OPTIMIZING AN RBF INTERPOLATION SOLVER

An ARM Porting Story

Patrick Schiffmann

AVL List GmbH (Headquarters)

OUTLINE

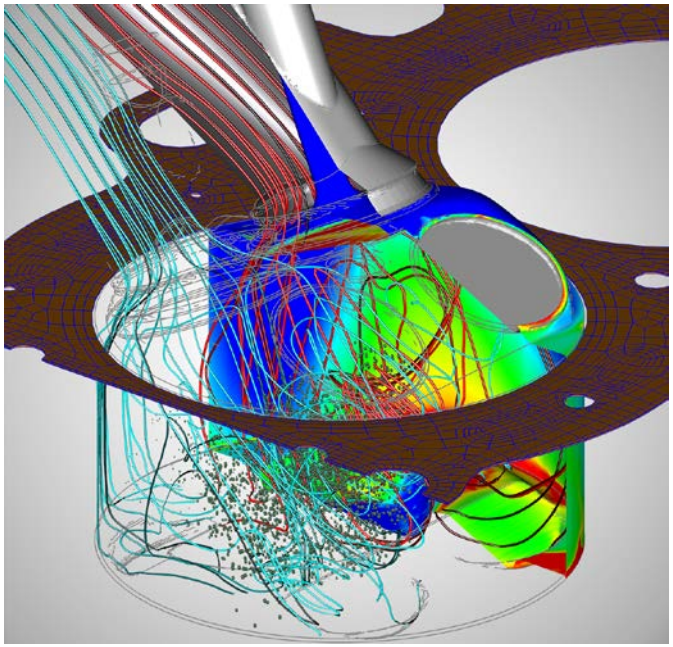
1. Motivation & Background
2. Porting to ARM
3. Baseline Comparison
4. Optimizations
5. Lessons learned

AVL

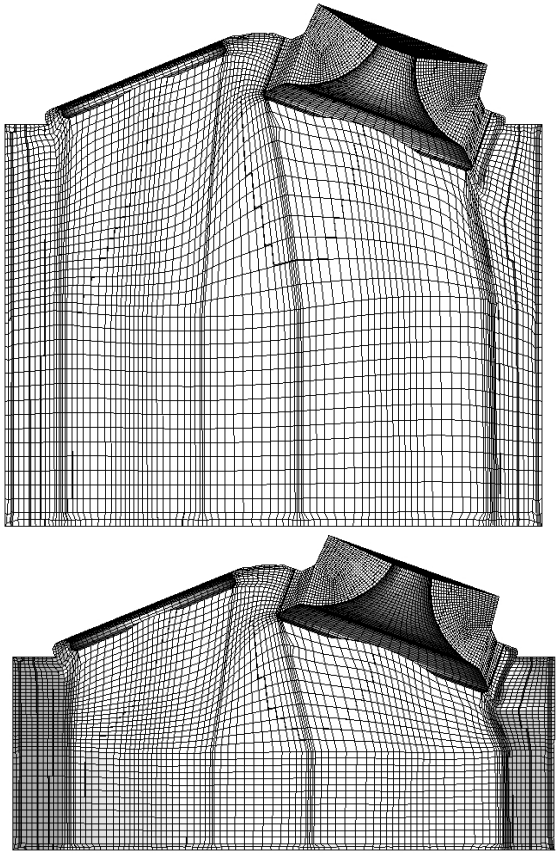
- World's largest independent company for development, simulation and testing technology of powertrains

- 8600 employees, 65% scientists and engineers
- About 1.5bn € revenue, 10% R&D

- **A**dvanced **S**imulation **T**echnologies
  - Development of new simulation methods
  - HPC mostly relevant for Fluid Dynamics



MOTIVATION



MESHES IN COMPUTATIONAL FLUID DYNAMICS (CFD)

- In our application, meshes are
  - unstructured
  - O(1M) – O(100M) cells
- When geometry changes, mesh must adapt
  - meshing is expensive
  - frequent interpolation combined with rare re-meshing is preferred
- Mesh quality important for
  - Quality of results
  - Convergence speed

MATHEMATICAL FORMULATION

General approximation:

- set of points $\mathcal{X} = \{x_i\}_{i=1}^{N}$ is given
- function values $f_i = f(x_i)$ are given (f unknown)
- search for an approximating function s : $\quad s|_{\mathcal{X}} = f|_{\mathcal{X}}$.

In the context of RBF interpolation we seek for an interpoland of the form

$$s(x) = \sum_{i=1}^{N} \lambda_i \phi(\|x_i - x\|) + p(x), \quad \lambda_i \in \mathbb{R}, p \in \mathbb{P}^M. \tag{1}$$

Polynomial term p is required for the existence and uniqueness of a solution.

MATHEMATICAL FORMULATION

Requiring the interpolation condition $s|_{\mathcal{X}} = f|_{\mathcal{X}}$ in all given points and the unisolvency of the set $\mathcal{X}$ for $\mathbb{P}_d^M$, thus $p|_{\mathcal{X}} = 0 \Rightarrow p \equiv 0$ and demanding a side condition on the coefficients of the polynomial term leads to a system of linear equations for the determination of the coefficients $\underline{\lambda}$ and $\underline{\pi}$:

$$\sum_{i=1}^{N} \lambda_i \phi(\|x_i - x_k\|) + \sum_{j=1}^{M} \pi_j p_j(x_k) = f(x_k), \qquad 1 \leq k \leq N,$$

$$\sum_{i=1}^{N} \lambda_i p_l(x_i) = 0, \qquad 1 \leq l \leq M, \qquad (2)$$

or, in short notation

$$\begin{pmatrix} \Phi & \Pi \\ \Pi^\top & 0 \end{pmatrix} \begin{pmatrix} \underline{\lambda} \\ \underline{\pi} \end{pmatrix} = \begin{pmatrix} \underline{f} \\ \underline{0} \end{pmatrix}. \qquad (3)$$

Solving (3) provides all information to evaluate the RBF approximate $s(x)$.

PROBLEM SIZE

- Dense linear system in N variables
  - N = number of points which define the deformation

- Direct Solution
  - Complexity O(N^3)
  - System Matrix 8TB Memory for 1M points
  - Possible on supercomputers, but we target
    - Work stations
    - Production jobs up to 1000 cores
    - Mont-Blanc 3 Prototype

- By exploiting the properties of the problem
  - Reduce memory and runtime
  - Trade scalability and parallel efficiency

FGP ALGORITHM

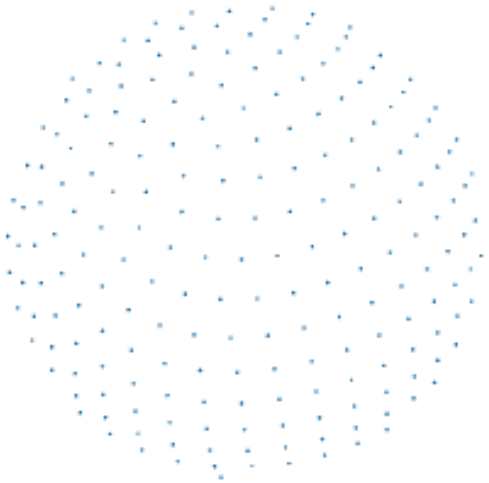
RBF: $\sqrt{r^2 + c^2}$ (multiquadric biharmonics) and constant polynomial terms.
The system , $i = 1, \ldots, n$, $j = 1, \ldots, M$

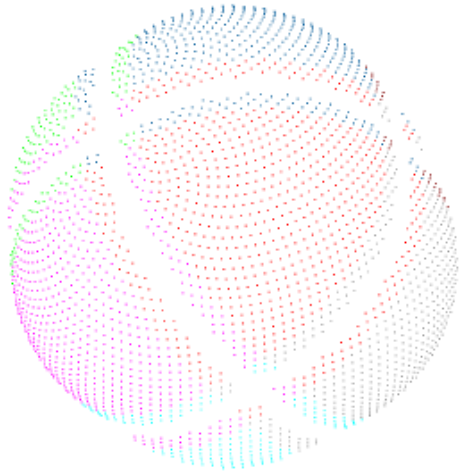
$$\begin{pmatrix} \Phi & \Pi \\ \Pi^\top & 0 \end{pmatrix} \begin{pmatrix} \underline{\lambda} \\ \underline{\pi} \end{pmatrix} = \begin{pmatrix} \underline{f} \\ \underline{0} \end{pmatrix} ; \qquad \Phi_{ij} = \Phi(x_i - x_j) , \quad P_{ij} = P_j(x_i)$$

is solved via FGP algorithm, a special Krylov subspace algorithm for RBF [Faul/Goodsell/Powell'05]:

- no matrix is stored
- operation Matrix$*$Vector directly implemented
  - Brute force: direct implementation of $\Phi * \underline{\lambda}$ $\mathcal{O}(N^2)$
  - Multipole approx. $\widetilde{\Phi}$ of Φ is used. $\mathcal{O}(N \log N)$
- Krylov operator construction for FGP
  - appropriates for our RBF with constant polynomial
  - approximates Φ^{-1} by 51 entries per row.
    Octree is used for neighborhood relations: $\mathcal{O}(N \log N)$
- $<$ 30 iterations to solve the system

ADDING TWO LEVEL DOMAIN DECOMPOSITION





```
while not converged do
    for each level in MultiLevel-DD do                    MPI
        for each box in DD[level] do
            if points in box < threshold then
                solveDirectly(box)
            else
                solveFMM(box)                             OMP
            end
        end
        applyResultsToNextLevel()
    end
    calcResidual()
end
```

PORTING TO ARM I

- Starting point
  - AVL joined Mont-Blanc 3 project, I joined AVL
  - Code was 15k LOC C++
    - Designed for doing a math PhD, not HPC production use
    - Only ever built / ran using Intel tools on Intel CPUs
    - Depending on internal library
      - Only available for x86 due to other dependencies
    - Sharing of code open question

- Preparation
  - Build with GCC on x86, try to match performance
  - Remove dependencies for which code to build on ARM is unavailable
  - Hopefully, most would start here

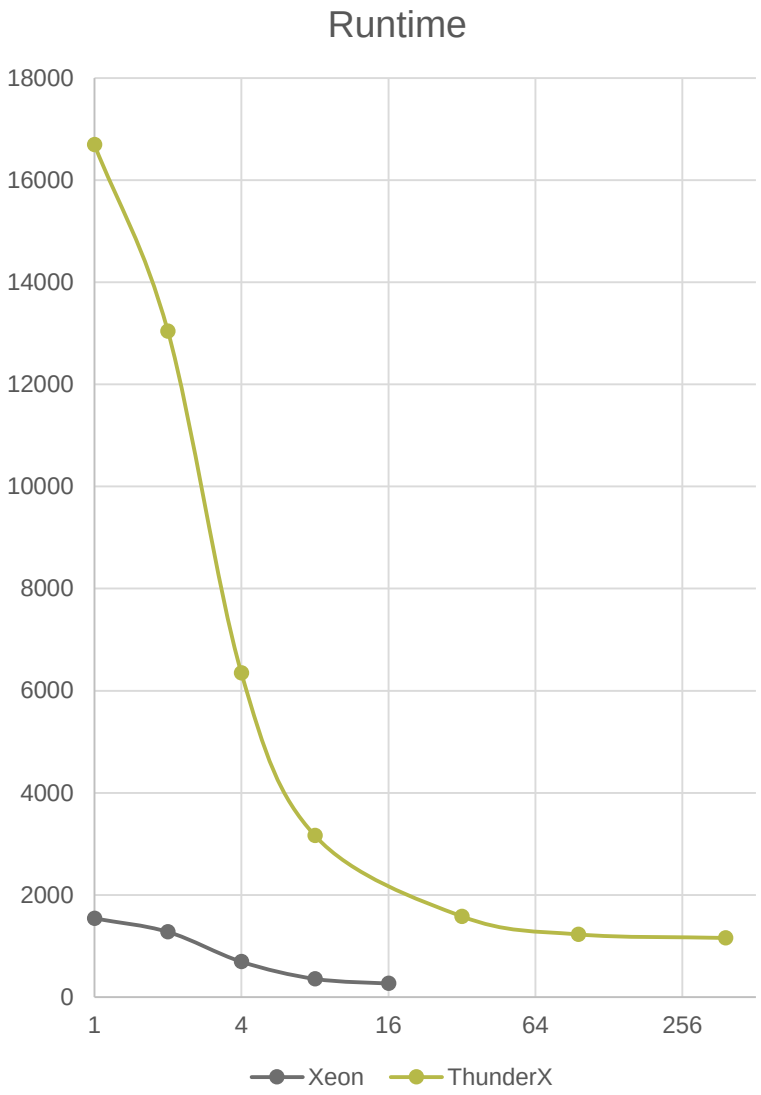
PORTING TO ARM II

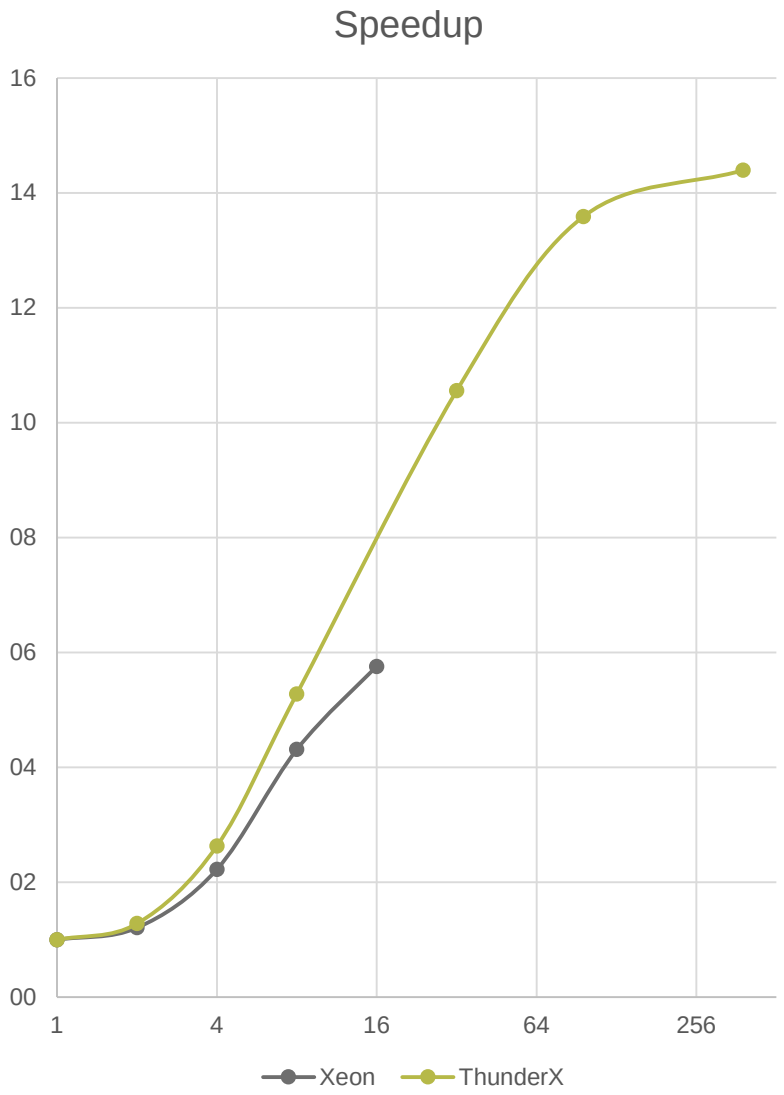
- Mont-Blanc mini clusters at Barcelona Supercomputing Center
  - 4x ThunderX (2 Socket, 48 Cores each)
  - 16x Jetson-Tx (4 Cores)

- First compilation via emulated Ubuntu VM inhouse
  - Works ~50x slower than native
  - Avoid if possible, but can be helpful

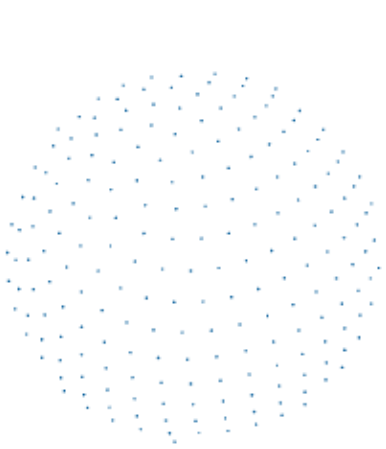
- Issues
  - Lack of documentation
    - Good personal support and trial & error helped
  - Some problems with
    - Length of data types, struct packing, BLAS/LAPACK includes and naming conventions, …
  - Porting made code quality issues visible

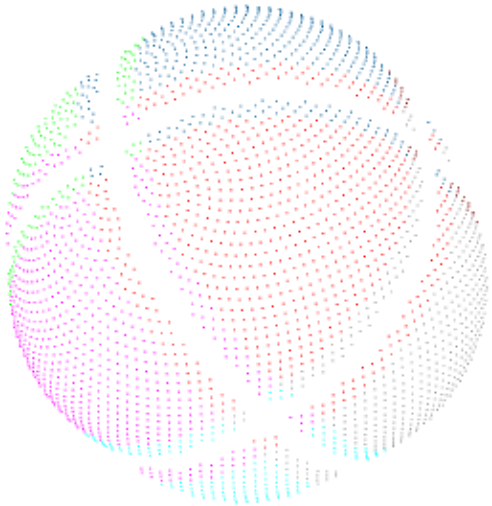
FIRST RESULTS

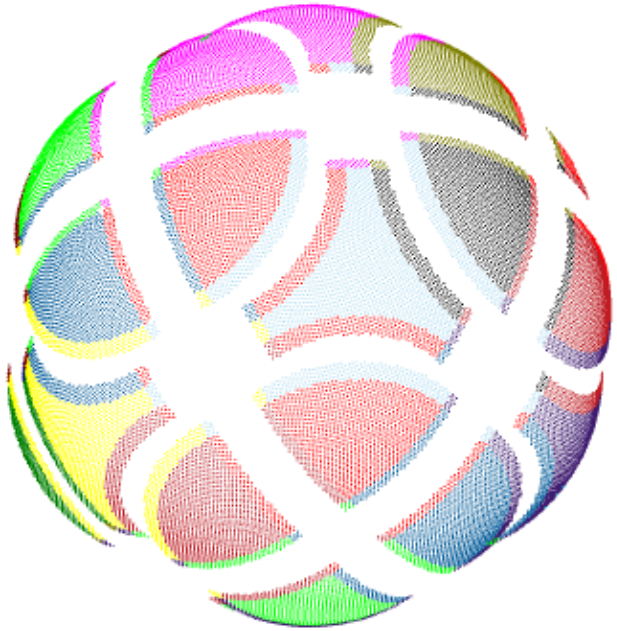




MULTI LEVEL DOMAIN DECOMPOSITION WITH HALOS







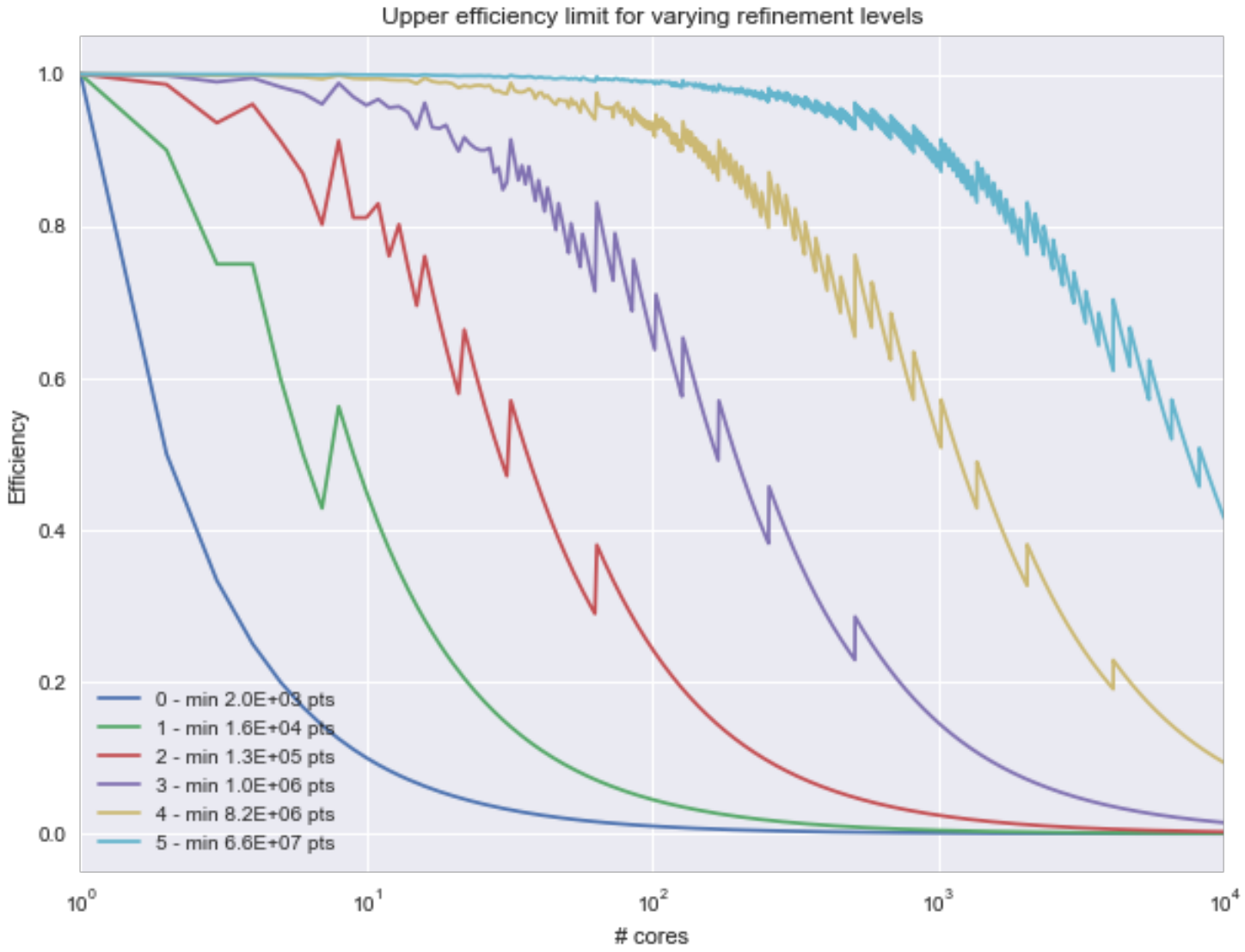
$$X_\ell \subset X_{\ell+1}$$

HYBRID ALGORITHM

```
while not converged do
    for each level in MultiLevel-DD do
        for each box in DD[level] do          MPI+OMP
            if points in box < threshold then
                solveDirectly(box)
            else                              OpenMP
                solveFMM(box)
            end
        end
        applyResultsToNextLevel()
    end
    calcResidual()
end
```

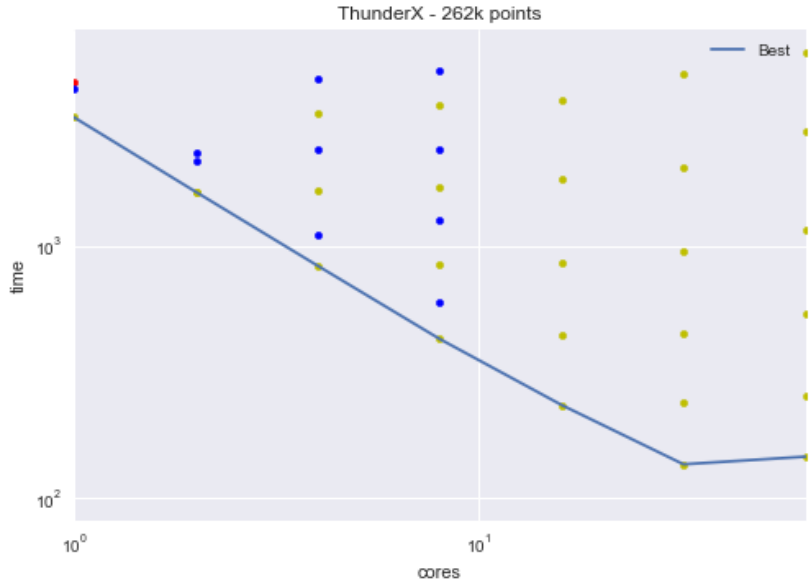
- Inherent load balance issues
  - In level N there are 8^N boxes
  - Boxes need at least ~2k points for algorithm to converge fast
  - Final level(s) need more boxes than MPI processes

THEORETICAL PERFORMANCE LIMITS



HYBRID CODE

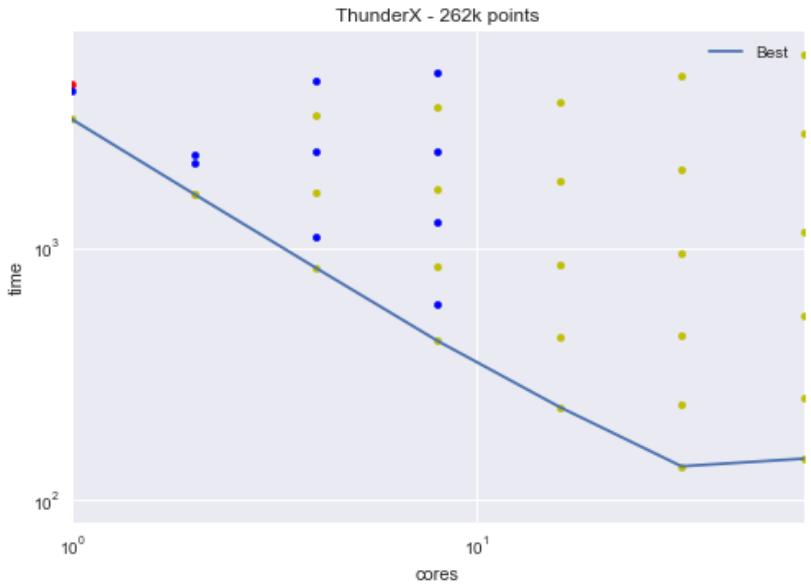
- Sweep over process/thread space for all refinement levels
- Better scaling on both platforms
- Low variability on Xeon, pure threading fastest
- Very high var. on ThunderX, one process per socket

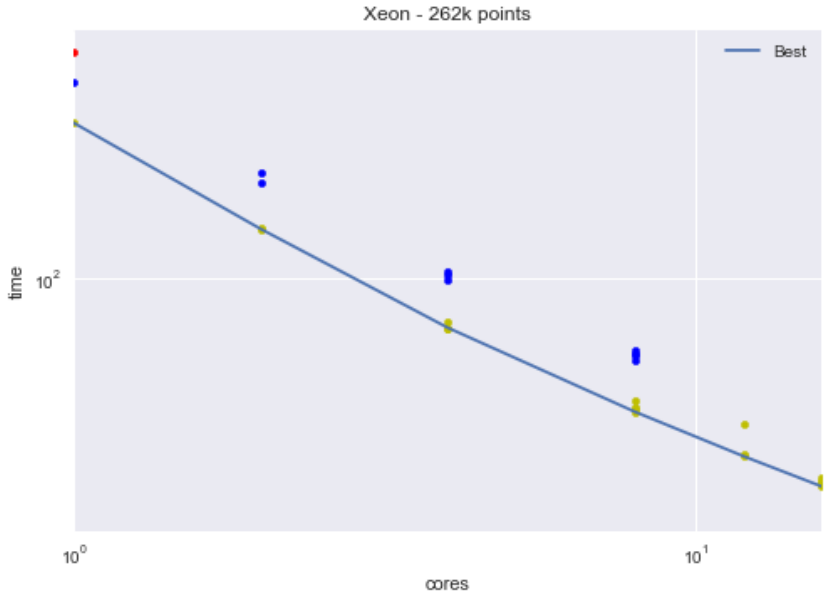


cores	level	points	processes	threads	time	Speed-Up
1	3	262144	1	1	3237.930	1.000000
2	3	262144	1	2	1637.280	1.977628
4	3	262144	1	4	832.815	3.887934
8	3	262144	1	8	426.710	7.588128
16	3	262144	1	16	233.300	13.878826
32	3	262144	1	32	135.645	23.870618
64	3	262144	2	32	145.447	22.261924

HYBRID CODE

- Sweep over process/thread space for all refinement levels
- Better scaling on both platforms
- Low variability on Xeon, pure threading fastest
- Very high var. on ThunderX, one process per socket





EXPLICIT VECTORISATION

- Kernel: Evaluating polynomial of degree 12 in 3 variables
  - Sum of exponents limited, reducing work to 1/6th
    - Valid: $x, y^2, x + y^2, \ldots x + y + z^{10}$
    - Invalid: $x^{13}, x^{10} + y^3, ..$
  - Auto vectorisation only worked without exploiting this reduction

- Kernel speed-up
    - Xeon 2.1x
    - ThunderX 1.8x
    - Vectors are in [N][455] array, so not well aligned

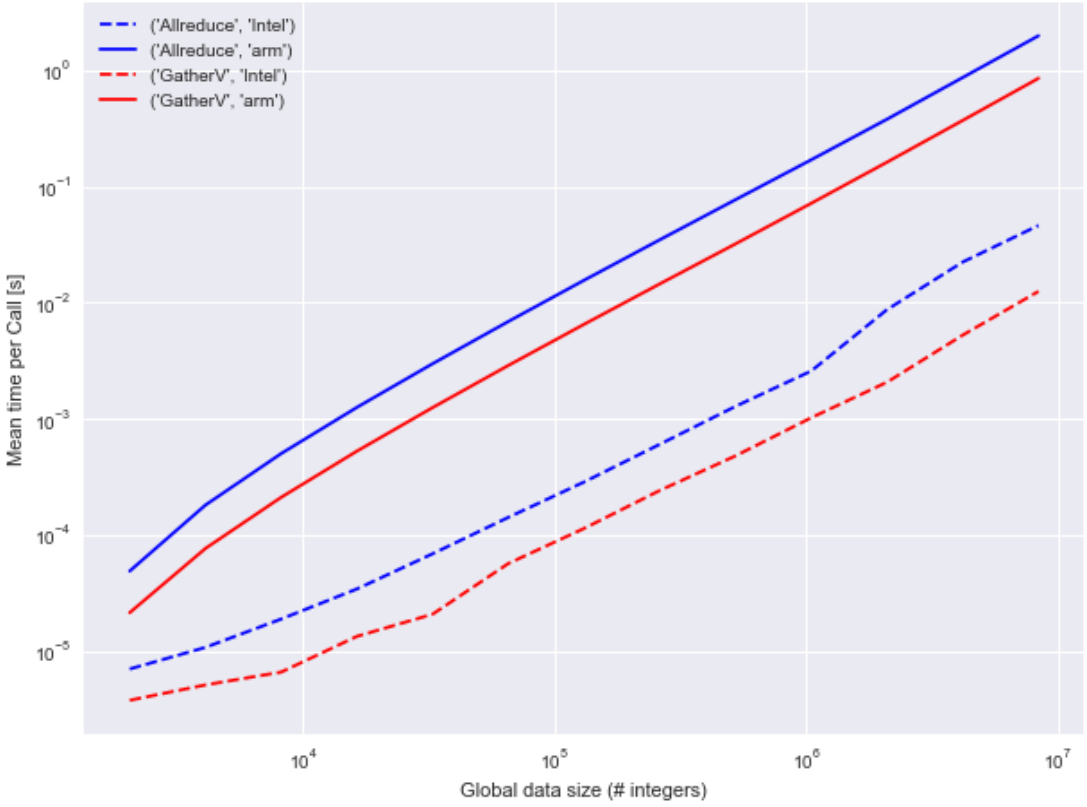
- Different intrinsics per ISA/compiler are cumbersome
  - Elegant solution would make application writers happy
  - E.g. VCL Vector Class Library, www.agner.org/optimize
    - Not supporting ARM out of the box, GPL licensed

ALLREDUCE VS ALLGATHERV

Code used basic MPI Methods

Moving from to better suited methods gave ~5x

Communication was no bottleneck, but results for large process counts became more stable



NEXT STEPS

- Auto-tune parameters to system
  - At compile or run time?

- Find solution for coarse iterations
  - Hybrid with more threads, less processes?
  - Smart potential parallelism as with TBB vs OMP_NESTED?
  - Use serial and parallel BLAS depending on level

- Study on energy / total cost of operations
  - Potential for use in non time critical parameter studies

- Heterogeneous systems
  - For real work loads, boxes are not exactly same amount of work
  - Take imbalance into account when distributing work
  - Have dedicated (accelerated?) node for work on critical path

LESSONS LEARNED

- Porting code is easy when
  - No binary dependencies
  - No specific intrinsics, inline asm, …
- Performance
  - 5 - 10x difference in serial
  - Similar or better scaling on ARM
  - Scalability really limited by application, not system
- Vendor lock-in is painful
  - Unexpected number of bugs discovered
  - Desirable be able to move fast to a new platform

REFERENCES & ACKNOWLEDGEMENTS

- G. Haase, D. Martin, P. Schiffmann and G. Offner: "A Domain Decomposition Multilevel Preconditioner for Interpolation with Radial Basis Functions", In Large Scale Scientific Computing LSSC'17
- G. Haase, D. Martin and G. Offner: "*Towards RBF Interpolation on Heterogeneous HPC Systems*", In Large Scale Scientific Computing LSSC'15

- Thanks to BSC, ARM and UVSQ
- This work is supported by Horizon 2020 grant No 671697