Imperial College London

Towards resilience at exascale: memory-conservative fault tolerance in Nektar++

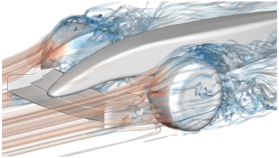
Chris Cantwell

ExaFLOW

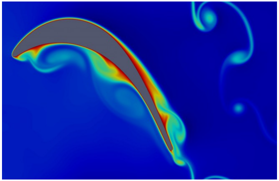
Nektar++ Workshop 2017,
Imperial College London
15th June 2017

Application Area: Aerodynamics

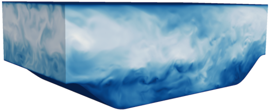
Formula 1 Car (Lombard, Sherwin)



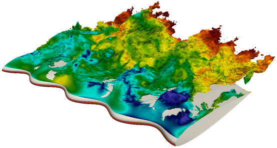
T106C turbine blade (Mengaldo)



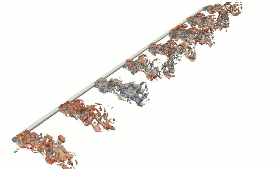
Turbulent hill (Moxey)



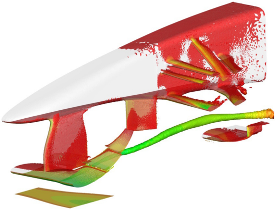
NACA 0012 with wavy leading-edge (Serson)



Vortex-induced vibration of cylinder (Bao)

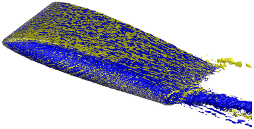


Y250 Wing on F1 car (Lombard, Sherwin)

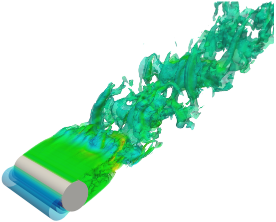


LES of wingtip vortex at $Re = 1.2$M

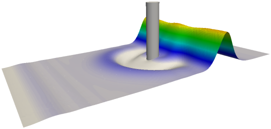
(Lombard, Moxey, Sherwin)



Compressible Flow over a cylinder (Mengaldo)



Shallow Water Equations (Eskilsson)

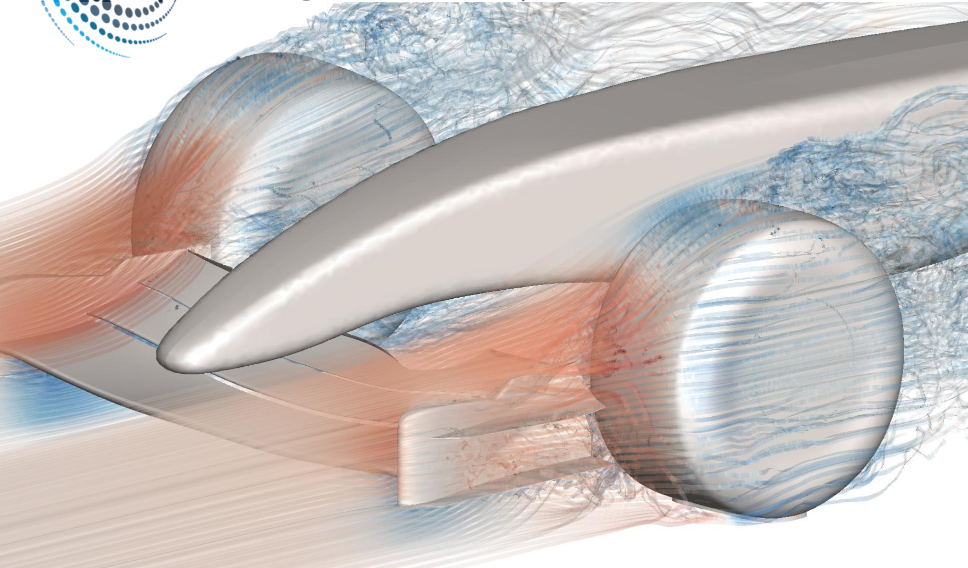


ExaFLOW

Horizon 2020

2015-2018

Enabling Exascale Fluid Dynamics Simulations



ExaFLOW

Horizon 2020

2015-2018

Enabling Exascale Fluid Dynamics Simulations

"*address current* **algorithmic bottlenecks** *to enable the use of* **accurate CFD** *codes for problems of practical engineering interest*"

Objectives

Adaptive error control and mesh refinement

Solver efficiency

Strategies for fault tolerance and resilience

Heterogeneous modelling

Extreme Parallel I/O and data reduction

Energy awareness of high-order methods

The ExaFLOW project has received funding from the European Union Horizon 2020 Framework Programme (H2020) under grant agreement number 671571

ExaFLOW

Enabling Exascale Fluid Dynamics Simulations

Horizon 2020

2015-2018

"*address current* **algorithmic bottlenecks** *to enable the use of* **accurate CFD** *codes for problems of practical engineering interest*"

Objectives

Adaptive error control and mesh refinement

Solver efficiency

Strategies for fault tolerance and resilience

Heterogeneous modelling

Extreme Parallel I/O and data reduction

Energy awareness of high-order methods

Codes

Nek5000

Nektar++

SBLI

NS3D

The ExaFLOW project has received funding from the European Union Horizon 2020 Framework Programme (H2020) under grant agreement number 671571

ExaFLOW

Horizon 2020

2015-2018

Enabling Exascale Fluid Dynamics Simulations

"*address current* **algorithmic bottlenecks** *to enable the use of* **accurate CFD** *codes for problems of practical engineering interest*"

Objectives

Adaptive error control and mesh refinement

Solver efficiency

Strategies for fault tolerance and resilience

Heterogeneous modelling

Extreme Parallel I/O and data reduction

Energy awareness of high-order methods

Codes

Nek5000

Nektar++

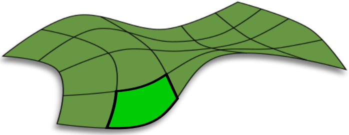
SBLI

NS3D

Work with: Allan Nielsen, David Moxey, Jan Hesthaven, Spencer Sherwin

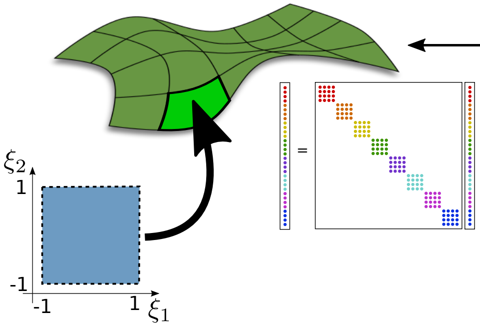
The ExaFLOW project has received funding from the European Union Horizon 2020 Framework Programme (H2020) under grant agreement number 671571

Incompressible Navier-Stokes solver



Read mesh

Incompressible Navier-Stokes solver

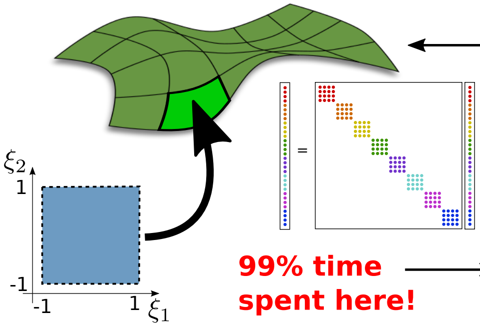


Read mesh

Preprocessing
Build matrices

STATIC PHASE

Incompressible Navier-Stokes solver



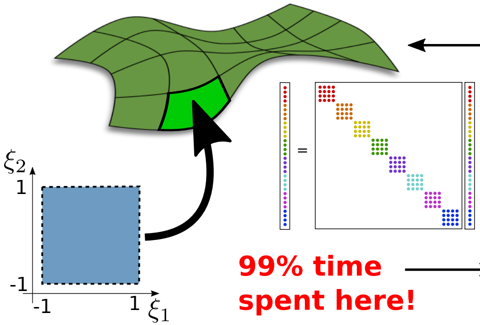
99% time spent here!

- Read mesh
- Preprocessing Build matrices
- Time-integrate
  - Non-linear advection
  - Conjugate Gradient
  - Checkpoint
- Write final solution

STATIC PHASE

DYNAMIC PHASE

Incompressible Navier-Stokes solver



Read mesh

Preprocessing
Build matrices

99% time spent here!

Time-integrate

Process-local operation

Non-linear advection

Reductions each iteration
Assembly each iteration

Conjugate Gradient

Checkpoint

How often should we do this?

Write final solution

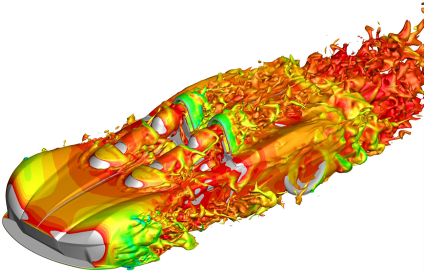
STATIC PHASE

DYNAMIC PHASE

Challenges of Exascale

Solve bigger problems in more detail

Contours of CP0, coloured by pressure



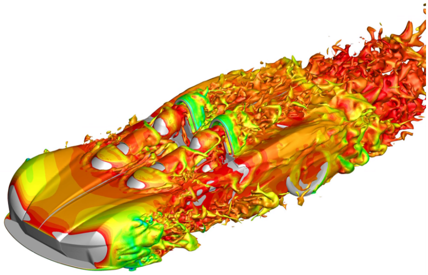
RP1 car by Elemental Cars

- 2M elements
  (697k prisms, 1653k tets)
- 5th-order, 4 variables
- 488M local DOFs.
- 2k cores on an SGI ICE-XA
- 122k DOFs/core
- 72hrs runtime, 30min start-up

Challenges of Exascale

Solve bigger problems in more detail

Contours of CP0, coloured by pressure



RP1 car by Elemental Cars

- 2M elements
  (697k prisms, 1653k tets)
- 5th-order, 4 variables
- 488M local DOFs.
- 2k cores on an SGI ICE-XA
- 122k DOFs/core
- 72hrs runtime, 30min start-up

Massive increase in parallelism

- Thousands/millions of cores - frequent hardware failures
- Low memory/core - costly (power and time) to move data

...so how do we make efficient use of such machines

The need for Resilience at Exascale

- Errors due to both hardware failure and software bugs
- *Hard* (permanent) and *soft* (transient) errors

The need for Resilience at Exascale

- Errors due to both hardware failure and software bugs
- *Hard* (permanent) and *soft* (transient) errors
- Exascale systems expected to contain $>$100,000 nodes
- Increased probability of system failure: more CPUs, memory, disks

The need for Resilience at Exascale

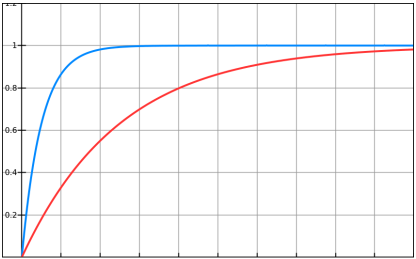
- Errors due to both hardware failure and software bugs
- *Hard* (permanent) and *soft* (transient) errors
- Exascale systems expected to contain $>$100,000 nodes
- Increased probability of system failure: more CPUs, memory, disks
- Model time of failure with exponential distribution with PDF: $f(t) = \frac{1}{M} e^{-t/M}$
- M is the mean time to interrupt (MTTI)

The need for Resilience at Exascale

- Errors due to both hardware failure and software bugs
- *Hard* (permanent) and *soft* (transient) errors
- Exascale systems expected to contain $>$100,000 nodes
- Increased probability of system failure: more CPUs, memory, disks
- Model time of failure with exponential distribution with PDF: $f(t) = \frac{1}{M} e^{-t/M}$
- M is the mean time to interrupt (MTTI)
- Integrate to get prob. of failure before time T: $\mathbb{P}(t \leq T) = 1 - e^{-T/M}$

MTTI of system:

$$\frac{1}{M_{sys}} = n \times \frac{1}{M_{node}}$$



Blue: $M = 1$, Red: $M = 5$

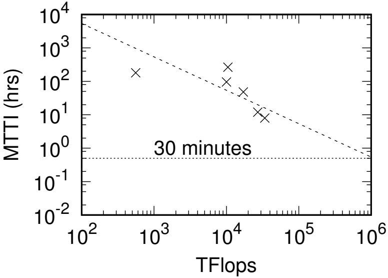
The need for Resilience at Exascale

- Errors due to both hardware failure and software bugs
- *Hard* (permanent) and *soft* (transient) errors
- Exascale systems expected to contain $>$100,000 nodes
- Increased probability of system failure: more CPUs, memory, disks
- Model time of failure with exponential distribution with PDF: $f(t) = \frac{1}{M} e^{-t/M}$
- M is the mean time to interrupt (MTTI)
- Integrate to get prob. of failure before time T: $\mathbb{P}(t \leq T) = 1 - e^{-T/M}$

MTTI of system:

$$\frac{1}{M_{sys}} = n \times \frac{1}{M_{node}}$$

Name	PF	MTTI
Mira (BG/Q)	10.0	4-7 days
Titan	27.11	12 hours
Tianhe-2	33.86	8 hours
(Exascale)	1000	$<$ 30 min



At exascale, failures will be the norm

NEKTAR++
SPECTRAL/HP ELEMENT FRAMEWORK

Check-pointing to disk

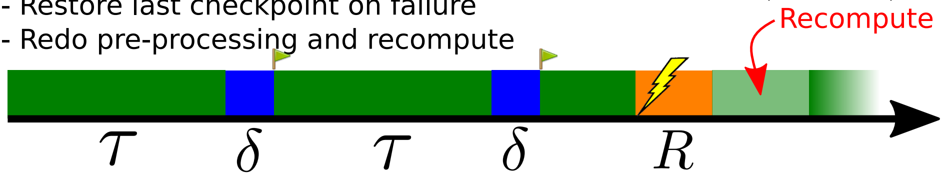
"Classic" resilience methodology
- Periodically save state
- Restore last checkpoint on failure
- Redo pre-processing and recompute

Check-pointing to disk

Proportion of useful compute:

$$\frac{t_s}{t_w} = e^{-\frac{R}{M}} \left(\frac{\tau}{M\left(e^{\frac{\tau+\delta}{M}} - 1\right)} \right)$$

"Classic" resilience methodology
- Periodically save state
- Restore last checkpoint on failure
- Redo pre-processing and recompute



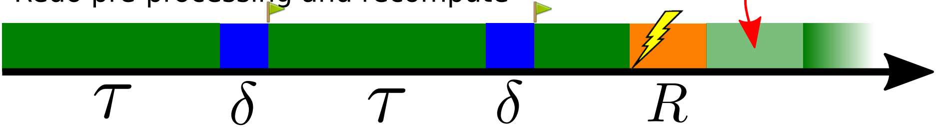
"A higher order estimate of the optimum checkpoint interval for restart dumps", J.T. Daly, Future Generation Computer Systems, 22:300–312, 2006

Check-pointing to disk

Proportion of useful compute:

$$\frac{t_s}{t_w} = e^{-\frac{R}{M}} \left(\frac{\tau}{M\left(e^{\frac{\tau+\delta}{M}} - 1\right)} \right)$$

"Classic" resilience methodology
- Periodically save state
- Restore last checkpoint on failure
- Redo pre-processing and recompute



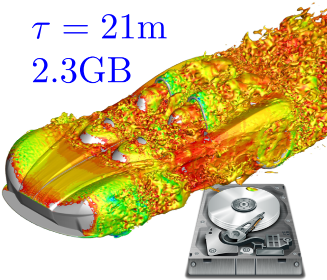
Optimal checkpoint interval:

$$\tau_{opt} = \sqrt{2\delta(M+R)} \quad \tau + \delta << M$$

$\delta = 2.5\text{s}$

$\tau = 21\text{m}$

2.3GB



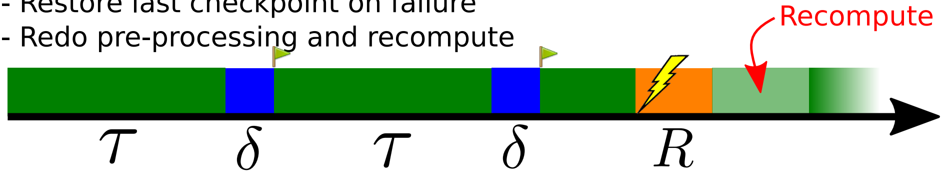
"A higher order estimate of the optimum checkpoint interval for restart dumps", J.T. Daly, Future Generation Computer Systems, 22:300–312, 2006

Check-pointing to disk

"Classic" resilience methodology
- Periodically save state
- Restore last checkpoint on failure
- Redo pre-processing and recompute

Proportion of useful compute:

$$\frac{t_s}{t_w} = e^{-\frac{R}{M}} \left(\frac{\tau}{M\left(e^{\frac{\tau+\delta}{M}} - 1\right)} \right)$$



Optimal checkpoint interval:

$$\tau_{opt} = \sqrt{2\delta(M+R)} \quad \tau + \delta << M$$

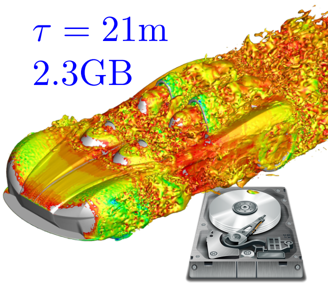
$\delta = 2.5\mathrm{s}$

$\tau = 21\mathrm{m}$

$2.3\mathrm{GB}$

Massive I/O load on distributed file systems

At Exascale we might consider:
- MTTI is order of minutes, M smaller
- Larger simulations, δ larger

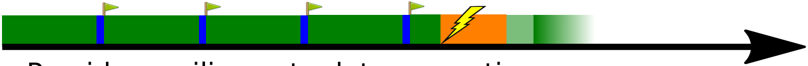


At exascale disk checkpointing time $\approx$ MTTI

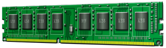
"A higher order estimate of the optimum checkpoint interval for restart dumps", J.T. Daly, Future Generation Computer Systems, 22:300–312, 2006

Checkpointing to (remote) memory

Local in-memory check-pointing

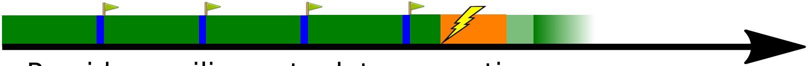


- Provides resilience to data corruption errors
- More frequent check-points, reduced recompute
- Scalable
- No resilience to hardware / node failure
- Combine with less-frequent disk check-pointing



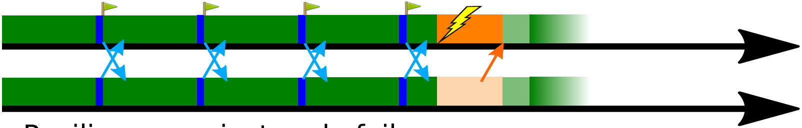
Checkpointing to (remote) memory

Local in-memory check-pointing

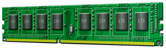


- Provides resilience to data corruption errors
- More frequent check-points, reduced recompute
- Scalable
- No resilience to hardware / node failure
- Combine with less-frequent disk check-pointing

Remote in-memory check-pointing

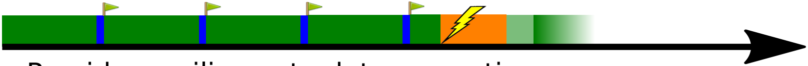


- Resilience against node failure
- Intelligent 'buddying' of processes e.g. 'netloc'
- Greater demand on network
- Write checkpoint to disk on failure



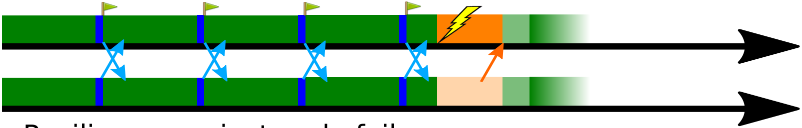
Checkpointing to (remote) memory

Local in-memory check-pointing



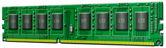
- Provides resilience to data corruption errors
- More frequent check-points, reduced recompute
- Scalable
- No resilience to hardware / node failure
- Combine with less-frequent disk check-pointing

Remote in-memory check-pointing



- Resilience against node failure
- Intelligent 'buddying' of processes e.g. 'netloc'
- Greater demand on network
- Write checkpoint to disk on failure

If node fails, we are short of compute-capacity.



A new strategy for surviving failure

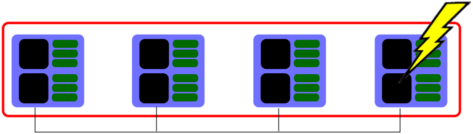
Main challenges:

- Data is costly to move around, low memory-per-core at exascale
- Writing data to disk is slow and energy inefficient
- Restart and redistribution of work is expensive
- Static phase requires collective operations
- Complexity of existing codes

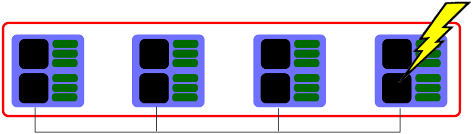
Proposed strategy:

- Exploit proposed *User-Level Failure Mitigation (ULFM)* in MPI to enrol a spare node to replace failed nodes
- Record *result* of communication during initialisation (static) phase
- Rapidly reconstruct process state locally on spare and continue
- Utilise remote-memory checkpointing for (dynamic) solution
- Minimally intrusive changes to existing code required

ULFM: Recovering from node failure



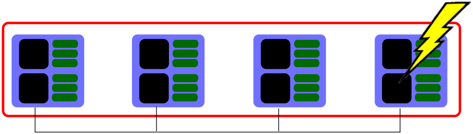
ULFM: Recovering from node failure



User-level Failure Management (ULFM)

MPIX_Comm_agree
MPIX_Comm_shrink
MPIX_Comm_revoke

ULFM: Recovering from node failure



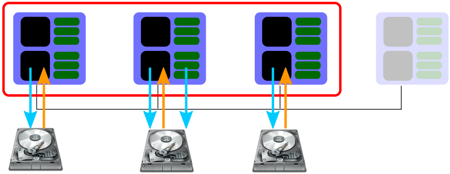
User-level Failure Management (ULFM)

MPIX_Comm_agree
MPIX_Comm_shrink
MPIX_Comm_revoke

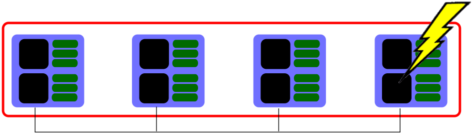
Two options:

1. Shrink MPI communicator

- Remove dead process
- Redistribute computation
- Preprocess / rebuild matrices
- Restart job?



ULFM: Recovering from node failure



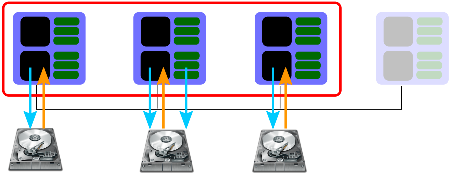
User-level Failure Management (ULFM)

MPIX_Comm_agree
MPIX_Comm_shrink
MPIX_Comm_revoke

Two options:

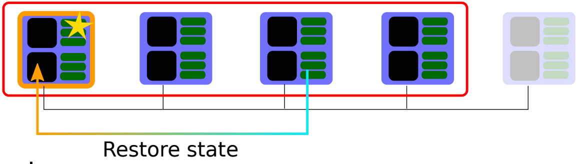
1. Shrink MPI communicator

- Remove dead process
- Redistribute computation
- Preprocess / rebuild matrices
- Restart job?



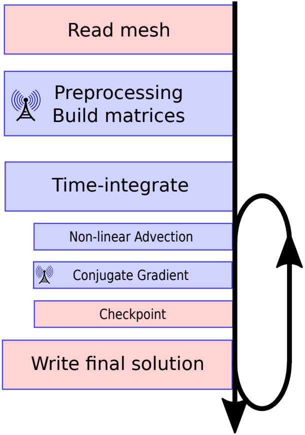
2. Add a spare

- Initialise a spare node
- Recover remote in-memory check-point
- Continue computation
- All other nodes untouched

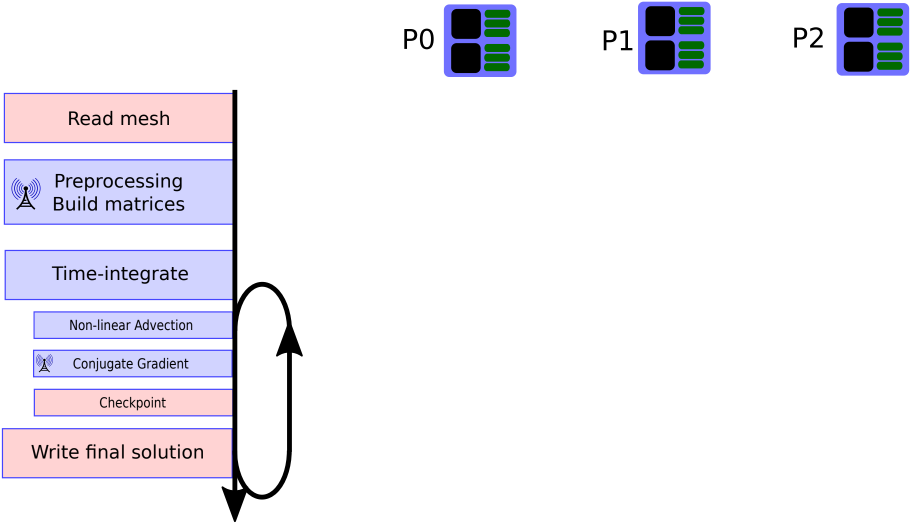


Are spares a waste of resources?

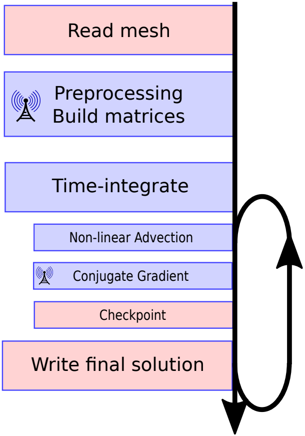
Putting everything together

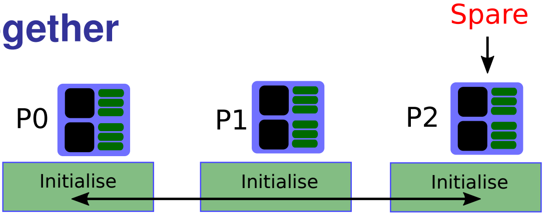


Putting everything together



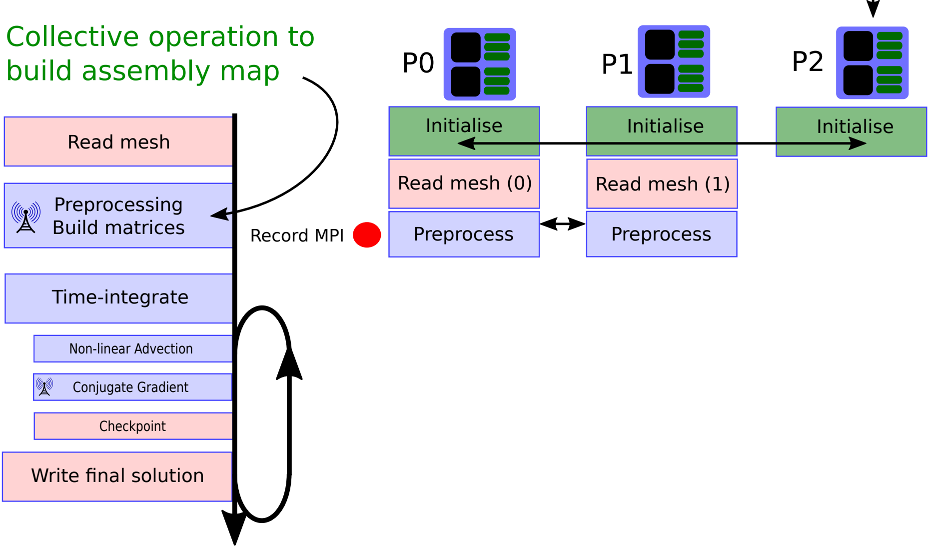
Putting everything together



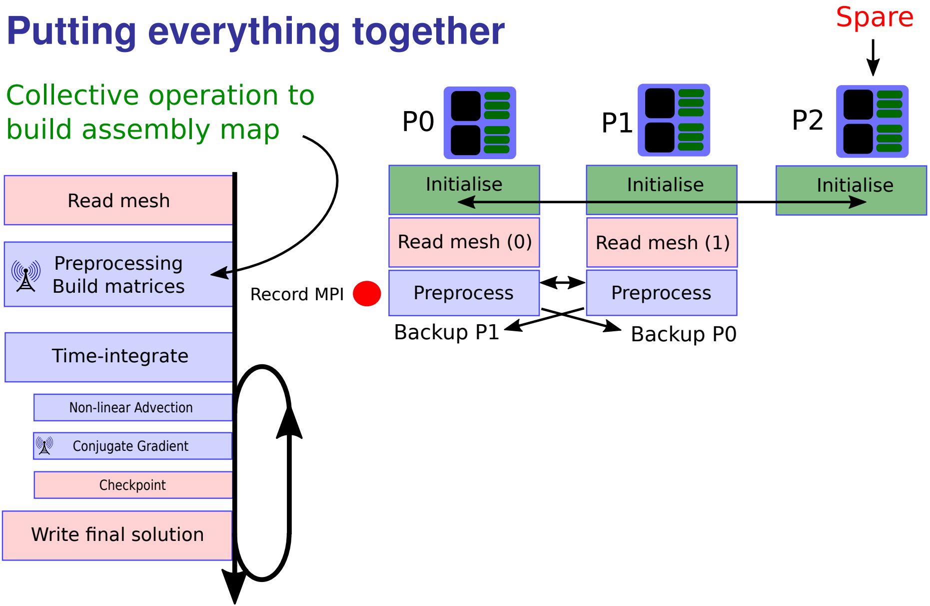


Putting everything together

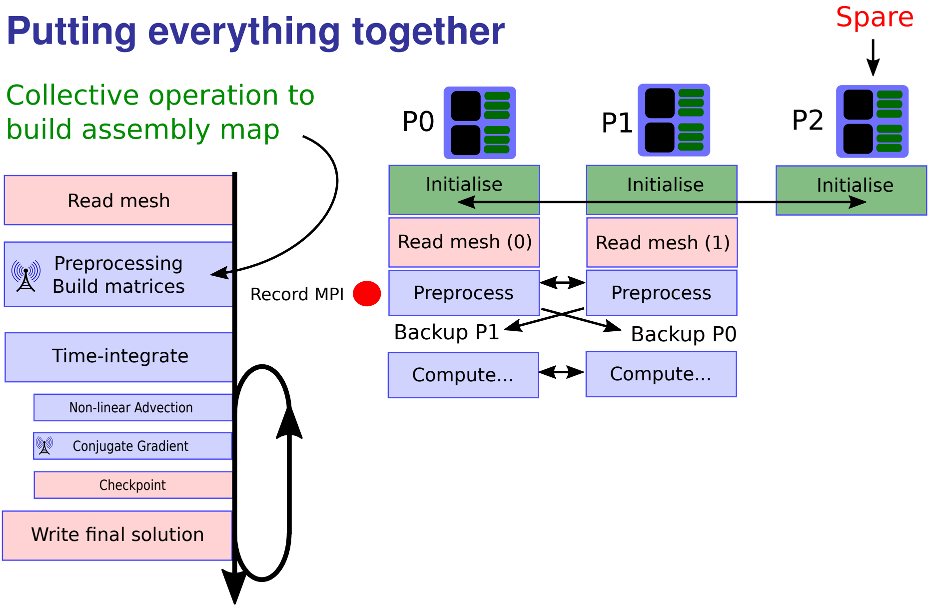
Spare



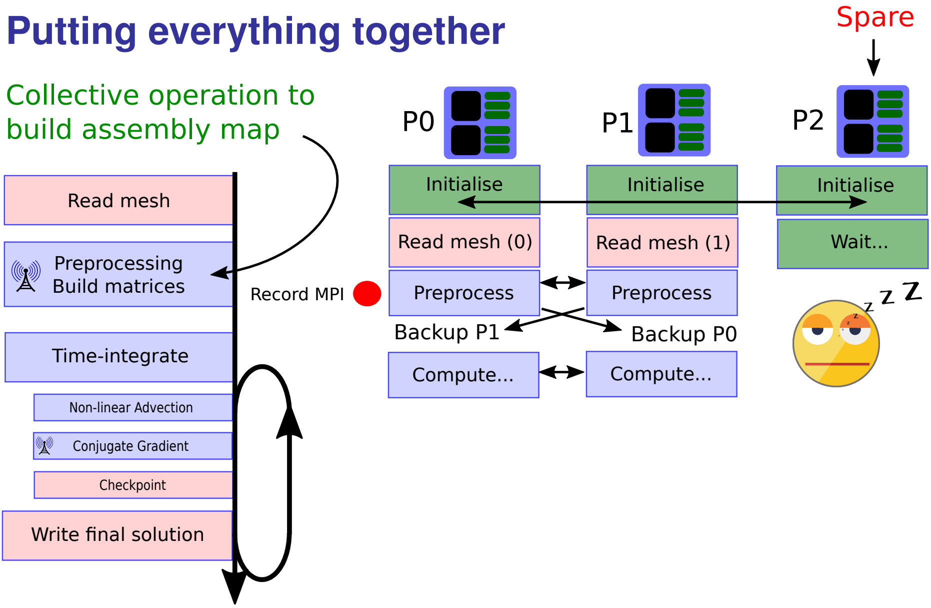
Putting everything together



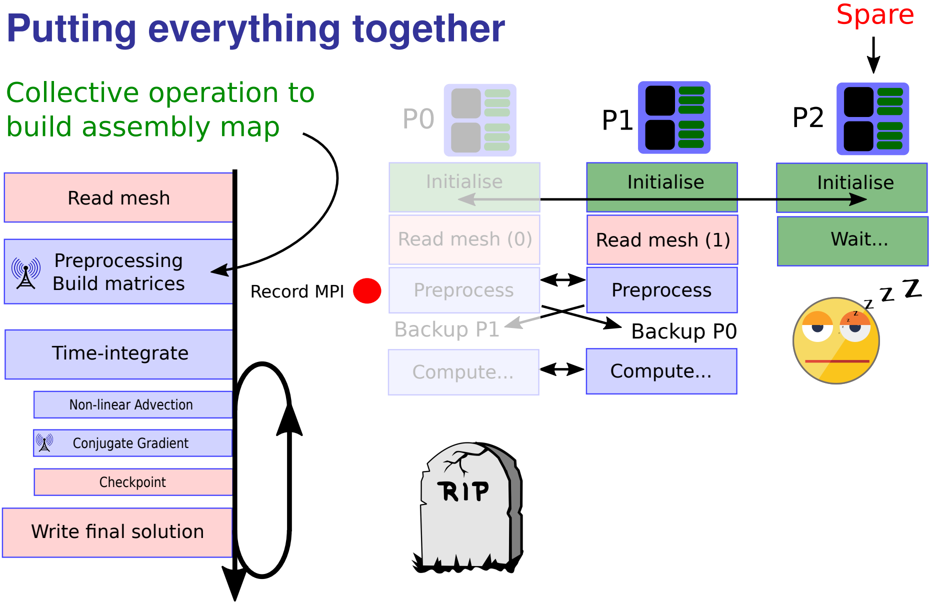
Putting everything together



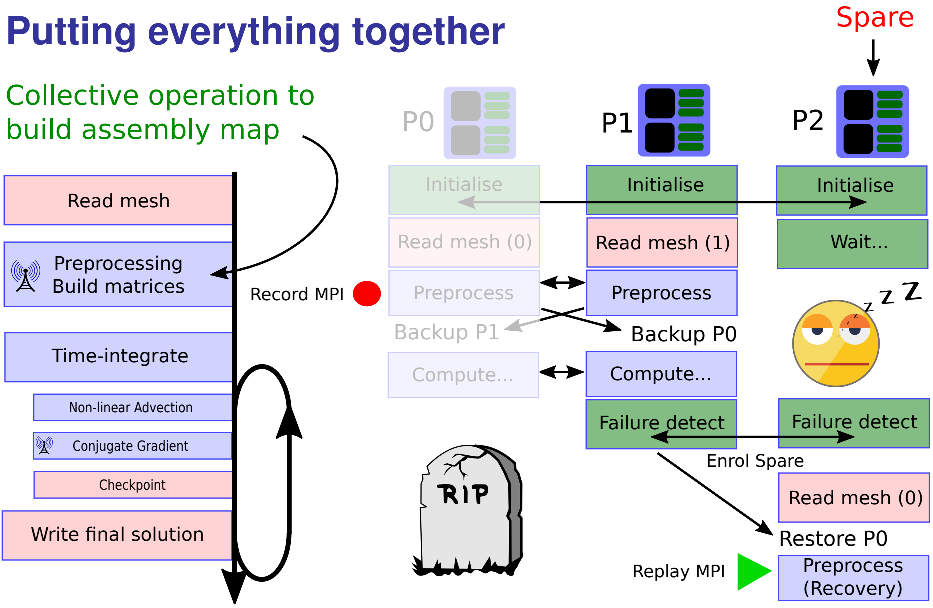
Putting everything together



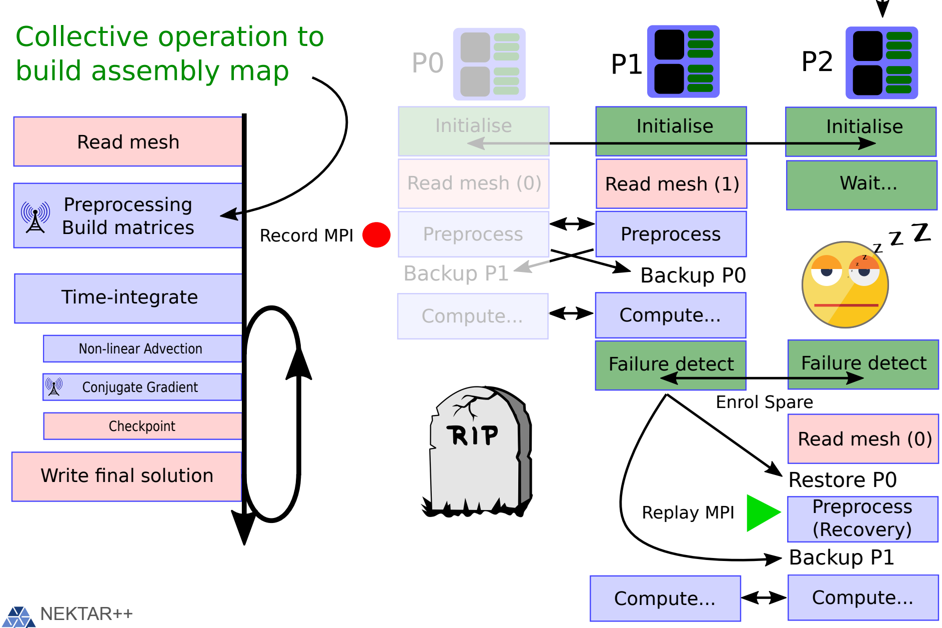
Putting everything together



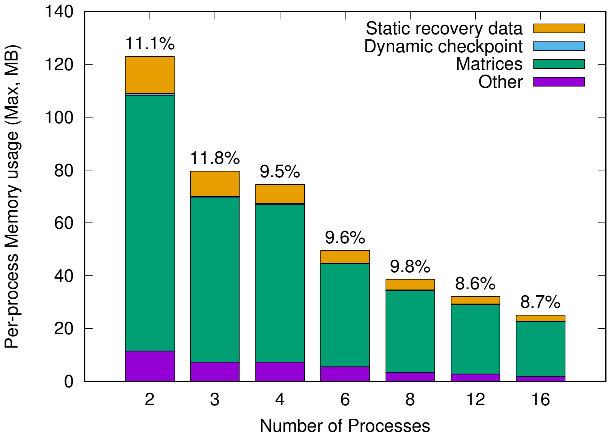
Putting everything together



Putting everything together



Indicative memory usage

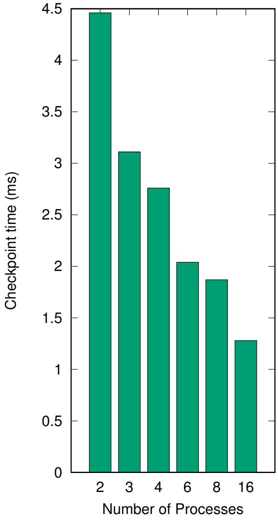


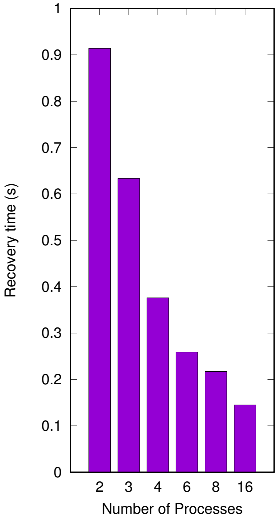
Flow past a cylinder
830 elements, P=8
≈30k DOF

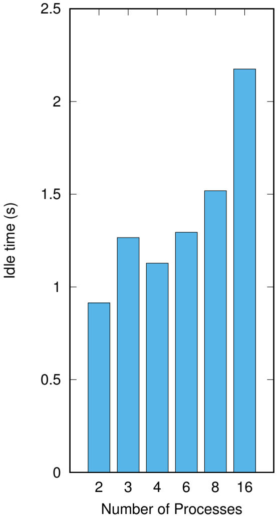
NProc	≈DOF/Proc
2	15000
3	10000
4	7500
6	5000
8	3750
12	2500
16	1900

- Memory required for recovery generally around 10% of total
- Dynamic checkpoint data less than 0.5%

Preliminary benchmarking







Summary

Exascale machines will likely have low memory/core and short MTTI.

- Algorithms at exascale need to be resilient to hardware failure
- Traditional approaches using disk check-pointing will be infeasible
- Propose new memory-conservative strategy for time-dependent solvers
  - Use ULFM to avoid costly restarts
  - Partition algorithm into *static* and *dynamic* phases
  - Remote in-memory checkpointing through asynchronous pairwise exchange
  - Only store the result from MPI calls to allow independent local recovery of the lost partition

Summary

Exascale machines will likely have low memory/core and short MTTI.

- Algorithms at exascale need to be resilient to hardware failure
- Traditional approaches using disk check-pointing will be infeasible
- Propose new memory-conservative strategy for time-dependent solvers
  - Use ULFM to avoid costly restarts
  - Partition algorithm into *static* and *dynamic* phases
  - Remote in-memory checkpointing through asynchronous pairwise exchange
  - Only store the result from MPI calls to allow independent local recovery of the lost partition

Future work:

- Test scalability on large-scale platforms (requires ULFM MPI!)
- Optimise remote-memory checkpoint placement

Thank you for listening!