LA-UR-23-32613

Approved for public release; distribution is unlimited.

Title:	Implicit Monte Carlo at LANL-Progress and Challenges
Author(s):	Long, Alex Roberts
Intended for:	Monte Carlo Computational Summit, 2023-10-25/2023-10-26 (South Bend, Indiana, United States)
Issued:	2023-11-06

Implicit Monte Carlo at LANL— Progress and Challenges

Alex Long, Los Alamos National Laboratory

Jayenne IMC Team: Matt Cleveland, Ryan Wollaeger, Kendra Long, Ben Ryan, Irina Sagert, Daniel Holladay

10/25/2023

LA-UR-2023-XXXXX

The TRT equations are non-linear in temperature

- The Thermal Radiative Transfer (TRT) equations, without physical scattering:

$$\frac{1}{c}\frac{\partial I}{\partial t} + \Omega \cdot \nabla I + \sigma_{\mathrm{a}} I = \sigma_{\mathrm{a}} B,$$

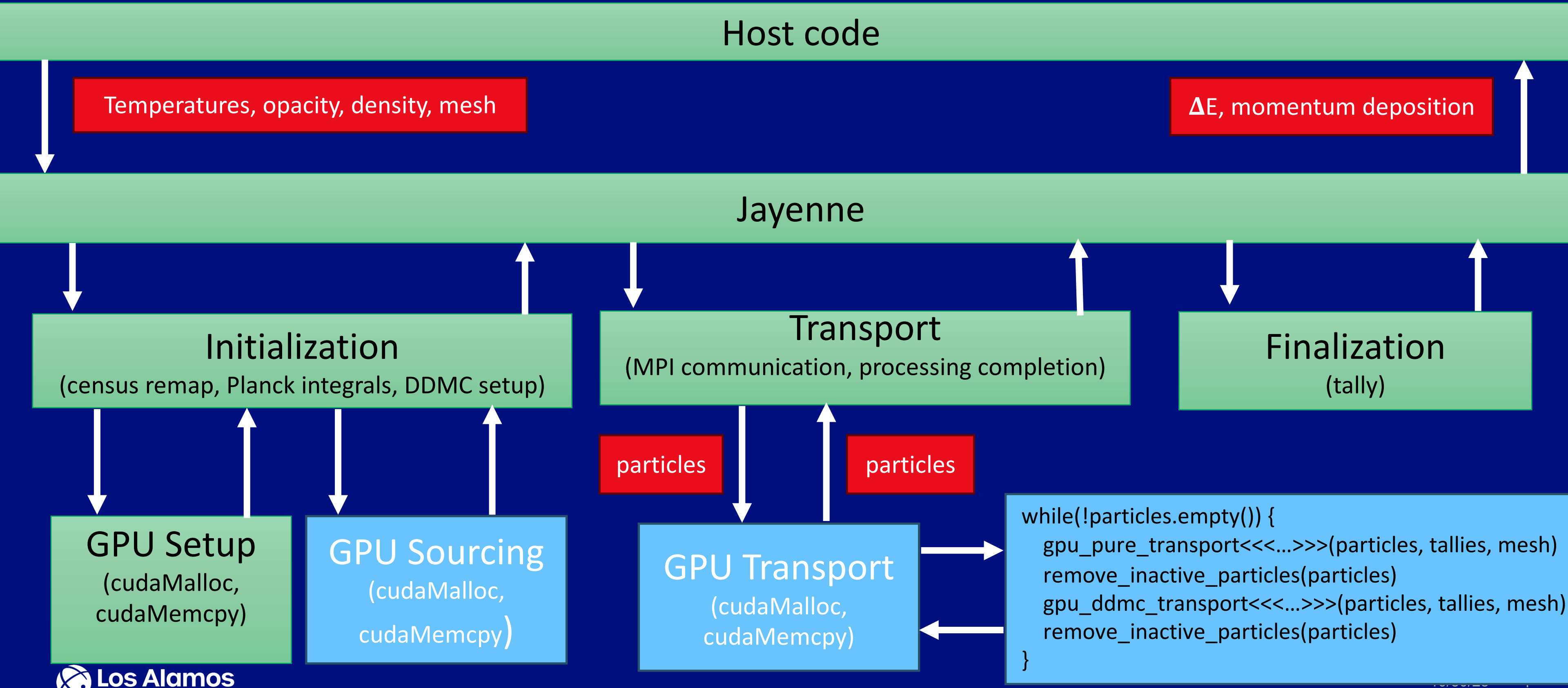
$$\frac{\partial U_{\mathrm{m}}}{\partial t} = \int_0^\infty \int_0^{4\pi} \sigma_{\mathrm{a}} I \, d\Omega d\nu - \int_0^\infty \sigma_{\mathrm{a}} B \, d\nu + S_{\mathrm{m}}.$$

- These equations are coupled by the absorption (σI) and emission terms (B)
- The emission term is proportional to T^4, making the equations nonlinear in T
- The equations are linearized by assuming opacity, heat capacity and emission are fixed at the beginning of the timestep

IMC linearizes the Thermal Radiative Transfer equations

- Implicit Monte Carlo introduces “effective scattering” to stabilize TRT
- A RHS term in energy and momentum equations of hydrodynamics
- Key points for computation:
  - Runs on the hydro mesh
  - Monte Carlo noise can seed hydrodynamic instabilities
  - Mean free paths can be very small (nanometers)
  - Diffusion acceleration methods are required for all practical problems
- IMC code at LANL is called Jayenne
  - 25 years old!
  - C++ with Fortran API to Cassio and Flag
  - Many improvements to standard Fleck and Cummings paper

Rough outline of Jayenne code flow from host code to GPU transport



Major refactoring coincided with GPU port

- Moving away from object-oriented design patterns at a low level
- Moving towards functional programming for tracking and sampling
- Focus on most-used features—2D AMR mesh, multigroup, basic tallies
- Dropping from four template arguments everywhere to one
- Simplify particle to 128 bytes and cell tallies to 88 bytes
- First targeted unaccelerated transport, then Random Walk acceleration, then Discrete Diffusion Monte Carlo (DDMC)

Core design strategy

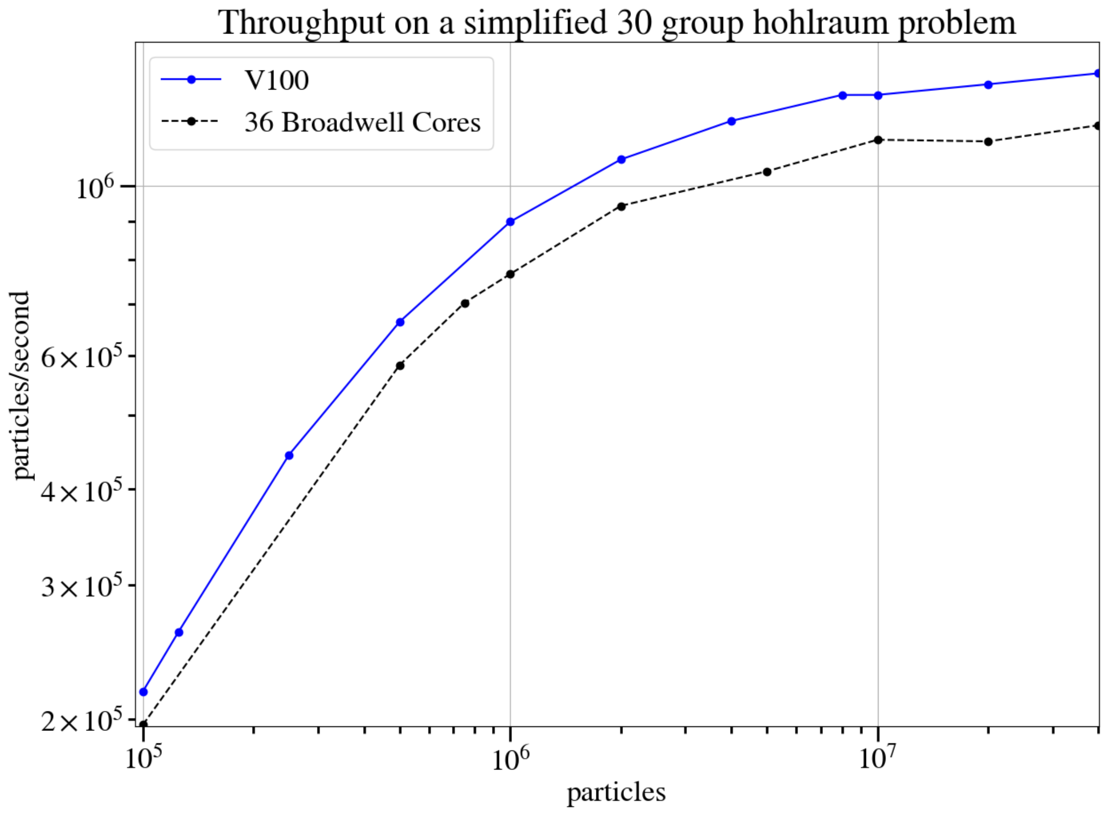
- GPU transport kernel templated on mesh type and dimension allows for a single compilation unit
- Explicit memory model and no allocations in device code
- One particle per thread, particles immediately loaded in to shared memory
- Truncated history, with particles sorted and requeued after 25 steps
- No virtual or recursive functions
- Hybrid history/event-based with two events: unaccelerated transport and Discrete Diffusion Monte Carlo (DDMC)
- CMake and simple preprocessor macros for mostly single-source (and HIP compilation)

Notes and oddities

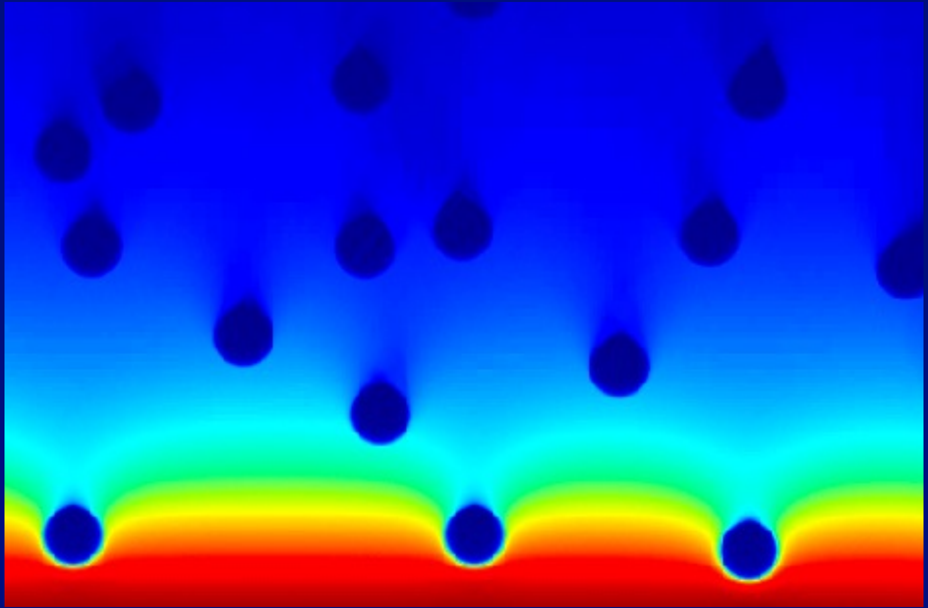
- Most of our experience is on Sierra-like systems
- Better performance with GCC than XL
- CUDA 11.2+ showed significant slowdown (~2x)
- Best performance with CUDA's Multi-Process Service on
- Running with 40 ranks per node for best host code configuration
- Kernel sizes: No acceleration 113 registers, DDMC acceleration 96 registers
- Speedup from Broadwell nodes to both DDR and HBM Sapphire Rapids nodes is about 3.5x (roughly power scaling)

Throughput saturates around 10 million particles

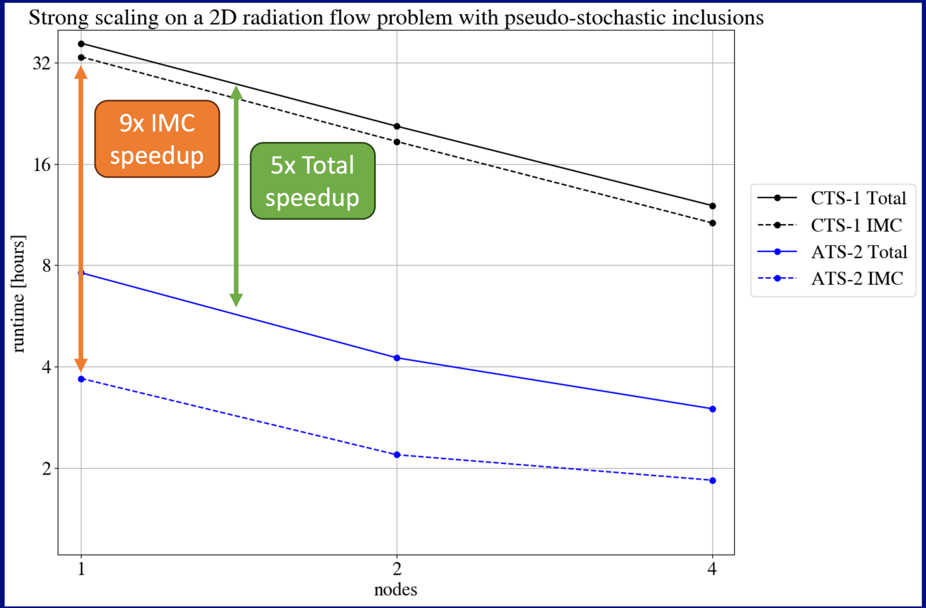
- Why doesn't CPU cross GPU?
- A bit odd looking, possibly due to relative expense of initialization with low particle counts
- Generally, expecting 4x speedup node-to-node



Excellent speedup on smaller, transport dominated problems



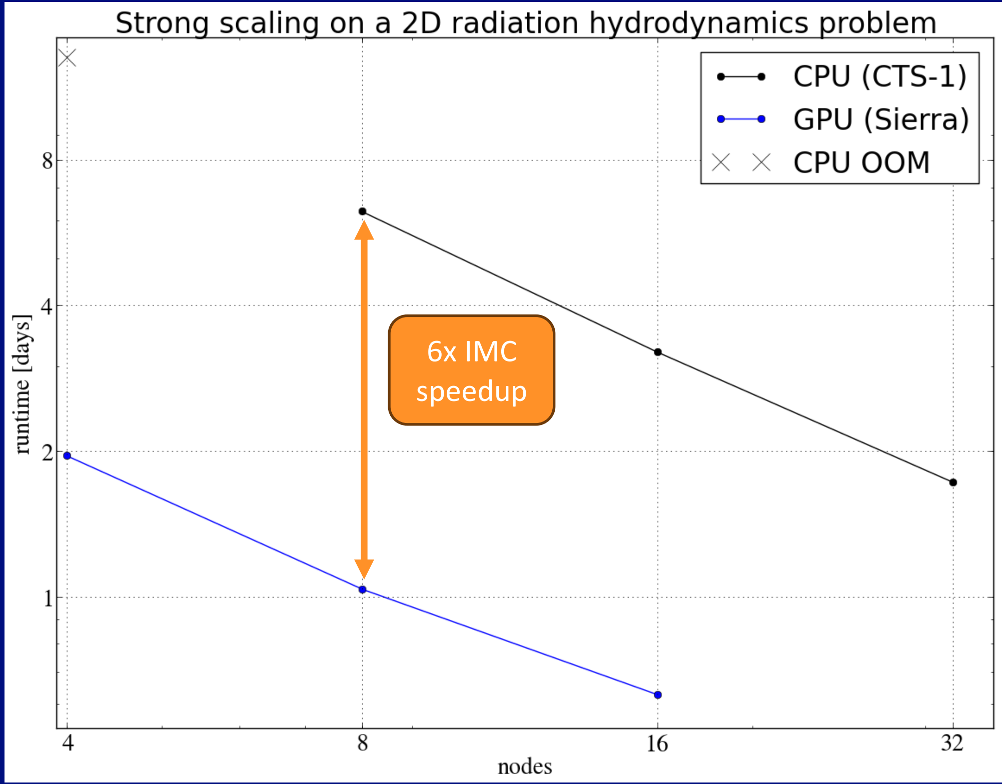
Radiation temperature with an incident radiation source and background "inclusions"



At one node, 10 million particles per GPU (95% peak throughput)

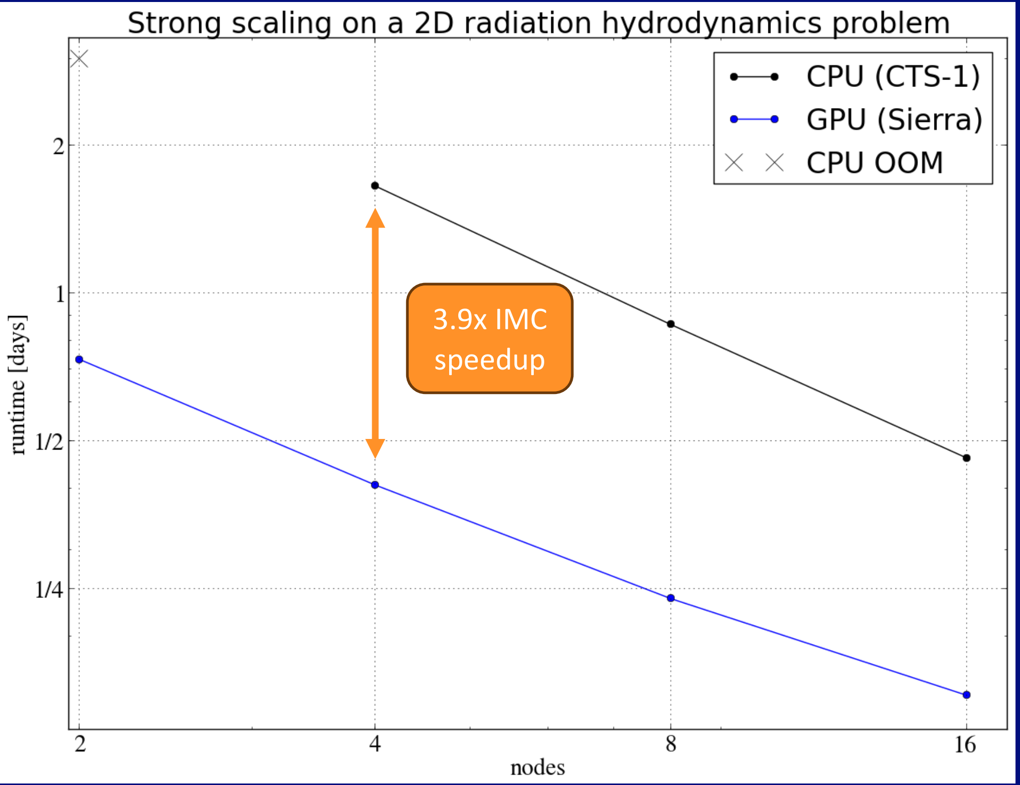
Good speedup on large, transport dominated problems

- 15 million particles per GPU at 8 nodes, about max throughput
- Runs out of memory on 4 nodes of CPU cluster
- Currently working with simple hydro mesh decomposition, moving towards internal decomposition and acquiescing MPI ranks



Limited speedup on standard user runs

- Default cell counts gives 5 million particles per GPU at 8 nodes, 90% of peak throughput
- Does throughput curve take longer to saturate in domain-decomposed problems? How much longer?
- Way to measure this tail effect?



Future work

- Port initialization routines
  - 45% time in transport at 80% of peak throughput
- Simplify momentum integral
  - Moving from dynamic quadrature points to many static points gives 20% speedup of GPU kernel
- GPU feature addition
  - External surface tallies
  - Compton scattering options
  - LD-IMC

Conclusions and questions

- Trying to change how IMC is used at LANL by suggesting much more particles and fewer GPU nodes
- Is the GPU hiding MC memory latency? Does definitively knowing this matter for procurement or optimization work?
- I’m wary of event-based transport for IMC because particle can have many collisions within a timestep (10k kernel launches)
- After initialization, try a “persistent thread” approach
- Speedup above 8x is of limited utility without hydrodynamics and EOS on the GPU

Compiling with –O2 and -res-usage on V100 with CUDA 11.1

```
ptxas info    : Compiling entry function '_ZN14rtt_imc_solver21hybrid_ddmc_transportIN12rtt_tracking20CylindricalCoord_sysILi2EEEN6rtt_mc16Cylindrical_MeshELi2EEEvPK
NS1_9Cell_DataEPKNS_13Cell_IMC_DataEPKNS_14Cell_DDMC_DataIXT1_EEEPKNS1_2U3EPKdNS1_9FrequencyENS_16Scattering_ModelEPNS_10Cell_TallyEPNS_8ParticleEjdPi' for 'sm_70'
ptxas info    : Function properties for _ZN14rtt_imc_solver21hybrid_ddmc_transportIN12rtt_tracking20CylindricalCoord_sysILi2EEEN6rtt_mc16Cylindrical_MeshELi2EEEvPKNS
1_9Cell_DataEPKNS_13Cell_IMC_DataEPKNS_14Cell_DDMC_DataIXT1_EEEPKNS1_2U3EPKdNS1_9FrequencyENS_16Scattering_ModelEPNS_10Cell_TallyEPNS_8ParticleEjdPi
    2880 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Used 96 registers, 8192 bytes smem, 2496 bytes cmem[0], 432 bytes cmem[2]
ptxas info    : Function properties for __internal_accurate_pow
    0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Function properties for __internal_trig_reduction_slowpathd
    0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Compiling entry function '_ZN14rtt_imc_solver25hybrid_no_accel_transportIN12rtt_tracking20CylindricalCoord_sysILi2EEEN6rtt_mc16Cylindrical_MeshELi2EE
EvPKNS1_9Cell_DataEPKNS_13Cell_IMC_DataEPKNS_14Cell_DDMC_DataIXT1_EEENS1_9FrequencyENS_16Scattering_ModelEPKiPKNS1_5ArrayIdXmlT1_Li2EEEEPNS_10Cell_TallyEPNS_8Particl
eEmdPid' for 'sm_70'
ptxas info    : Function properties for _ZN14rtt_imc_solver25hybrid_no_accel_transportIN12rtt_tracking20CylindricalCoord_sysILi2EEEN6rtt_mc16Cylindrical_MeshELi2EEEv
PKNS1_9Cell_DataEPKNS_13Cell_IMC_DataEPKNS_14Cell_DDMC_DataIXT1_EEENS1_9FrequencyENS_16Scattering_ModelEPKiPKNS1_5ArrayIdXmlT1_Li2EEEEPNS_10Cell_TallyEPNS_8ParticleE
mdPid
    2768 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Used 113 registers, 8192 bytes smem, 2504 bytes cmem[0], 480 bytes cmem[2]
ptxas info    : Function properties for __internal_accurate_pow
    0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
ptxas info    : Function properties for __internal_trig_reduction_slowpathd
    0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
```

- No acceleration transport uses 113 registers
- DDMC acceleration uses 96 registers