This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

An Exascale Programming, Multi-objective Optimisation and Resilience Management Environment Based on Nested Recursive Parallelism

AllScale – Pilots Applications

AmDaDos

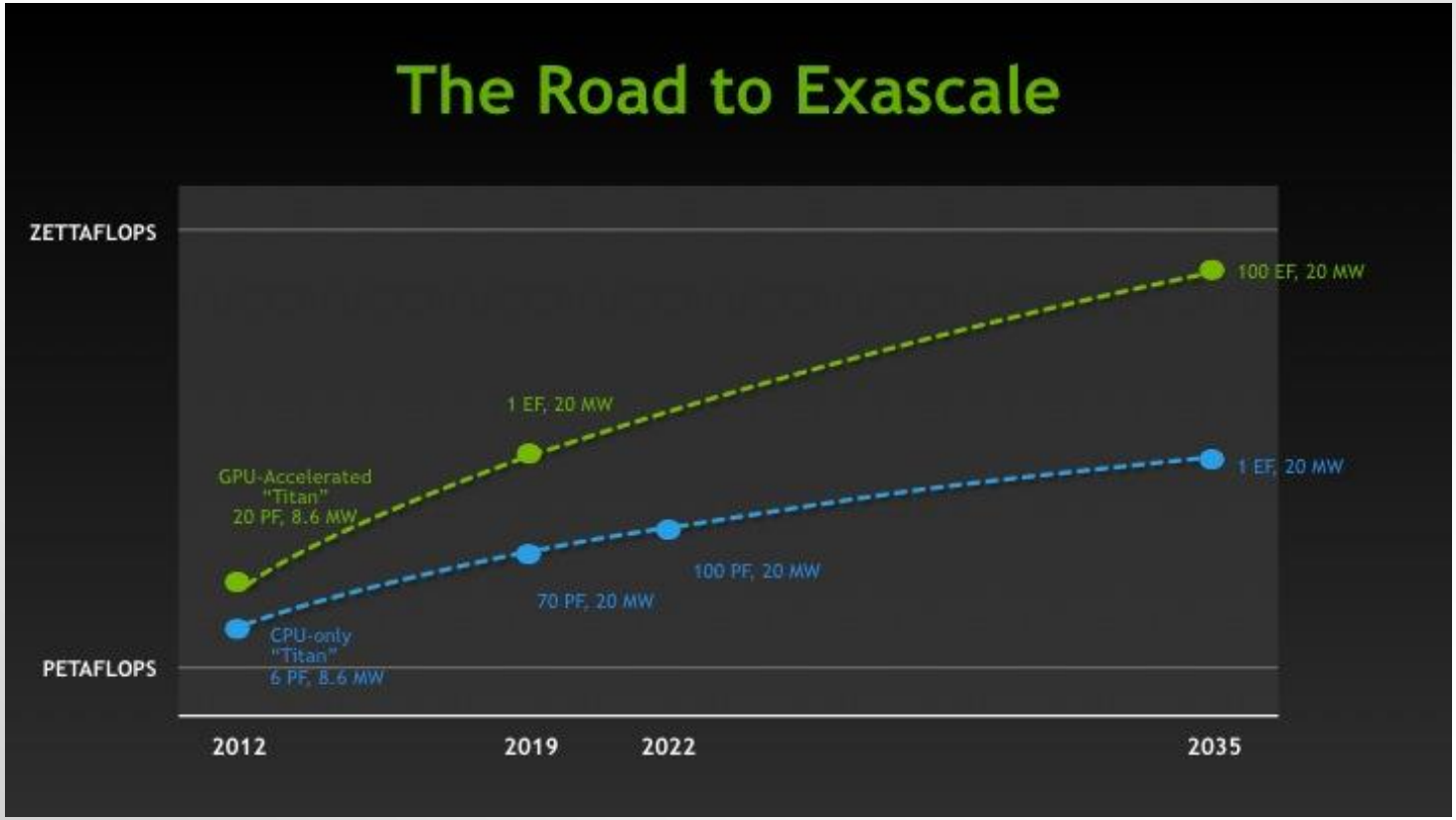
Adaptive Meshing and Data Assimilation for the Deepwater Horizon Oil Spill

Emanuele Ragnoli, Fearghal O Donncha
IBM Research, Ireland

Agenda

- Exascale
- AllScale
- AmDaDos (Why DD)
- Early experiments and MPI
- Conclusion and future work

Exascale *computing refers to computing systems capable of at least one exaFLOPS, or a billion billion calculations per second. Such capacity represents a thousand fold increase over the first petascale computer that came into operation in 2008.*



Motivation

- **Exascale** systems will likely be
  - multi-node
  - multi-core (millions)
  - accelerator based

  architectures exhibiting multiple levels of parallelism, including
  - nodes
  - sockets
  - cores
  - vector units, and
  - instruction level parallelism

How to **harness** this power?

Motivation

- **Exascale** systems will likely be
  - multi-node
  - multi-core (millions)
  - accelerator based

  architectures exhibiting multiple levels of parallelism, including
  - nodes
  - sockets
  - cores
  - vector units, and
  - instruction level parallelism

How to **program** such systems?

Problems

- **Dominating HPC languages** are
  - tailored for **specific** architecture **designs**
  - largely **static** (e.g. fixed number of threads)

- Most languages promote **flat parallelism** like parallel loops, which imposes the need for **global synchronization**

- Accelerator languages and MPI:
  - Low-level style of programming – **everything left to the developer**

- **Hybrid parallel programs** suffer from
  - hard-coded problem decompositions schemas
  - lack of coordination among runtime systems

AllScale Vision

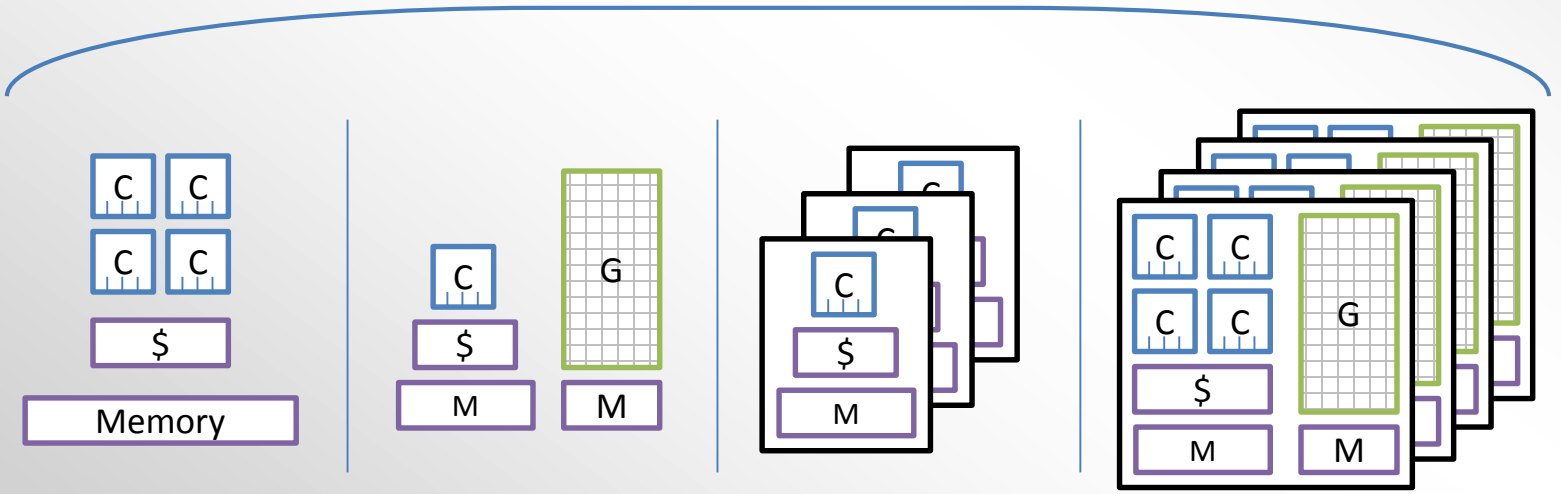
Application

Parallel Algorithm

Unified Parallel Programming Model

Toolchain

Portability, Tuning, and Resilience



Multicore | Accelerators | Clusters | Heterogeneous Clusters

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

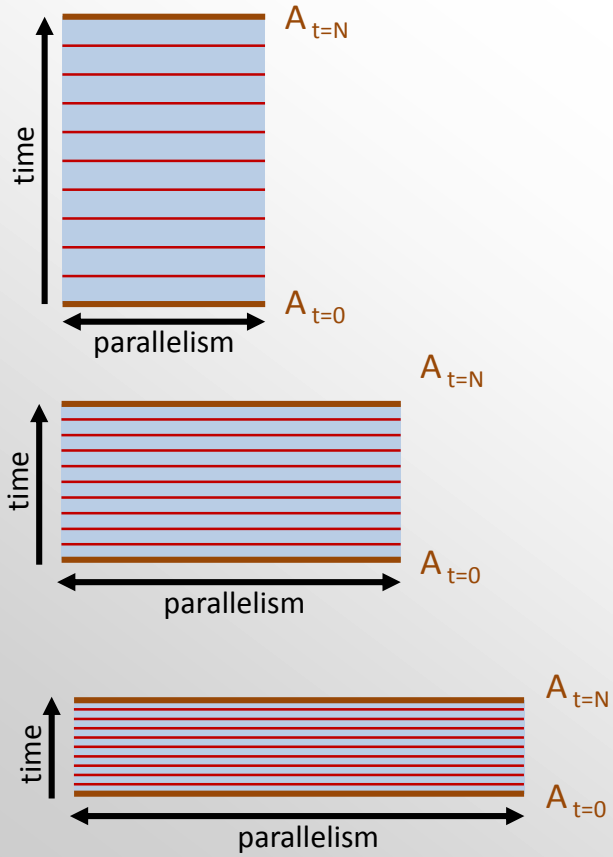
Objectives

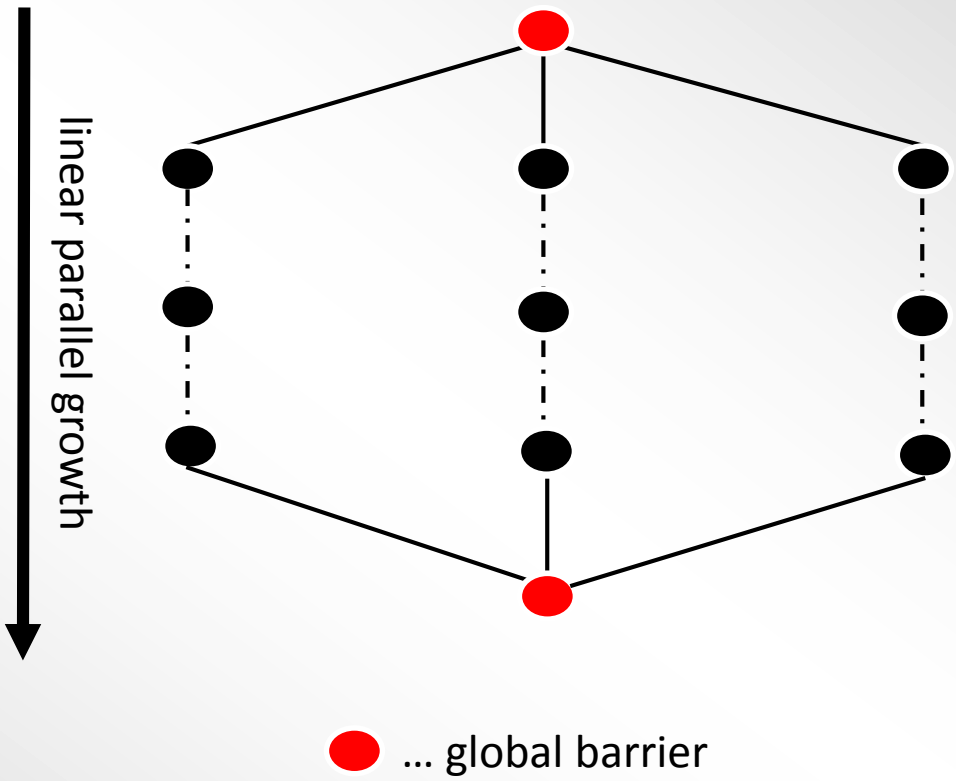
- **Objective 1** – Single Source to Any Scale
  - substantial improvement in productivity
- **Objective 2** – Exploit Recursive Parallelism
  - foundation of scalability
- **Objective 3** – Multi-Objective Optimization
  - time, energy, resource usage
- **Objective 4** – Unified Runtime System
  - one to rule them all (objects and resources)
- **Objective 5** – Mitigating Hardware Failures
  - let system manage recovery
- **Objective 6** – Integrated Monitoring
  - runtime system supported online/offline profiling

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

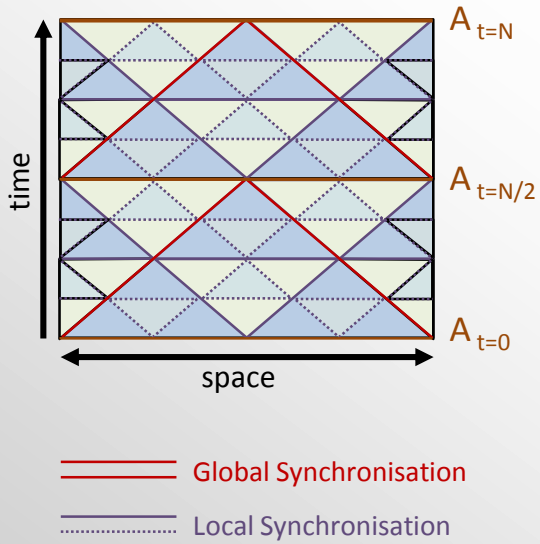
Conventional Flat Parallelism

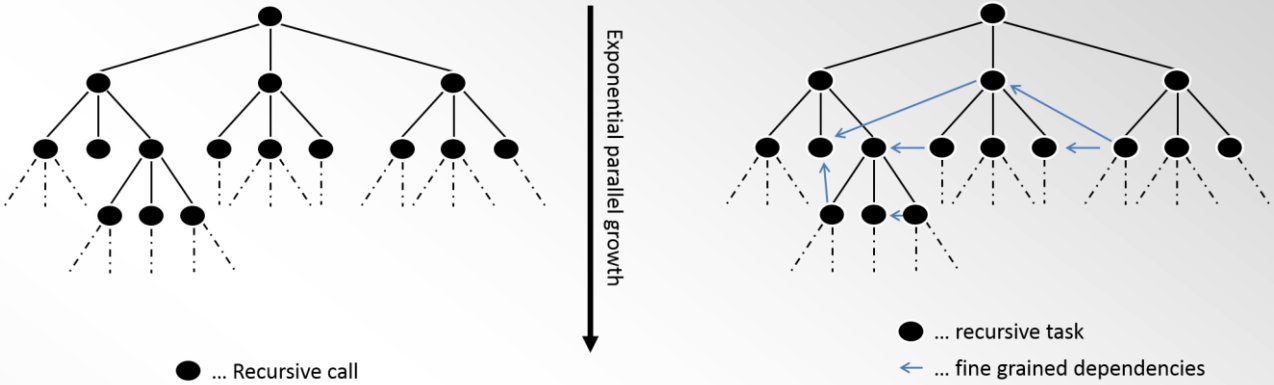
How to map flat parallelism to a hierarchical parallel architecture?
Complex handling of errors – global operations

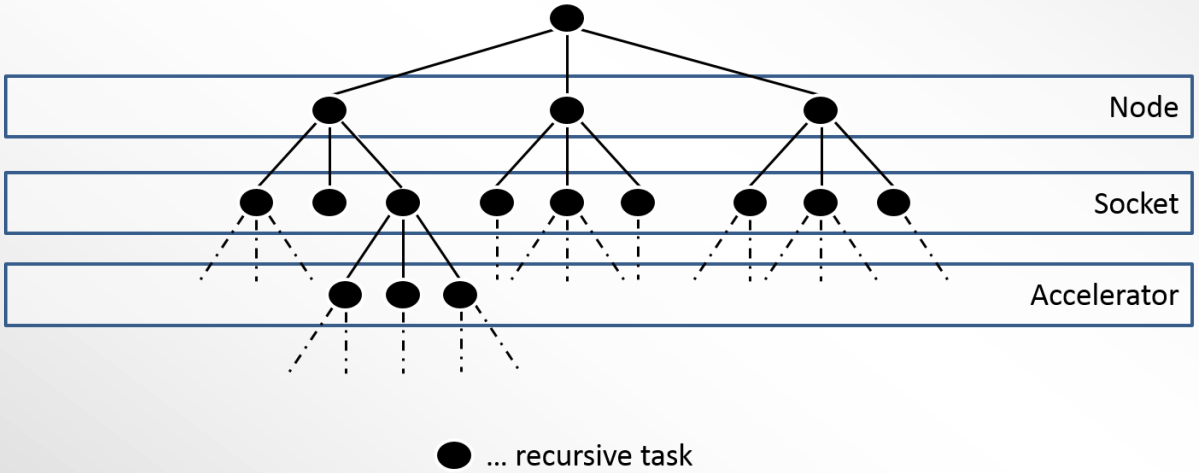




Recursively Nested Parallelism

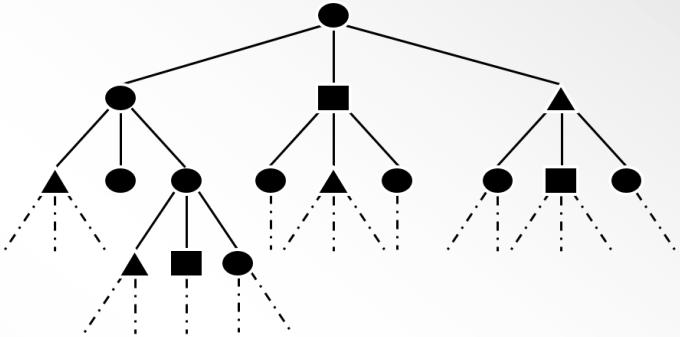






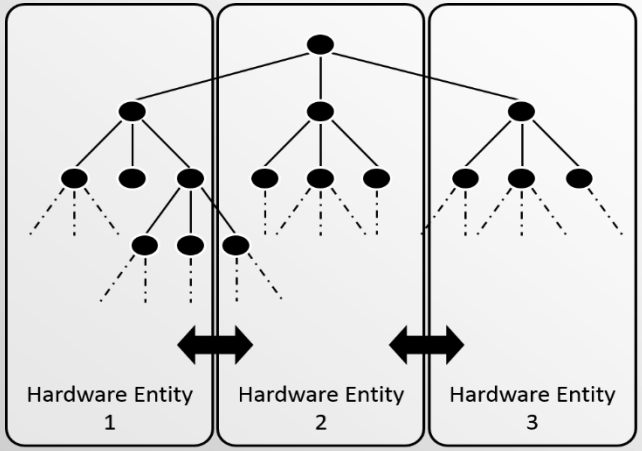
Maps naturally to multiple levels of HW parallelism

Recursive Parallelism

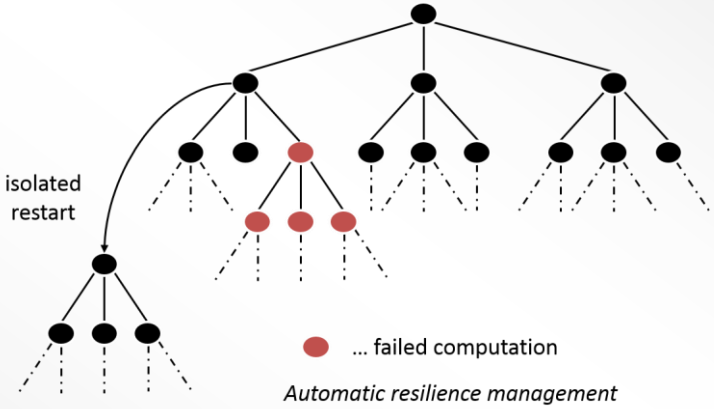


▲ ■ ● ... task versions

Multiversioning allows adaption to hardware & system state



Dynamic load balancing and data migration

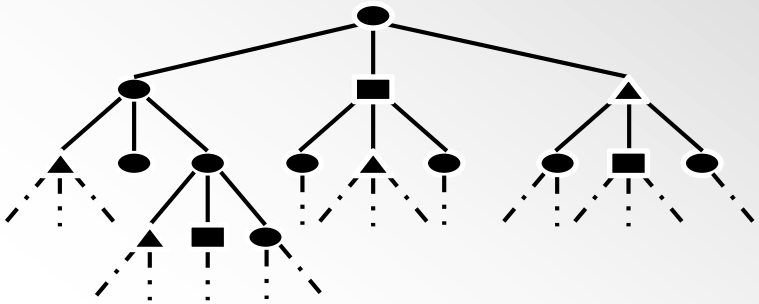


Automatic resilience management

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

Compiler

- **Analyzes** rec primitive **usage** and **data accesses**
- Generates **multiple code versions** for each step
  - Sequential
  - Shared memory parallel
  - Distributed memory parallel
  - Accelerator

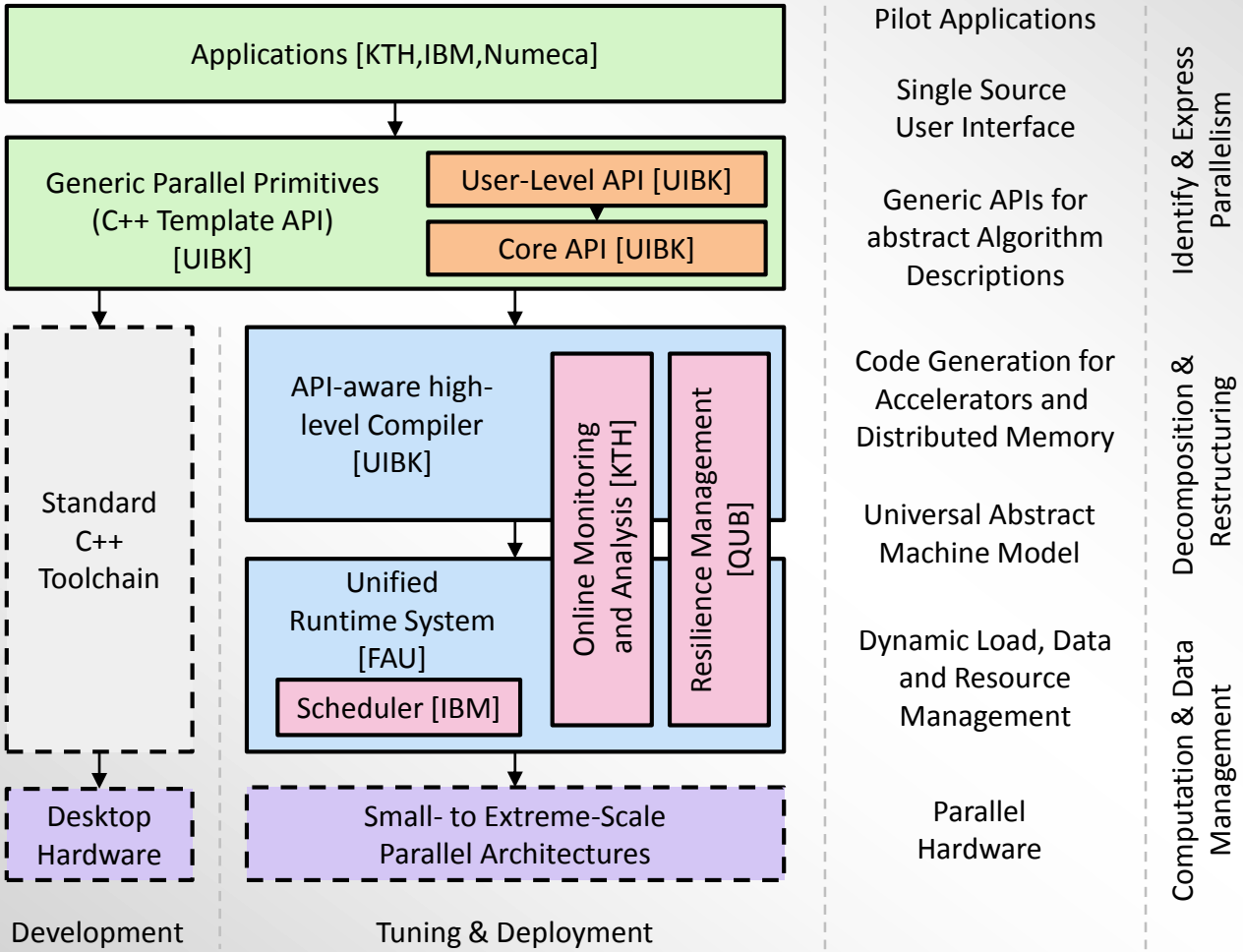


- **Reports** potential **issues** to programmer
  - Data dependencies, race conditions, …
- Provides **additional information to runtime**
  - E.g. type of recursion and data dependencies
  - Improves dynamic optimization potential

Runtime System

- Provides an **abstract parallel machine** as target for compiler-generated code
- **Manages distributed resources**
  - Data locality
  - Communication & synchronization
  - Accelerators
  - Dynamic load balancing
- **Selects** from compiler-generated **code versions**
  - Depending on hardware and execution context

Components



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

AmDaDos

Motivation

- Large Scale Oil Spills requires quick and prompt response.
- Tracking at high resolution the impact of the oil is key to proper emergency management operations.
- The computational complexity of high resolution models for oils spills tracking is computationally very difficult if real time is required.

Components

- Transport of a chemical constituent
- Data assimilation
- Adaptive meshing

Algorithms

- First generation FEM advection-diffusion model
- Scalable data assimilation algorithms

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

AmDaDos - Approach

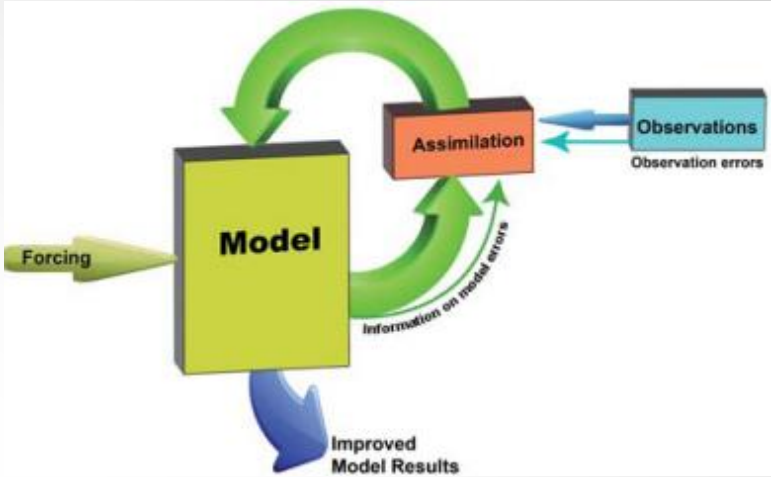
- First generation advection-diffusion model with data assimilation

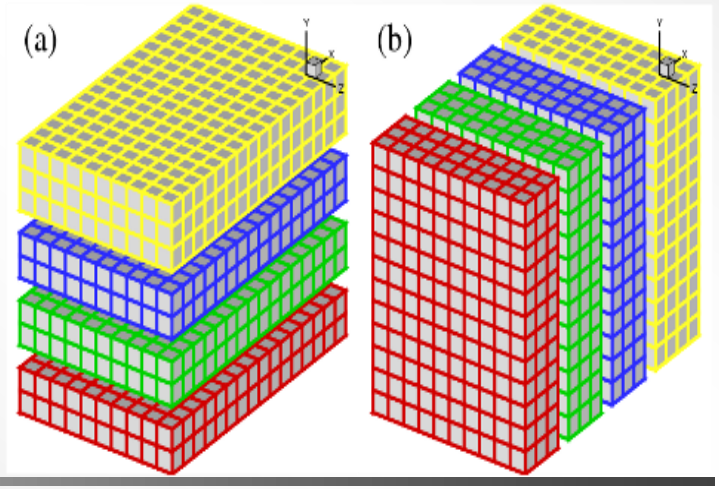
$$\frac{\partial C}{\partial t} = \mu \frac{\partial C}{\partial x} + \mu \frac{\partial C}{\partial y}$$

- Domain Decomposition with FEM (ADN)
  - Make data assimilation algorithms computationally feasible
  - Requires coordination of solution across adjacent subdomains

- Adaptive Meshing with Data Assimilation
  - refinements at observations and boundaries

- Computational expense dictated by number of cells in each domain





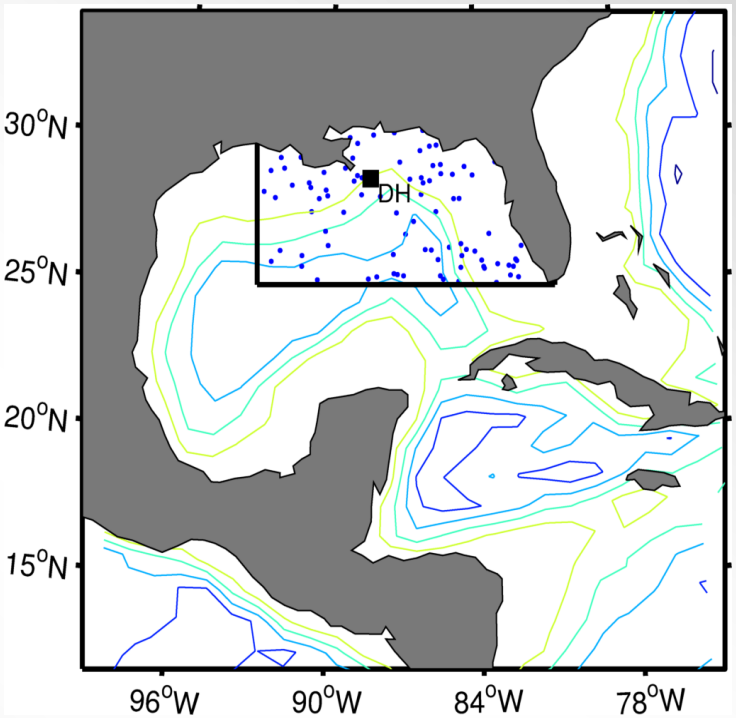
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

AmDaDos - Approach

- First generation advection-diffusion model with data assimilation

$$\frac{\partial C}{\partial t} = \mu \frac{\partial C}{\partial x} + \mu \frac{\partial C}{\partial y}$$

- Domain Decomposition with FEM (ADN)
  - Make data assimilation algorithms computationally feasible
  - Requires coordination of solution across adjacent subdomains
- Adaptive Meshing with Data Assimilation
  - refinements at observations and boundaries
- Computational expense dictated by number of cells in each domain



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

AmDaDos - Challenge

Nesting Region	Resolution (meters)	# cells (10^8)	AM FLOPS	DA FLOPS	Data per day (TB)
1) Global Model	100	0.03	$6*10^{13}$	$3*10^{18}$	0.2
2.1) AM oil	20	0.01	$1*10^{14}$	$1*10^{18}$	0.07
2.2) AM oil	**4**	**62.5**	**$3*10^{18}$**	**$6*10^{22}$**	**432**
3.1) AM coast	20	0.03	$3*10^{14}$	$3*10^{18}$	0.2
3.2) AM coast	**4**	**0.34**	**$2*10^{16}$**	**$3*10^{20}$**	**2.35**
4.1) AM observations	20	25.0	$1*10^{17}$	$2*10^{19}$	172
4.2) AM observations	**4**	**156.25**	**$4*10^{18}$**	**$2*10^{21}$**	**1078**
Total		244.17	$8*10^{18}$	N/A	1686

- Issues
  - Actual MPI existing benchmarks in DA are not scalable and cannot achieve real time

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

AmDaDos - Utilizing AllScale

- ca 20k lines of C++ code
- FEM code
- Sequential code and MPI code
- API with 20+ subroutines

- Synchronization
  - global at each time step
  - global at checkpoints
- Main solvers:
  - DiscretizeSubProblemByFEM
  - SolveRiccatiEquation
  - SolveFilterEquation
- Global synchronization
  - UpdateBoundaryData
- Libraries dependencies:
  - Armadillo
  - OPenBlas

Algorithm 1 Algorithm of localised minimax filter method

```
Require:
  T                    // number of time steps
  globalproblem        // description of global physical problem
  errorlevel           // acceptable level of Schwartz iteration error
  GetInterfaceError()  // computes the difference between estimates on the interface
                       // nodes obtained from adjacent subdomains
  subproblems = DecomposeProblem(globalproblem)
  for t = 1 to T do
    for subdomain in subdomains do
      DiscretizeSubproblemByFem(subproblem, t)
      UpdateBoundaryData(subproblem, subproblems, t)
      if HasObservations(subproblem) then
        InitObservations(subpoblem, t)
      else
        InitPseudoObservations(subpoblem, t)
      end if
      SolveRiccatiEquation(subproblem, t)
      SolveFilterEquation(subproblem, t)
    end for

    error = GetInterfaceError(subproblems, t)
    while error > errorlevel do
      for subdomain in subdomains do
        UpdateBoundaryData(subproblem, subproblems, t)
        SolveFilterEquation(subproblem, t)
      end for
      error = GetInterfaceError(subproblems, t)
    end while
  end for
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

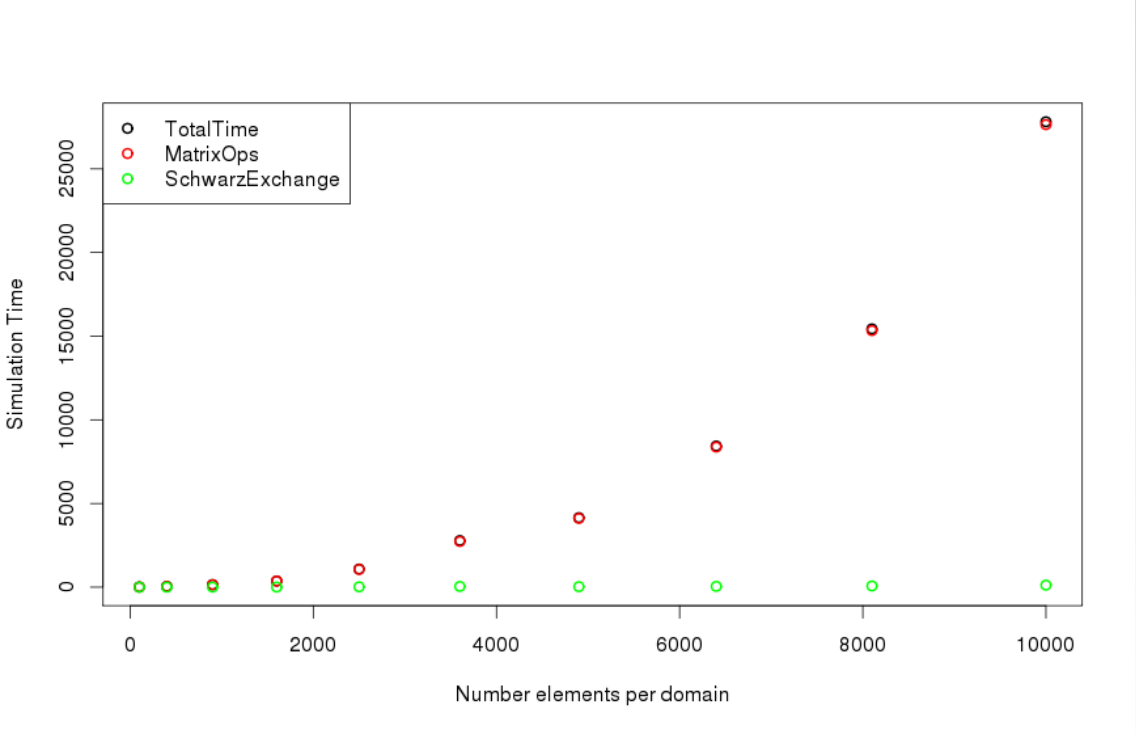
Serial performance 1) function of number Elements

1. **Experiments done on 3 x 1 domain varying number elements per domain**

2. **Total simulation time:**
   1. **Subdomain solution**
   2. **Boundary Exchange**

3. **Subdomain solution involves matrix operations on n x n element matrix – exponential compute complexity**

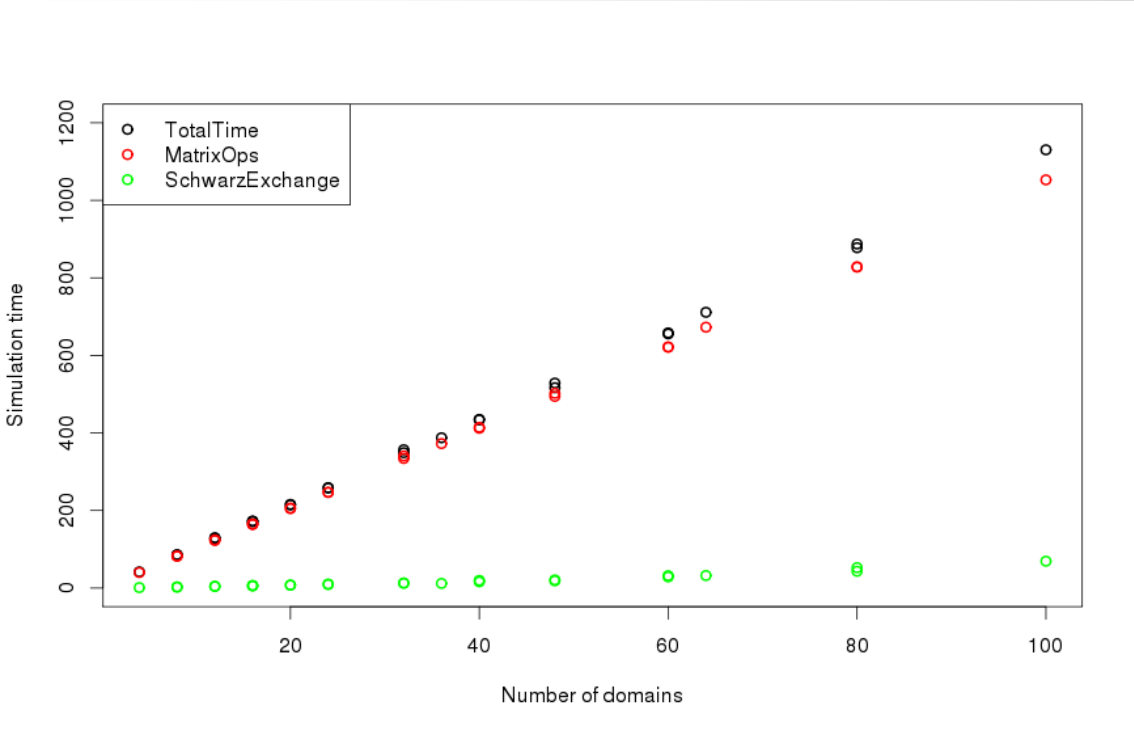
4. **Schwarz exchange passes boundary information between subdomains (flat profile)**



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

Serial performance: 2) function of number domains

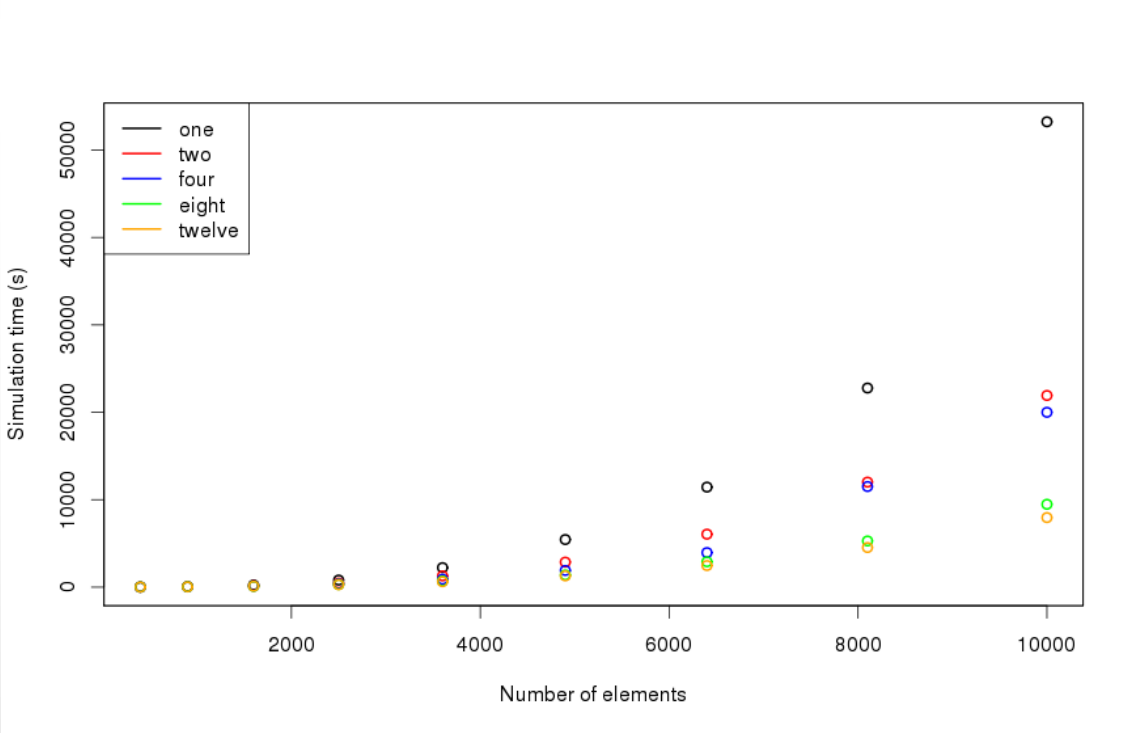
1. **Increase number of subdomains from 1x1 to 10x10 (100 subdomains total).**
2. **For all sims, each subdomain composed of 20 x 20 elements**
3. **Simulation time increases linearly as function of number of subdomains.**
4. **Schwarz does not become punitive (green)**
5. **Obviously this is serial, in parallel one could expect better scaling with concurrent task allocations...**



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

Pseudo parallel performance – Scaling as function of OpenMp threads

1. **In serial, matrix operation significant component of total simulation time**
2. **Linear algebra built on optimized blas library with intrinsic OpenMP parallelization**
3. **Simulation time using different number of openMP threads**
4. **At 10000 elements 2 OMP threads gives superlinear speedup of 2.43**



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

Pseudo parallel performance – Speedup & parallel efficiency

1. **Parallel speedup**

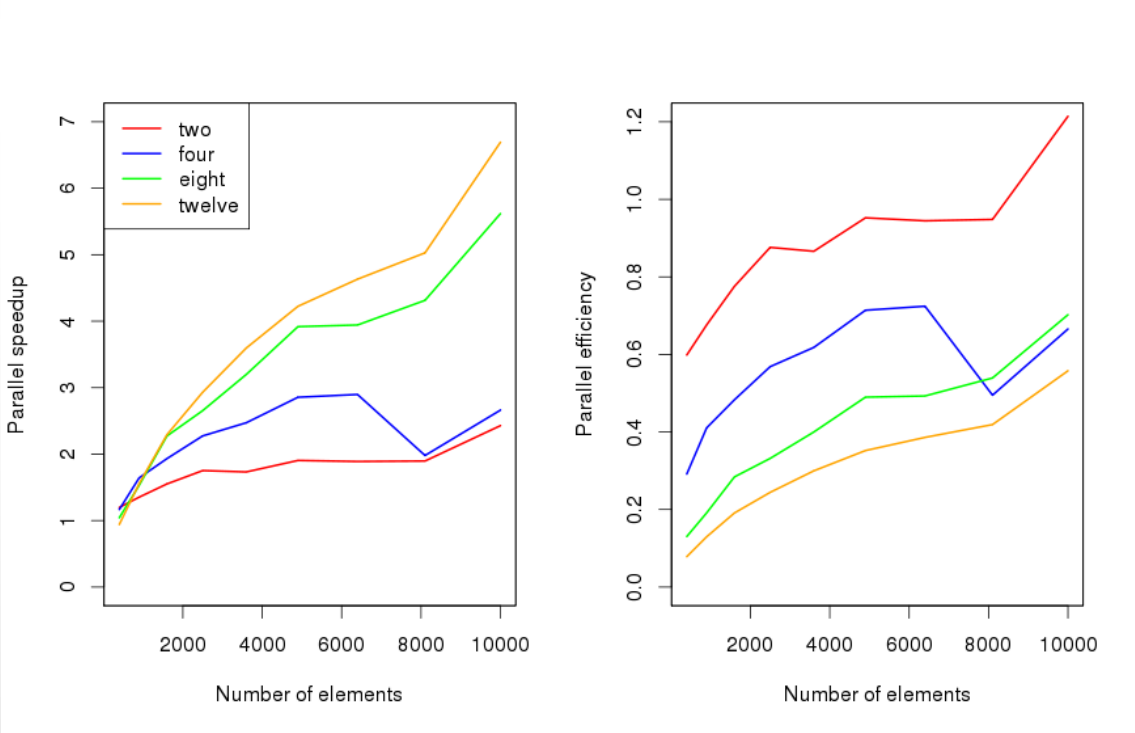
$$S = \frac{T_1}{T_p}$$

2. **Parallel efficiency**

$$E = \frac{1}{N_P}\frac{T_1}{T_p}$$

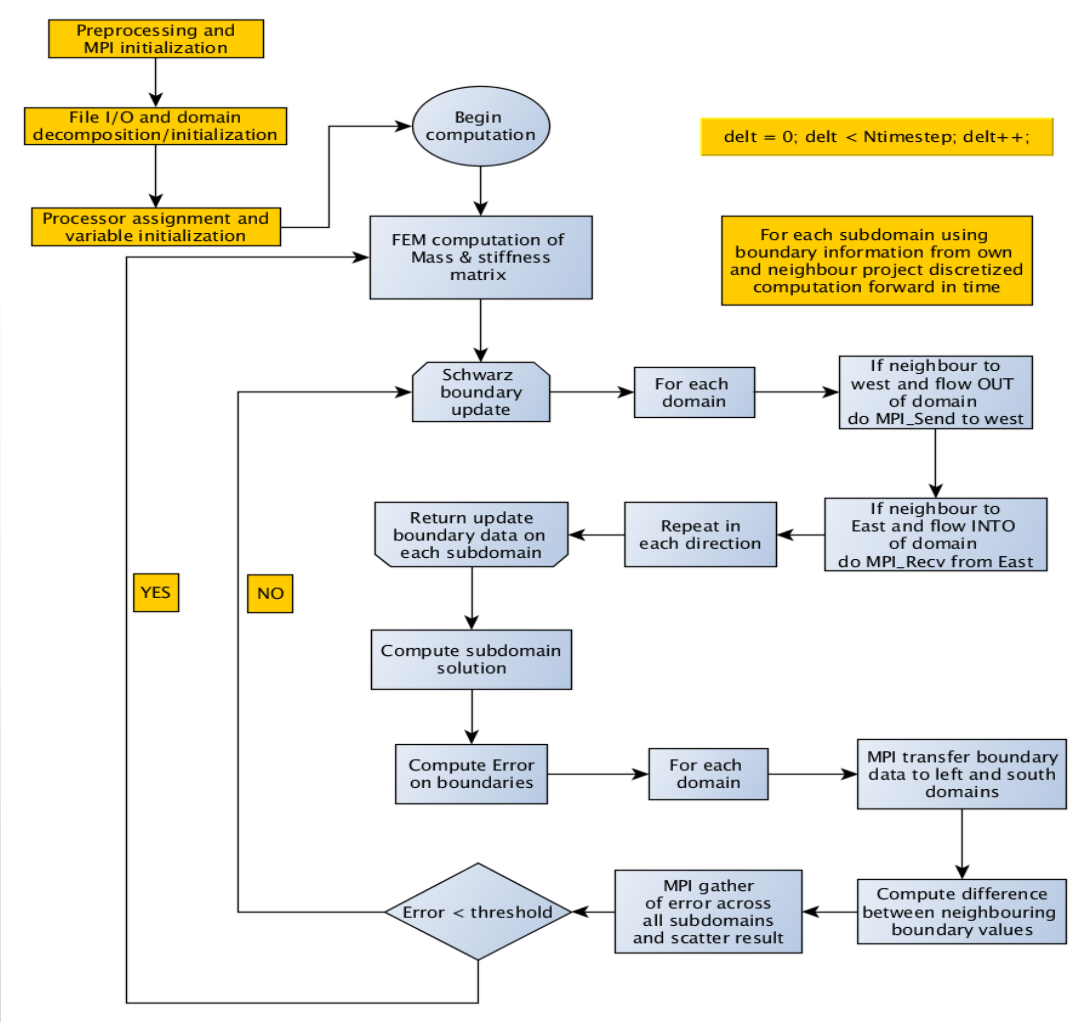
3. **At 12 OMP threads speedup is 6.7 corresponding to parallel efficiency of 55%**

4. **Suggests 2 OMP threads optimum configuration...**



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

Parallel structure



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

Parallel performance – Weak Scaling

1. **Experimental design: 40 x 40 element subdomain attached to each MPI domain**

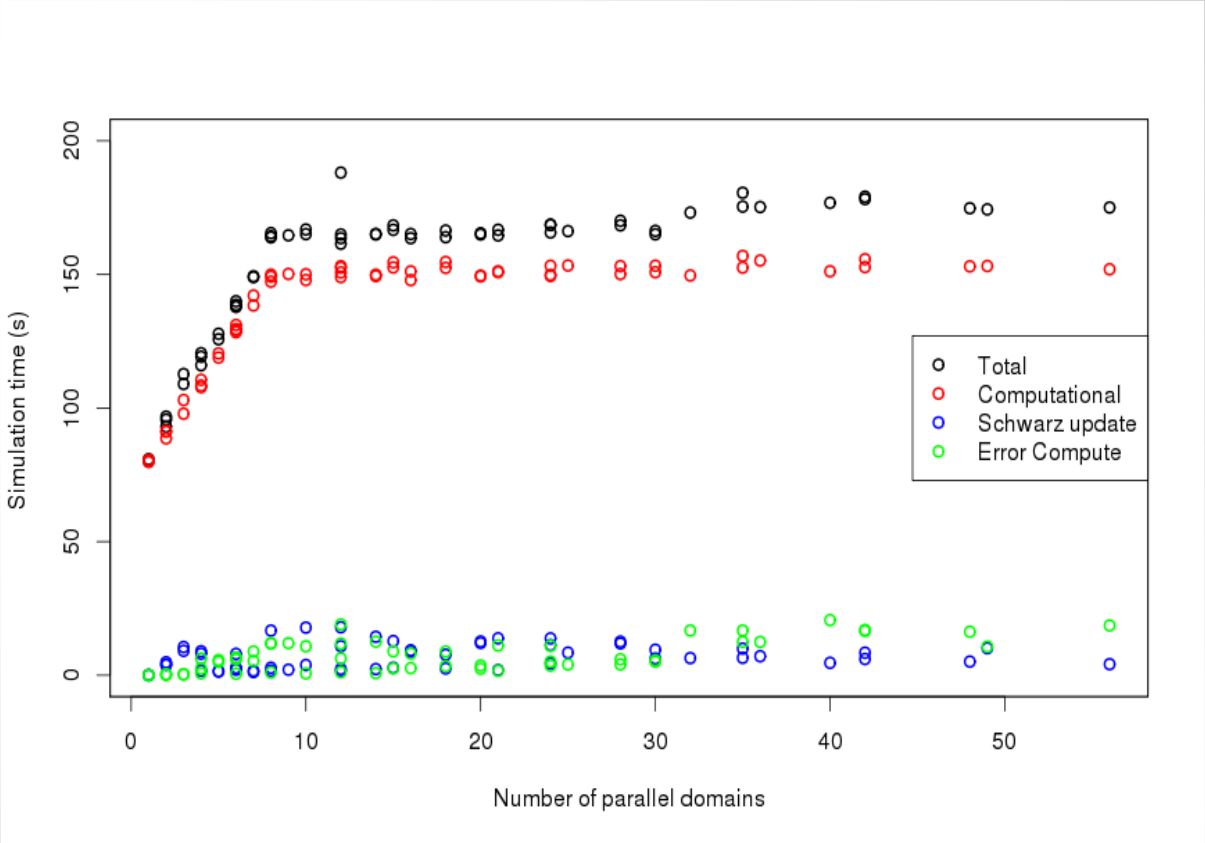
2. **Distributed across 7 nodes (24 cores) with max eight MPI processes (with two OMP threads) on each node**

3. **Problem size increases as number of MPI processes increases**

4. **Provides insight into parallel scalability of algorithm**

5. **Schwarz update routine (MPI_Send/Recv) and Error Compute (MPI_reduce) contains MPI functions**

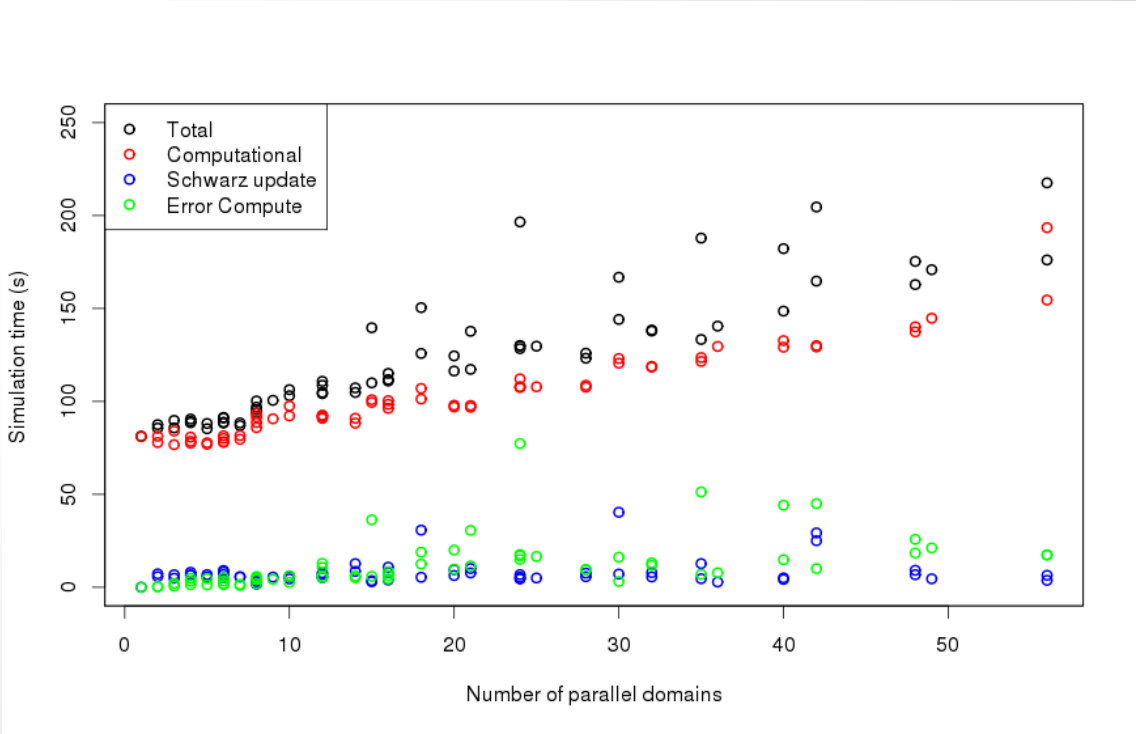
6. **Significant increase in simulation time up to ~10 MPI processes and then levels off**



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

Parallel performance – Weak Scaling – Distributed across cores equally

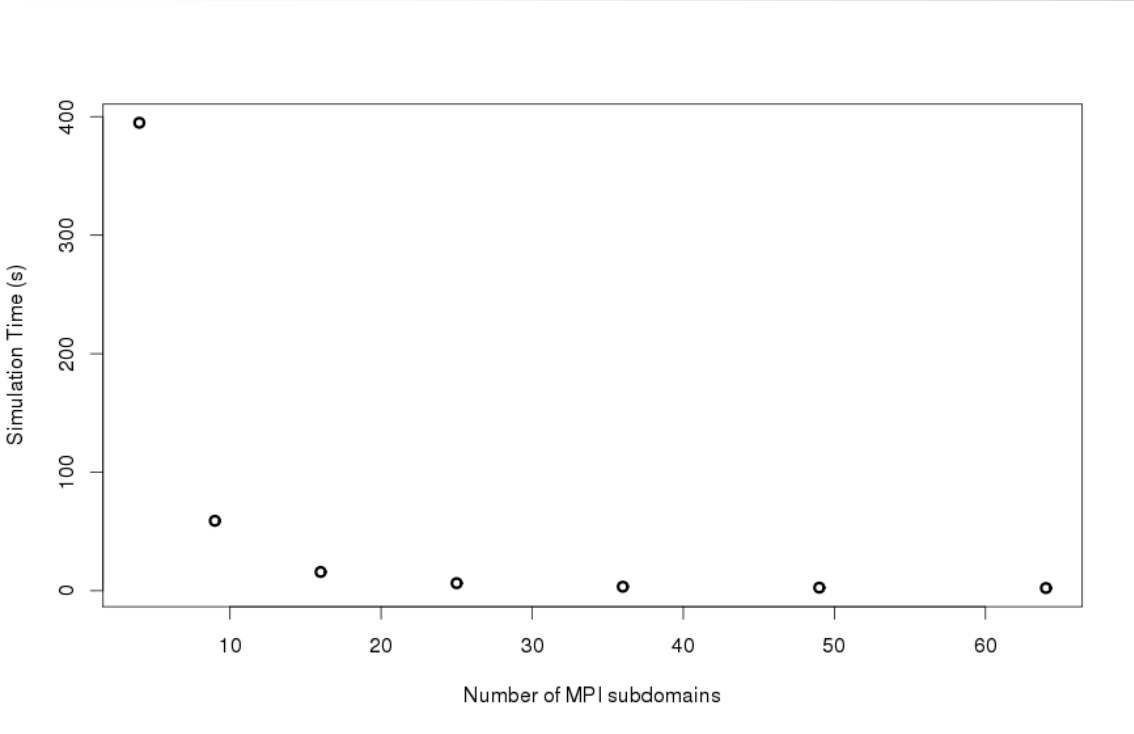
1. **Same experimental design as previous**
2. **Equal distribution of MPI processes across nodes (round robin), e.g. for 5 MPI processes one process assigned to each node**
3. **Simulation time flat up to ~ 10 MPI processes**
4. **A more linear increase in simulation time as MPI process number increases beyond this**
5. **As before significant increase in simulation time due to Computational component (not MPI)**
6. **Of MPI component, Error computation the most expensive (global reduction)**



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

Parallel performance – Strong Scaling

1. **Simulation time for fixed size problem**
2. **Provides insight into potential throughput of application**
3. **A 100 x 100 element global domain distributed across MPI domains**
4. **Rapid decrease in simulation time resulting from both mathematical implementation and work distribution**
5. **Levels off as computation attached to each core becomes too small**



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603

Conclusion and Future Work

- Domain decomposition as an approach to reduce computational time
- Parallel improvements from OpenMP paradigm in optimized blas library for linear algebra
- MPI parallelization within subdomains:
  - Efficient means to reduce simulation times
  - Error computation and domain synchronization requires global MPI call which is the most expensive parallel module
- As number of MPI processes per node increases computational time increases:
  - Multiple calls to linear algebra library
  - Data locality of finite element codes
  - Cache misses
  - expensive component is error computation due to global calls (same as conj grad methods)

- Future work
  - detailed analysis of computational expense of blas calls as number of processes increase
  - move to HPX and AllScale runtime

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671603