An Implicitly Parallel Meshfree Solver in Regent

Rupanshu Soi[1], Nischay Ram Mamidi[1], Elliott Slaughter[2], Kumar Prasun[1],

Anil Nemili[1], and SM Deshpande[3]

[1]Birla Institute of Technology and Science - Pilani, Hyderabad Campus, India

[2]SLAC National Accelerator Laboratory, Menlo Park, USA

[3]Jawaharlal Nehru Centre for Advanced Scientific Research, Bengaluru, India

3^{rd} Parallel Applications Workshop, Alternatives To MPI+X

SC 20, November 12-13, 2020

Outline

Introduction

- Numerical simulations of fluid flow problems are computationally intensive
- Current CFD solvers do not exploit full computational resources (both CPUs and GPUs)
- Need a CFD code that can fully exploit heterogeneous platforms
- For example: SU2 uses CPUs, OpenFOAM uses CPUs or GPUs (not both)

Introduction

- Numerical simulations of fluid flow problems are computationally intensive
- Current CFD solvers do not exploit full computational resources (both CPUs and GPUs)
- Need a CFD code that can fully exploit heterogeneous platforms
- For example: SU2 uses CPUs, OpenFOAM uses CPUs or GPUs (not both)
- Current CFD solvers are explicitly parallel
- It will be advantageous if the code is implicitly parallel

Introduction

- Numerical simulations of fluid flow problems are computationally intensive
- Current CFD solvers do not exploit full computational resources (both CPUs and GPUs)
- Need a CFD code that can fully exploit heterogeneous platforms
- For example: SU2 uses CPUs, OpenFOAM uses CPUs or GPUs (not both)
- Current CFD solvers are explicitly parallel
- It will be advantageous if the code is implicitly parallel

Objective: Develop an implicitly parallel meshfree solver in Regent

Meshfree q-LSKUM Solver for 2D Euler Equations

Least Squares Kinetic Upwind Method (LSKUM):

- Euler equations: Govern the inviscid compressible fluid flows

$$\frac{\partial \boldsymbol{U}}{\partial t} + \frac{\partial \boldsymbol{G}}{\partial x} + \frac{\partial \boldsymbol{H}}{\partial y} = 0$$

- Introduce upwinding using Kinetic Flux Vector Splitting (KFVS) (Mandal-1994)

$$\frac{\partial \boldsymbol{U}}{\partial t} + \frac{\partial \boldsymbol{G}^{+}}{\partial x} + \frac{\partial \boldsymbol{G}^{-}}{\partial x} + \frac{\partial \boldsymbol{H}^{+}}{\partial y} + \frac{\partial \boldsymbol{H}^{-}}{\partial y} = 0$$

- Basic idea of LSKUM: Approximate the spatial derivatives using Least Squares

Meshfree q-LSKUM Solver for 2D Euler Equations

Least Squares Kinetic Upwind Method (LSKUM):

- Euler equations: Govern the inviscid compressible fluid flows

$$\frac{\partial \boldsymbol{U}}{\partial t} + \frac{\partial \boldsymbol{G}}{\partial x} + \frac{\partial \boldsymbol{H}}{\partial y} = 0$$

- Introduce upwinding using Kinetic Flux Vector Splitting (KFVS) (Mandal-1994)

$$\frac{\partial \boldsymbol{U}}{\partial t} + \frac{\partial \boldsymbol{G}^{+}}{\partial x} + \frac{\partial \boldsymbol{G}^{-}}{\partial x} + \frac{\partial \boldsymbol{H}^{+}}{\partial y} + \frac{\partial \boldsymbol{H}^{-}}{\partial y} = 0$$

- Basic idea of LSKUM: Approximate the spatial derivatives using Least Squares
- Input: Set of points and their neighbours (known as connectivity)
- Operates on structured, unstructured, cartesian, chimera point distributions, etc.
- Higher-order accuracy in space: Using $\boldsymbol{q}$-variables (q-LSKUM) (Deshpande-2002)
- Time accuracy: Strong Stability Preserving Runge-Kutta Schemes (SSP-RK3)

Pseudo Code

Algorithm 1: Meshfree solver based on q-LSKUM

```
subroutine q-LSKUM
    call preprocessor()
    for n ← 1 to n ≤ N do
        call timestep()
        for rk ← 1 to 4 do
            call q_variables()
            call q_derivatives()
            call flux_residual()
            call state_update(rk)
        end
        call residue()
    end
    call postprocessor()
end subroutine
```

Regent Implementation of the Solver

Regent's Data Model

- Regions are the primary unit of Regent's data model
- Partitions (array of subregions) help expose data parallelism in an application
- Regent has an expressive framework for defining partitions

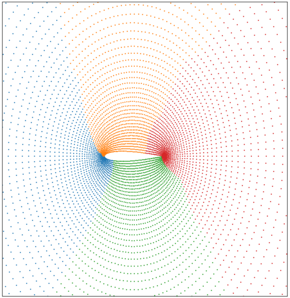
Regent Implementation of the Solver

Regent's Data Model

- Regions are the primary unit of Regent's data model
- Partitions (array of subregions) help expose data parallelism in an application
- Regent has an expressive framework for defining partitions

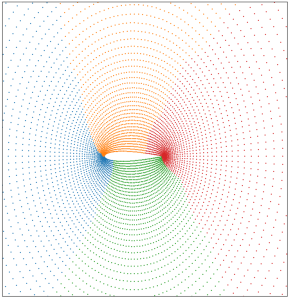
Domain Decomposition

- Point distribution is partitioned into subregions of local and ghost points
- Employed METIS for partitioning (Optional, but important for performance)

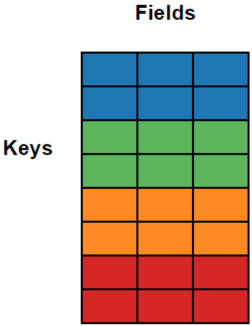


Decomposition of a NACA 0012 airfoil

Regent Implementation of the Solver



Decomposition of a NACA 0012 airfoil



Region

Regent Implementation of the Solver

Regent Tasks

- Tasks receive regions as input and declare privileges on them (RO, WO, RW)
- Task dependencies are inferred automatically
- Compiler, runtime (Legion) extract parallelism
- Tasks execute after their dependencies are satisfied

```
task state_update(r : region(ispace(int1d),
    DomainPt), rk : int)
where
  reads(r.{nx, ny, prim, flux_res}),
  writes(r.prim)
do
  var sum_res_sqr = 0.0
  for pt in r do
    -- computations
  end
  return sum_res_sqr
end
```

Regent task declaration for state-update()

Regent Implementation of the Solver

Data Communication

- No user-written data communication code
- All data copies inserted automatically to maintain correctness

```
var p_nbhs = p_local | p_ghost
for i = 0, N do
  __demand(__index_launch)
  for color in p_local.colors do
    time_step(p_local[color])
  end
  var res : double = 0.0
  for rk = 1, 5 do
    __demand(__index_launch)
    for color in p_local.colors do
      q_variables(p_local[color])
    end
    __demand(__index_launch)
    for color in p_local.colors do
      q_derivatives(p_local[color], p_nbhs[color])
    end
    __demand(__index_launch)
    for color in p_local.colors do
      flux_residual(p_local[color], p_nbhs[color])
    end
    __demand(__index_launch)
    for color in p_local.colors do
      res += state_update(p_local[color], rk)
    end
  end
  residue(res)
end
```

Regent code for q-LSKUM

Regent Implementation of the Solver

Regent Specific Optimizations:

- Index launches: Amortizes the analysis cost of a loop that launches tasks
- Dynamic Control Replication: An optimization technique for scalability (Slaughter-SC17)
- OpenMP code generation: Converts serial loops to OpenMP style loops
- Mapper customization: To disable load balancing for better performance on AMD nodes

Numerical Results

Test Case Details:

- Inviscid flow over a NACA 0012 airfoil
- $Ma = 0.85$ and $AoA = 1^o$
- Five levels of point distributions: 0.8M to 40M

Language Specifications

- Regent, Fortran 90 and Julia 1.5.1

Node Configuration

- AMD EPYC™ 7542 (32x2) with 256 GB RAM
- Mellanox EDR 100 Gbps Interconnect

Numerical Results: Performance on a single node

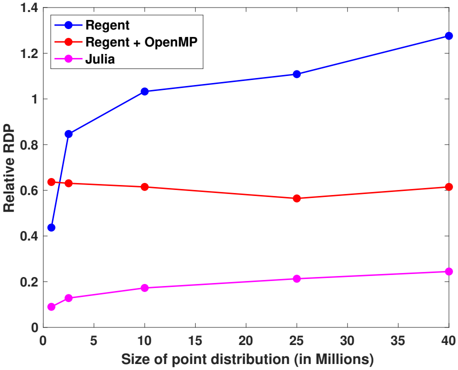
No. of points	Regent	Regent + OpenMP	Fortran	Julia
RDP $\times 10^{-7}$ (Lower is better)				
$804,824$	9.9266	6.8145	4.3367	48.2093
$2,642,264$	4.8180	6.4662	4.0788	31.8098
$9,992,000$	3.7195	6.2460	3.8406	22.2528
$25,330,172$	3.3717	6.6212	3.7374	17.5542
$39,381,464$	2.8772	5.9714	3.6717	15.0160

Table: Comparison of RDP values on a single node.

- RDP = Total wall clock time in seconds/No. of iterations/No. of points
- Number of iterations = 1000

Numerical Results: Performance on a single node

Comparison of relative RDP on a single node:



Relative RDP of Regent = RDP of Fortran / RDP of Regent

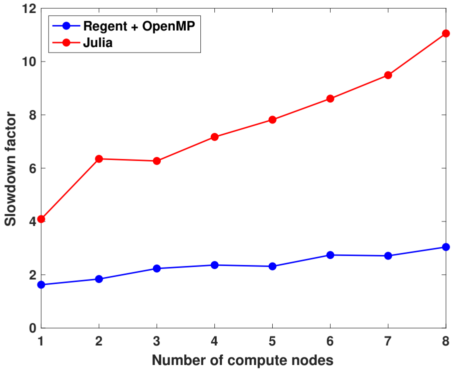
Numerical Results: Performance on multiple nodes

Nodes	Regent + OpenMP	Fortran	Julia
RDP values (Lower is better)			
1	5.9714×10^{-7}	3.6717×10^{-7}	1.5016×10^{-6}
2	3.2912×10^{-7}	1.7886×10^{-7}	1.1356×10^{-6}
3	2.8706×10^{-7}	1.2845×10^{-7}	8.0546×10^{-7}
4	2.2686×10^{-7}	9.5952×10^{-8}	6.8814×10^{-7}
5	1.8809×10^{-7}	8.1205×10^{-8}	6.3482×10^{-7}
6	1.8947×10^{-7}	6.9134×10^{-8}	5.9520×10^{-7}
7	1.6165×10^{-7}	5.9616×10^{-8}	5.6575×10^{-7}
8	1.5186×10^{-7}	4.9933×10^{-8}	5.5204×10^{-7}

Table: Comparison of RDP values on multiple nodes.

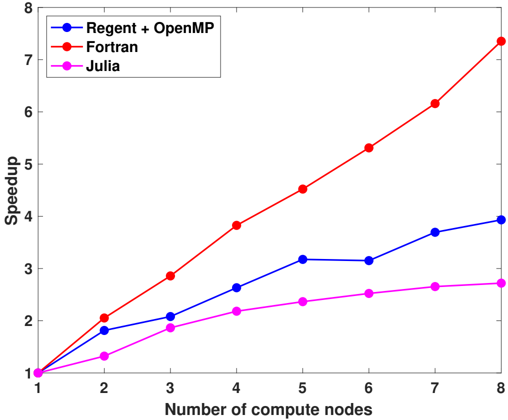
Numerical Results: Performance on multiple nodes

Comparison of the slowdown factor on the finest point distribution:



Slowdown factor of Regent+OpenMP = RDP of Regent+OpenMP / RDP of Fortran

Numerical Results: Performance on multiple nodes



Strong scalability on the finest point distribution

Conclusions & Future Work

Conclusions:

- Developed an implicitly parallel meshfree q-LSKUM solver based on Regent
- On a single node for fine point distributions, Regent exhibited superior performance

Conclusions & Future Work

Conclusions:

- Developed an implicitly parallel meshfree q-LSKUM solver based on Regent
- On a single node for fine point distributions, Regent exhibited superior performance

Future Work:

- Working on enhancing the computational efficiency of Regent on multi-node clusters
- Extending the solver to three-dimensional compressible flows
- Constructing a truly hybrid solver based on Regent, which can exploit the full computational potential on heterogeneous platforms

Conclusions & Future Work

Conclusions:

- Developed an implicitly parallel meshfree q-LSKUM solver based on Regent
- On a single node for fine point distributions, Regent exhibited superior performance

Future Work:

- Working on enhancing the computational efficiency of Regent on multi-node clusters
- Extending the solver to three-dimensional compressible flows
- Constructing a truly hybrid solver based on Regent, which can exploit the full computational potential on heterogeneous platforms

Thank you very much!