The GPGPU & Many Vector Core Folly ... is there hope for HEP?

Software Technology Forum, CERN

Matevž Tadel, UCSD

April 27 2016

Preliminaries

- Why am I speaking here ...
    - Since early 2014: Kalman filter based tracking on novel architectures
        - Cornell, Princeton, UCSD; CMS + NSF PIF project
        - Multi-threading + SIMD (AVX, KNC); recently porting to CUDA.
            - I'm the guy who gets most of the low level fun (e.g., perl scripts that write intrinsics code)
        - The master plan was (and mostly still is) to try everything that makes sense.
    - In 2008: Parallelization of ALICE Simulation (Geant3, VMC, TGeo)
- This talk is about alternatives to GPGPUs for offline.
    - Based on experience from the Tracking project.

Outline

- Main claims
- Experience / known hardware limitations from the Kalman tracking project, based on Xeon / KNC
    - Would like to learn if GPUs could do better (even theoretically).
- Reducing hardware constraints
- Near future

Main points

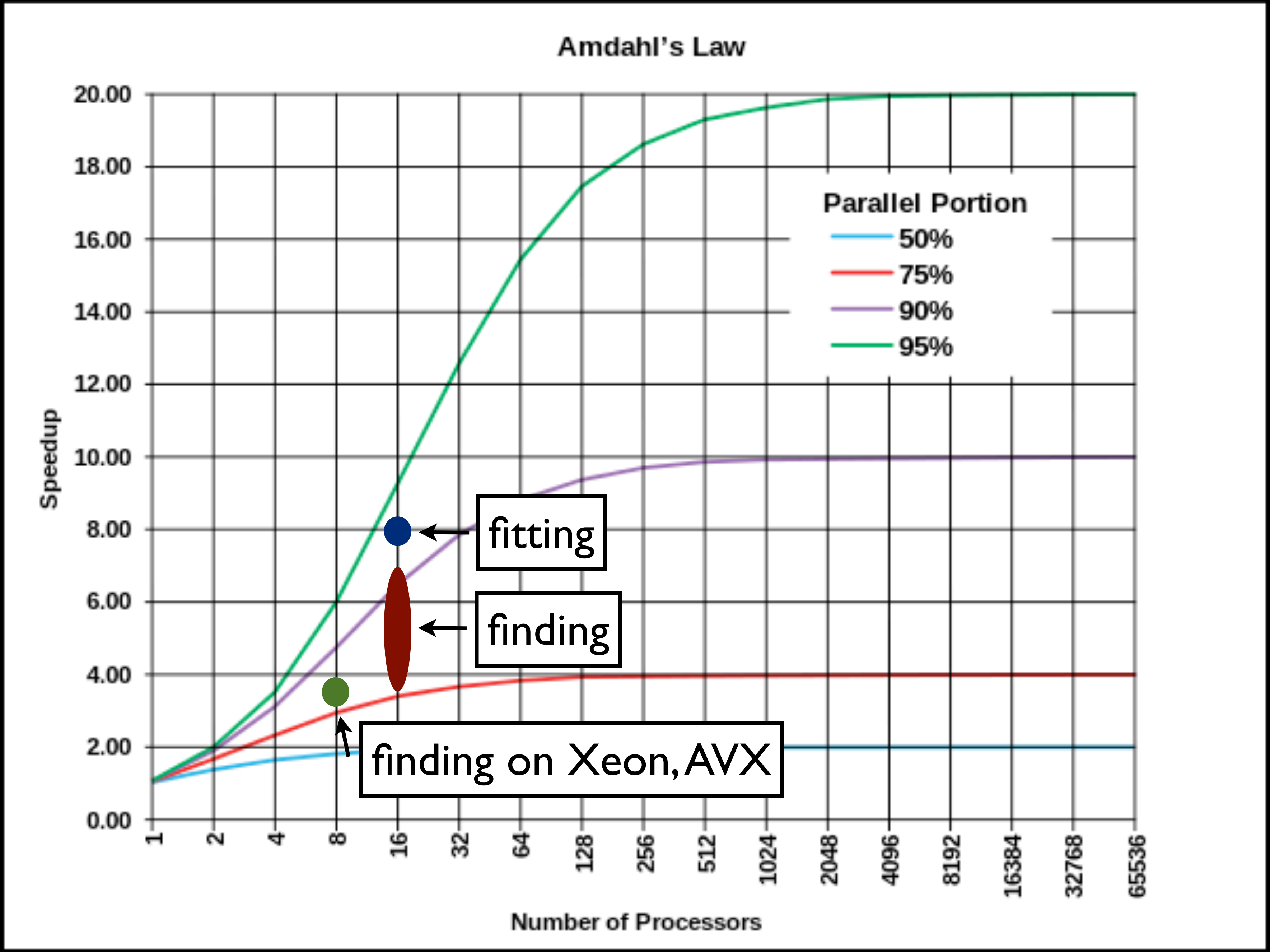
- Chip makers are on a binge making FLOPSy hardware.
  - We are not the only ones feeling frustrated.
- There are only few <u>offline</u> algorithms in HEP that can make **full** use of SIMD / SIMT
  - E.g.: TPC tracking, calo reconstruction, parts of silicon tracking, particle flow reconstruction (?)
  - However: many algorithms can make good use of it!
  - <u>Online</u>: *latency* is all that matters ... best physics per $ ➙ do what you can
- Reconstruction is overtaking simulation in overall computing time!
  - We have CPUs with AVX on the floor **now** (well, for quite a while) ... we should be focusing on that.
- Deployment of hardware is only half of the story
  - Has anybody been following the WLCG saga about multi-core job support?

Track fitting

- This was our first step
  - Initial speedup on MIC: ~30x compared to SMatrix implementation
    - x 8: vectorization
    - x 2: take into account known 0, 1 elements of matrices
    - x 2: manual (auto generated) intrinsics code for matrix multiplication
  - icc-16:
    - xeon & mic: auto-vectorization improved, manual intrinsics give 2% (xeon), 8% (mic) speedup
    - on mic: scalar mode (W=1) uses x87 ... which is twice faster than vector operations
      - makes vectorization speedup less appealing :)
- Side note: icc produces 2x faster code than gcc
    - that was with gcc-4.8, need to recheck with gcc-5.3

Between fitting & finding

- Amdahl's law also applies to vectorization!



Fitting:

x) close to 95%
x) includes repacking of hits from object to Matriplex format

Finding:

x) at 75%, can get to 90%
x) slightly better on xeon - out of order execution

Track finding

- Note: We are actually writing a new tracking code.
  - Most importantly, simplify hit search by using somewhat global coords.
  - We're trying to keep it geometry independent.
- Vector propagation of W tracks is relatively straightforward.
  - Auto-vectorized, requires about 40 temporary variables.
- Match N tracks with M_{cand} hits ... this is hard!
  - Need data reformatting to have a track & a hit in the same vector slot.
  - Prefetching of hit data in L2 & L1 helps ... but not too much and can be overdone.
  - L1 cache is getting real tight for 512-bit vectors & two threads per core.
  - L2 is ok, but one has to be careful with problem partitioning.

Cache & Data repacking

- Xeon - 2 hw threads, MICs - 4 hw threads per core.
- On MIC, L1 is already tight for 2 threads, sigh.
    - Could have separate Inst. and Data caches per two threads?
    - Alternatives: 1. Think harder; 2. use narrower vectors; 3. dumb down algorithms.
        - And above all: profile, also at full load.
- Could we use hyper threads for data prefetching & reformatting?
- C++/hardware support for programming hyper-threads:
    - Prefetching into L2.
    - Repacking: need extra 17 cache lines =~ 1.2 kB
    - Problem is synchronization if running two threads.
        - Need smart interleaving, like giving 2 lambdas.
    - We are able to promise that writes do not overlap.

Avoiding output data races

- Embarrassing parallelism continues to work for us - we can easily avoid write races and have practically no need for inter-thread communication.
- Well, not in GeantV case - this is a real hard problem.
    - Geometry (voxelization), physics ... a lot of incoming data.
    - Collecting compatible particles across multiple cores is super costly.
    - This stretches L1 and memory system / hierarchy.
  - Ideal solution: have the ability to stream data from simulation cores to a special core that shuffles particles into bins and schedules them for processing.
    - Data in flight is not part of memory, i.e., does not have an address.
    - Avoid any locking/synchronization and memory coherency.
    - Special cores have no need for VPUs ➙ can use that space for input buffers.
    - Core interconnects are there ...
- Do the same with output (hits).
- Applies to any data-gather operation, including output serialization!

Unzipping

- This is the bottleneck in several stages of analysis.
  - Could one do SIMD (multi-stream) zip/unzip?
    - Especially if streams / baskets share the compression dictionary.
    - Why is there no hardware support for data compression?
  - Passing compressed data into the core!
    - Decompress on the fly ... reduce memory B/W
    - Hydrological kernels → claim x3 speedup, x8 energy efficiency
    - Fine for single / few variables, probably hard for full deserialization.

What's coming our way

- Skylake: AVX-512
- KNC: out of order execution + double L2 size
  - Systems coming online ... US ATLAS / CMS encouraged to use them!

Trinity, LANL & SNL:
Phase II: > 9500 KNLs

Cori, NERSC
Phase II: 9300 KNLs, single socket, 96 GB RAM

Unlikely partners?

- Game engines have very similar problems as reconstruction.
    - I’m not talking about the final rendering part.
  - AI, physics + collision detection, heart beat, pre-render passes (culling, z-sorting, attribute sorting)
  - Very active discussions in C++ SG14 (low-latency C++: games + nano second trading)
- 2013:
    - PlayStation-4   35 M
    - Xbox One   >10 M
  - Both use custom AMD chips.
    - AMD does seem to be investing in SIMD / SIMT fusion → APU
  - Intel also packs GPUs (“Compute Architecture”) on mobile/desktop CPUs (actually do SSE!)
- Are custom APUs the future of HPC / HTC?
  - Select number of cores, cache sizes, SIMD / SIMT functionality, ...
- Cuncurrency SG1: heated discussions about how to tag tasks for execution on specific “core” types.

Conclusion

- There is a lot of work to be done!
    - That is, real hard work for people who like challenges :)