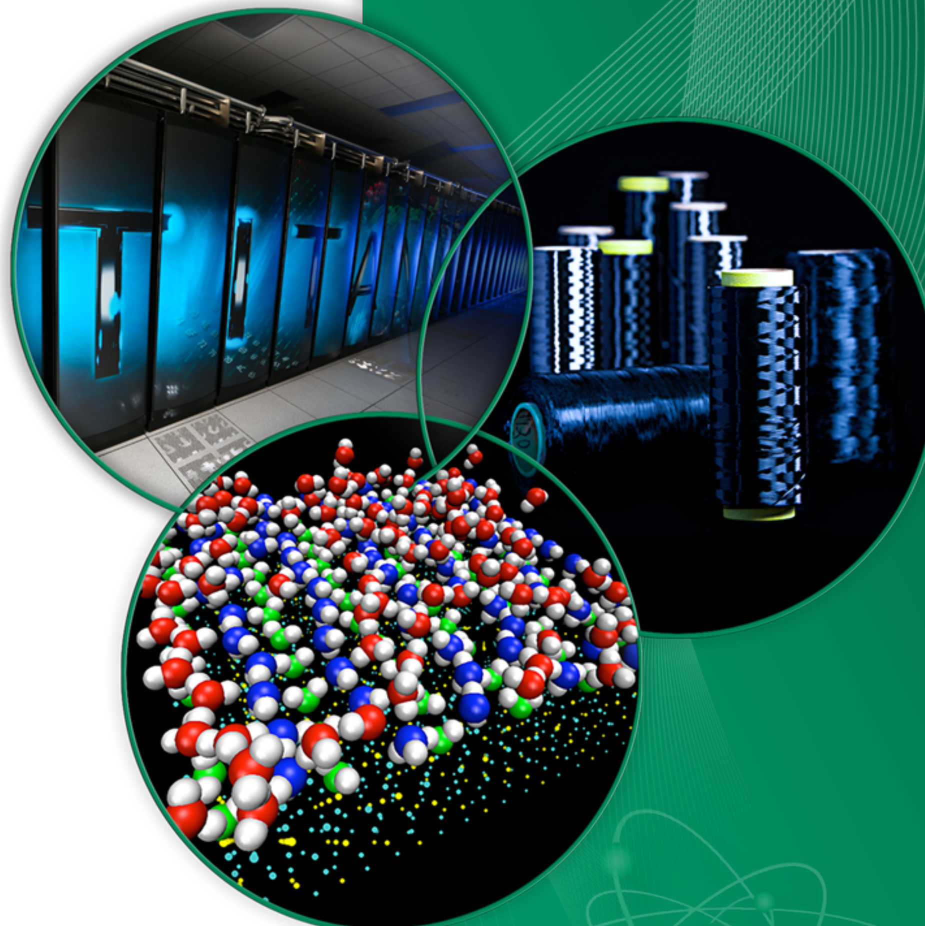
Tracking a Value's Influence on Later Computation

Philip C. Roth

Future Technologies Group
Computer Science and Mathematics Division

Oak Ridge National Laboratory
Oak Ridge, TN USA



Value Influence Tracking

- Understanding how values are propagated through time and space helps us recognize:
  - Inefficient/unnecessary computation (e.g., cut-off distance)
  - Incorrect computation (e.g., this value should have been accessed)
  - Values for which high reliability is needed

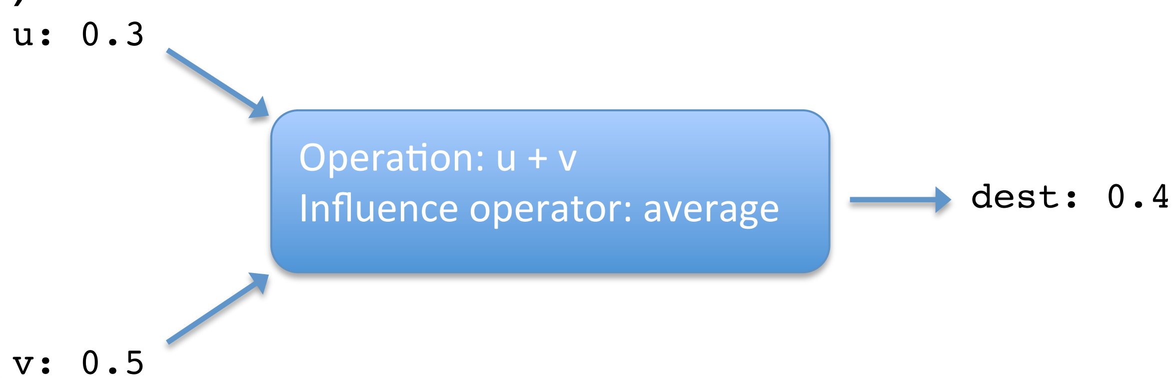
- A value is not the same as a variable: a variable holds a value at a given point during a program run

- We are researching an empirical approach for tracking how a value contributes to later computation, called its *influence*

- We are evaluating this approach by implementing the Value Influence Tracking (VIT) tool, and applying it to parallel applications

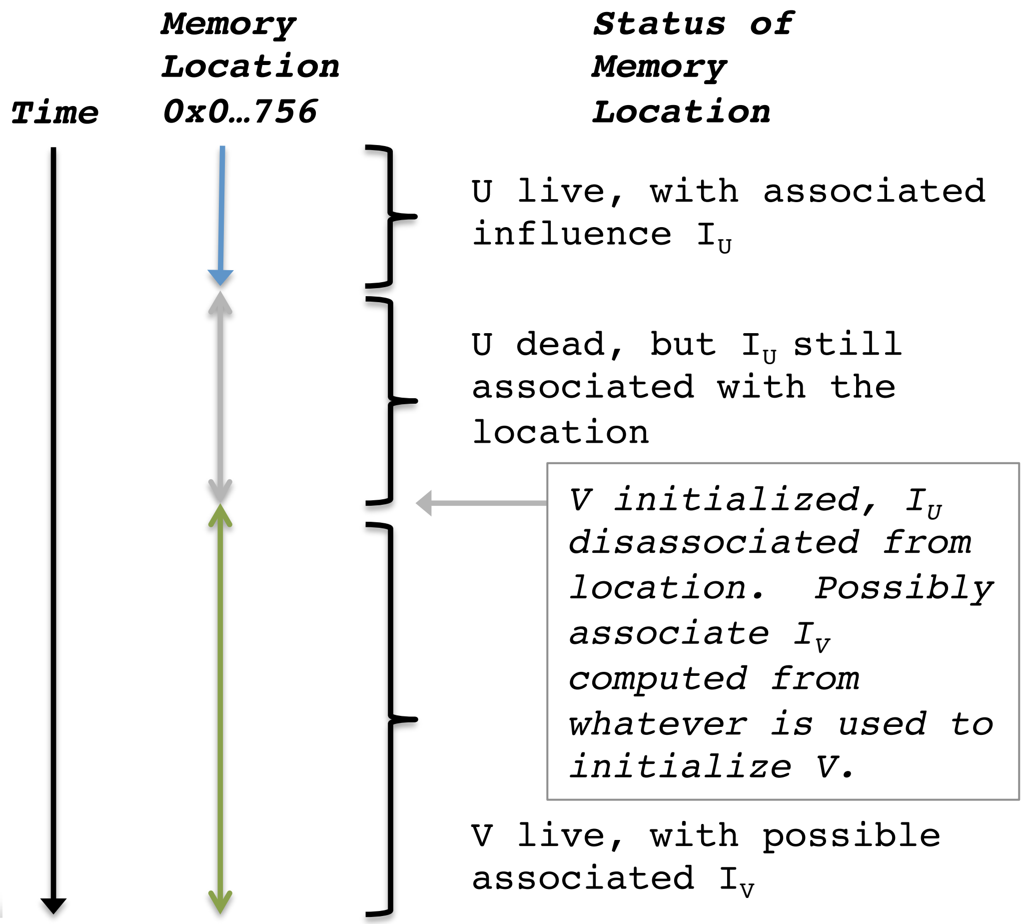
Assigning and Combining Value Influence

- A value's influence is real number in [0,1]
- When value is indicated as being of interest, we assign an influence value of 1
- Values that have not been assigned an influence value are assumed to have influence value of 0
- When a value is combined with other(s) by a program, we combine their influence data and associate it with the output
- Influence operators can be simple (e.g., AND, average) or complex (e.g., max, something depending on input magnitude)



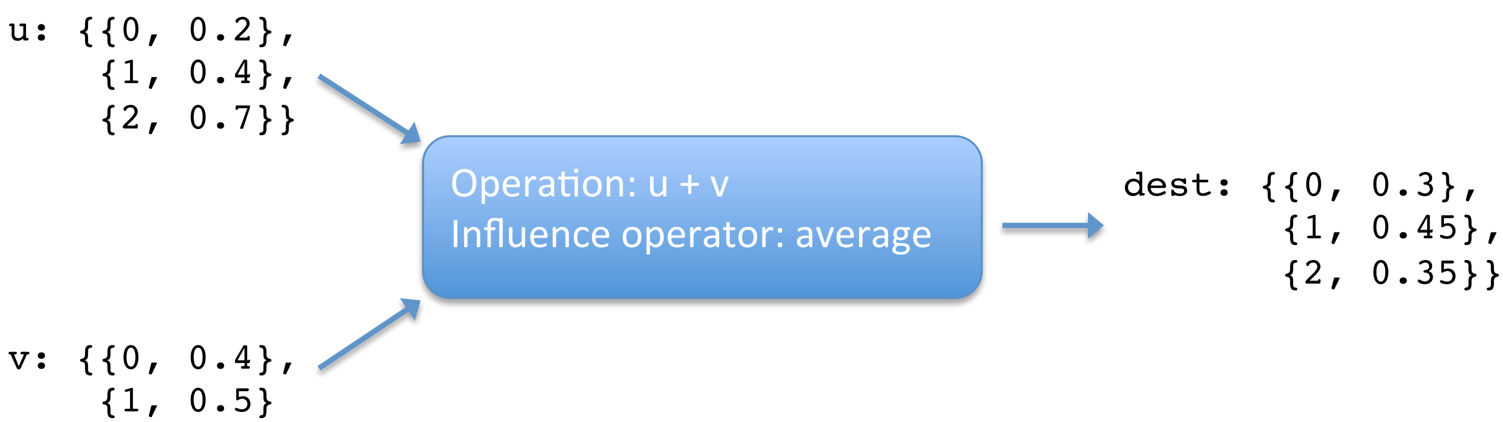
Variable Lifetime

- At different times during a program run, different variables may share the same location (register or memory)
- Our approach respects variable lifetime naturally through location initialization



Tracking Influences for Multiple Values

- We can track influence for multiple values
  - Each variable has a vector of associated influence data
  - Each influence value has a color
- Combining function performs per-color combination, and associates resulting vector to output location



Multithreaded Programs

- We track influence data differently for values in registers and in memory
  - All threads share influence of value in memory, just as all threads share the actual value
  - We use thread-local storage to store influence of values in registers
  - When running thread executes an operation that combines values from registers and produces an output, we use influence data from thread-local storage associated with the active thread
- Thread-local storage plus natural variable lifetime handling provides support for multithreaded programs

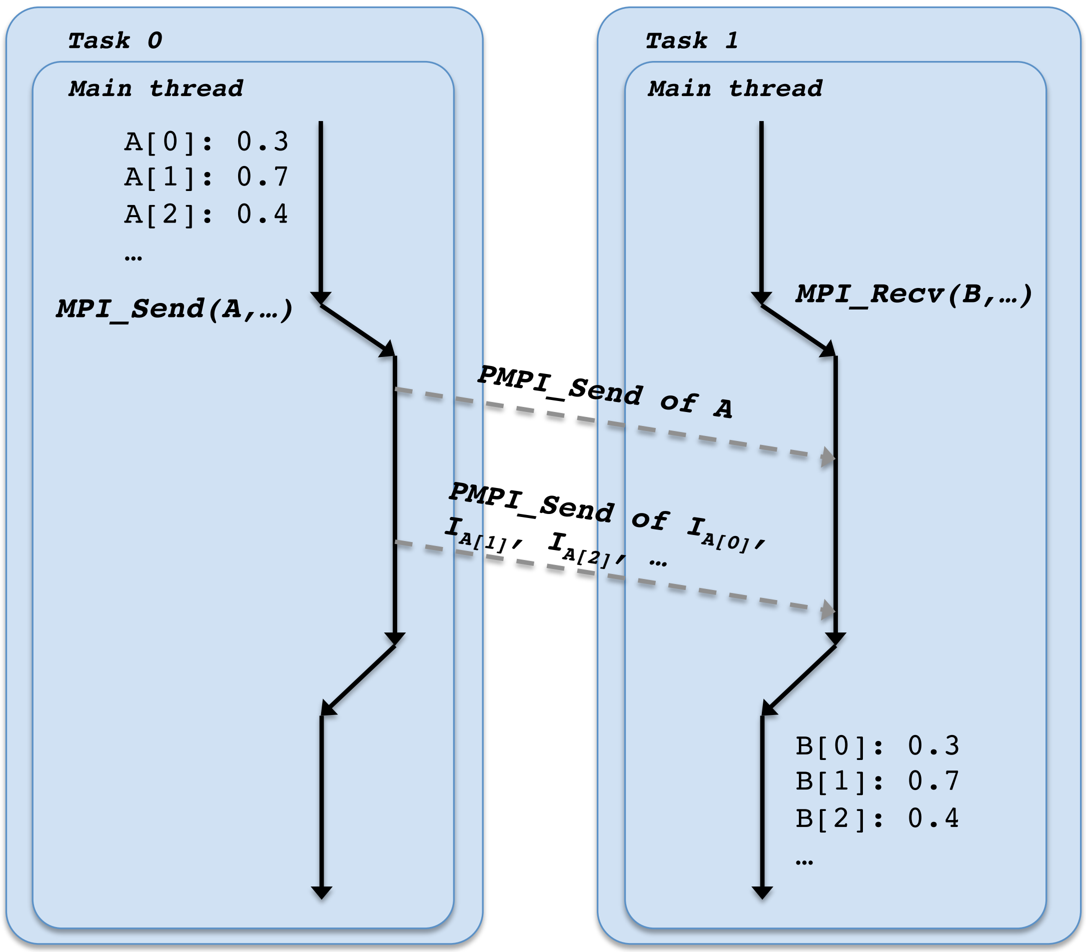
MPI Programs

- Whenever data is transferred from one address space to another, we need to:
  - Know what data is being transferred
  - Know where it is being transferred
  - Know when transfer is guaranteed to be done

- Our approach: use PMPI profiling interface to interpose tool functionality whenever MPI function is called

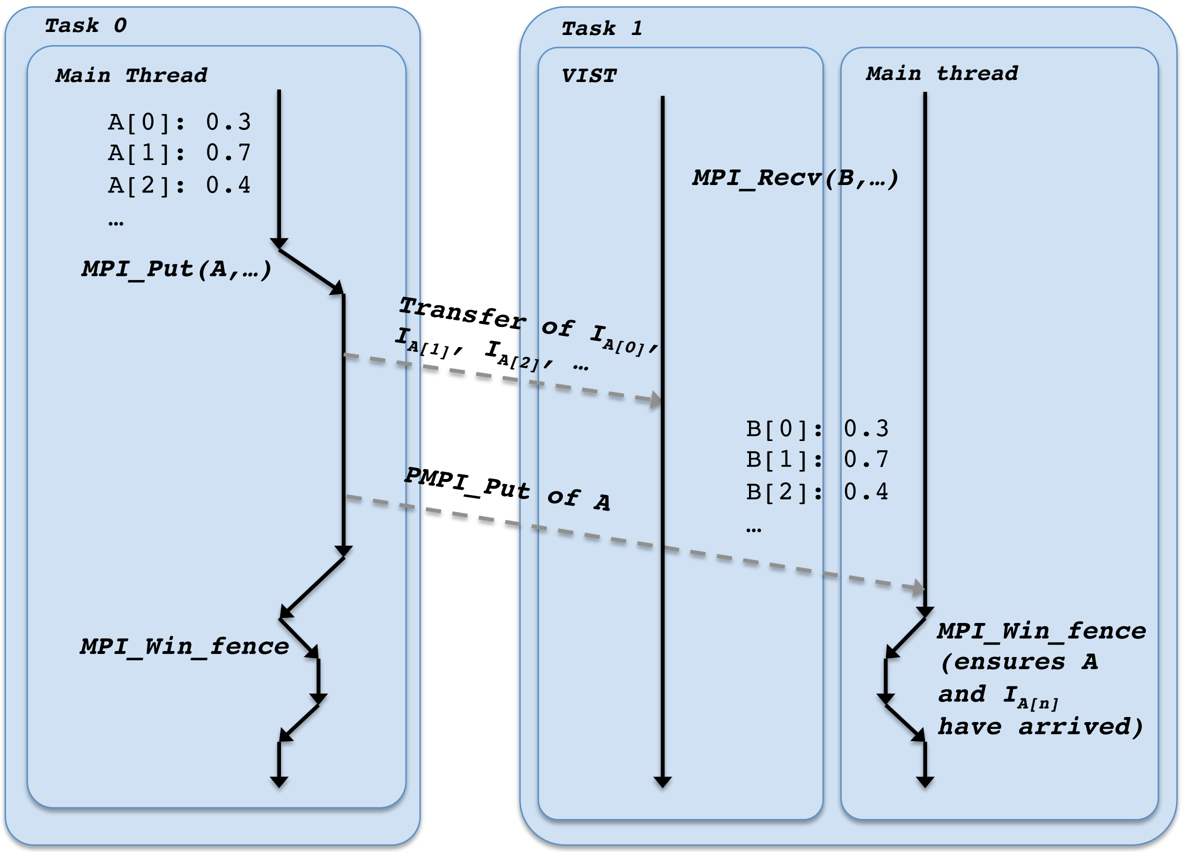
MPI Programs: Two-Sided, Collectives

- Within interposed function, transfer application data and then any associated influence data



MPI Programs: One-Sided

- Needs separate thread (Value Influence Service Thread, VIST) for asynchronous access to influence data
- PMPI_Win_fence needs to ensure value influence data has been transferred before continuiing



Implementation

- VIT: Value Influence Tracker
- Uses *dynamic instrumentation* to propagate influence data
  - Intel Pin: VIT is a *pintool*
  - Uses *trace instrumentation mode*: when executing a program, VIT sees a sequence of instructions to be executed (a “trace”) at a time
  - VIT examines each instruction in the trace to see if it writes to a destination (memory or register); if so, VIT instruments the instruction
  - VIT instrumentation combines influence data from inputs, associates resulting influence value with output

Tool Control

- C-based API for control
  - VIT_Track: to start tracking a value
  - VIT_Report: to output influence data
  - VIT_Reset: to forget any influence data

- Why a C API? Only thing that makes sense:
  - Variable is in scope
  - Variable is live
  - (Hopefully) variable has been initialized

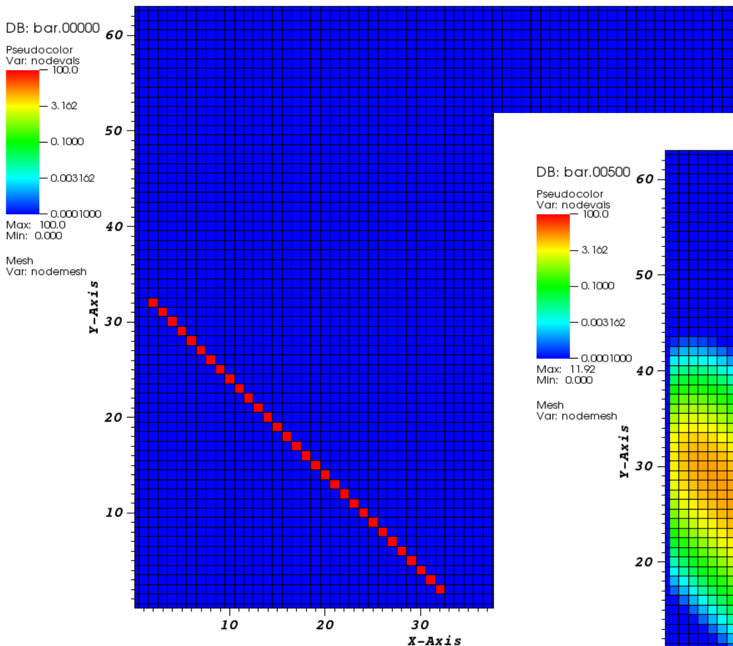
VIT Output

- Sequence of (address, influence) pairs
- Implementation does not currently report influence data for values in registers
- With care, can convert address into symbolic name using symbol information and something like GNU's Binary File Descriptor library or SymtabAPI library

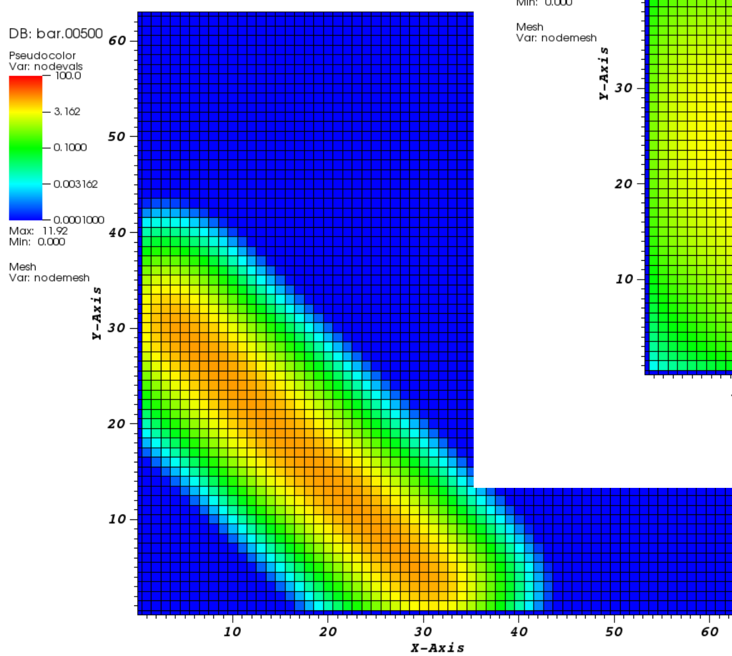
```
VIT_Report called
In memory:
0x000000000192cc98: 1.80845e-05
0x000000000192ce90: 1
0x000000000192ce98: 0.000711323
0x000000000192cea0: 1.20563e-05
0x000000000192d098: 6.02816e-06
0x0000000001934ea0: 0.166667
0x0000000001934ea8: 0.00347222
```

Case Study

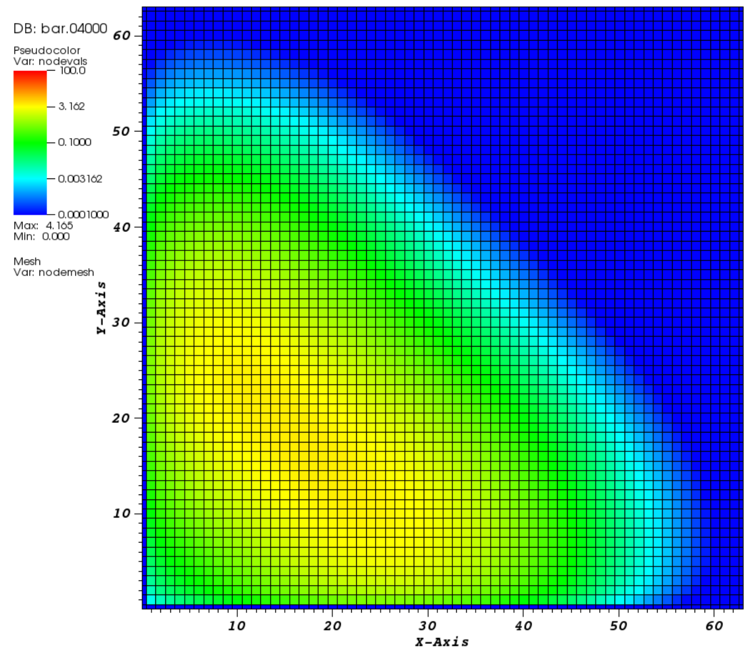
- Used VIT with 2D heat transfer application
  - Explicit discretization
  - Forward time-centered space with 5-point stencil
  - Double-buffered



Before time steps



After 500 time steps



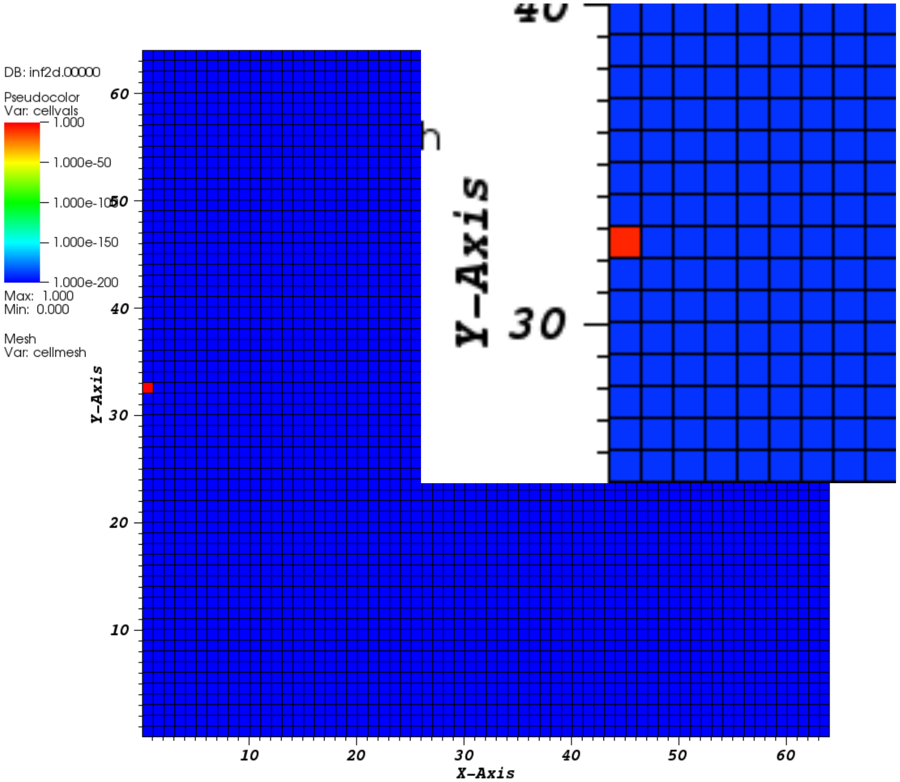
After 4000 time steps

Case Study (II)

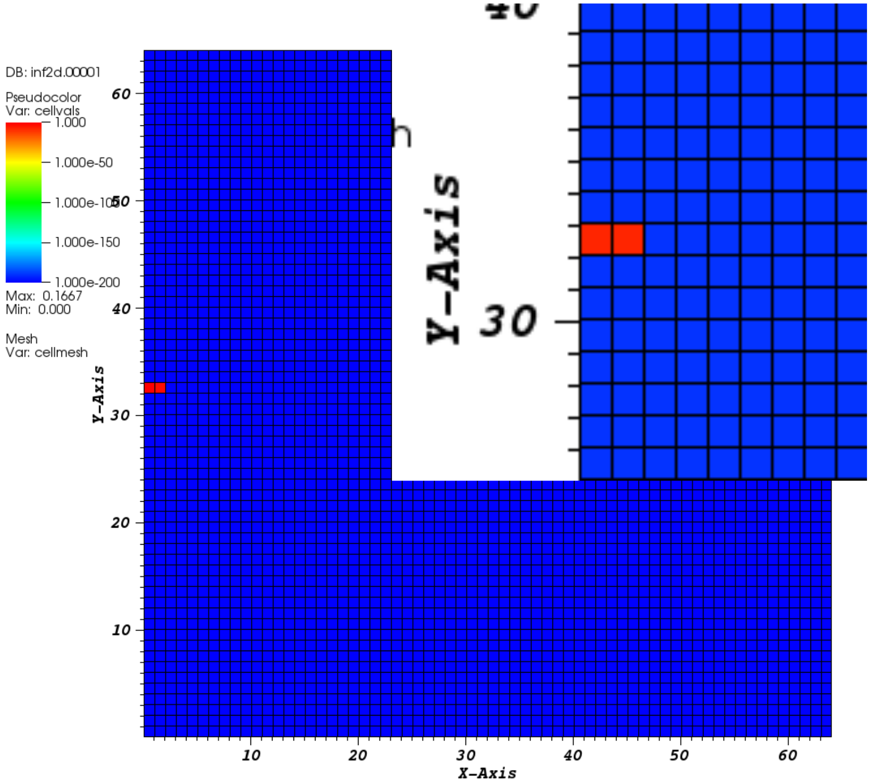
- For this test, tagged a boundary value and “watched” its influence propagate through the array
  - Each time around main loop, used API to dump current influence data
  - Visualized this data using VisIt

Case Study (III)

- Before time steps

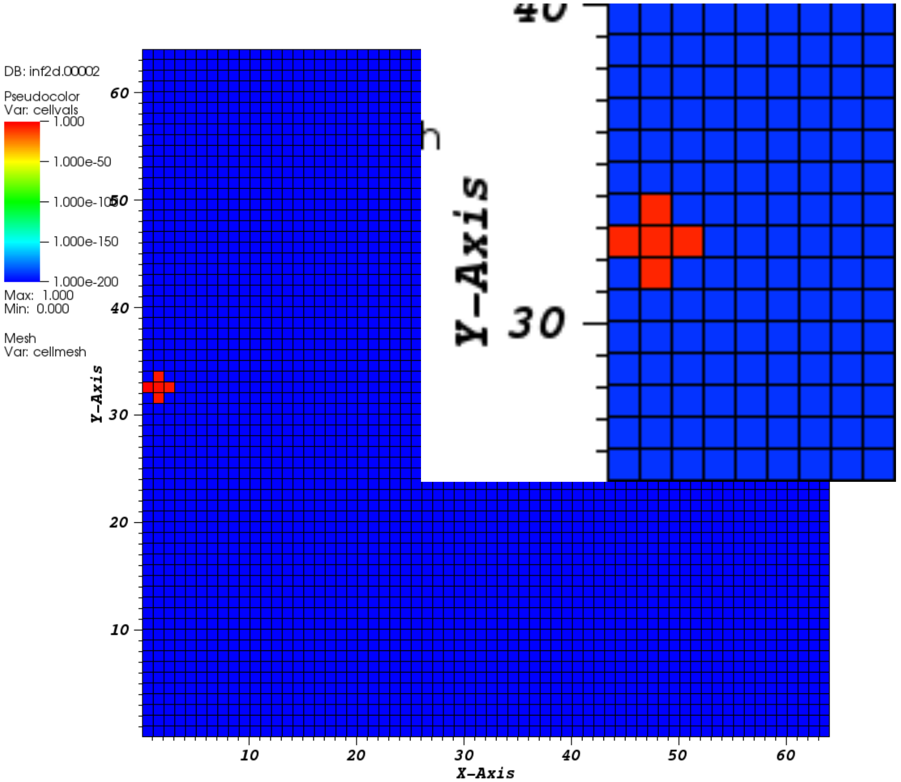


- After 1 time step

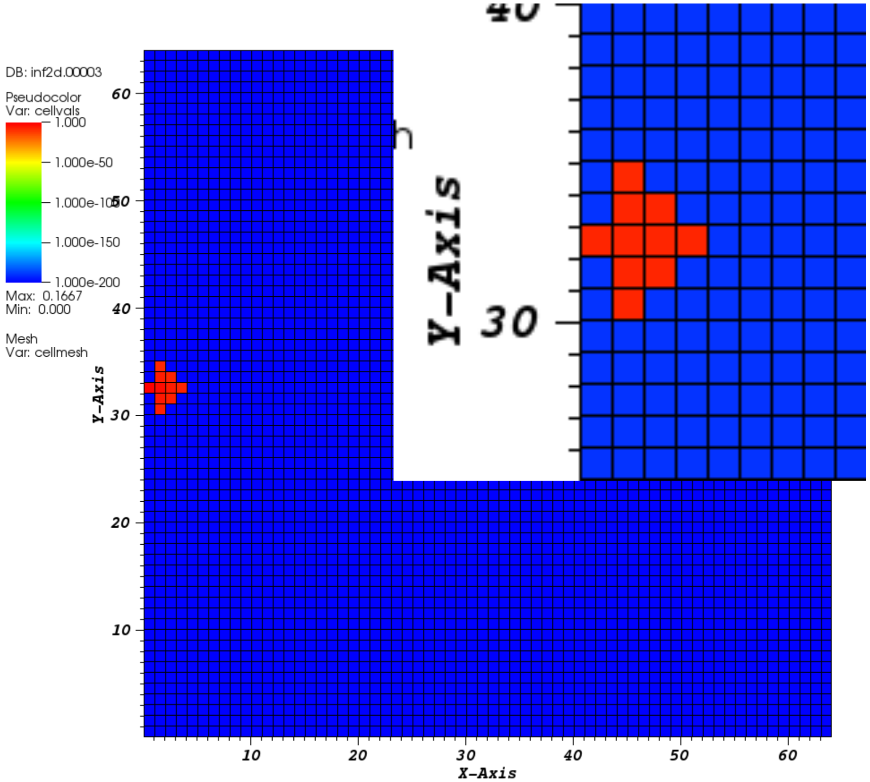


Case Study (IV)

- After 2 time steps

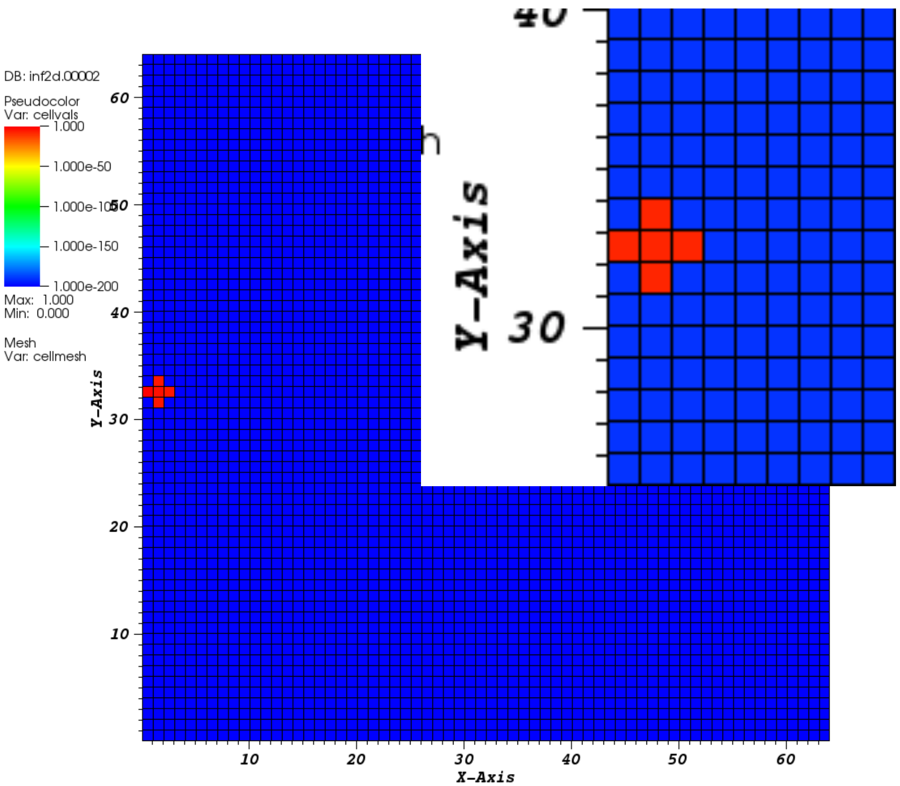


- After 3 time steps

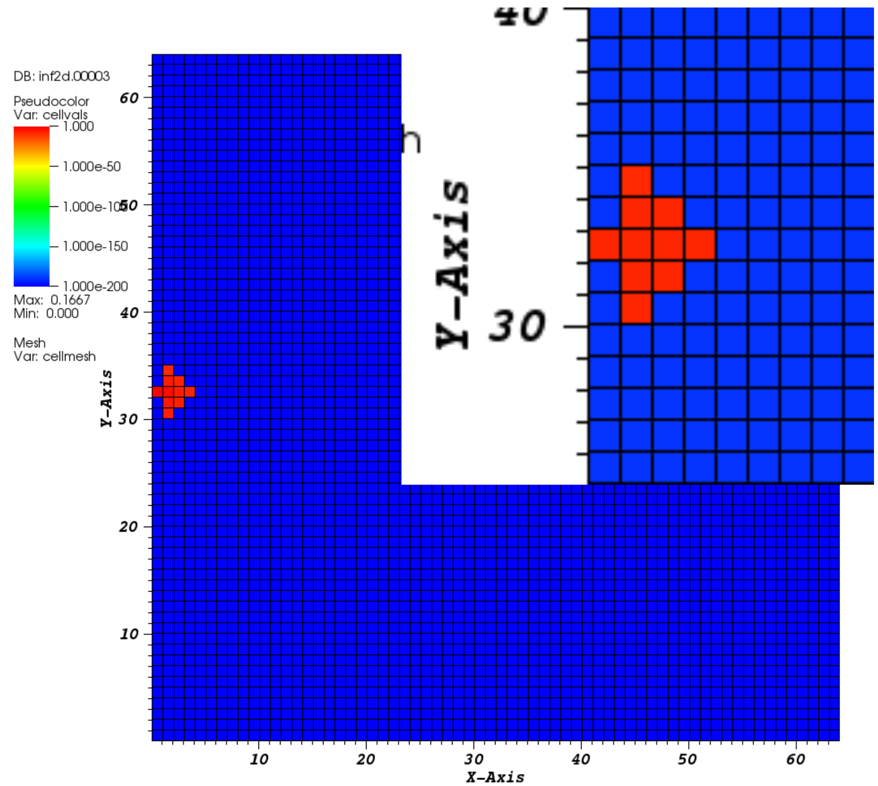


Case Study (IV)

- After 2 time steps

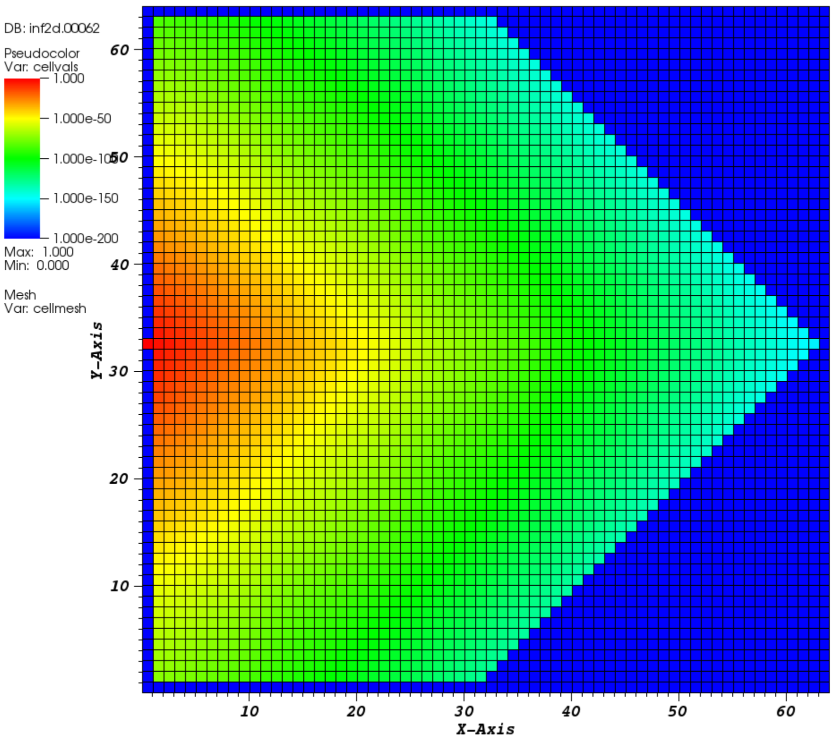


- After 3 time steps



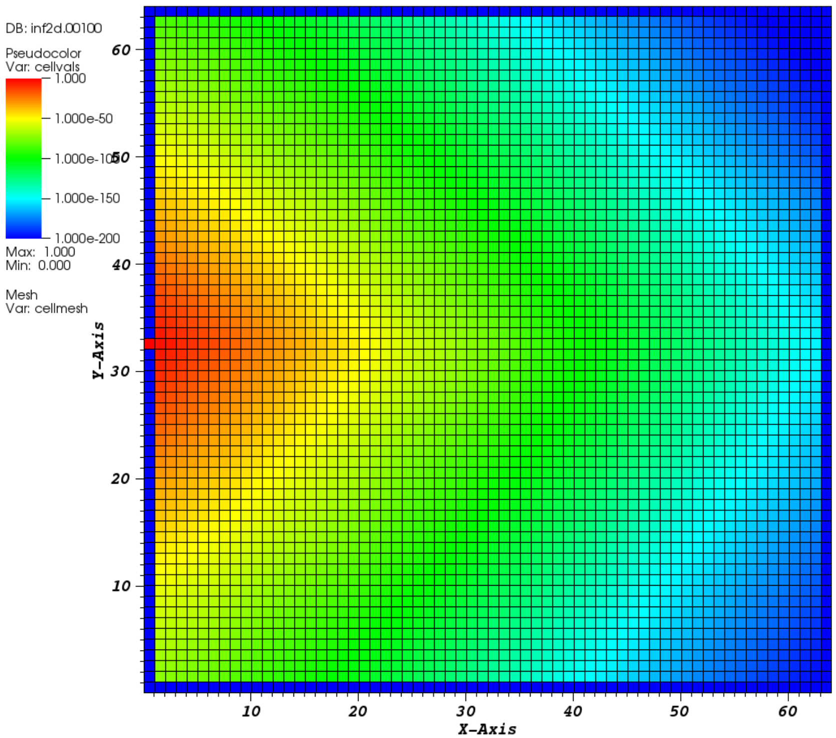
Case Study (V)

- After 62 time steps



Non-zero influence "reaches" opposite boundary, since array is 64x64 with boundary of size 1

- After 100 time steps



Related Work

- Program slicing, chopping
  - Our approach is data-centric, not code-centric
  - Unlike original slicing, dynamic and operates at machine instruction level
  - Forward program slicing could be used in conjunction with VIT to improve time for value influence analysis
- Taint Analysis
  - Our approach uses a real-valued influences, not binary
  - Our approach could be used to implement Taint Analysis
- Automated Differentation
  - Automatically evaluate derivative of a function/program
  - May be usable for paper's use case, but not clear that can solve same problems in general

Ongoing and Future Work

- Implementing support for multiple address spaces
- Addressing a problem in handling register-based addressing modes
  - We currently do not distinguish between using value of a register as an address in indirect addressing mode versus using value in a register as an input to arithmetic operation
  - Causes magnitude of influence values to decrease more quickly than expected
  - Can be solved in Pin by checking addressing modes on inputs
- Support for SIMD instructions
  - Some compilers use SIMD instructions even for scalar operations because the SIMD hardware gives higher performance
  - We support such scalar use of SIMD, but do not yet track handle SIMD instructions that operate on multiple data items
- Accelerator support
  - Distinct address spaces (not a stretch)
  - Lack of dynamic instrumentation infrastructure (analogous to Pin)

Summary

- We are researching an empirical approach for tracking how a value contributes to later computation, called its *influence*

- We are evaluating this approach by implementing the Value Influence Tracking (VIT) tool, and applying it to parallel (MPI, multithreaded) applications

- rothpc@ornl.gov
- http://ft.ornl.gov/~rothpc