Accelerating Probabilistic Inference of Hypergraph Algorithms Using PGAS and Chapel

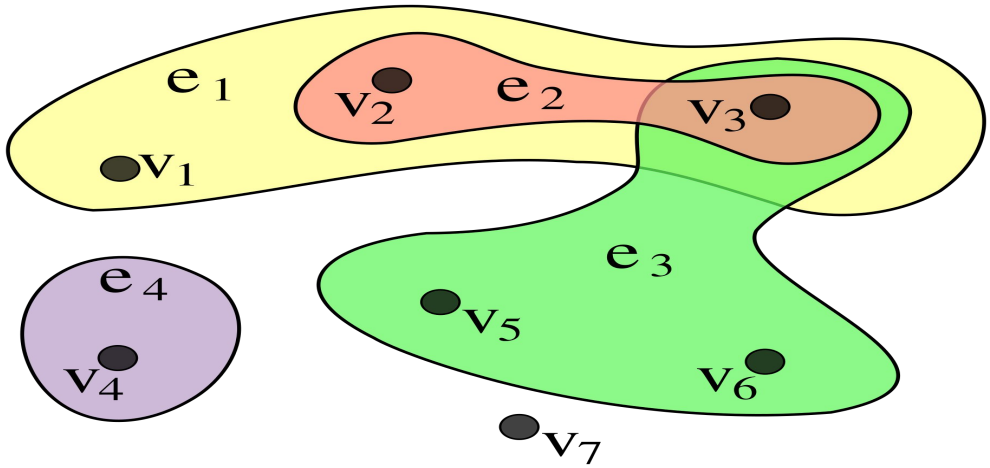
- Rinor Ramli

A presentation for ChapelCon '25

rinorramlifreelance@gmail.com

Hypergraph

A generalization of a graph in which a hyperedge can join any number of nodes.



Hypergraph PGM (Probabilistic Graphical Model)

1. Every node is a random variable.

2. Every hyperedge encodes rules that bind random variables together to form a child random variable.

P (X = 1 | A, B) = {

0.8 if A = 1 and B = 0

0.2 otherwise.

}

Applications of PGM systems.

1. Already present in many critical use cases in industry i.e genomics research, quant trading, fraud detection analysis.

2. Precursor to Bayesian neural networks (BNNs) and HPCAI.

3. HPSC - High Performance Statistical Computing.

Challenges in scaling to HPC level

1. Dataflow

- Components of an algorithm must be executed in a strict sequence.
- Later components must wait, causing bottlenecks as the chain becomes longer.
- Many hypergraph algorithms and statistical computing in general are inherently dataflow-heavy

A - > B -> C -> D -> E -> F -> G

Challenges in scaling to HPC level

2. Branching logic

- Existing architectures reliant on monolithic GPU units are ill-equipped for large scale branching logic.

- Yet PGMs and many algorithms in statistical computing in general involve extensive uses of branching logic

- SIMD thrives on embarrassingly parallel, contiguous code.

Challenges in scaling to HPC level

3. Dependency on data parallelism and single-locale task parallelism.

- PGMs and statistical computing in general rely extensively on task parallelism.

- GPUs are very poorly equipped for task parallelism.

- task parallelism is offloaded to CPUs, but CPUs themselves have limited cores to produce threads.

Challenges of scaling to HPC level

4. Dependency on GPUs

- Very expensive.

- Region locks by HPC providers.

- Geopolitical risks i.e tariffs, raw earth logistics, tensions with China.

- Massive carbon footprint hinders corporate ESG compliance.

- Singular architecture, vulnerable to shocks from hardware failure.

Challenges of scaling to HPC level

5. Dependency on physical HPCs

- Overkill for smaller tasks

- Zero flexibility in configuring individual locales.

- Not suitable for computations where certain locales only process low powered tasks.

Challenges of scaling to HPC level

6. Misalignment against market sentiment

- What we want: fast, fast, fast! Chasing hype, grants, speed at all costs…including literal costs of running.

- What end users want: cheap, cheap, cheap! Chasing practicality as the end game. Speed is only important for the pragmatic purpose of reducing costs of usage.

- Discourages mass adoption of HPC except for research or for high stakes critical use cases in industry.

Hypothesized alternative solutions

1. Do away with massive monolithic GPUs - use CPUs, TPUs or smaller GPU units instead.

2. Use multiple locales instead of singular ones.

3. Use cloud computing such as Google Cloud to set up a virtual HPC made up of heterogeneous VMs as locales.

3. Don't embrace raw speedups as the end goal. Embrace **superlinear acceleration** instead as enabled by multi-locale computation.

Control: K-core decomposition of a hypergraph PGM

Objective: identify nodes with a given K-core threshold and a percentage of certainty.

i.e identify nodes of a hypergraph whose core count is greater than 600, with a certainty of 80%.

1. Observe values from sampling of child random variable represented by a hyperedge.

2. Calculate core rating for each node treating observed value of child random variables as a hyperedge weights.

3. Use sigmoid functions to fit the observed core with the given threshold and certainty percentage.

Control: K-core decomposition of a hypergraph PGM

4. Run the observed sigmoid values on an array of particles that represents each node.

5. Use the sigmoid values to update the weights and normalize the weights of each particle.

6. Repeat the process for a given number of epochs i.e 3000 rounds.

7. Select particles with the highest weight to represent the most likely probability values of each node to fit the objective.

Hypothesized methods for acceleration

1. **LXT (locale x thread) task parallelism**

- My coined term for task parallelism that also involves locales in addition to threads, multiplicatively, for task parallelism.

- Divide sampling of hyperedges, core calculation, sigmoidization, particles weights update and normalization of particles across locales, then divide them even further across threads within each locale.

- Ensure complete domain alignment for each locale for zero communication overhead and maximum efficiency.

Hypothesized methods for acceleration

2. **Particle pruning**

- SIMD-sort particles i.e with quickselect, then remove lowest performing particles via array slicing.

- Focus on computing only the particles with the highest weights.

- Conditions to trigger pruning: fixed intervals (i.e. every 1000 epochs), weight variance threshold, performance threshold, KL divergence trigger (entropy), multi-armed banditry with Hoeffding boundary.

Hypothesized methods for acceleration

2. **Particle pruning**

- compile all conditions into a Bayesian decision maker with probabilistic output.

- each conditions are also probabilistic.

- why? Because probabilistic programming dictates no deterministic conditions should ideally be used upstream in order to not impede on proper convergence of stochastic data.

- this is a complex algorithm but only executed sporadically, and triggered by a separate locale. Just a low-core cheap CPU VM is enough to act as this locale.

Hypothesized methods for acceleration

3. **Stochastic histogram memoization**

- LLN and CLT at play - no matter how byzantine a random process' underlying fundamentals are, given static anchors are present, sampling at large enough amounts will always eventually reveal a certain predictable structure ie a histogram.

- PGM algos inherently involve massive amounts of sampling, making this implementation a natural fit.

- Store past solutions prior to sending to particles additionally in a histogram. Only store the probability each solution appears - hence a stochastic histogram.

Hypothesized methods for acceleration

3. **Stochastic histogram memoization**

- the histogram in effect acts like a X-nominal random variable by itself.

- this implementation greatly relies on clever engineering of solution spaces to collapse to a small enough range to allow for predictable, repeated outputs.

- i.e use sigmoid functions to reduce observations to [0,1] then use binning to further reduce them to match the size of the particle array. Simons Cone, holonomy transitivity, manifold solving, etc for observations with dependency / hierarchy with each other.

Hypothesized methods for acceleration

3. **Stochastic histogram memoization**

- memoized data in the form of a particle array just for the observations (a particle array technically IS a histogram).

- mirror the pruning of the particles in parallel. Chapel can do this elegantly.

- again, a single ultra-cheap low core CPU VM locale dedicated to this is enough.

Hypothesized methods for acceleration

4. **Speculative execution**

- Use the memoized data from earlier to generate speculative observations via sampling.

- Sample at O(2) complexity instead of O(f(a, b)) at the absolute barest minimum where a = number of hyperedges, b = dataflow steps.

- x - rng roll, SIMD-powered, O(1). y - pick a particle associated with the RNG, again SIMD-powered, O(2).

- can even be combined together via branch-masking to approach O(1).

Hypothesized methods for acceleration

4. **Speculative execution**

- Split locales into two - one for speculative execution, the other for definitive execution.

- epoch results become solely determined by speculative execution, with the results from definitive execution lagging behind but used to correct the speculative execution results similar to packet-based rubberbanding in lookahead speculation in MMORPGs.

- locale splitting triggered when pruning of particles had reached 50 percent.

Hypothesized end result: superlinearity

- The longer the algo runs, the faster it gets.

- The acceleration becomes more pronounced as we add more locales while maintaining the same theoretical sum of computing power.

- This makes for a strong selling point for end-users : incentivizing them to actually subscribe for the largest portion of locale usage in a HPC network.

- Cost savings for massive usage that goes beyond discounts that barely eat into profit margins.

Massive ESG benefits

- Dependency on multiple locales mean we do away with extremely expensive and polluting monolithic GPUs

- We stop depending on physical HPCs either, since cloud HPCs such as Google Cloud allows for customizing locales at an individual level.

- Use very cheap VMs for locales that only do low powered tasks - this flexibility is NOT possible with physical HPCs where every locale is exactly the same and forcibly soldered with GPUs.

- While ethernet is a bit slower than physical (but improving rapidly), they suffer much less from communication overhead at larger volumes.

Future research

1 - Replace data parallelism with LXT task parallelism entirely.

2 - Delve into Bayesian neural networks.

3 - Explore applying multi-locale superlinearity in other domains ie blockchain, AI, compilers.

4 - Create a dedicated hypergraph database employing this feature in-house for HPSC and HPCAI.

Thank you!

Email : rinorramlifreelance@gmail.com