VV/UQ Implications of Performance Models for Large Scale Computing

Ryan G. McClarren
Texas A&M University
PSAAP V&V/UQ Meeting August 2012

A Performance Model Enables VV/UQ Insight

- In this talk I will argue that performance models are more than just a way to predict how a given algorithm, code or method will scale on a particular machine.

- This information can be vital in planning run sets for UQ investigations.

- A perhaps novel application of a performance model is for *algorithmic* verification.

- I'll talk about a particular example of this:
  - Radiation (or other particle) transport algorithms for large scale, parallel computing.

- In the end, I hope to motivate the investment in performance models for large scale codes in the VV/UQ context.

The was a large collaborative effort

- TAMU Nuclear Engineering: Marvin Adams, Daryl Hawkins, Michael Adams
- TAMU Computer Science: Timmie Smith, Lawrence Rauchwerger, Nancy Amato
- Hawkins, Smith, et al., "Efficient Massively Parallel Transport Sweeps", to appear in *Transactions of the American Nuclear Society*

Allocation of computational resources can be a difficult challenge

- In a UQ campaign it is often the case that the size of the campaign is limited by the available computational resources.

- In many UQ strategies one desires to complete many different simulations to study the importance of important parameters.

- This is further complicated by the fact that one often doesn't know how long a given run will take.
  - This is often partially due to the fact that the run sets are meant to explore input space---likely in regions of parameters you've never tried before.

- Therefore, you might not know how many runs you can afford.

- At CRASH this has lead to some clever approaches to right-size our run sets.
  - For a run set of 3D rad-hydro calculations, the design consisted of a Latin-Hypercube design of size X plus two smaller sets to fill in the design.

A solution can be a robust, flexible performance model

- For a given problem *and* computer if one knows
  - The problem
    - Size (Degrees of freedom, number of time steps, etc.)
  - The machine
    - Clock speed
    - Communication latency
    - Number of nodes/procs

- One can, in many instances, predict the performance and, as a corollary the run time, for a given problem.

- Specifically, we are talking about first principles type performance models where we aggregate the cost of several smaller pieces of the calculation.

- One can, in principle, develop statistical models for performance where the runtime model is inferred from actual results.
  - These can be useful in the absence of a first principles model, but can have problems outside the domain of previous runs.
  - This model may be less useful for algorithmic verification.

Performance can be a verification metric

- With a performance model, it is possible to test the implementation of the parallel algorithms.
- We call this algorithmic verification.
- Just like in a verification exercise where one looks at code convergence an verifies that the convergence rate is correct
  - One can look at the parallel performance and verify that the scaling is as expected.
  - This can point to failing in the implementation, machine, runtime environment, etc.
- Without a performance model it's easy to attribute anomalous scaling to things out of the developer/user's control.
- Of course, one needs a believable performance model.

AN EXAMPLE OF THE BENEFIT OF A PERFORMANCE MODEL

A performance model for parallel, particle transport calculations.

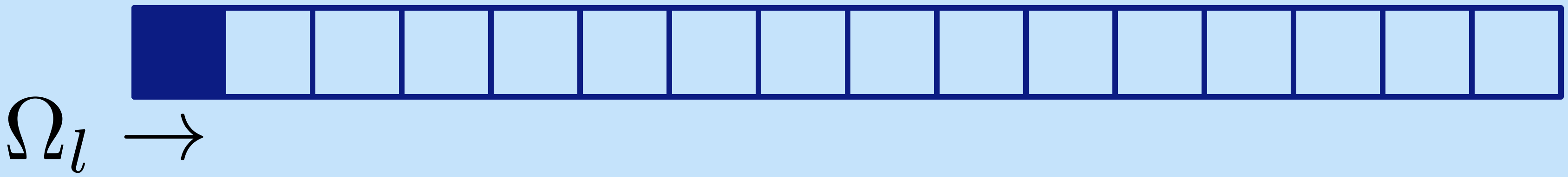
- Particle transport calculations are often the most expensive piece of simulation.

- This is because the kinetic density of particles varies over a seven-dimensional phase space (3 space, 3 momentum, and 1 time)

- The discrete ordinates method is the most common deterministic transport method.
  - This method solves a series of advection-reaction equations of the form
  $$\partial_t \psi_l + \Omega_l \cdot \nabla \psi_l = C(\psi_1, \psi_2, \ldots, \psi_L)$$
  - These equations are advection equations with constant speed, which can be solved via a simple iteration scheme
  $$\partial_t \psi_l^{n+1} + \Omega_l \cdot \nabla \psi_l^{n+1} = C(\psi_1^n, \psi_2^n, \ldots, \psi_L^n)$$
  - In practice, more complicated iterations are used, but they all have the same underpinnings.

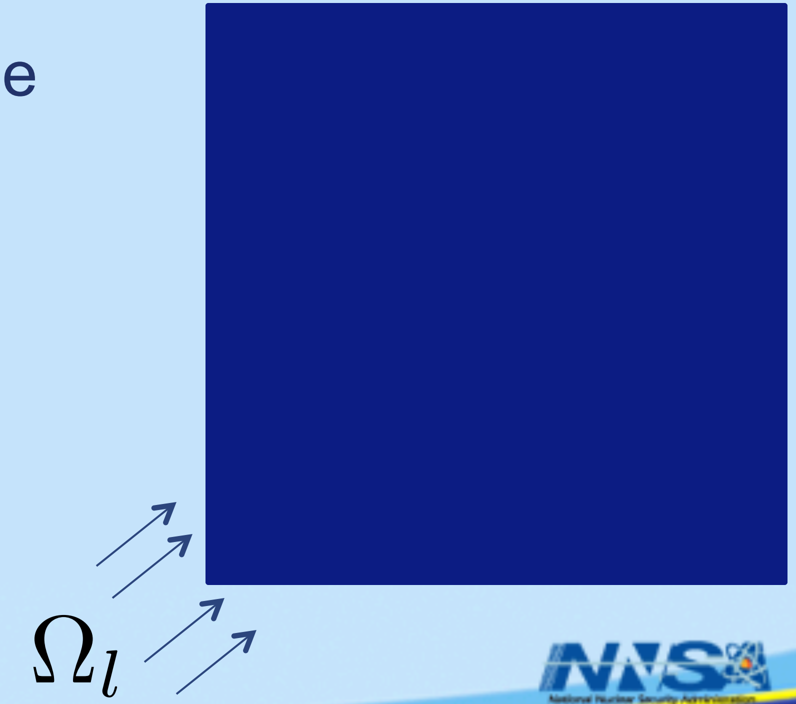
Each iteration involves a “sweep” across the grid

1-D Example



2-D Example

Starting at the boundary, the computation moves across the grid.

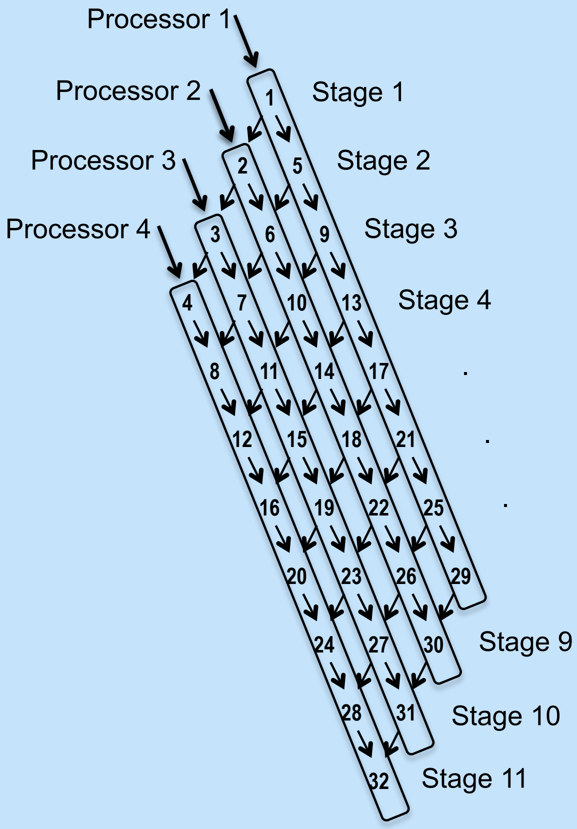


The sweeps have a particular dependencies for parallel processing

- To compute a sweep in parallel using spatial domain decomposition, there is a particular order in which processors can do their work.

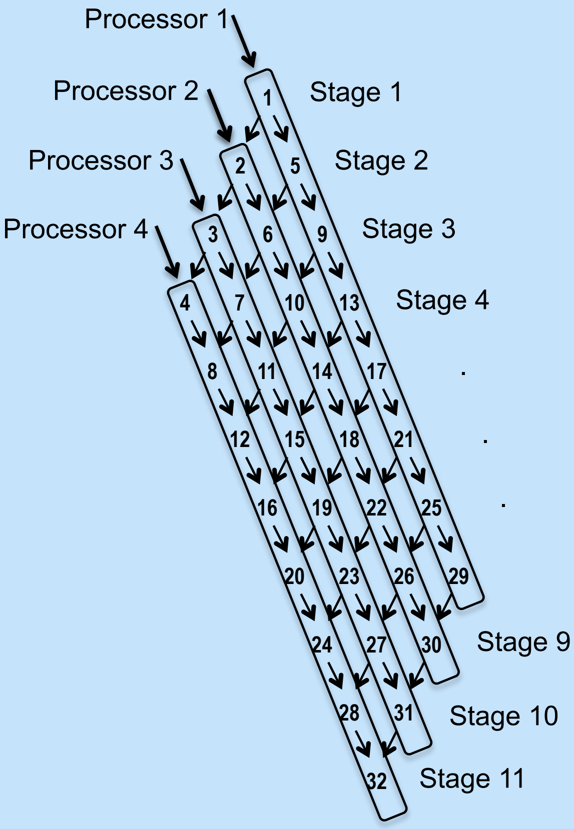
- This can be represented in a task dependency graph.

- In the example, notice that processor 4 is idle in steps 1-3, and processor 1 is idle in stages 9-11.



Improving efficiency: pipefill

- The idleness of processor 1, can be remedied by having it start on the next angle in the same octant.
- Then when stage 11 is complete, processor 4 can begin without being idle.
- This pipe filling helps efficiency but has it's limits.
- The task graph width scales as $P^{2/3}$



Optimal Sweep Algorithms

- A sweep algorithm is defined by its
  - Partitioning (how the domain is divided among procs)
  - Aggregation (grouping of cells, directions, energy groups into tasks)
  - Scheduling (choosing what task to execute if several are available)
- It is possible to choose the best possible parameters for the algorithm so that it is provably optimal.
- This algorithm has been implemented in the `PDT` code developed at Texas A&M and built on the Standard Template Adaptive Parallel Library (STAPL).

Parallel Efficiency for Optimal Algorithm

- For a 3-D problem with N_x x N_y x N_z cells, partitioned with a P_x x P_y x P_z processor layout, with G groups and M directions, and
- With tasks containing $A_xA_yA_z$ cells, A_m directions, and A_g groups.
- The optimal weak scaling efficiency is

$$\epsilon_{opt} = \frac{1}{\left[1+\frac{P_x-r_x+P_y-r_y+K_z(P_z-r_z)}{8MGN_{zp}/(A_mA_gA_z)}\right]\left[1+\frac{T_{comm}}{T_{task}}\right]}$$

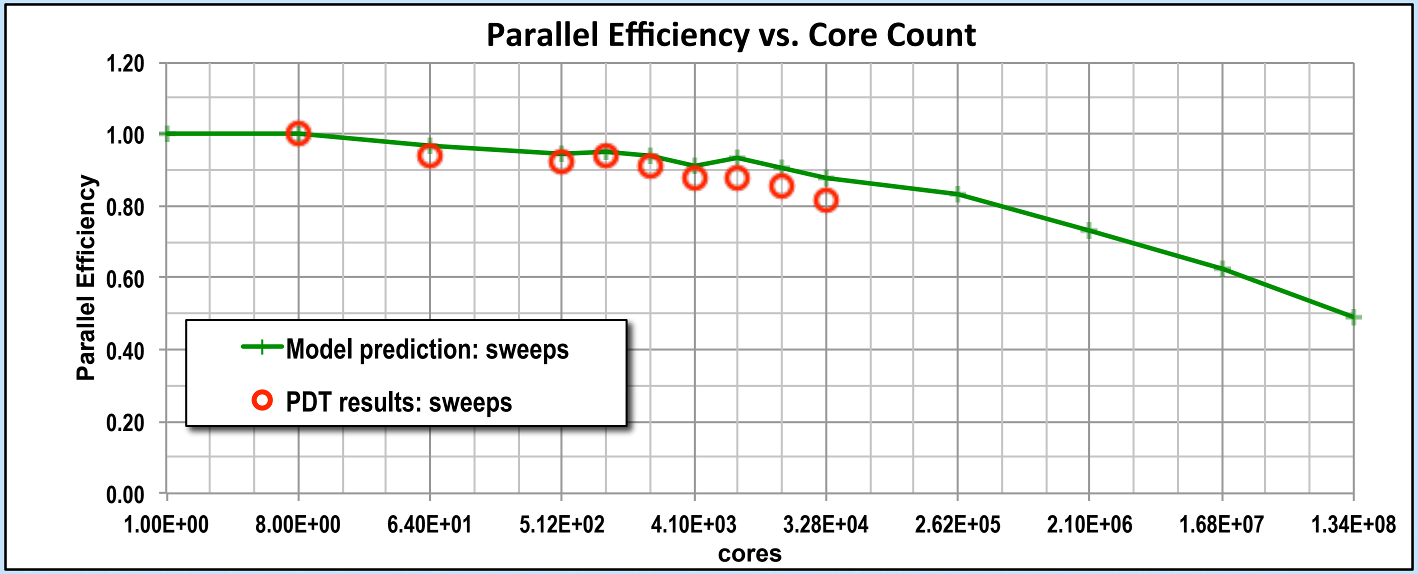
- Where

$$N_{zp} = N_z/P_z \qquad K_z = N_{zp}/A_z$$

$$r_i = \begin{cases} 1 & P_i \text{ even} \\ 2 & P_i \text{ even} \end{cases}$$

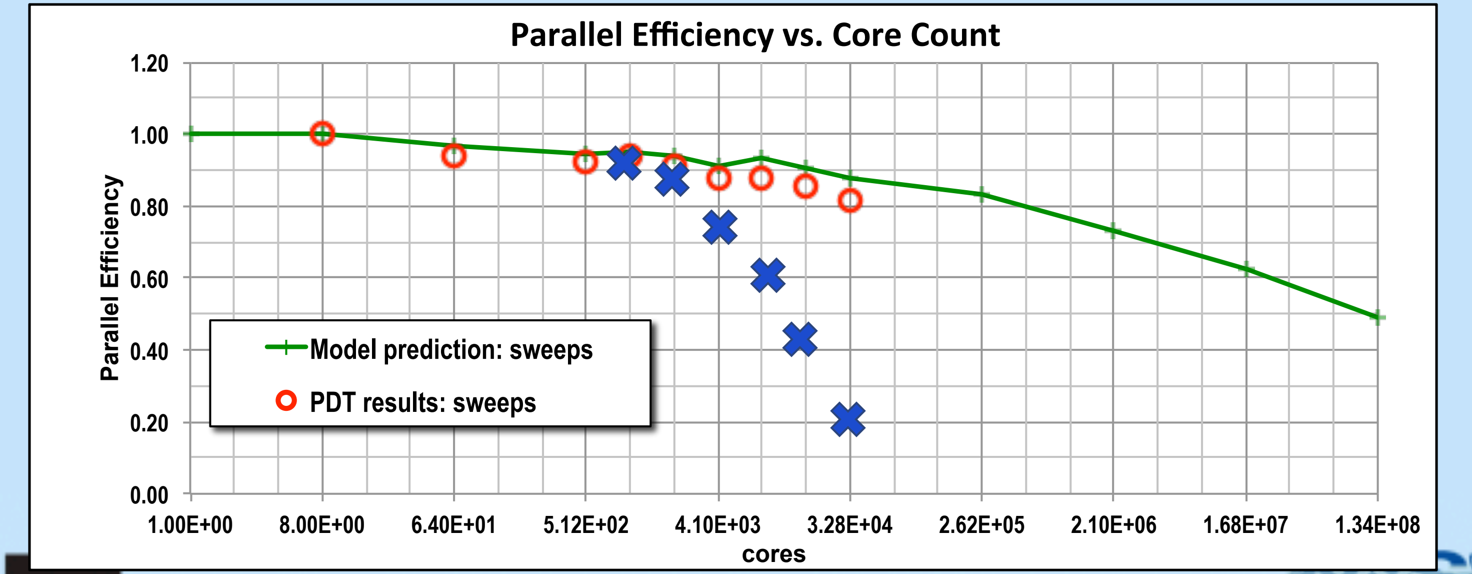
The implementation of the optimal schedule

- We've used particular test problem designed to test parallel scaling (the Zerr-Azmy problem).
- Constant 4096 cells/core; **results normalized to 1 processor performance.**
- Model predicts above 70% efficiency at 1 million cores



It wasn't always so rosy

- In the graph, at 32k cores we are achieving above 80% efficiency.
- Does not exactly agree with model, but the slope appears to be the same, and the dips and bumps in the model appear.
- Not that long ago, the results looked much worse.
- Given that we had a performance model, we knew there was an O(P) communication somewhere in the implementation.



Performance models are part of the VV/UQ discussion

- Can we perform the runs we want to do?
- Are we getting the right efficiency (do we know?)?
- Is there a bug in our parallel implementation.
- A performance model helps to answer all of these questions.