Accelerating Quicksilver – a Monte Carlo Proxy App on Multicores

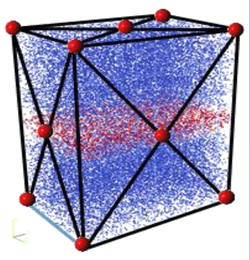
Jesmin Jahan Tithi, Xing Liu*, Fabrizio Petrini (jesmin.jahan.tithi@intel, xing.research@gmail, Fabrizio.Petrini@intel).com

Parallel Computing Labs, Intel Corporation, Santa Clara, CA, USA

Abstract

We show how to **accelerate Quicksilver** – a Monte Carlo proxy app on state-of-the-art multicores and **obtain 1.8x speedup** compared to its original implementation [1]

Quicksilver and Its Relevance



- A proxy app representing Mercury Monte Carlo particle transport simulator used in Lawrence Livermore National Laboratory, USA
- Tracks particles moving through a polyhedral domain of material - decomposed into domains, meshes, cells, facets, …
- Replicates the control divergence, memory access & communication patterns of Mercury
- Lessons learned in optimizing Quicksilver have direct implications for Mercury
- Quicksilver is being used in novel architecture co-design efforts
- Performance of Quicksilver influences hardware procurements at DOE

Challenges - Hard to Parallelize Efficiently

- Complex and irregular kernels with multiple execution paths
  - In Mercury, execution flow can reach 100K lines of code
- Control flow is dominated by branching
- Performance is dominated by latency bound look-ups, branch divergence
- Uses arrays of structures, requires multiple-levels of indirections for look-up
- Accesses multi-GB data randomly or using non-unit strides
  - difficult to cache or coalesce
- Low vectorization opportunities
- GPUs provide little performance advantage
  - 35% when one branch is dominating, but slow when branches taken equally

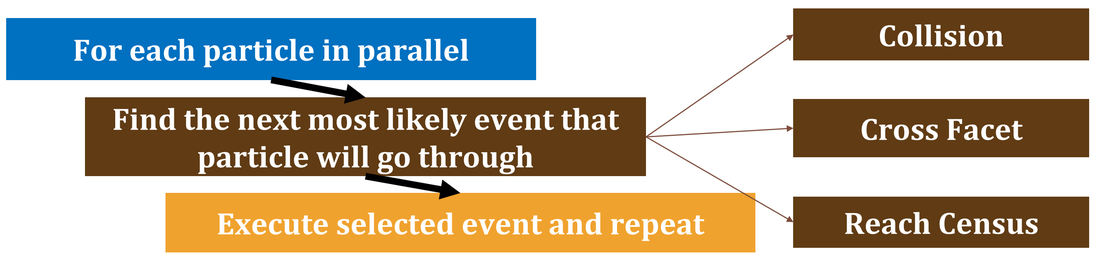
Event Breakdown	Speedup (P100 GPU vs POWER8) original code
Collision Dominated	1.35x
Facet Dominated	1.32x
Balanced	0.56x

Source: [1]

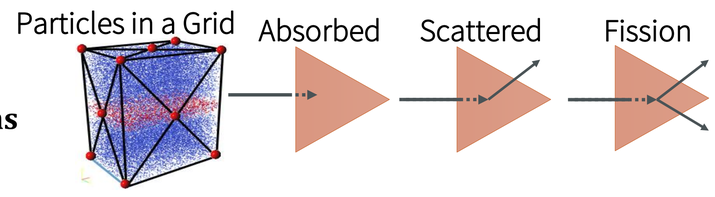
Performance of original implementation shows minimal benefits of using GPU

How Does Quicksilver Work?

- Tracks a particle from its beginning to end in one pass - **history-based** approach



Particles go through three types of collisions



Pseudocode

```
Quicksilver(...){
  //pre-processing and initialization
  cycle_init(...){
    source particles
    perform population control
    initialize tallies
  }
  //core-kernel, particle tracking
  cycle_tracking(...){
    track each particle in parallel
  }
  //post-processing
  cycle_finalize(...){
    reduce all tallies
  }
}
```

Core kernel

```
cycle_tracking(...){
  for all particles {
    do{
      //select segment with the shortest distance
      segmentOutcome=segment_outcome(...)
      //executed selected event and update tallies
      if(segmentOutcome == facet){
          facet_crossing_event(...)
          increment tallies
      }
      if(segmentOutcome == collision){
          collision_event(...)
          increment tallies
      }
      if(segmentOutcome == census){
          census_event(...)
          increment tallies
      }
    } while(!absorbed or
            !incensus or
            !escaped)
  }
```

Intel® Advisor Shows Optimization Targets

Two most time consuming modules are **segment_outcome and collision_event**
These are our optimization targets

Function	% Time Taken
collision_event	54%
segment_outcome	42%

Optimizations

Algorithm and data structure optimizations in collision_event

```
collision_event(...){
    find_isotope_and_reaction(...)
    nOut = compute_collision_produce(...)
    if(nOut > 1)
        add_extra_particle(...)
    if (nOut==1){
        find_new_energy_group(...)
        update_original_particle(...)
    }
}
```

find_isotope_and_reaction is the most time consuming part
- finds isotope & reaction id for a particle
- uses nested **for loops** with **break** to **linearly search in Nuclear data**
- six levels of indirections to access cross-section data

Optimizations
- Find isotope & reaction ids using **binary search** instead of linear search
- Reduces search time from **O(n) to O(logn):** n=O (isotopes **x** reactions)

Requires
- Copy content of Nuclear data for materials into a contiguous TABLE
- Prefix-sum collision probabilities into a contiguous TABLE
- Use binary search to search isotope and reaction ids, instead of a linear search

Benefits
- Reduction in computational cost from linear to logarithmic
- Compacts frequently accessed data -> fits data in cache and improves access

Other optimizations in collision_event and segment_outcome

- Function inlining, scalar replacement, common sub-expression and dead-code elimination in segment outcome and collision event
- Improvements from above optimizations: 5-7% (maximum)

Experimental Results

Platform used: Intel(R) Xeon(R) Platinum 8280 CPU @ 2.70GHz,
Number of threads: 56, Number of cores: 56, Number of sockets: 2, turbo boost: on

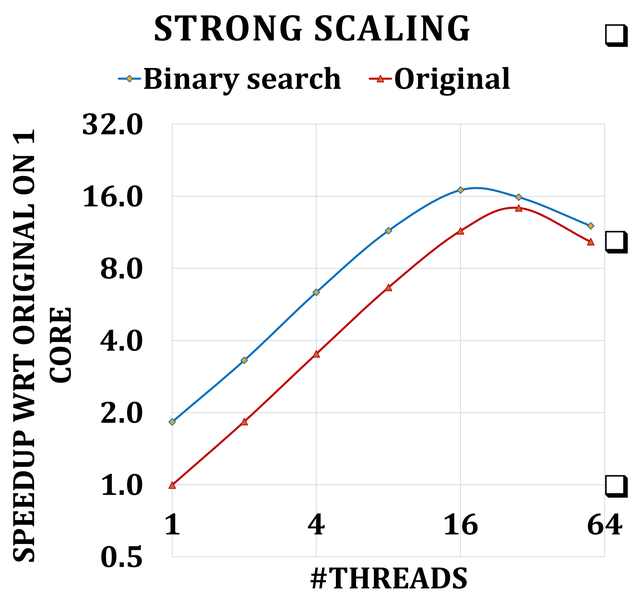
Compiler: Intel(R) 64, Version: 19.0.2.187 Build 20190117

Input: Coral2_P1_1.inp, 163840 particles, 16x16x16 mesh, 100 time steps

Performance of full application run (Speedup)

Performance in terms of segments per sec: The higher the better	Original	Optimized with restructuring, hand-tuning, better atomics, and Binary Search
Segments per sec	1.05E+06	1.92E+06
Speedup wrt original	1	1.83x

Performance of full application run (Scaling)



- Quicksilver is a throughput scaling problem
  - uses MPI to weak scale across ranks and OpenMP to strong scale inside an MPI rank
- Improvement inside a single MPI rank which may occupy a single socket (28 cores) or sub-socket (e.g., 2 ranks each using 14 cores) should be faithfully replicated across ranks during throughput scaling
- Thus, the algorithmic improvement shown in this work is valuable at scale

Future Direction

- Exploration of other optimization techniques – event-based Particle tracking, selective data privatization to reduce numa-overheads, etc
- Mapping to special-purpose accelerators

Acknowledgements

Thanks to the lead developer of Quicksilver. David Richard for his help

References

[1] D. F. Richard and, R. C. Bleile and P. S. Brantley, S. A. Dawson and M. S. McKinley and M. J. OBrien, *Quicksilver: A Proxy App for the Monte Carlo Transport Code Mercury*, IEEE International Conference on Cluster Computing, 2017.

* Xing Liu has left Intel. His contribution to this work happened while he was at Intel.