkpoint file.
  3. The reduction in migration time is critical when migration is used to prevent application failures proactively.
  4. New version requires less memory storage than the serialized classical version.
Improving an MPI application-level migration approach through checkpoint file splitting

M. Rodríguez, I. Cores, M.J. Martín and P. González

Universidade da Coruña
Computer Architecture Group
ivan.coresg@udc.es