Literature review: Adaptive Neural Network-Based Approximation to Accelerate Eulerian Fluid Simulation

Yuan Chen

Motivation

Problem

The Eulerian fluid simulation, which requires prohibitively high computational resources

Navier-Stokes equations

$$\frac{\partial \vec{u}}{\partial t} = -\vec{u} \cdot \nabla \vec{u} - \frac{1}{\rho} \nabla p + \vec{g} \tag{1}$$

$$\nabla \cdot \vec{u} = 0 \tag{2}$$

Use (2) as a constraint and split (1) (momentum equation) into three parts: advection, adding external force, and pressure projection. Solve by finite difference method using MAC (marker-and-cell) grids.

Motivation

Algorithm 1 Velocity Update in the Euler Equation

Require: Simulation time step N;

Start with an initial divergence-free velocity field $\vec{u}^0$
Determine a good time step $\triangle t$ to go from time t_n to time t_{n+1}.
for $n \leftarrow 1$ to N **do**
  Advection. Set $\vec{u}^A$ = advect($\vec{u}^n$, Δt, q);
  Add body force. $\vec{u}^B = \vec{u}^A + \triangle t \vec{f}$;
  Pressure projection. set $\vec{u}^{n+1}$ = Project($\triangle t$, $\vec{u}^B$):
    1) Solve the Poisson eq. $\nabla \cdot \nabla \vec{p}_n = \frac{1}{\triangle t} \nabla \cdot \vec{u}^B$
      //Use a PCG solver to to update $\vec{p}_n$.
      Set initial guess $\vec{p}_n$=0 and residual vector r=d (if r=0, then return $\vec{p}_n$)
      Set search direction $\vec{s} = ApplyPreconditioner(r)$;
      while residual doesn't reach the convergence criteria **do**
        Set $\alpha = \frac{r^T r}{s^T A s}$;
        Calculate the residual $r = r - \alpha A \vec{p}_n$;
        Update the solution $\vec{p}_n = \vec{p}_n + \alpha \vec{s}$;
        Update the conjugated direction $\vec{s} = r + \beta \vec{s}$;
      end while
    2) Apply velocity update: $\vec{u}^{n+1} = \vec{u}^B - \Delta t \frac{1}{\rho} \nabla \vec{p}_n$;
end for
return 0

- Most crucial and time-consuming step: solving the Poisson's equation (to preserve the divergence-free constraint on the velocity and maintain simulation accuracy.)
- Preconditioned Conjugate Gradient (PCG) method, which involves large computation that iteratively converges to meet a convergence criteria

Motivation

Idea: Use neural network to augment existing simulations by improving accuracy and significantly reducing latency, by replacing some execution phases.
Here, use Neural Network to approximate PCG method – to solve poisson equation.

Challenges

1. Generate multiple neural network models, given the user requirements on performance and simulation quality. Each of them has different topologies and different implications on performance and simulation quality.
2. Select neural network models at runtime
3. Provide a high-quality approximation for a large number of input problems and ensure overall performance benefit.

Tompson's model(existing work)

a CNN model with five stages of convolution and ReLU layers to approximate the PCG solver.

$$\hat{p}_t = f_{conv}(\nabla \cdot \vec{u}_t^*, g_{t-1}; W)$$

Objective function:

$$DivNorm = \sum_i w_i \{\nabla \cdot \vec{u}_t\}_i^2$$

Table 1: Execution time and simulation quality loss of three models for solving the Poisson's equation.

Method	Execution Time (ms)	Avg. Quality Loss
PCG	2.34×10^8	$--$
Tompson [10]	7.19×10^4	1.3×10^{-2}
Yang [11]	3.20×10^4	4.9×10^{-2}

Proposal

Smart-fluidnet framework

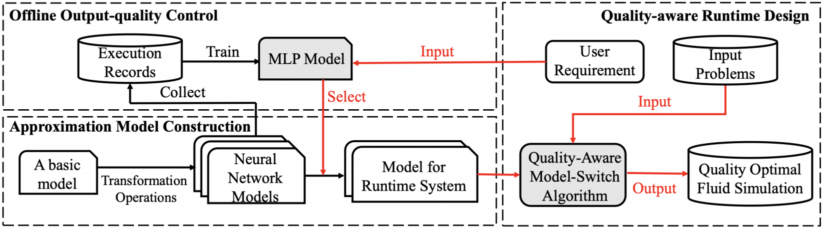

Figure 2: Workflow of the proposed Smart-fluidnet.

metric

- time cost
- quality loss: The simulation output in mantaflow is a smoke dense matrix of a rendered smoke frame.

we have quality loss. The simulation quality loss (Q_{loss}) is formally defined by:

$$Q_{loss} = \frac{1}{N \times M} \sum_{i=1}^{N} \sum_{j=1}^{M} \left| \rho^*_{ij} - \rho_{ij} \right|, \quad (3)$$

where ρ refers to the smoke density matrix generated in the original

Auto-Keras with extension

Auto-Keras uses the Bayesian optimization to generate a single model with the best accuracy. Use 4 transformation operations to change Auto-Keras:

- deleting a layer of the neural network. $shallow(G,L)$
- narrowing a layer of the neural network by reducing neurons in an intermediate layer. $narrow(G,L,r)$
- pooling. (downsample, max/average pooling). $pooling(G,L,m)$
- dropout. probability p, $dropout(G,L,p)$

do not apply the operation more than twice in the input model.
1-shallow(5)-narrow*10 times(5+50)-pooling(55+55)-dropout(110+18)
133 models in total: including 5 accurate models generated by Auto-Keras
14 model candidates: selected by Pareto optimality (the lowest time cost, the lowest quality loss, or both)

2. OFFLINE OUTPUT-QUALITY CONTROL

user requirement

$U(q,t)$: The final quality loss and execution time of the fluid simulation should be less than q and t

success rate

the ratio of those input problems with which the fluid simulation can reach the simulation quality and time requirement to the total number of input problems.
Developed by a non-linear MLP model.

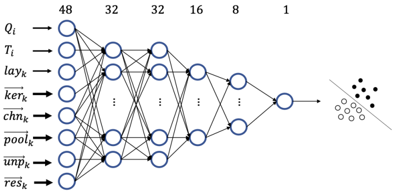

Figure 4: The network architecture of our MLP model.

Construction of Training Samples

- $14 \times N$ execution records (i.e., simulation quality and execution time) for the 14 neural network models)
- Given a neural network, random picking up a user requirement (q and t), build **feature vector** of the sample; denote the ratio of those execution records to N as label.

MLP Model Construction and Loss Function

- Neural Network NN_k,feature vector F, MLP output $\hat{r}_{k,q,t} = f_{MLP}(F_{k,q,t})$
- minimize the loss between $\hat{r}_{k,q,t}, r_{k,q,t}$
- a balanced trade-off between prediction accuracy and model size.

Usage of MLP

$$T_{total} = \hat{r}_{k,q,t} \times T_{M_k} + (1 - \hat{r}_{k,q,t}) \times T'$$

Select NN_k where $T_{total} < t$

CumDivNorm: A Metric for Runtime Quality Control

$$CumDivNorm = \sum_{i=1}^{n} DivNorm_i$$

Shows strong correlation with final simulation quality Q_{loss}
Use linear regression to predict $CumDivNorm^{final}$; use knn, including offline and online phase to predict Q_{loss}

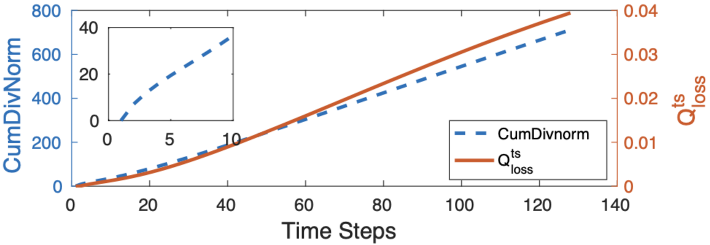

Figure 6: Relationship between $CumDivNorm$ **and** Q^{ts}_{loss}**.**

3. QUALITY-AWARE RUNTIME DESIGN

Algorithm 2 The quality-aware model-switch runtime algorithm

Require: The user requirement $U(q, t)$.

Choose a neural network model M_k with the highest success rate according to MLP.
while t does not reach the final time step **do**
    Send M_k to predict the final simulation quality.
    Prediction of quality loss:
    1) Build a linear regression model with $DivNorm$ values measured in the last five time steps;
    2) Predict $CumDivNorm^{final}$ by the linear regression model;
    3) Predict Q'_{loss} of the current neural network model by the KNN algorithm;
    Model Switch:
    if Q'_{loss} is close to q **then**
        Continue using the current neural network model for L steps;
    else if Q'_{loss} less than q **then**
        Switch to a faster (less accurate) neural network model;
    else if Q'_{loss} is larger than q **then**
        Switch to a slower (more accurate) neural network model;
    else if Cannot find any model **then**
        Restart by the PCG method;
    end if
    $t \leftarrow t + L$; //L is the check interval.
end while
return 0

Evaluation

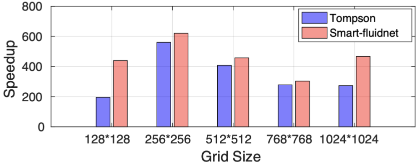

Figure 8: Performance (execution time) of the Tompson's model and Smart-fluidnet.

Table 2: Percentage of input problems with which the simulation reaches the requirement on quality.

Grid size	128*128	256*256	512*512	768*768	1024*1024
Tompson	68.22%	67.16%	85.27%	71.06%	46.38%
Smart-fluidnet	88.27%	87.14%	91.36%	86.47%	91.05%

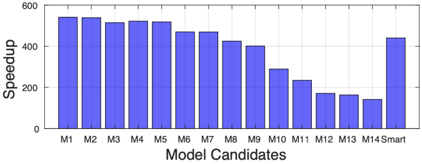

Figure 10: Performance (execution time) for the Tompson's

Evaluation

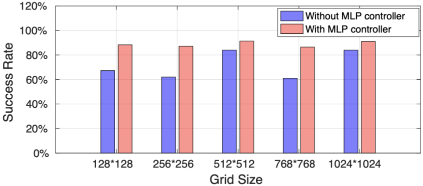

Figure 12: Success rate of reaching target quality with or without using MLP.

Table 4: Resource usage of different methods.

Methods	FLOP (single step)	GPU Memory
PCG	~1,250 M	332 MB
Tompson	243.79 M	299 MB
Smart-fluidnet	110.97 M	1,069 MB