TOWARDS A PARADIGM SHIFT IN HOW WE DEVELOP PARALLEL HIGH-PERFORMANCE COMPUTING APPLICATIONS

Gihan R. Mudalige
Oxford e-Research Centre, University of Oxford
gihan.mudalige@oerc.ox.ac.uk

03rd March 2016
"a hand-held apparatus on which we fling birds at pigs has greater computational capabilities than the arsenal of machines used for guiding crafts through outer space some 45 years ago"

Michio Kaku
MORE TRANSISTORS THAN WE COULD AFFORD TO SWITCH ON!

- Intel Haswell (18 cores) 5.56 Billion transistors
- AMD FirePro S9100 2560 Stream Processors 6.2 Billion transistors
- Nvidia K80 GPU (2 x GK210 GPUs) 4992 cuda cores, 7.1 billion transistors
- Intel Xeon Phi (Knights Corner) 61 cores, ~ 8 Billion transistors
SINGLE THREAD SPEEDUP IS DEAD – MUST EXPLOIT PARALLELISM

NEED TO UTILIZE AVAILABLE MASSIVE PARALLELISM, BUT …

1. Programming models are not suitable to utilize this level of parallelism
2. Type of parallel hardware keeps changing rapidly – moving target

*Its “parallelism” Jim but not as we know it!*
THE SOFTWARE CHALLENGE – DATA MOVEMENT

- Computing is cheap and massively parallel while data movement dominates energy and performance costs
  - Bandwidth is the main bottleneck
    - It’s not about the FLOPS, it’s about data movement” – Bill Dally, NVIDIA/Stanford University (2011)
  - Reduce communications to reduce energy
    - “Even just moving a double precision value across a chip will cost over a factor of ten more [Energy] than computing with it “ - ASCR Exascale Tools Workshop (2011)

- Current programming environments
  - Ignore incurred cost of communication
  - Simply rely on the hardware cache coherency to virtualize data movement
  - No easy way to express data locality and affinity and describe how to decompose and how to layout data in the memory (currently use manual ad-hoc techniques)
  - Communication libraries implicitly assume all the processing elements are equidistant to each other (e.g. MPI, OpenMP)

Needs to refactor applications to align with the a more data-centric programming model and evolve to express information about data locality.
Need easier ways to program for optimized data movement
THE SOFTWARE CHALLENGE – A MOVING TARGET

- Increasingly complex programming skills set needed to extract best performance for your workload on the newest architectures.
  - Need a lot of platform specific knowledge
  - Not clear which architectural approach is likely to “win” in the long-term,
  - Cannot be re-coding applications for each “new” type of architecture or parallel system.
  - Nearly impossible for re-writing legacy codes

- Currently the common approach for utilizing novel hardware is to
  - Manually port legacy applications
    - Usually converting key kernels of the application
    - Re-write parts of the code to optimize communications
  - Maintain several “efficient” versions of the source code

The development of HPC applications designed to “future proof” their continued performance and portability on a diverse range of hardware and future emerging systems is of critical importance.
Provide the application developer with a **domain specific** abstraction

- To declare the problem to be computed
- Without specifying its implementation
- Use domain specific constructs in the declaration
- Provide a lower implementation level

- To apply automated techniques for translating the specification to different implementations
- Target different hardware and software platforms
- Exploit domain knowledge for better optimisations on each hardware system
The OP2 API – An Example
OP2 declarations

```c
int nedges = 12; int ncells = 9;

int edge_to_cell[24] =
{0, 1, 1, 2, 0, 3, 1, 4,
  2, 5, 3, 4, 4, 5, 3, 6,
  4, 7, 5, 8, 6, 7, 7, 8};

op_set edges = op_decl_set(nedges, "edges");
op_set cells = op_decl_set(ncells, "cells");

op_map pecell =
op_decl_map(edges, cells, 2, edge_to_cell, "edge_to_cell_map");

op_dat dcells =
op_decl_dat(cells, 1, "double", cell_data, "data_on_cells");
```
OP2 declarations

double cell_data[9] = 
{0.128, 0.345, 0.224, 0.118, 0.246, 
0.324, 0.112, 0.928, 0.237};

double edge_data[12] =
{3.3, 2.1, 7.4, 5.5, 7.6, 3.4, 10.5, 
8.9, 6.4, 4.4, 3.6};

op_dat dcells = op_decl_dat(cells, 1, "double", cell_data, 
"data_on_cells");

op_dat dedges = op_decl_set(edges, 1, "double", edge_data, 
"data_on_edges");
**The OP2 API – An Example**

**OP2 loop over edges**

```c
void res(double* edge, double* cell0, double* cell1){
    *cell0 += *edge;
    *cell1 += *edge;
}
```

```c
op_par_loop(res,"residual_calculation", edges,
            op_arg(dedges, -1, OP_ID, 1, "double", OP_READ),
            op_arg(dcells, 0, pecell, 1, "double", OP_INC),
            op_arg(dcells, 1, pecell, 1, "double", OP_INC));
```

*Directly accessed*

*Indirectly accessed via mapping*

Access descriptors

The mapping + index resolve the address of the actual parameter
DEVELOPING AN APPLICATION WITH OP2

Unstructured Mesh Application ➔ OP2 Application (Fortran/C/C++ API) ➔ Platform Specific Optimized Application Files ➔ Platform Specific Binary Executable ➔ Hardware

OP2 Source-to-Source translator (Python)

Modified Platform Specific OP2 Application ➔ Conventional Compiler (e.g. icc, nvcc, pgcc)+ compiler flags ➔ Mesh (hdf5) ➔ Hardware

Link

OP2 Platform Specific Optimized Backend libraries

Sequential, Vectorized, CUDA, OpenMP, MPI, OpenCL
Hydra is an unstructured mesh production CFD application used at Rolls-Royce for simulating turbo-machinery of Aircraft engines.

- Production code is written in FORTRAN 77
  50K LoC with 1000 parallel loops

- For real production problems, simulation time is in the order of hours, up to days for large calculations.
Hydra Performance – Distributed Memory (2.5M Edges)

HECToR (Cray XE6): (up to 128 nodes = 4K cores)
1 Node = 2 x 16-core AMD Opteron 6276 (Interlagos) 2.3GHz + 32GB
Cray Gemini Interconnect
NVIDIA GPU Cluster:
1 Node = 2Tesla K20m GPUs + Intel Xeon E5-1650 (sandy-bridge) 3.2GHz
FDR InfiniBand

```
SUBROUTINE res_calc(x1,x2,q1,q2,adt1,adt2,res1,res2)
  IMPLICIT NONE
  REAL(kind=8), DIMENSION(2), INTENT(IN) :: x1
  REAL(kind=8), DIMENSION(2), INTENT(IN) :: x2
  REAL(kind=8), DIMENSION(4), INTENT(IN) :: q1
  REAL(kind=8), DIMENSION(4), INTENT(IN) :: q2
  REAL(kind=8), INTENT(IN) :: adt1
  REAL(kind=8), INTENT(IN) :: adt2
  REAL(kind=8), DIMENSION(4) :: res1
  REAL(kind=8), DIMENSION(4) :: res2
  REAL(kind=8) :: dx,dy,mu,ri,p1,vol1,p2,vol2,f

  !computations such as:
  res(1) = res(1) + q1(1)*(x1(1) - x2(1))
  ... 
END SUBROUTINE
```

```
SUBROUTINE op_wrap_res_calc(
   & opDat1Local, opDat3Local, opDat5Local, opDat7Local, &
   & opDat1Map, opDat1MapDim, opDat3Map, opDat3MapDim, &
   & bottom,top)
real(8) opDat1Local(2,*)
real(8) opDat3Local(4,*)
real(8) opDat5Local(1,*)
real(8) opDat7Local(4,*)
INTEGER(kind=4) opDat1Map(*)
INTEGER(kind=4) opDat1MapDim
INTEGER(kind=4) opDat3Map(*)
INTEGER(kind=4) opDat3MapDim
INTEGER(kind=4) opDat5Map(*)
INTEGER(kind=4) opDat5MapDim
INTEGER(kind=4) bottom,top,i1
INTEGER(kind=4) map1idx, map2idx, map3idx, map4idx
DO i1 = bottom, top-1, 1
   ! set up the indirect access index
   map1idx = opDat1Map(i1 * opDat1MapDim + 0)+1
   map2idx = opDat1Map(i1 * opDat1MapDim + 1)+1
   map3idx = opDat3Map(i1 * opDat3MapDim + 0)+1
   map4idx = opDat3Map(i1 * opDat3MapDim + 1)+1
   ! kernel call -- use the indirect index
   ! to access the data in the ops_dat
   CALL res_calc( &
      & opDat1Local(1,map1idx), &
      & opDat1Local(1,map2idx), &
      & opDat3Local(1,map3idx), &
      & opDat3Local(1,map4idx), &
      & opDat5Local(1,map3idx), &
      & opDat5Local(1,map4idx), &
      & opDat7Local(1,map3idx), &
      & opDat7Local(1,map4idx) )
END DO
END SUBROUTINE

SUBROUTINE res_calc_host( userSubroutine, set, &
   & opArg1, opArg2, opArg3, opArg4, opArg5, opArg6, &
   & opArg7, opArg8 )
   ...
   n_upper = op_mpi_halo_exchanges( [... ])
   CALL op_mpi_wait_all(numberOfOpDats,opArgArray)
   ! setup c to f pointers
   CALL c_f_pointer(opArg1%data,opDat1Local,(/shape/))
   CALL c_f_pointer(opArg1%map_data,opDat1Map,(/shape/))
   ...
   ...
   CALL op_wrap_res_calc( &
      & opDat1Local, opDat3Local, opDat5Local, opDat7Local, &
      & opDat1Map, opDat1MapDim, opDat3Map, opDat3MapDim, &
      & 0, n_upper)
   ...
   END SUBROUTINE
#define SIMD_VEC 4

SUBROUTINE res_calc_vec(x1,x2,q1,q2,
ad1,ad2,rest1,rest2,idx)
  !dir attributes vector :: res_calc_vec
  IMPLICIT NONE
  real(8), DIMENSION(SIMD_VEC,2), INTENT(IN) :: x1, X2
  real(8), DIMENSION(SIMD_VEC,4), INTENT(IN) :: q1, q2
  real(8), DIMENSION(SIMD_VEC,1), INTENT(IN) :: ad1, ad2
  real(8), DIMENSION(SIMD_VEC,4) :: rest1, rest2
  INTEGER(4) :: idx
  !computations such as:
  res(idx,1) = res(idx,1) + &
  & q1(idx,1) * (x1(idx,1) - x2(idx,1))
END SUBROUTINE

SUBROUTINE op_wrap_res_calc( opDat1Loc, opDat3Loc, &
  & opDat5Loc, opDat7Loc, opDat1Map, opDat3MapDim, &
  & opDat5Map, opDat7MapDim, bottom,top)
  ...
  real(8) dat1(SIMD_VEC,2), dat2(SIMD_VEC,2), &
  & dat3(SIMD_VEC,4), dat4(SIMD_VEC,4), dat5(SIMD_VEC,1), &
  & dat6(SIMD_VEC,1), dat7(SIMD_VEC,4), dat8(SIMD_VEC,4)
  !loop SIMD_VEC number of iterations at a time
  DO i1 = bottom, ((top-1)/SIMD_VEC)*SIMD_VEC-1, SIMD_VEC
    !gather loop -- vectorized
    !DIR SIMD
    DO i2 = 1, SIMD_VEC, 1
      CALL res_calc_vec(dat1,dat2,dat3,dat4,dat5,dat6, &
        & dat7, dat8, i2) ! -- vectorized kernel call
    END DO
  / scatter loop -- non-vectorized
  DO i2 = 1, SIMD_VEC, 1
    map7idx = opDat3Map(1+(i1+i2-1)*opDat3MapDim+0)+1
    map8idx = opDat3Map(1+(i1+i2-1)*opDat3MapDim+1)+1
    opDat7Loc(1,map7idx)=opDat7Loc(1,map7idx)+dat7(i2,1)
    opDat7Loc(2,map7idx)=opDat7Loc(2,map7idx)+dat7(i2,2)
    opDat7Loc(3,map7idx)=opDat7Loc(3,map7idx)+dat7(i2,3)
    opDat7Loc(4,map7idx)=opDat7Loc(4,map7idx)+dat7(i2,4)
    opDat7Loc(1,map8idx)=opDat7Loc(1,map8idx)+dat8(i2,1)
    opDat7Loc(2,map8idx)=opDat7Loc(2,map8idx)+dat8(i2,2)
    opDat7Loc(3,map8idx)=opDat7Loc(3,map8idx)+dat8(i2,3)
    opDat7Loc(4,map8idx)=opDat7Loc(4,map8idx)+dat8(i2,4)
  END DO
END DO

! remainder - non-vectorized kernel call
  DO i1 = ((top-1)/SIMD_VEC)*SIMD_VEC, top-1, 1
    CALL res_calc( ... )
  END DO
END SUBROUTINE
Vectorization

- Vectorisation of an application with indirect accesses (not previously auto-vectorizable)
- Can use OP2 to generate
  - Aligned memory allocations
  - Explicit gather-scatter to achieve SIMD vectorization
- Promising results for achieving auto-vectorization with Intel compilers (v 16) for an unstructured-mesh benchmark (Airfoil)

<table>
<thead>
<tr>
<th>Airfoil version</th>
<th>Runtime (seconds)</th>
<th>BW(achieved) GB/s (Ind. Loop)</th>
<th>BW(Peak) GB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>20xMPI+ non-vectorized (Fortran API)</td>
<td>32.21</td>
<td>75.21</td>
<td>136</td>
</tr>
<tr>
<td>20xMPI+ Auto-Vectorized (Fortran API)</td>
<td>28.04</td>
<td>79.89</td>
<td>136</td>
</tr>
<tr>
<td>20xMPI+ Vector Intrinsics (C/C++ API)</td>
<td>26.93</td>
<td>86.43</td>
<td>136</td>
</tr>
</tbody>
</table>

Results from a 2 x 10 core Haswell CPU (28 million edge mesh)

~13% speedup

~4% speedup
Vectorization - Hydra

- Nearly 40% speedup with vectorization for loops that has a significant number of floating-point operations.

- But slowdown due to vectorization for loops with less computational intensity.

<table>
<thead>
<tr>
<th>Loop</th>
<th>Dir/Ind</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vfluxedge</td>
<td>indirect</td>
<td>3.13</td>
</tr>
<tr>
<td>wflluxedge</td>
<td>indirect</td>
<td>0.17</td>
</tr>
<tr>
<td>wvfluxedge</td>
<td>indirect</td>
<td>0.15</td>
</tr>
<tr>
<td>ifluxedge</td>
<td>indirect</td>
<td>0.72</td>
</tr>
<tr>
<td>edgecon</td>
<td>indirect</td>
<td>0.68</td>
</tr>
<tr>
<td>volapf</td>
<td>direct</td>
<td>0.27</td>
</tr>
<tr>
<td>srccanode</td>
<td>direct</td>
<td>1.02</td>
</tr>
<tr>
<td>srck</td>
<td>direct</td>
<td>0.21</td>
</tr>
<tr>
<td>updatek</td>
<td>direct</td>
<td>0.65</td>
</tr>
<tr>
<td>accumedges</td>
<td>indirect</td>
<td>0.82</td>
</tr>
<tr>
<td>invjacs</td>
<td>direct</td>
<td>0.12</td>
</tr>
</tbody>
</table>

Total runtime: 9.25 sec

20xMPI+ non-vectorized

<table>
<thead>
<tr>
<th>Loop</th>
<th>Dir/Ind</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vfluxedge</td>
<td>indirect</td>
<td>2.20</td>
</tr>
<tr>
<td>wflluxedge</td>
<td>indirect</td>
<td>0.18</td>
</tr>
<tr>
<td>wvfluxedge</td>
<td>indirect</td>
<td>0.15</td>
</tr>
<tr>
<td>ifluxedge</td>
<td>indirect</td>
<td>0.80</td>
</tr>
<tr>
<td>edgecon</td>
<td>indirect</td>
<td>0.85</td>
</tr>
<tr>
<td>volapf</td>
<td>direct</td>
<td>0.28</td>
</tr>
<tr>
<td>srccanode</td>
<td>direct</td>
<td>0.57</td>
</tr>
<tr>
<td>srck</td>
<td>direct</td>
<td>0.20</td>
</tr>
<tr>
<td>updatek</td>
<td>direct</td>
<td>0.66</td>
</tr>
<tr>
<td>accumedges</td>
<td>indirect</td>
<td>0.53</td>
</tr>
<tr>
<td>invjacs</td>
<td>direct</td>
<td>0.12</td>
</tr>
</tbody>
</table>

Total runtime: 7.94 sec

20xMPI+ Auto-Vectorized
Vectorization - Hydra

- Overall 14% speedup if we use vectorization for all loops.
- But can use auto-tuning to select only the loops that gain a speedup with vectorization and leave the others.

CUDA on 1/2 NVIDIA K80 GPU
Compiled with PGI CUDA FORTRAN 15.1

<table>
<thead>
<tr>
<th>Loop</th>
<th>Dir/Ind</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vfluxedge</td>
<td>indirect</td>
<td>3.29</td>
</tr>
<tr>
<td>wfluxedge</td>
<td>indirect</td>
<td>0.11</td>
</tr>
<tr>
<td>wvfluxedge</td>
<td>indirect</td>
<td>0.08</td>
</tr>
<tr>
<td>ifluxedge</td>
<td>indirect</td>
<td>2.05</td>
</tr>
<tr>
<td>edgecon</td>
<td>indirect</td>
<td>3.21</td>
</tr>
<tr>
<td>volapf</td>
<td>direct</td>
<td>0.19</td>
</tr>
<tr>
<td>srcsanode</td>
<td>direct</td>
<td>0.39</td>
</tr>
<tr>
<td>srck</td>
<td>direct</td>
<td>0.26</td>
</tr>
<tr>
<td>updatek</td>
<td>direct</td>
<td>0.43</td>
</tr>
<tr>
<td>accumulated</td>
<td>indirect</td>
<td>2.59</td>
</tr>
<tr>
<td>invjacs</td>
<td>direct</td>
<td>0.28</td>
</tr>
<tr>
<td>Total runtime</td>
<td></td>
<td>8.32</td>
</tr>
</tbody>
</table>

½ K80 ~ 150W
1 x K80 ~ 300 W
2 x Haswell CPUs ~ 210W (2x105W)

½ K80 is ~28% power efficient than 2 Haswell CPUs
But only ~5% slower than 2 Haswell CPUs.
OPS for Multi-Block Structured-Mesh Applications

- Essentially these are application computing over unstructured collection of structured meshes
- Each block is a structured mesh on which data is defined on
- Provides an API to specify the interface and dependencies between blocks – mismatching interfaces, sliding planes
- Similar to OP2 but uses a stencil (as oppose to a mapping) to access neighbouring elements
- Sets up semantics of parallel execution of blocks
- Addresses issues such as how you partition the multiple blocks
OPS : CLOVERLEAF

- Mini-app from Mantevo/UK-MAC benchmark suite from Sandia National Labs
  - [https://mantevo.org](https://mantevo.org), [http://uk-mac.github.io](http://uk-mac.github.io)

- 2D and 3D Structured Hydrodynamics
- Explicit solution to the compressible Euler equations
- Finite volume predictor/corrector
  - Lagrangian step followed by an advective remap

- Single material

- A range of hand-optimized parallel versions available (written in F90)
  - (MPI, OpenMP, CUDA, OpenCL, OpenACC
    - CAFArrays, OpenSHMEM, IntelOffload)

- Each about 6000 LoC

Re-engineered (within a month) with OPS (~6000 LoC high-level source)
Can automatically code generate - MPI, OpenMP, CUDA, OpenCL, OpenACC
CLOVERLEAF - SINGLE NODE (3480² - 2D MESH)

- 2 x 8-core Intel Xeon E5-2680 (Sandy bridge) 2.70GHz (with SMT) + 64GB RAM
- NVIDIA Tesla K20c GPU, 5GB/GPU
CLOVERLEAF PERFORMANCE SINGLE NODE (96³ - 3D MESH)

-10%

2 x 8-core Intel Xeon E5-2680 (Sandy bridge) 2.70GHz (with SMT) + 64GB RAM

NVIDIA Tesla K80 GPUs server

1 GPU card = 2 GPUs 2x12GB per 2 GPUs (ECC off)
CLOVERLEAF - DISTRIBUTED MEMORY (640^3 - 3D MESH)

Archer (Cray XC30)
Cray Aries Interconnect

1 Node =
2x 12-core Intel
Xeon E5-2697 2.70GHz
(Ivy Bridge) +
64 GB RAM

Up to 512 nodes = 12K cores

Titan (Cray XK7)
Cray Gemini interconnect

1 Node =
16-core AMD Opteron 6274
+ NVIDIA K20X
32GB Ram per node +
6GB/GPU (ECC on) +

Up to 2048 nodes = 32K cores
**CURRENT WORK: RE-ENGINEERING MULTI-BLOCK APPLICATIONS**

**with University of Bristol, Southampton and STFC**

- Re-Engineering production grade multi-block mesh applications to utilize OPS
  - ROTOR-SIM multi-block application from the Uni. of Bristol
  - SBLI multi-block application at Uni. of Southampton
  - Re-Engineer multi-block applications at STFC

- Performance optimisations and analysis on Titan and Archer
- Will be work done as part of the EPSRC funded OPS project
**Future Work: Optimizing Data Movement – Tiling**

```
for (i=0; i<N; i++) res[i] = 0.0; //loop 1 read/write res

for (i=0; i<N-1; i++) {
    flux = flux_function(q[i], q[i+1]);
    res[i] -= flux;
    res[i+1] += flux;
}

for (i=0; i<N; i++) q[i] += dt*res[i]; //loop 3
```

**Standard operation 8N transfers**

**Loop 1:** initialize in cache (0)
**Loop 2:** read q, update res in cache (1)
**Loop 3:** q, res in cache write out q, (using a separate q_new array) (2)

Total: 3N transfers – factor of 2.7x savings

Redundant Tiling - Both A and B towers can be computed in parallel with additional redundant computation

**Res data not get stored back in main memory, just hold a working set in cache**
Future Work: Multi-material Hydrodynamics Applications

- CloverLeaf was a Single Material Hydro-Dynamics Application

- In Multi-material Hydrodynamics applications there are many materials globally but very few locally

- Indirect addressing is usually used to reduce storage costs but this has very poor performance (e.g. no vectorisation achieved)

- Need new design to navigate and change sub-zonal data structures without hindering concurrency and memory bandwidth
CONCLUSIONS - THE NEED FOR A PARADIGM SHIFT

- Cannot be re-implementing HPC software
  - Need high-level development methods, tools and techniques
  - Need to identify the right level of abstraction for different application domains to support parallel applications development

- Need automated techniques for data-centric parallel programming
  - Can apply such optimisations automatically through high-level frameworks
RESEARCH COLLABORATORS

- Prof. Mike Giles, Ben Spencer (Oxford)
- Istvan Reguly, Endre László (Oxford, now at PPKE Hungary)
- Carlo Bertolli (Imperial College London, now at IBM TJ Watson)
- Prof. Paul Kelly, David Ham (Imperial College London)
- Prof. Simon McIntosh-Smith, Dan Curran, Michael Boulton, Prof. Chris Allen, Wayne Gaudin (Bristol)
- Adam Betts, Florian Rathgeber, Graham Markall, Lawrence Mitchell (Imperial College London)
- David Radford, Yoon Ho, Paolo Adami, Leigh Lapworth (Rolls Royce)
- Andrew Mallinson (Warwick, now at Intel)
- Prof. Stephen Jarvis, David Beckingsale, Andy Herdman (Warwick)
- John Pennycook (Intel)
- Prof. Neil Sandham, Sathya Jammy (Southampton)
- Prof. David Emerson, Jianping Meng (STFC)
- Prof. Nick Hills (Surrey)
- Thomas Bradley and others (NVIDIA)
- Brent Leback and others (PGI)
ACKNOWLEDGEMENTS

- OP2 is funded by the UK Technology Strategy Board and Rolls-Royce plc. through the SILOET project, and the UK EPSRC projects EP/I006079/1, EP/I00677X/1 on Multi-layered Abstractions for PDEs.
- OPS is funded by the UK Engineering and Physical Sciences Research Council projects EP/K038494/1, EP/K038486/1, EP/K038451/1 and EP/K038567/1 on “Future-proof massively-parallel execution of multi-block applications” and EP/J010553/1 “Software for Emerging Architectures” (ASEArch) project.
- CloverLeaf re-engineering research is funded by the UK AWE plc. under project “High-level Abstractions for Performance, Portability and Continuity of Scientific Software on Future Computing Systems”.
- This paper used the UK National Supercomputing Service – ARCHER and resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
- CloverLeaf development is supported by the UK AWE plc. under grants CDK0660 (The Production of Predictive Models for Future Computing Requirements) and CDK0724 (AWE Technical Outreach Programme)
- This research utilized the systems provided by Advanced Research Computing (ARC) at the University of Oxford in carrying out this work
OP2 available as open source software
https://github.com/OP2/OP2-Common

OPS available as open source software
https://github.com/gihanmudalige/OPS

Email: gihan.mudalige@oerc.ox.ac.uk
Web: http://www.oerc.ox.ac.uk/people/gihan-mudalige