Fault-tolerant protocol for hybrid task-parallel message-passing applications

Tatiana V. Martsinkevich
INRIA Saclay, Université de Paris Sud

Omer Subasi, Osman Unsal
Jesus Labarta
Barcelona Supercomputing Center

Franck Cappello
Argonne National Laboratory

Background

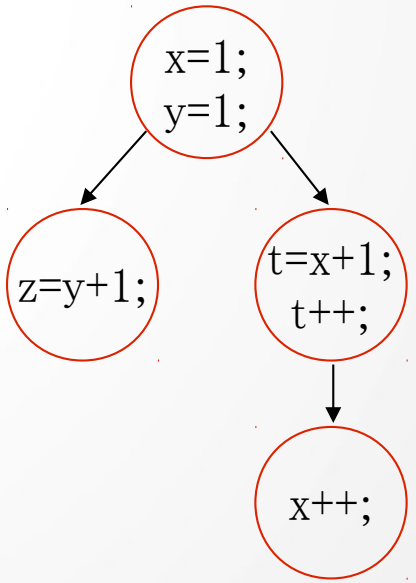
- Mean Time Between Failure (MTBF)
  - currently, few hours
  - will not get better on exascale
  - more frequent detected uncorrected errors (DUE)
- Programming models for large scale
  - hybrid MPI+threads or tasks
  - asynchronous execution
  - fault tolerance?

Background

- OmpSs – task-based PM (derived from OpenMP)
  - ➔ provides data directionality clauses (in, out, inout)
- Nanos – runtime that supports OmpSs apps
  - ➔ builds task dependency graph dynamically
  - ➔ data-flow execution

```
#pragma omp task output(x, y)
{       x = 1;
        y = 1;
}
#pragma omp task input(y) output(z)
        z=y+1;

#pragma omp task input(x) output(t)
{       t=x+1;
        t++;
}
#pragma omp task inout (x)
        x ++;
```



Background

- MPI+OmpSs PM
  - MPI calls inside tasks
  - communicating tasks run in parallel with other tasks
  - communication/computation overlap

Contributions

- Fault tolerance solution for MPI+OmpSs applications
  - combines task checkpointing + message logging
  - mitigates transient faults (DUEs) inside tasks
  - re-executes failed task
  - transparently handles MPI calls in tasks
- Evaluation
  - fault-free
  - execution with faults
- Factors impacting the execution overhead

Outline

- Related work
- Proposed FT solution
  - NanoCheckpoints
  - Message logging in tasks
- Evaluation
  - Fault-free execution
  - Scalability impact
  - Execution with faults
- Conclusion and future work

Related work

- Silent data corruption
  - task replication with majority voting [1]
  - task checkpointing+subgraph re-execution / ABFT [2]

- Differences with our solution
  - we do not handle SDC (yet)
  - we support MPI inside tasks

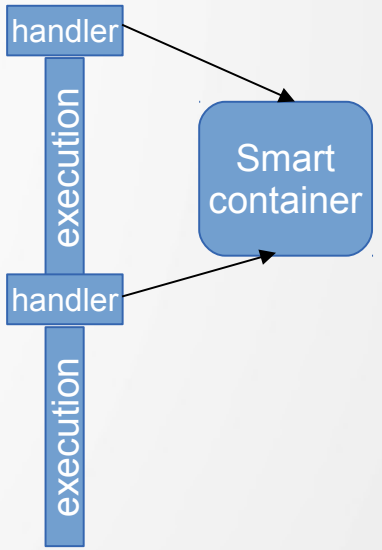
[1] O. Tahan, M. Shawky. *"Using Dynamic Task Level Redundancy for OpenMP Fault Tolerance"* (2014)
[2] C. Cao, T. Herault, G. Bosilca, J. Dongarra. *"Design for a soft error resilient dynamic task-based runtime".* (2015)

Proposed solution

- FT on the level of task
  - checkpoint task input parameters
  - message logging of task communication
- When a DUE happens the runtime:
  - catches the exception raised by the OS
  - restores input parameters
  - re-executes the task
  - (communication state is recovered from logs)
- Assume that DUE does not kill the process!

NanoCheckpoints

- Checkpoint `in/inout` params in memory
- Checkpoints not deleted after the task completes
    - avoid frequent `malloc()/free()`
- Rewrite old checkpoint for each new instance of the task
- Low fault-free overhead [1]



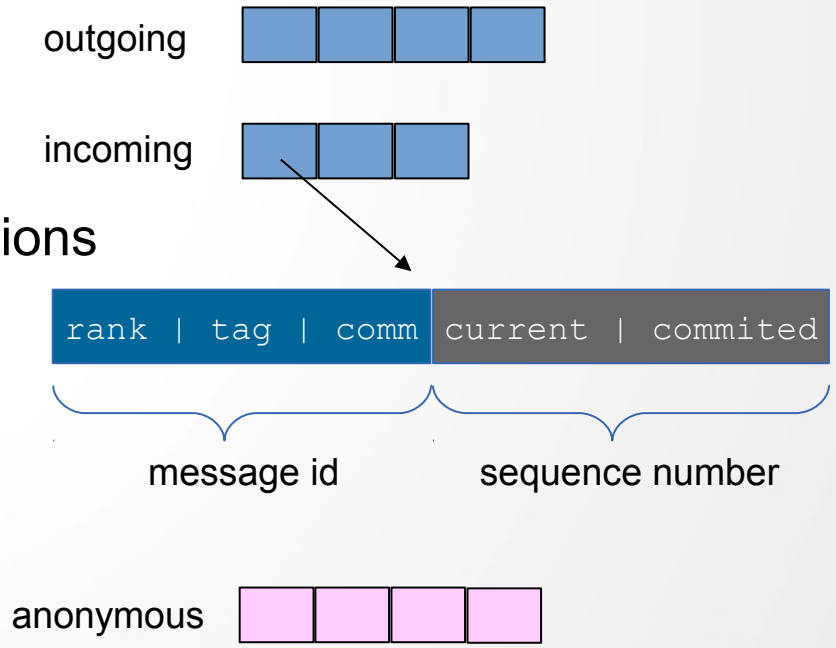
[1] O. Subasi, J. Arias, O. Unsal, J. Labarta, and A. Cristal, “NanoCheckpoints: A task-based asynchronous dataflow framework for efficient and scalable checkpoint/restart,” (2015)

Message logging

- Receiver-based logging
  - the task fails, but the process does not → logs not lost
- Logging within the context of a task
  - separate logs for tasks
  - delete upon task completion

Message logging protocol

- Sequence number for every message id
  - `current` and `commited` values
- Separate lists for in/out communication and anonymous reception order
- Fault-free task execution
  - Increment `current` and `commited` together
  - log incoming messages
  - log order of matched anonymous receptions
- Task recovery
  - Reset `current` to 0
  - Increment only `current` value
  - When `current = commited`, recovery finished



Evaluation

- MareNostrum@BSC
  - 2 Intel SandyBridge-EP E5-2670 CPU x 8 cores (16 cores per node)
  - 32 GB of RAM per node
  - InfiniBand FDR10 interconnect
- OpenMPI-1.6.4+PMPI wrapper library
- MPI+OmpSs benchmarks: Himeno, Nbody, Matmul (mxm)
- 64 ps-s with 16 threads

Fault-free overhead

- Message logging impacts execution more than checkpointing
  - ➔ receiver-based logging :-\

Runtime overhead

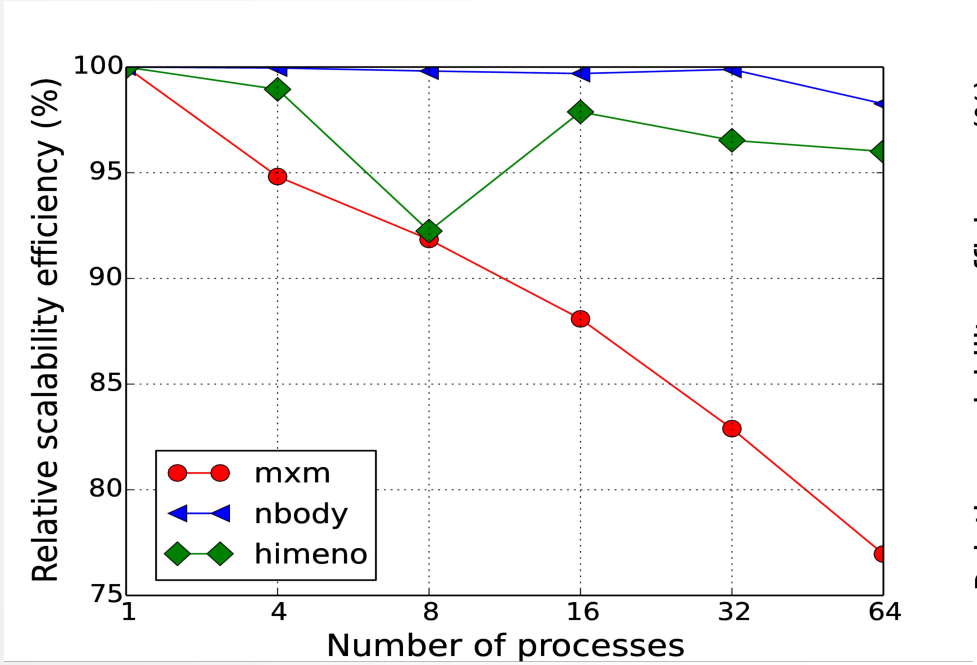
Himeno	Nbody	Matmul
0.89%	0.31%	4.45%

Memory overhead

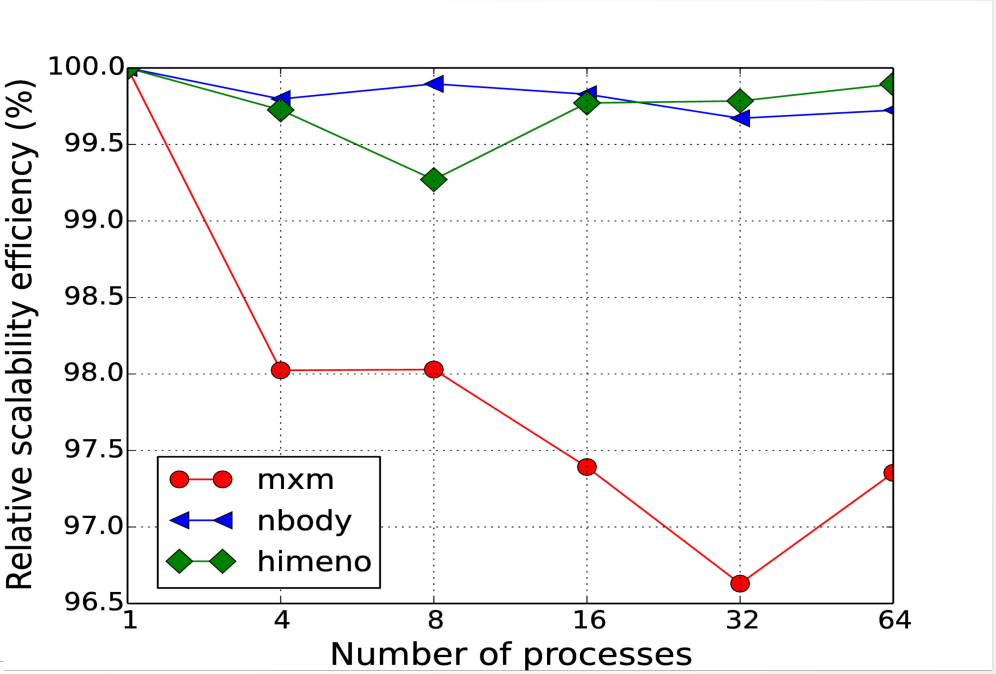
	Checkpoint Size (MB)	Total Message Log (MB)	Peak Message Log (MB)
Matmul	512	8064	128
Nbody	0.59	0.75	0.25
Himeno	1202	448	1.2

Scalability impact

- Moderate impact on weak scalability
- Message logging may decrease strong scalability



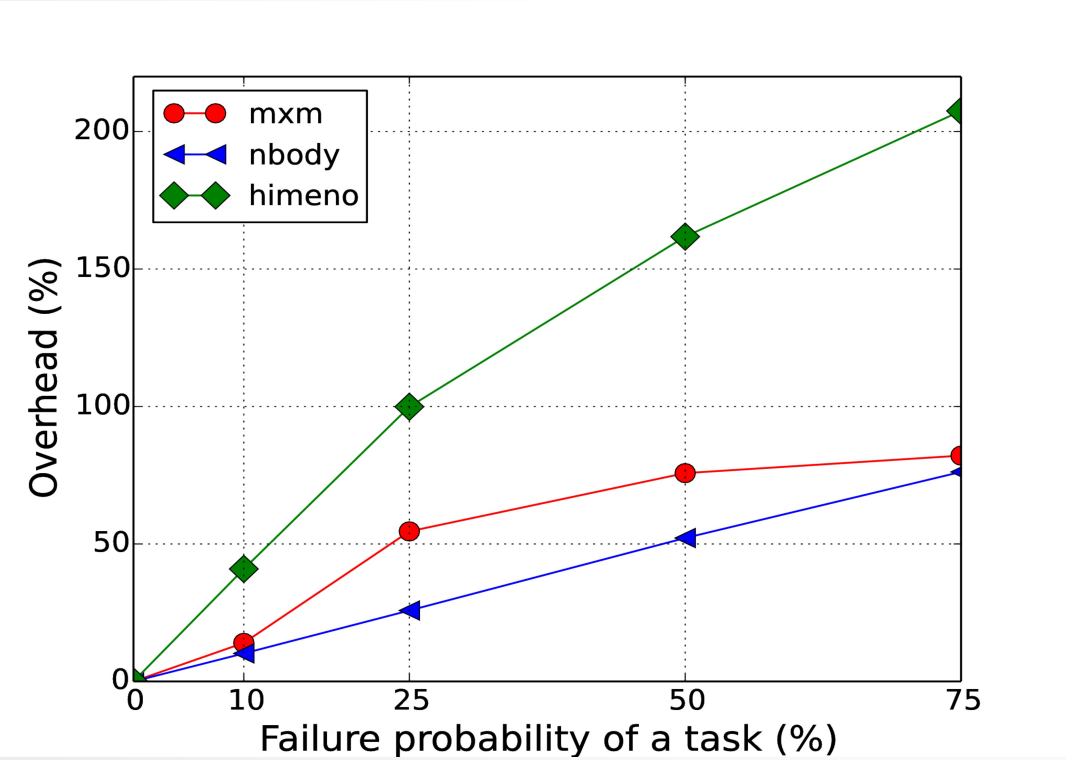
Strong scalability



Weak scalability

Execution with faults

- Assume worst case scenario: fault at the end of the task execution
- Overhead is higher if
  - → coarse task granularity
  - → strong task coupling



Fault rate (faults/sec)

	10	25	50	75
Himeno	1.59	2.82	4.24	5.44
Nbody	0.3	0.64	0.99	1.27
Matmul	0.16	0.31	0.53	0.76

Conclusion and future work

- FT solution on task level for handling DUEs
- The runtime overhead below 5%
- More independent tasks → easier to mask task recovery overheads

- Future work
  - support for SDC errors in message logging
  - tighter integration of message logging with task checkpointing