FEBRUARY 2019

OUTLINE OF A NEW COMPUTATIONAL ENGINE FOR SWMM

The National Center for Infrastructure Modeling and Management at the University of Texas at Austin
A research and outreach center funded by the US Environmental Protection Agency

BEN R. HODGES
Director of Research for NCIMM

A. CHARLES ROWNEY
Director of Operations for NCIMM

EHSAN MADADI-KANDJANI
Post-doctoral scholar

ICWMM 2019 TORONTO

Ben R. Hodges, Ph.D.

Director of Research,
National Center for Infrastructure Modeling and Managment
University of Texas at Austin

B.S. Marine Engineering (U.S. Merchant Marine Academy)
M.S. Mechanical Engineering (George Washington University)
Ph.D. Civil Engineering (Stanford University)

Developer of ELCOM 3D hydrodynamic model,
Research areas include: numerical algorithms; modeling lakes, rivers and estuaries; oil spill modeling; upscaling high-resolution topography; hydraulic of curb inlets; supersaturated dissolved gasses below high dams.

Acknowledgement and disclaimer

This presentation was developed under Cooperative Agreement No. 83595001 awarded by the U.S. Environmental Protection Agency to The University of Texas at Austin. It has not been formally reviewed by EPA. The views expressed in this document are solely those of the authors and do not necessarily reflect those of the Agency. EPA does not endorse any products or commercial services mentioned in this publication.

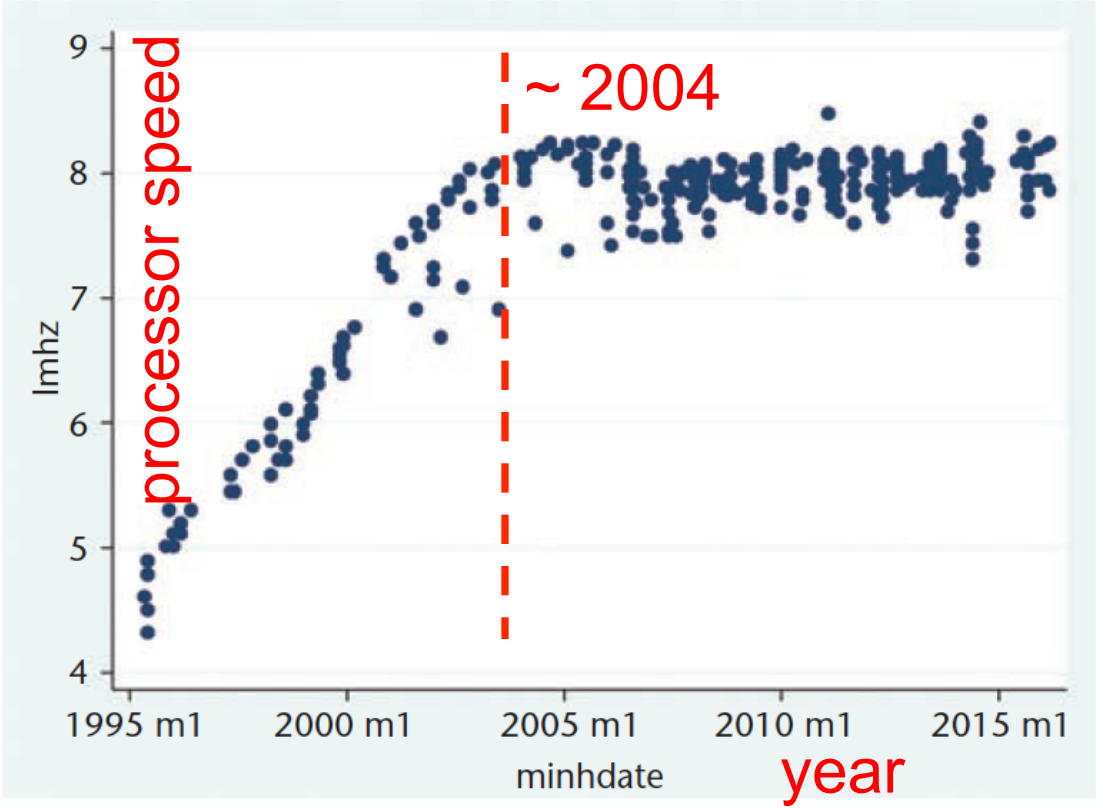
Our goal:

Building a new SWMM computational engine that is

- fast,
- stable,
- conservative,

for multi-thread parallel computation (i.e. order of 1000s of threads) on a desktop computer or cloud server

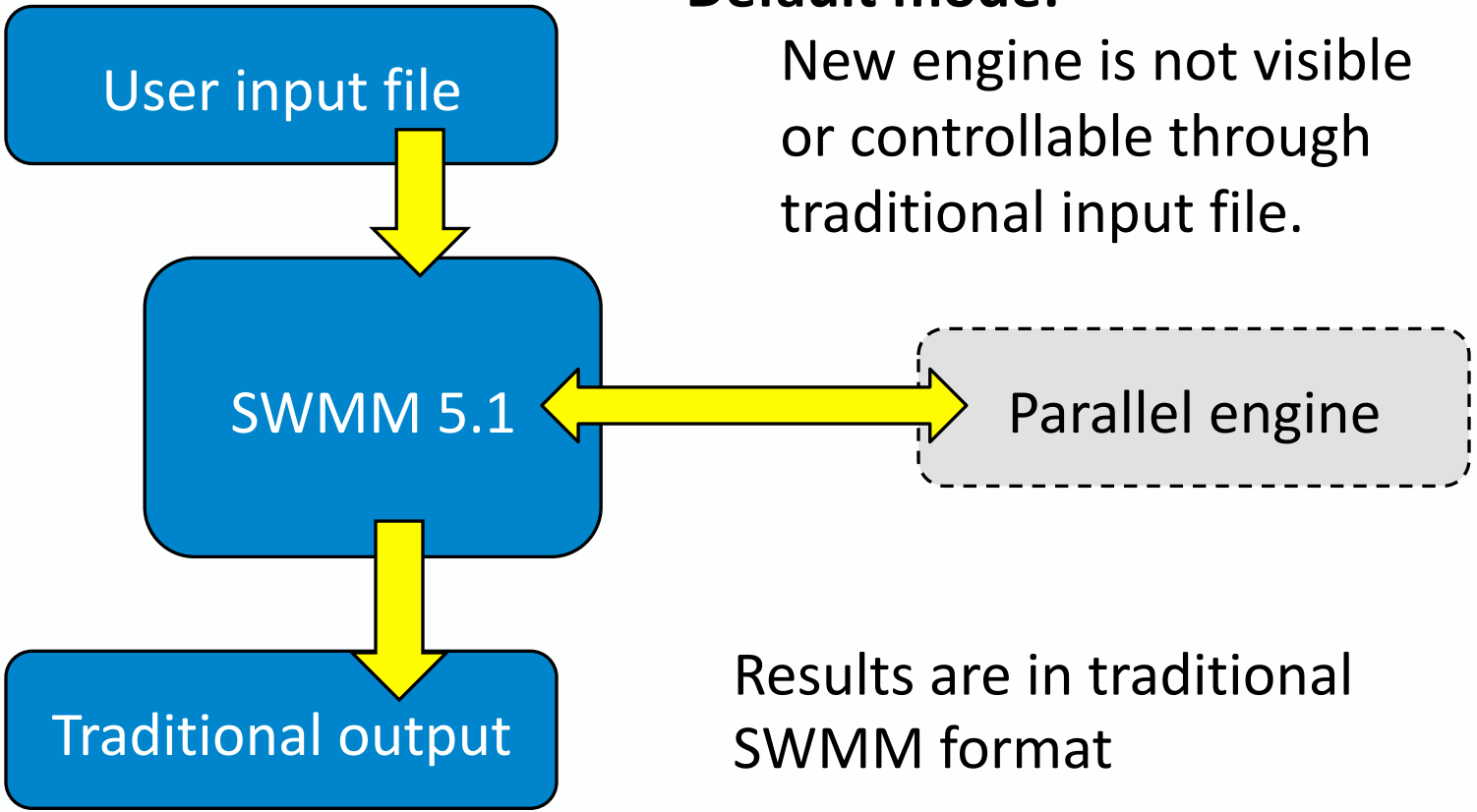
Why multi-thread computational engine?



Processors aren't getting faster, they are just getting more parallel threads.

Flamm, K. (2017) "Has Moore's Law been repealed? An economist's perspective", *Computing in Science & Engineering*, Mar-Apr. 2:29-40

Basic operation

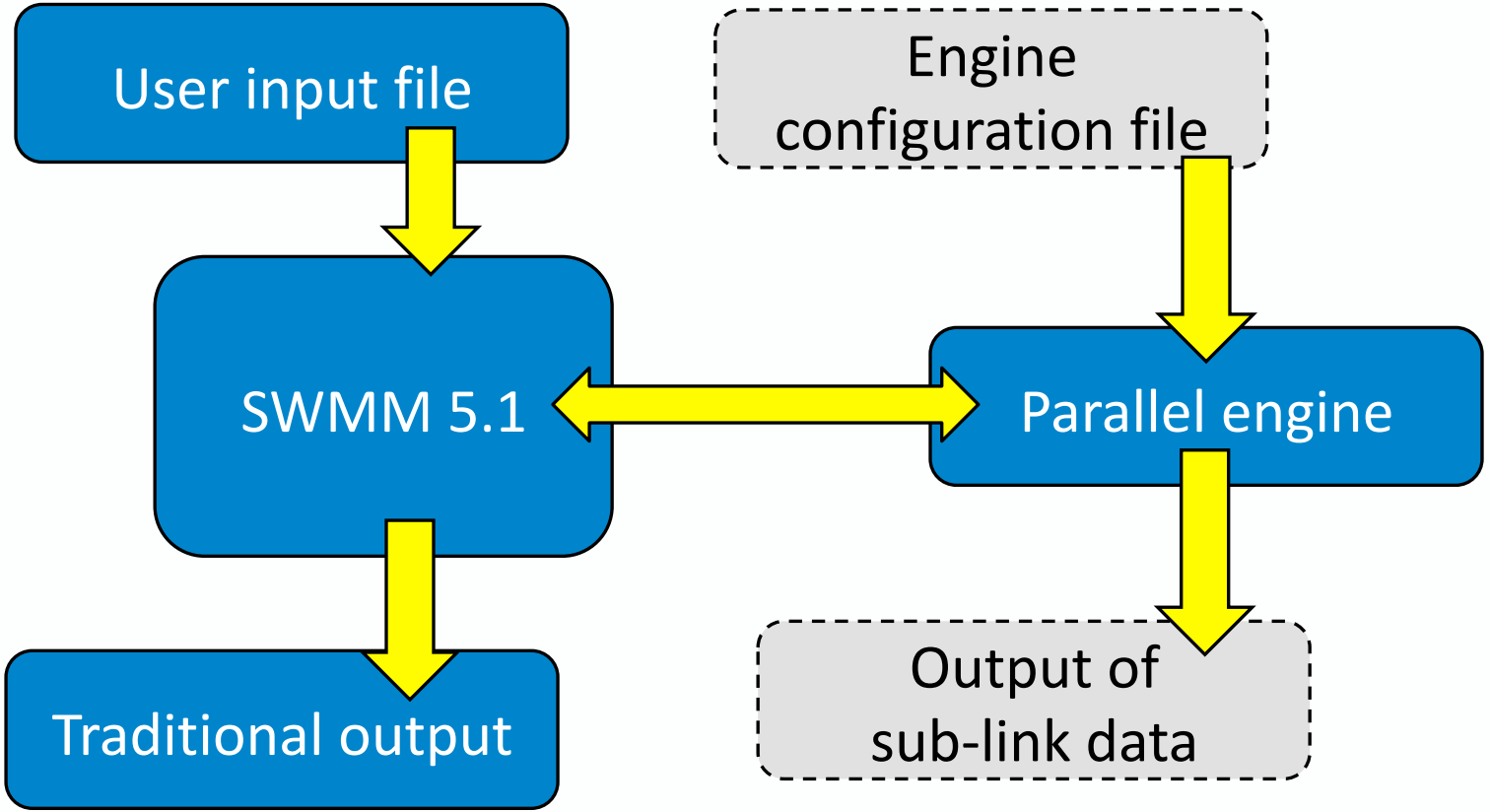


Default mode:

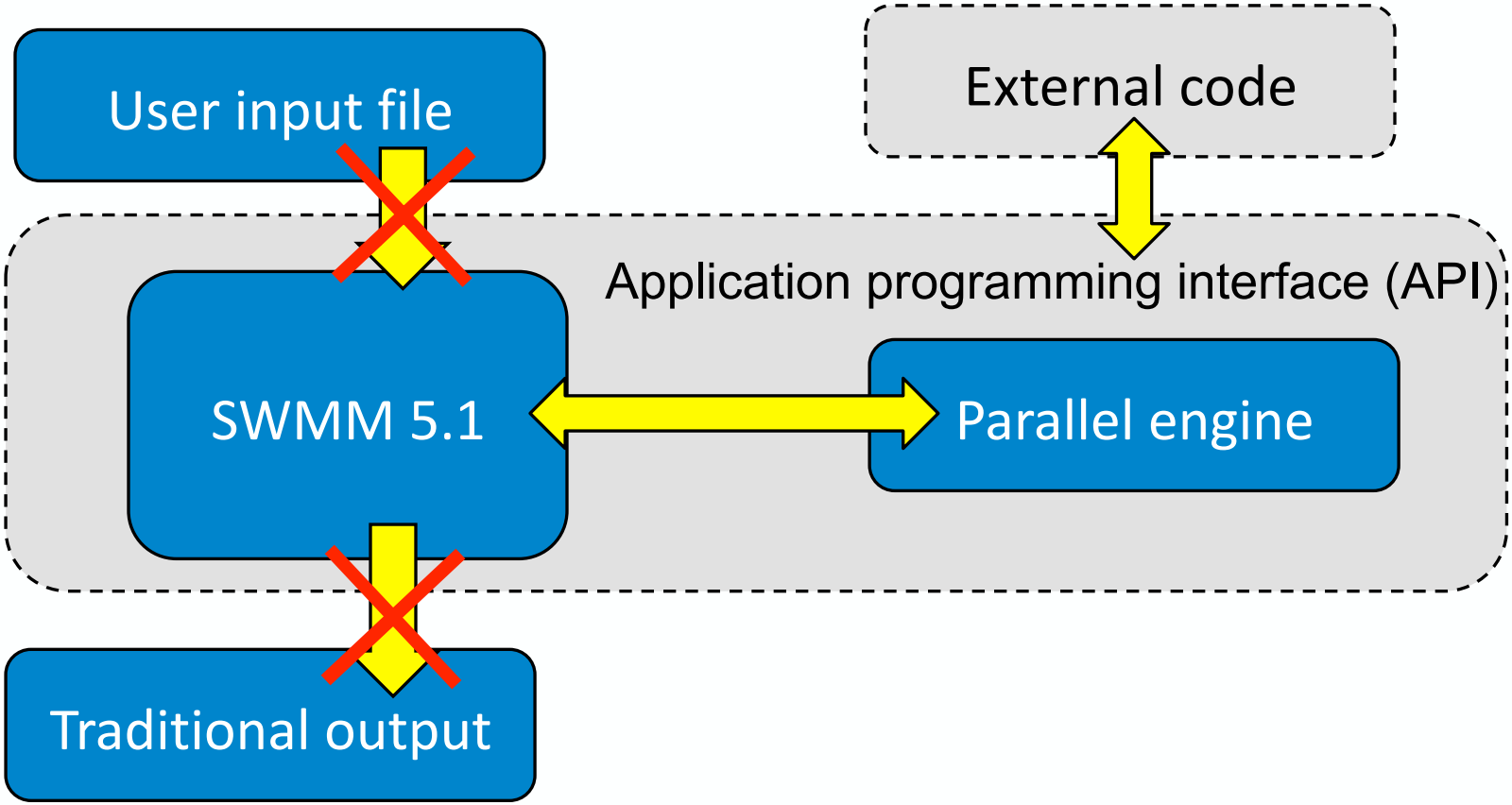
New engine is not visible or controllable through traditional input file.

Results are in traditional SWMM format

Advanced user operation



Programming mode



Summary of engine

- Initial focus on hydraulic solution.
- Finite-volume algorithms.
- Sub-discretization of links. **IMPORTANT – next slide!**
- Nodes as physical elements rather than infinitesimal points.
- Face interpolation uses a time-scale approach.
- Saint-Venant solution using Runge-Kutta 4^{th} order advance.
- Surcharged pipe using Artificial Compressibility with RK4 march in pseudo-time and 3^{rd} order backwards stencil in real time.
- Closely controlled mass conservation.

Why subdivision of links into elements/faces?

- Many stability/conservation problems in SWMM are associated with different link lengths.
- Subdivision is automated − user does not have to decide how to subdivide − input remains link/node.
- Program has control over time-stepping based on subdivisions
- Transitions are always at faces between elements:
  - An element is either open channel or closed conduit.
  - An open channel element is either subcritical or supercritical

Forward to Fortran...

- Parallel engine is being written in CoArray Fortran 2008
- Examined pros/cons of using C-MPI versus CoArray Fortran
- Major advantage of C-MPI is for large-scale 3D codes (e.g. climate modeling) where:
  - connection topology is uniform and predictable
  - number of elements is at the limit of the computer memory.
- CoArray Fortran
  - "Single Program Multiple Data (SPMD)" – copies data for each processor.
  - Processor configuration is determined at runtime (no custom compile)
  - Flexible for workstations, supercomputer, or cloud.
  - Parallelization with minimal compiler directives or checking.

Single Program Multiple Data (SPMD) parallel

- Each processor has a copy of ENTIRE data set in local memory.
- Each processor works only on a PORTION of the data.
- Communication (sync) is controlled across processors to improve efficiency.
- Optimal parallelization when main operating data can be held in memory cache. **More on this in a few slides.**
- Stormwater networks are ideal for SPMD as their relatively small size (compared to global climate circulation)

More on SPMD (single program multiple data) using CoArray Fortran

- With N cores the system is automatically divided into N parts.
- The entire data space (S) is duplicated N times, but each core only computes and updates its "own" S/N data.
- Each core has access to data outside of its "own", but that data is only synchronized as needed.
- If the data updated by an individual core can be stored in the L3 cache, then the communication to memory (bus time) can be reduced.

Where we can go...

- 32 cores per processor (AMD Ryzen Threadripper)
- processor has 64 MB L3 cache (fast access memory)
- Each 64 MB cache stores 8×10^6 numbers.
- With 100 pieces of data per element the cache handles 80,000 elements ***without*** communicating with main memory.
- So each core handles only 2500 elements if all stored in cache.
- 10^6 elements (huge SWMM system) requires only 13 processors (400 cores) for ***ALL*** the system to be in the local L3 cache.
- Note that 10^6 elements only takes about 1 GB of memory − which is easy to get with fewer processors. But the key parallel speed-up will be by effectively using the processor cache.

Advantage of Fortran 2008 with array processing

Say goodbye to do-loops...

```
do i = 1,N
   if B(i) < C(i) then
      A(i) = B(i) + C(j)
   endif
enddo
```

```
where B < C
      A = B + C
endwhere
```

The array code...

- automatically parallelizes
- is less buggy

CIMM

Data structure for SWMM engine

- All data stored in 2D arrays − here's real data

Element ID	Volume	Velocity	Surface Elevation	Cross-section area	...
1	1053	0.43	3.4	4.75	
2	953	0.44	3.3	4.8	
3	938	0.39	3.34	3.24	
4	1044	0.21	3.31	5.14	
5	1021	0.15	2.9	6.24	
6	965	0.04	2.85	9.5	
...	...	...	...		

Separate 2D arrays for different types of data

- Real data (e.g. volume, velocity, area)
- Integer data (e.g. neighbor maps, geometry type, roughness model)
- Logical data (e.g. IsSmallVolume, IsClosedConduit)

Row index in any array gives all the values for a single element (or face),

dataR(314 , :) stores all the real data for element 314

Column index gives data across all elements (or faces) of a given type

dataR(: , velocity) stores the velocity in all the elements

This structure makes data easy to push/pull through an API.

CIMM

Pointers used for code readability

Computing flowrate without pointers:

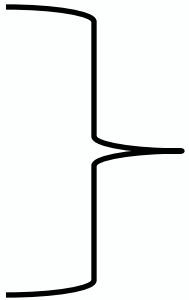
dataR(: , flowrate) = dataR(: , area) * dataR(: , velocity)

With pointers

Q => dataR(: , flowrate)

A => dataR(: , area)

V => dataR(: , velocity)



Q = A * V

Maps used for neighbor faces

mapUp => dElemInt(: , mapUpstreamFace)

mapDn => dElemInt(:, mapDownstreamFace)

FreeSurfaceUp => dFaceReal(mapUp , FreeSurfaceElevation)

FreeSurfaceDn => dFaceReal(mapDn , FreeSurfaceElevation)

ElementLength => dElemReal(: , Length)

thisGradient => dElemReal(: , FreeSurfaceGradient)

thisGradient = (FreeSurfaceUp − Free SurfaceDn) / Element Length

Readable code, no dangling pointers (aliases only), no memory allocation during execution, all arrays work as coarrays for distribution across processors.

CIMM

Separate arrays for channel/pipe elements, junction elements, and faces between elements

- Channel/pipe elements store one value for each parameter (e.g. there is only one flowrate in a channel)
- ***Junction elements store multiple values*** of some parameters (e.g. different flowrates in each branch of a junction)
- Faces are the connections between any two elements.
- Junctions are elements with multiple faces

  -- channel/pipe elements have only two faces.

This is a major (but subtle) change from the link-node system

CIMM

Link-node versus finite-volume

- **In link-node system:**
  - NODE has many connections
  - LINK connects to exactly 2 nodes.
- **In finite volume (FV) system**
  - FACE connects to exactly 2 elements
  - ELEMENT can connect to any number of faces.

Advantage of FV is that manholes are naturally allowed different heads for different branch pipes, depending on water elevation in the manhole itself.

Take-home message

Fortran 2008 (and soon 2018) is the way parallel coding should be − transferable across machines, transparent to user, and easy to code.

Let's compare "Hello World" in C-MPI and Coarray Fortran...

Hello world in C with MPI

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv) {
        MPI_Init(NULL, NULL);
        int world_size;
        MPI_Comm_size(MPI_COMM_WORLD, &world_size);
        char processor_name[MPI_MAX_PROCESSOR_NAME];
        int name_len;
        MPI_Get_processor_name(processor_name, &name_len);
        printf("Hello world from processor %s, of %d \n",
                processor_name, world_size);
        MPI_Finalize();
}
```

Adapted from http://mpitutorial.com/tutorials/mpi-hello-world/

Hello world in Coarray Fortran 2008

```
program hello_world
implicit none
write(*,*) 'Hello world from processor ' , this_image() , ' of ' , num_images()
end program hello_world
```

From: http://annefou.github.io/Fortran/coarrays/coarrays.html

Contact information

- Ben R. Hodges : hodges@utexas.edu
- http://www.ncimm.org/