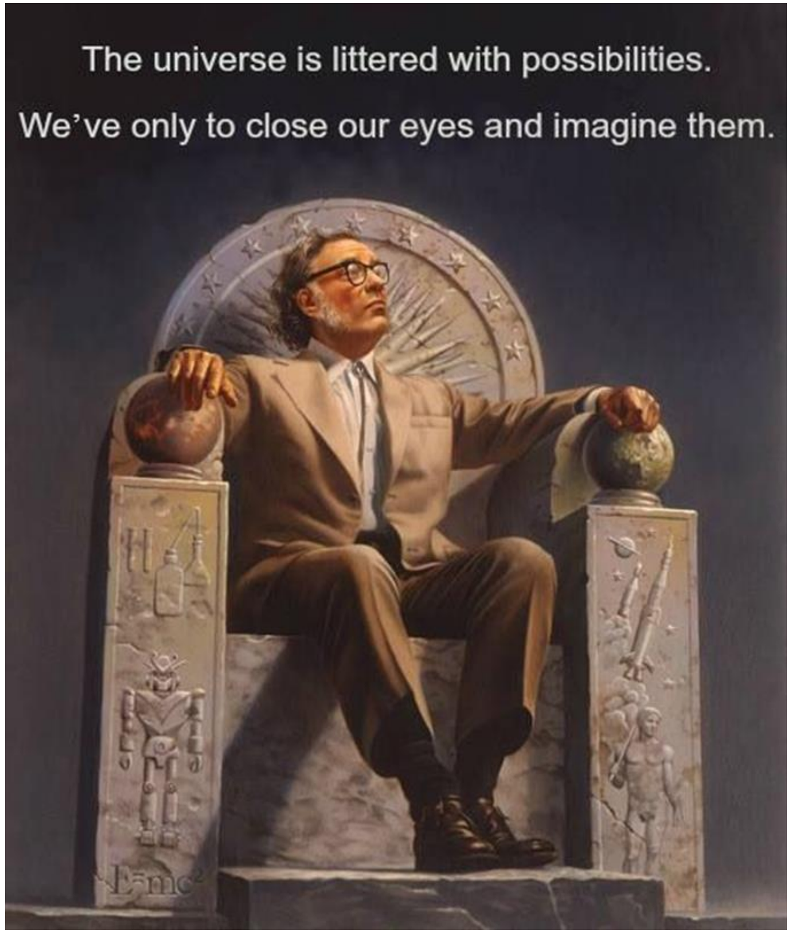
A study on the performance of reproducible computations

Nico Bombace, Michèle Weiland

Outline

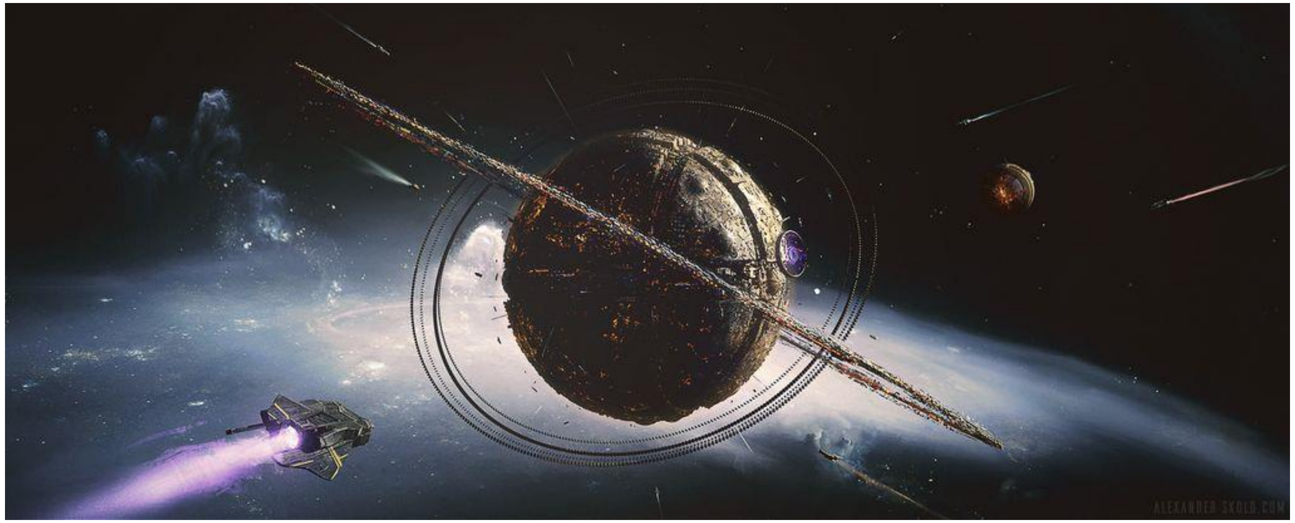
- Introduction
- The curse of Exascale
- Reproducibility
  - Software Verification
  - Functional Requirements
- Strategy
  - MiniFE
  - ReproBLAS
  - ReproFE
- Results
  - Performance
  - Reproducibility
- Conclusion and Further Work

Introduction


The universe is littered with possibilities.
We've only to close our eyes and imagine them.
E=mc²

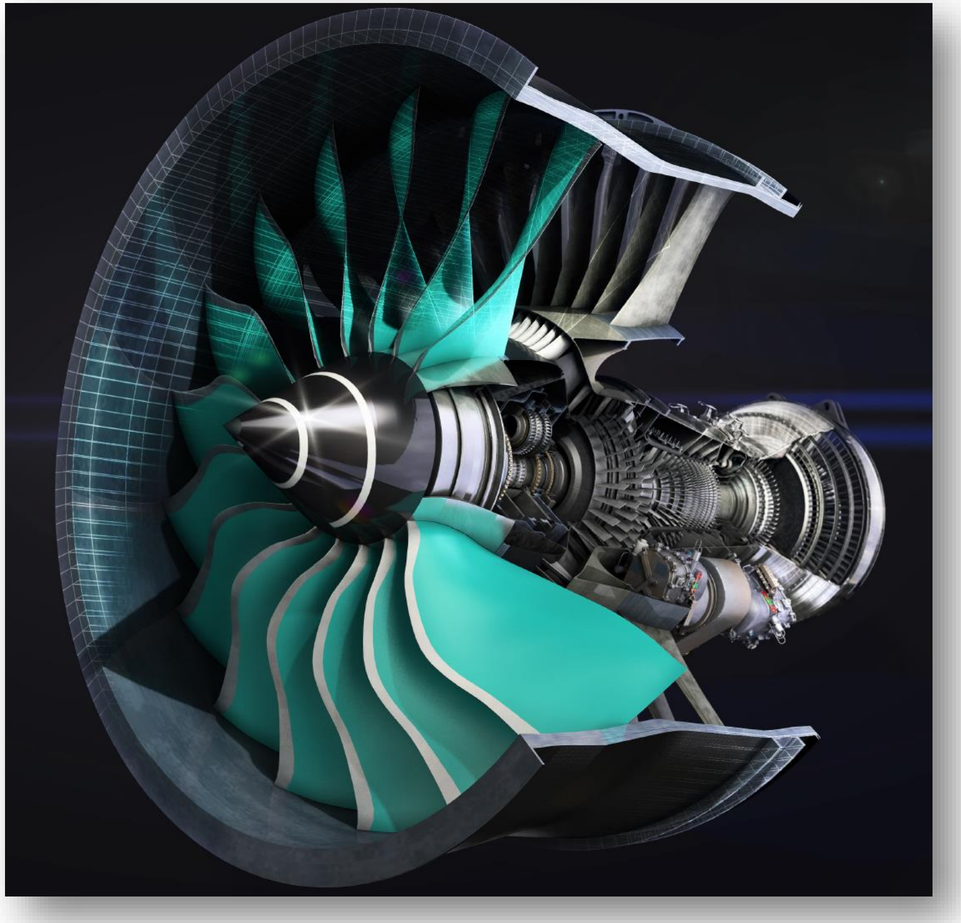
Advanced Simulation and Modelling of Virtual Systems (ASiMoV)

A convenient acronym



Introduction

- 5 year programme
- World’s first high-fidelity simulation of a gas turbine engine in operation
- Structure / Thermo dynamics / Fluid dynamics / Electromagnetics
- A trillion degrees of freedom
- An engineering challenge for the **Exascale era**
- Partners
  - Rolls Royce, Edinburgh, Warwick, Oxford, Cambridge, Bristol, Zenotech and CFMS

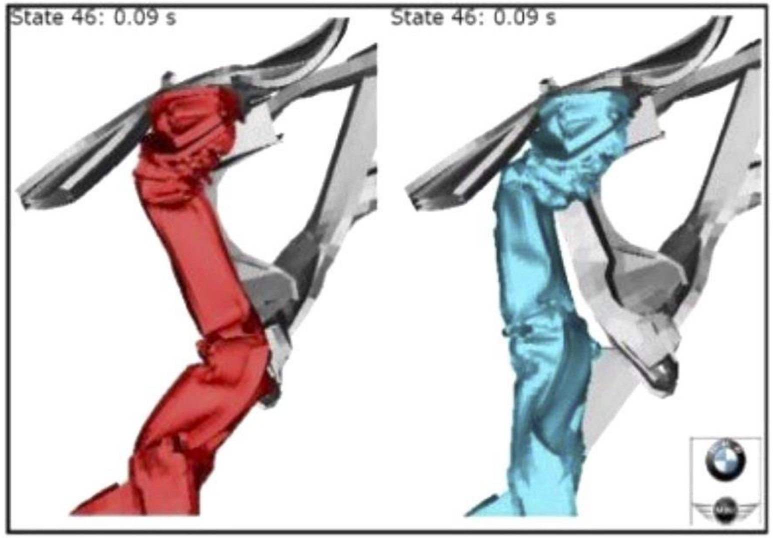


The curse of Exascale

- Parallel simulations with the same input, with the same conditions can produce different solutions.

Crashworthiness

Repeatability of Computations



[1]

Identical FEM models; identical software / hardware
and nevertheless largely differing computational results

[1] Reprinted from: Lenhard, Johannes, and Uwe Küster. "Reproducibility and the Concept of Numerical Solution." Minds and Machines (2019): 1-18.

Reproducibility

- Reproducibility: obtaining bit-wise identical results from different runs of the program on the same input data, regardless of different available resources[2].
- Cause of non-reproducibility: non associativity of floating point operations:

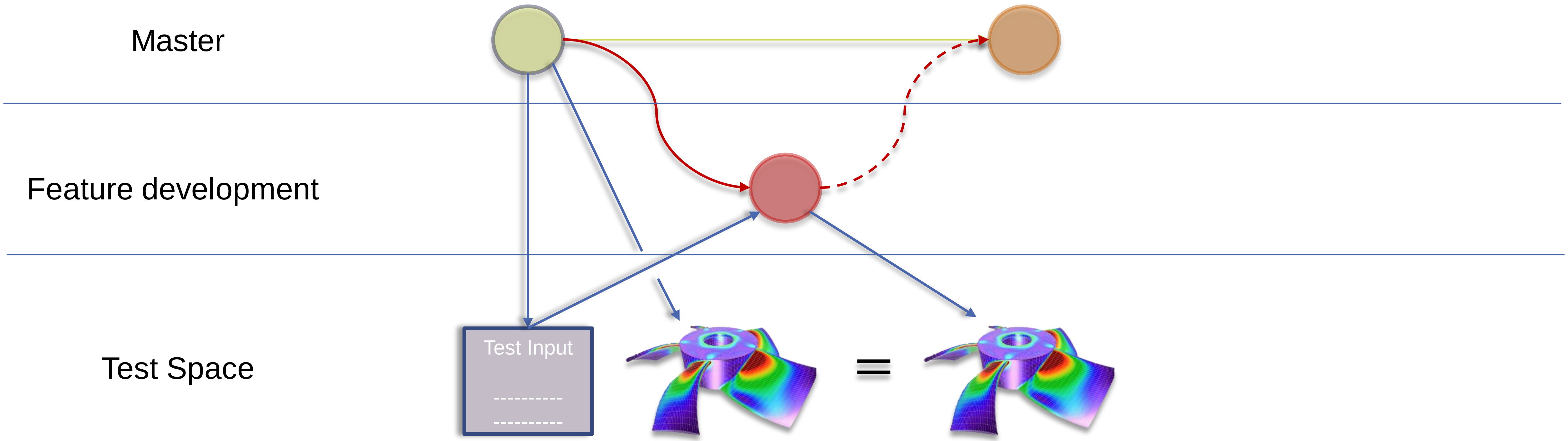
$$(a+b)+c \neq a+(b+c)$$

$$\begin{cases} a+\ b+d+e \neq a+b+c \\ d+e=c \end{cases}$$

[2] J. Demmel et al. : Reproducible BLAS (Basic Linear Algebra Subprograms). (Birds-of-a-Feather Session on: Reproducibility of High Performance Codes and Simulations: Tools, Techniques, Debugging) SC 2015, Austin, TX, Nov 15-20, 2015

Reproducibility in Software Verification

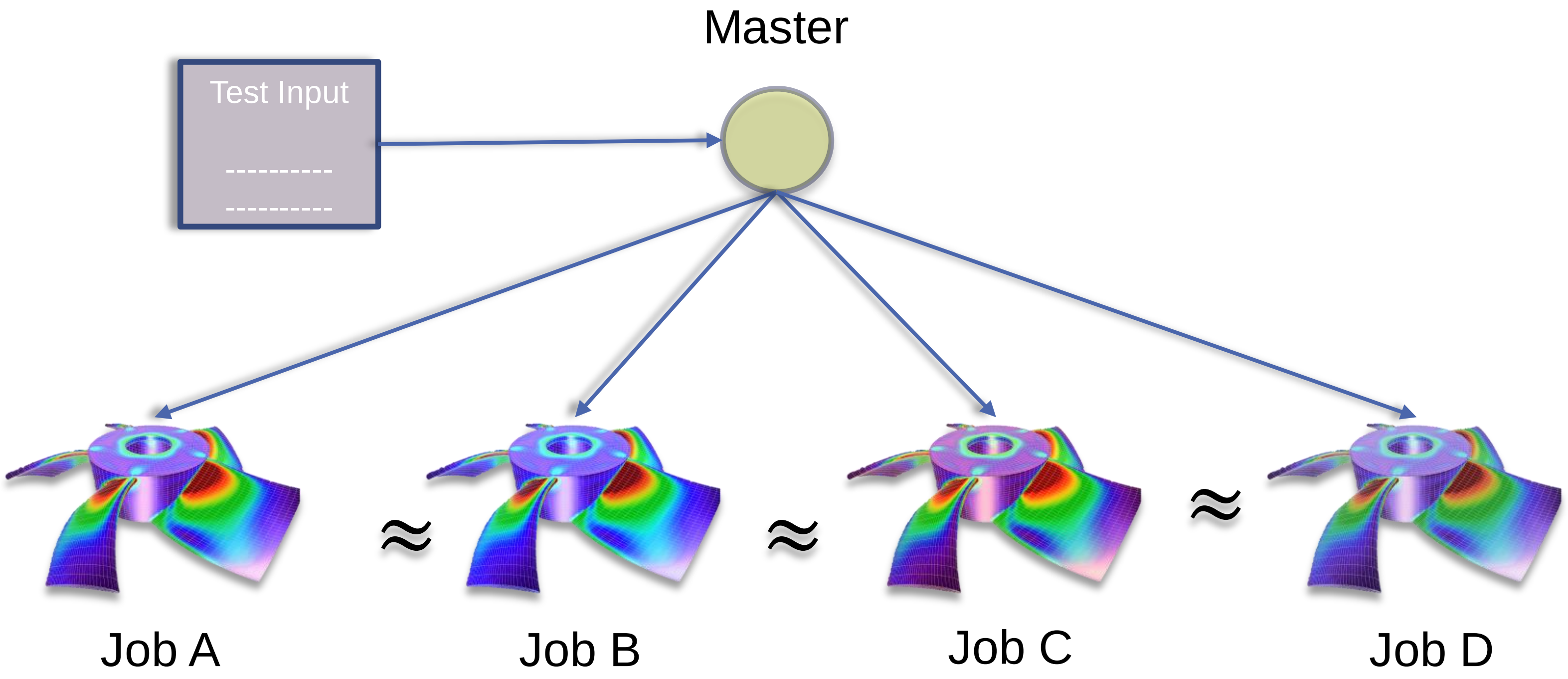
- The master branch contains input files and oracle solutions. These can be used to test new features before merging.



Reproducibility in Software Verification

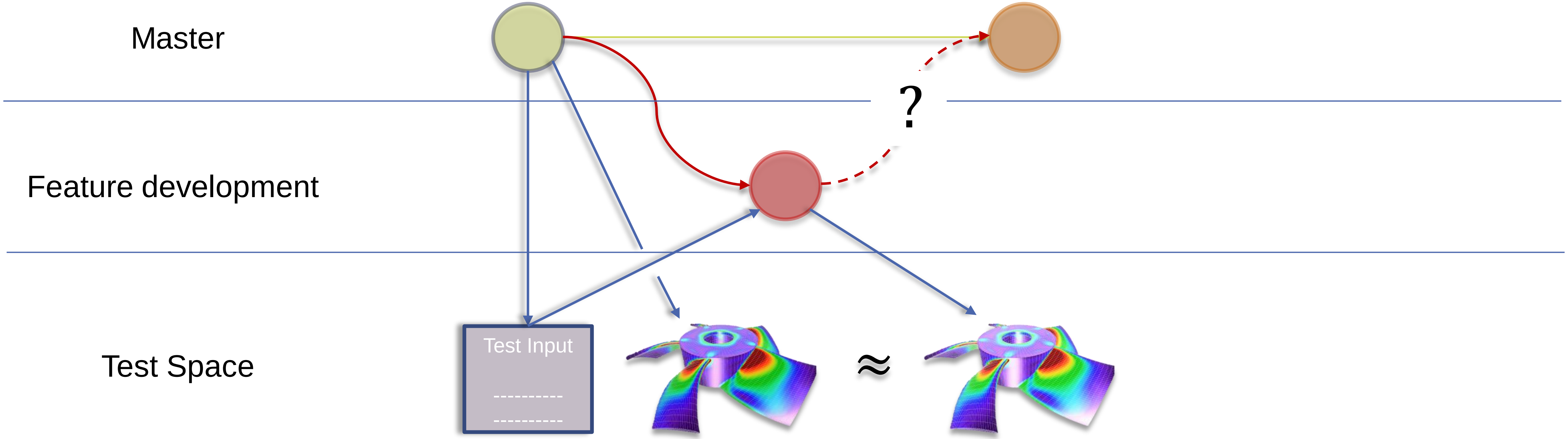
- However parallel simulations with the same input, can produce different solutions.

Conditions for every run	
	Number of Processes
Job A	40
Job B	50
Job C	100
Job D	500



Reproducibility in Software Verification

- How to understand if the oracle solution and the test solution differ because of the nature of parallel simulations or because of newly introduced bug?



Reproducibility: functional requirements

- Provide a pattern to test parallel libraries.
- Non intrusive.
- Minimise developer/user effort.
- Turn on reproducibility without major changes to source code.

C++ + ReproBLAS

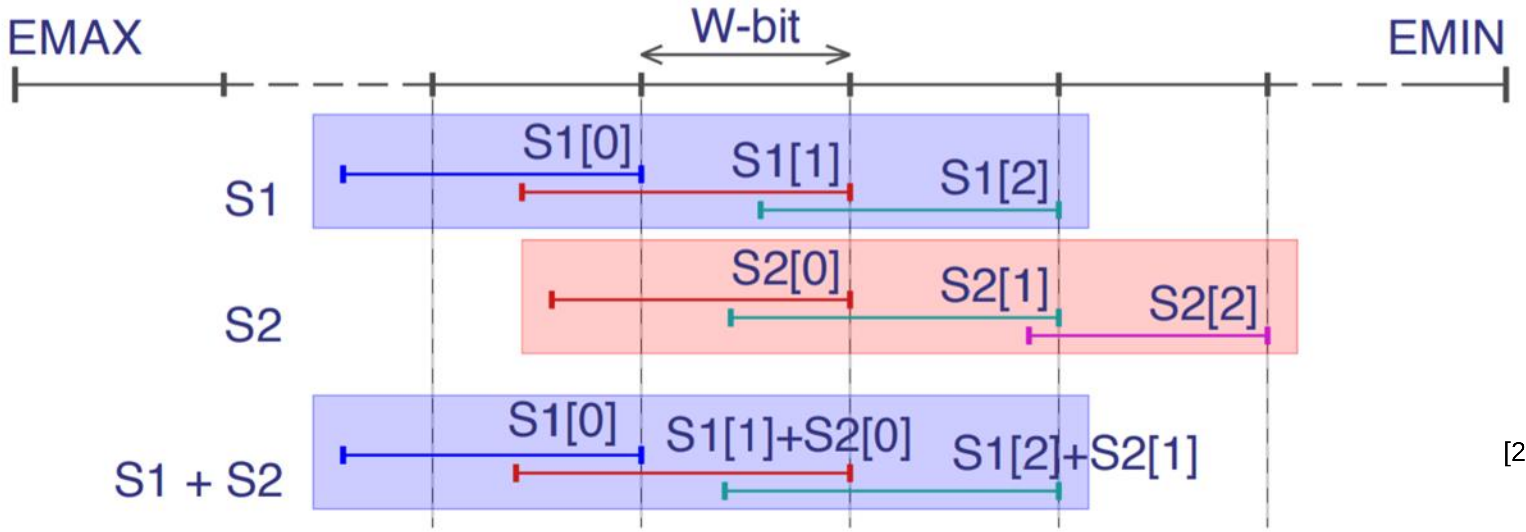
MiniFE

- Mini application that implements implicit finite elements.
- Support of different numerical types.
- Example generic metaprogramming.

```
template<  typename Scalar,
           typename LocalOrdinal,
           typename GlobalOrdinal>
struct
Vector{
    ...
    typedef Scalar        ScalarType;
    typedef LocalOrdinal  LocalOrdinalType;
    typedef GlobalOrdinal GlobalOrdinalType;
    ...
}
```

ReproBLAS: binned doubles

- The doubles are effectively split in 3 (or more bins) and then summed separately.



[2]

[2] **J. Demmel and H. D. Nguyen,** *Parallel Reproducible Summation,* IEEE Transactions on Computer, v.64, i. 7, July 2015. DOI: 10.1109/TC.2014.2345391

The reproducible namespace

- Wrap functionalities of ReproBLAS in new class.
- Respect encapsulation.
- Overload Arithmetic operators.

The reproducible namespace

```
namespace reproducible {
template <int K>
    class Double{
        double _d;
        std::array<double, 2*K> _binned;
    public:
       template <int K>
       reproducible::Double<K>::Double(double d): _d(d) {
          binned_dbsetzero(K, _binned.data());
          binned_dbdconv(K, _d, _binned.data());
       }
    }
}
```

Changes to MiniFE

- Changes to pragma directives to support operator overloading

```
#pragma omp parallel for reduction(+:result)
  for (int i=0; i<n; ++i) {
    result += xcoefs[i] * ycoefs[i];
}
```

```
#pragma omp parallel
{
  MINIFE_SCALAR result_private = 0;
  #pragma omp for nowait
  for (int i=0; i<n; ++i) {
    result_private += xcoefs[i] * ycoefs[i];
  }
  #pragma omp critical
  {
    result += result_private;
  }
```

Changes to MiniFE

- Changes to pragma directives to support operator overloading

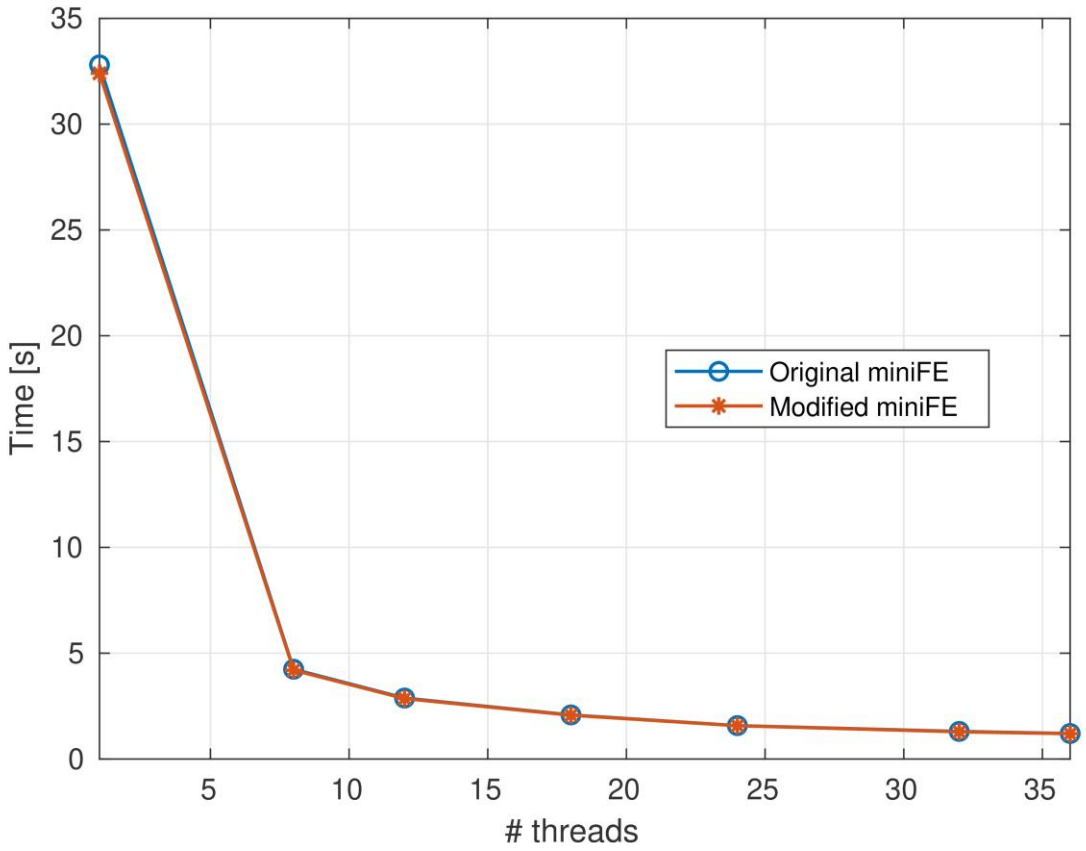
```
#pragma omp parallel for reduction(+:result)
   for (int  i=0;  i<n;  ++i)  {
      result  +=  xcoefs[i]  *  ycoefs[i];
}
```

```
#pragma omp parallel
{
   MINIFE_SCALAR result_private = 0;
   #pragma omp for nowait
   for (int  i=0;  i<n;  ++i)  {
      result_private += xcoefs[i]  *  ycoefs[i];
   }
   #pragma omp critical
   {
      result  +=  result_private;
   }
```

Changes to MiniFE

- How did these changes affect performance?

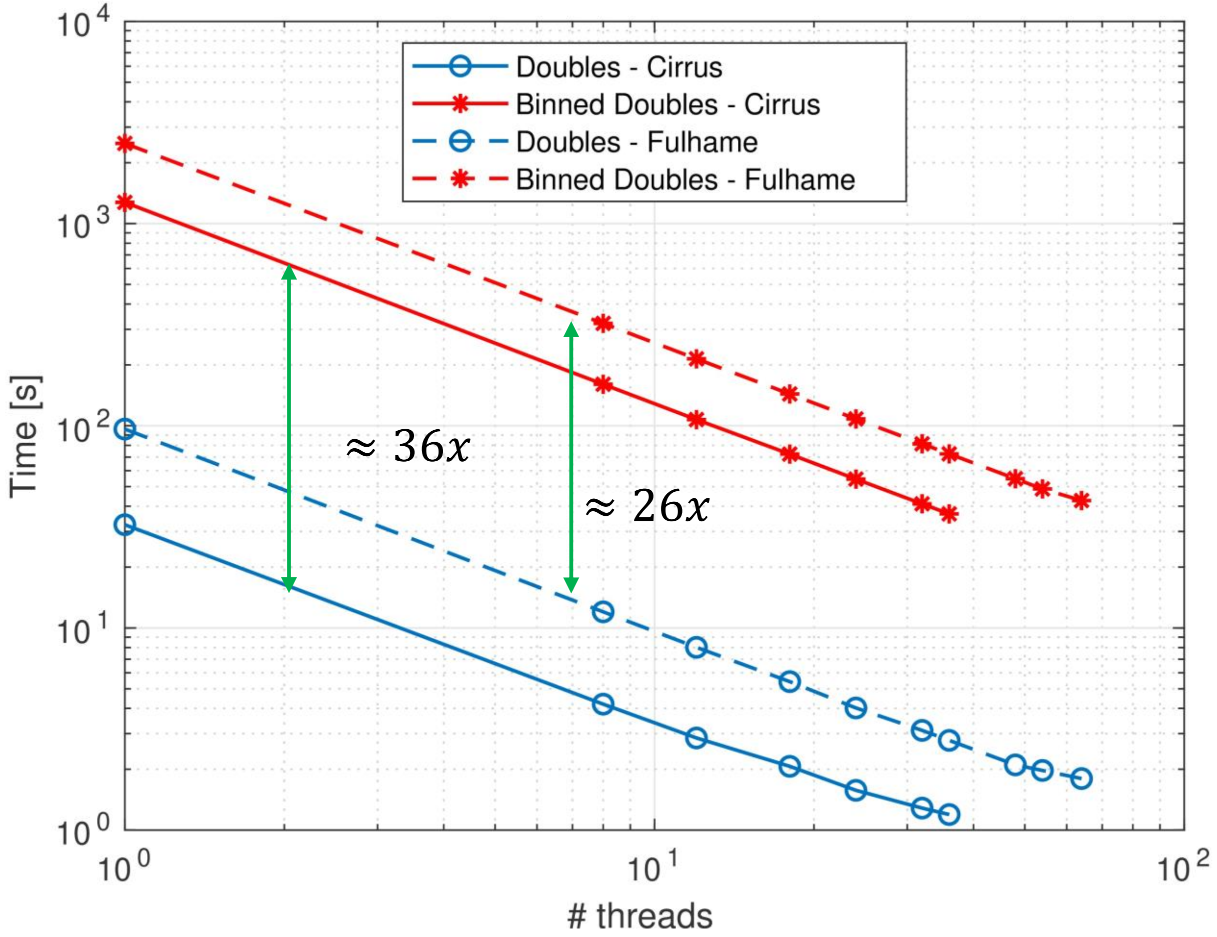
- 108x108x108 hex elements
- Two 2.1 GHz, 18-core Intel Xeon (Broadwell) processors
- GCC 6.3.0
  - O0 -novec
- 5 runs



Putting it all together: ReproFE

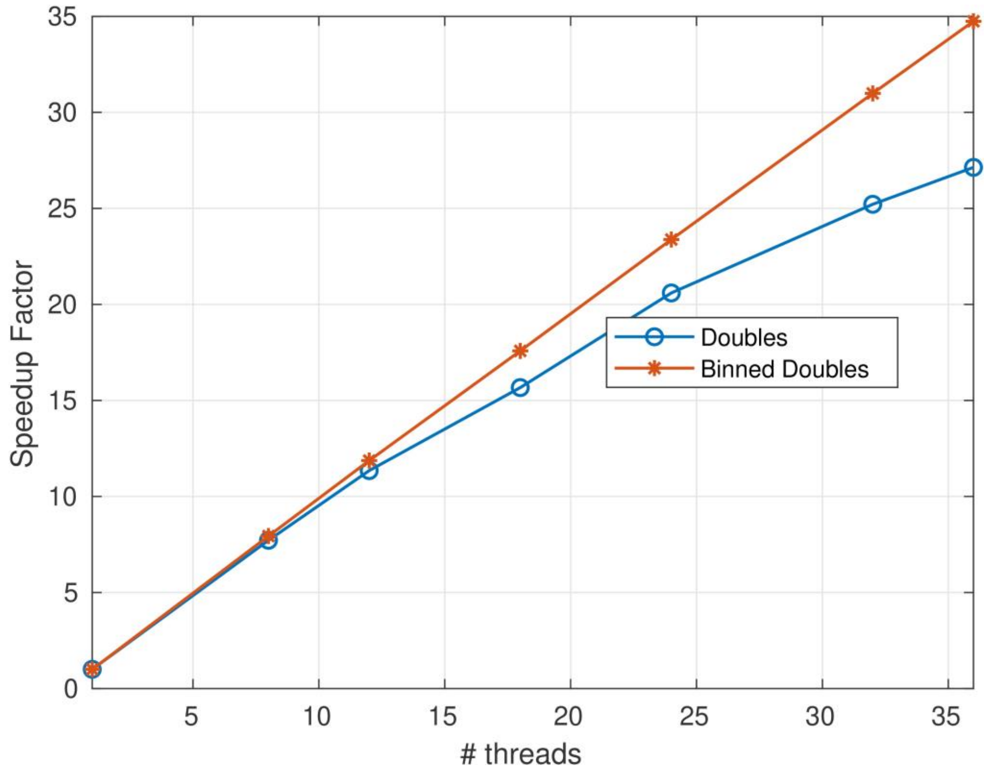
- Compile MiniFE using reproducible::Double<3>.
  - **Only change one parameter**
- Measure performance on two architectures
  - Two 2.1 GHz, 18-core Intel Xeon (Broadwell) processors. Cirrus
  - Arm-based HPE Apollo70 system, with two 32-core Marvell ThunderX2 processors. Fulhame
- GCC 6.3.0
    - O0 -novec
- 5 runs

ReproFE: performance

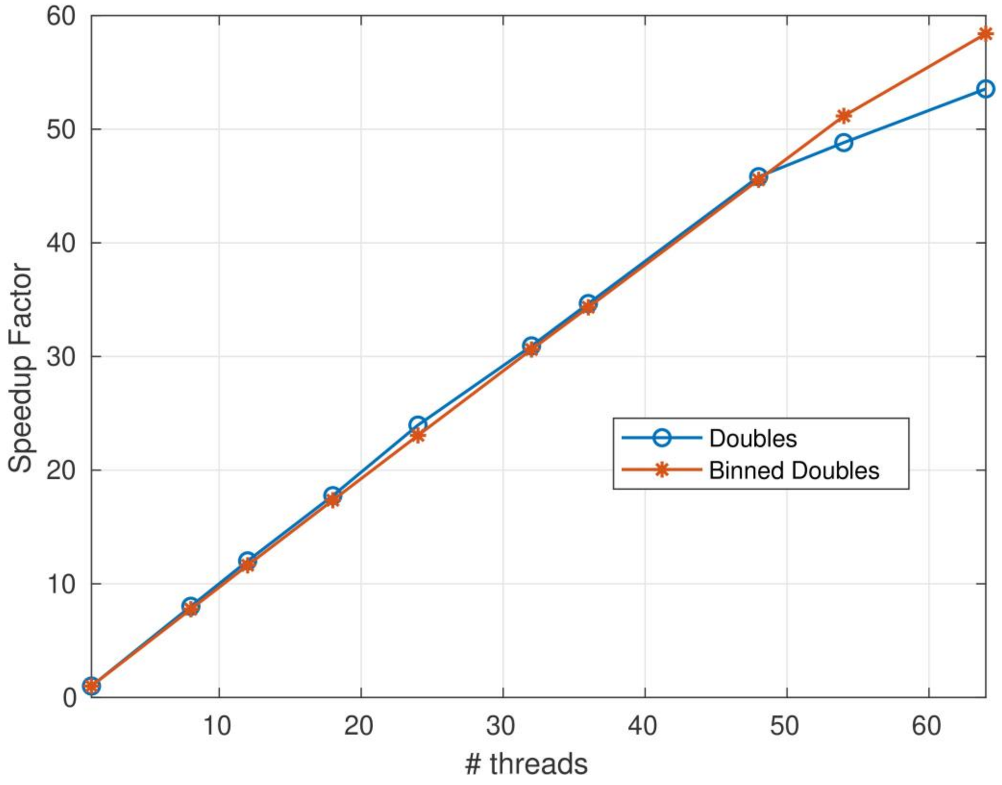


ReproFE: performance

Cirrus



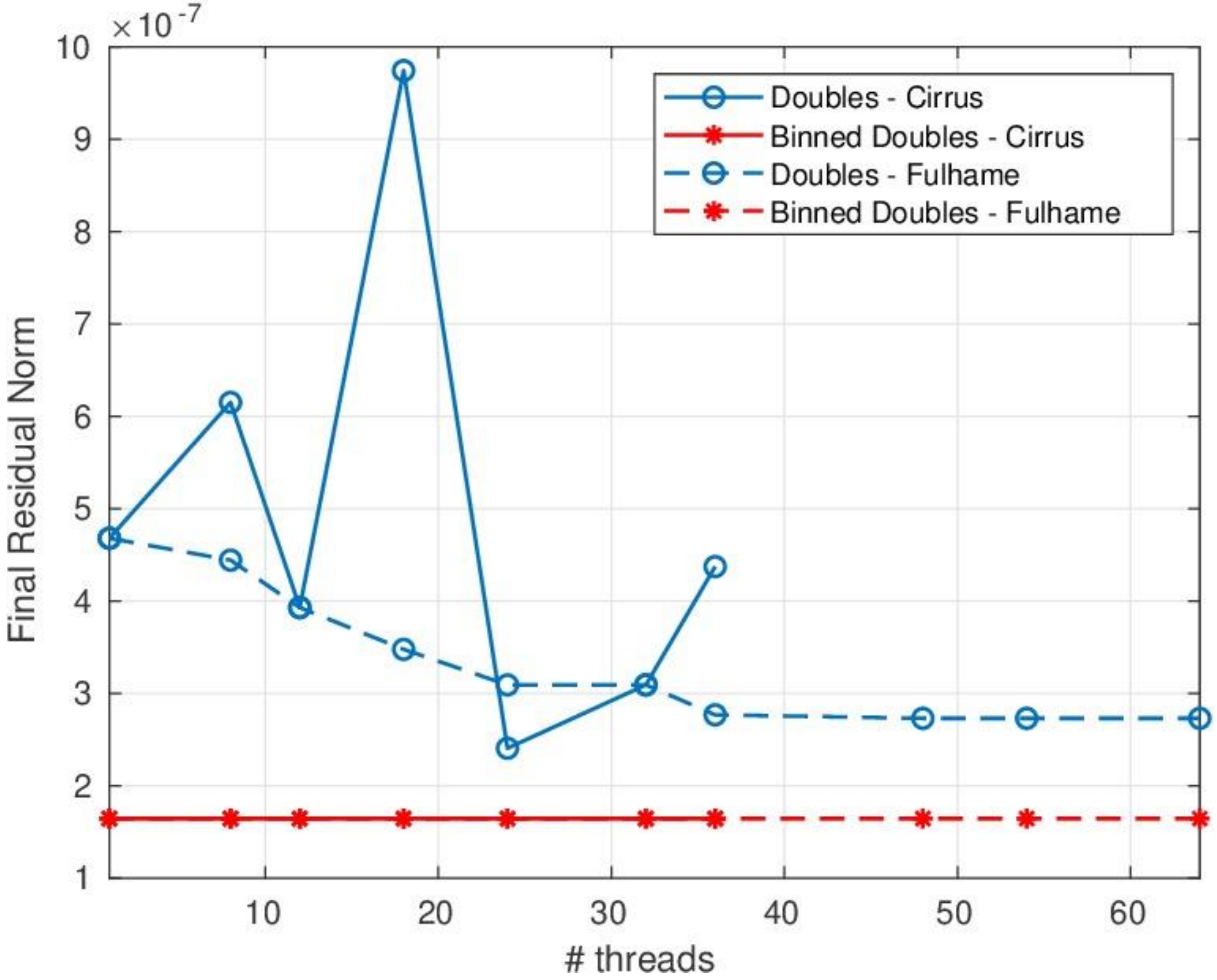
Fulhame



ReproFE: reproducibility

- Standard doubles cannot achieve reproducible results

- Reproducibility achieved with binned doubles



Conclusions and Further work

- We proposed a technique to achieve reproducible computations regardless of number of processes and/or architecture with minimum effort.
- We experienced a performance hit. However, in testing and certification code correctness is paramount.

Further work include:

- Trans-precision computing with binned doubles.
- Investigation on heterogenous parallel models (e.g. OpenMP + MPI)