Pitfalls and learnings in performance modeling

Phil Thierry*, Cedric Andreolli, Sai Chenna , Fabrice Dupros, Sunny Gogar,
Sylvain Jubertie, Nalini Kumar, Amine Mrabet and Mariam Umar
Intel Corp.

Introduction / Conclusion

One Simulator cannot satisfy all requirements and solve every questions

Rule number one:

- Define the objective :
  - One-shot question / project or should we develop of full modeling Workflow
- Define the input parameters available and how to get them (are the tools ready)
- Define the output results and needed accuracy
  - If a Lifetime is not enough, revisit the hardware and software granularities
  - In any case, use uncertainties
  - And better have at least 2 or 3 ways to validate the results

Analytical models

- Any Scale. Most complexities (comms. , compute) are implicitly accounted for.

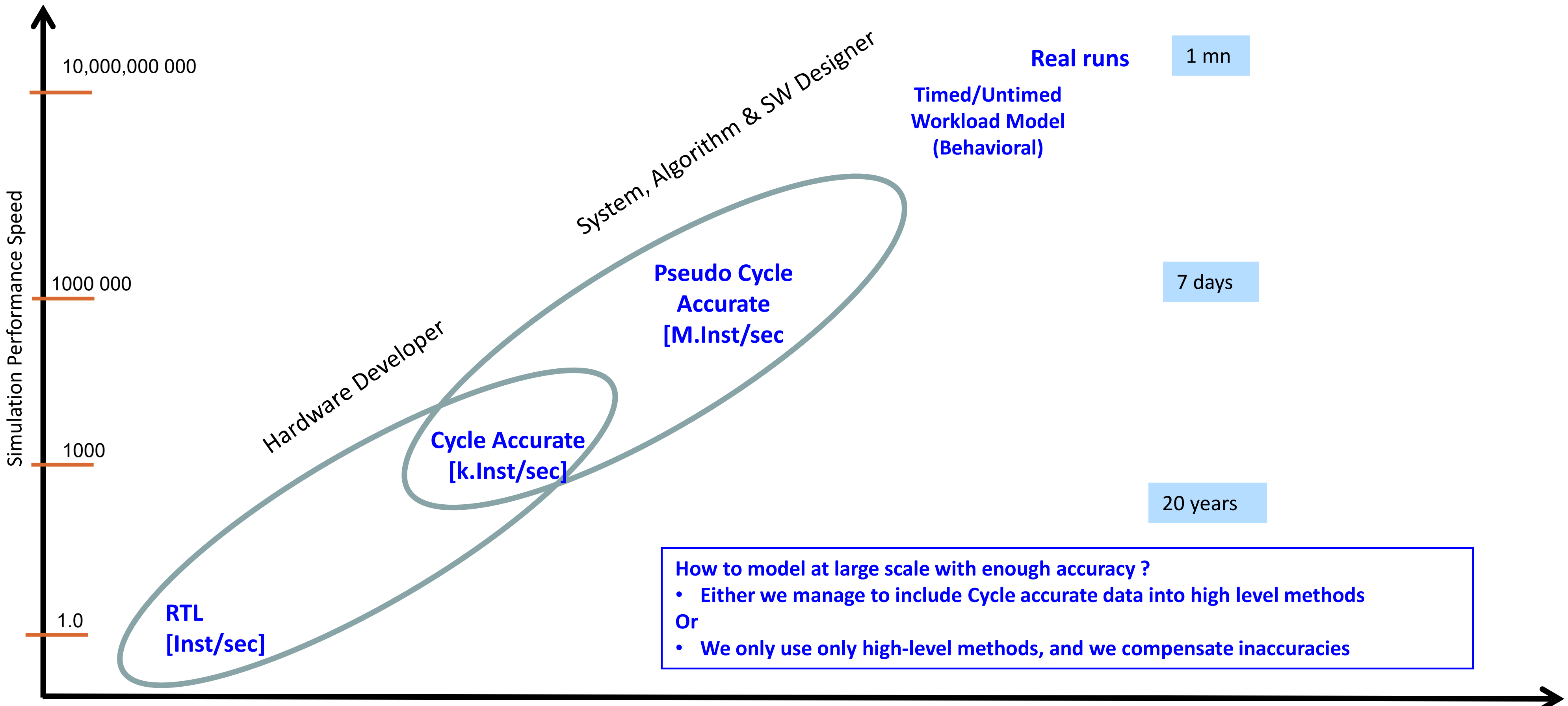
Graph-based modeling (AI-NN)

- Any scale but limited to NN

Measurement-based modeling (HPC and AI)

- Single CPU/GPU core to single CPU/GPU core
- Single homogeneous/heterogeneous node to Single homogeneous /heterogeneous node
- O(x) ranks to O(x) ranks
- O(x) ranks to O(10x) ranks

Accuracy and speed are orthogonal features !



How to model at large scale with enough accuracy ?
- **Either we manage to include Cycle accurate data into high level methods**

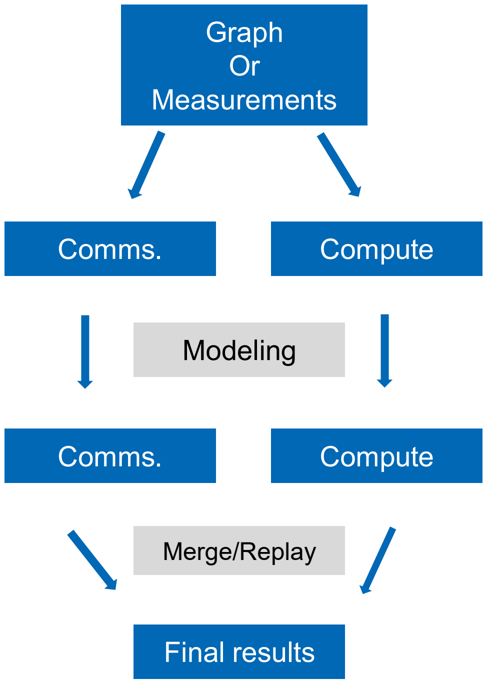
Or
- **We only use only high-level methods, and we compensate inaccuracies**

Very Small kernel. Millions Inst.
IP and SOC level
- design verification micro-architectural exploration and evaluation
- performance and power projections

Small kernel. Billions Inst.
SOC and node levels
- architectural exploration
- performance projections
- application, compiler, library optimization

Full application
Node up to Full system
- High level architectural exploration
- performance projections including communications
- application, compiler, library optimization

The right methods for the given applications



- **Graph-based methods for AI: (mostly python)**
  "Collective communication + GEMM and convolution"
- **Measurements-based methods for HPC and AI:**
  MPI, OMP, C/C++/ Sycl/ Fortran, etc...
  Distributed and Shared memory + Any numerical schemes

A way to mix accuracy and speed

- Parametric equation based on the numerical schemes
- Simple model based on F/B

Fast, Any scale. Can be adapted to specific requests
Complexity is implicitly accounted for.

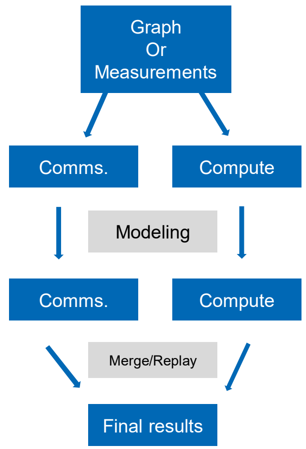
It’s all about the communication & compute relation (IO too)

Application can be described as Total_time = f [Comms (+IO), Compute (+ serial_init)]

Every modeling are doing comms $\oplus$ compute*

Where $\oplus$ can be either
- An addition : not accounting for overlap, dependencies, asynchronism
- A more complex “replay” (scheduling of comms and compute)

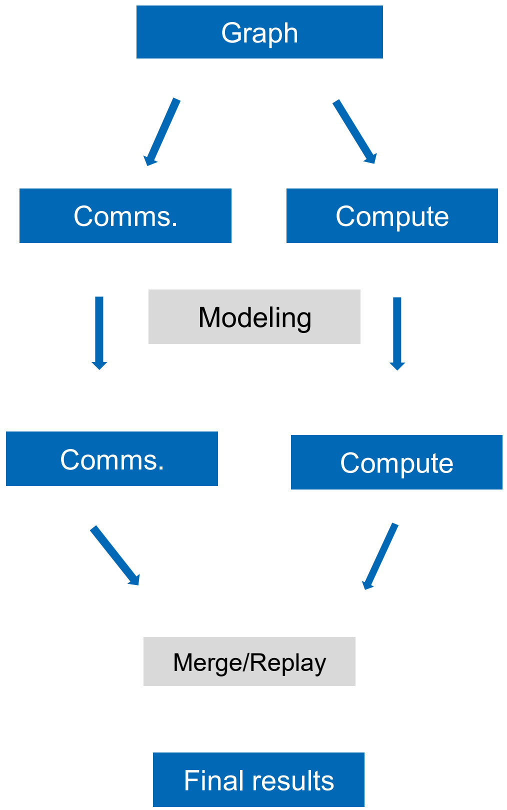
	Communications	Compute
profiling	APS, SMPI, SCOREP, ONNX	SDE, Advisor, MSR (Vtune, Emon, Papi), Unitrace
modeling	LogP , SIMGRID, SST	Analytical , NN, Sniper, Cy.Accurate



Then How to chose:
1. Speed : we may quantify them as processed Inst / sec
2. Accuracy: we can define ranges (<5% , [5-10], [10-20], [20,50],[50,100])
3. Scalability: how easy to move to another application

Back to first rule: Clearly define the objectives

AI-NN specific case



Identify volume of communication at every stages (mostly collectives)

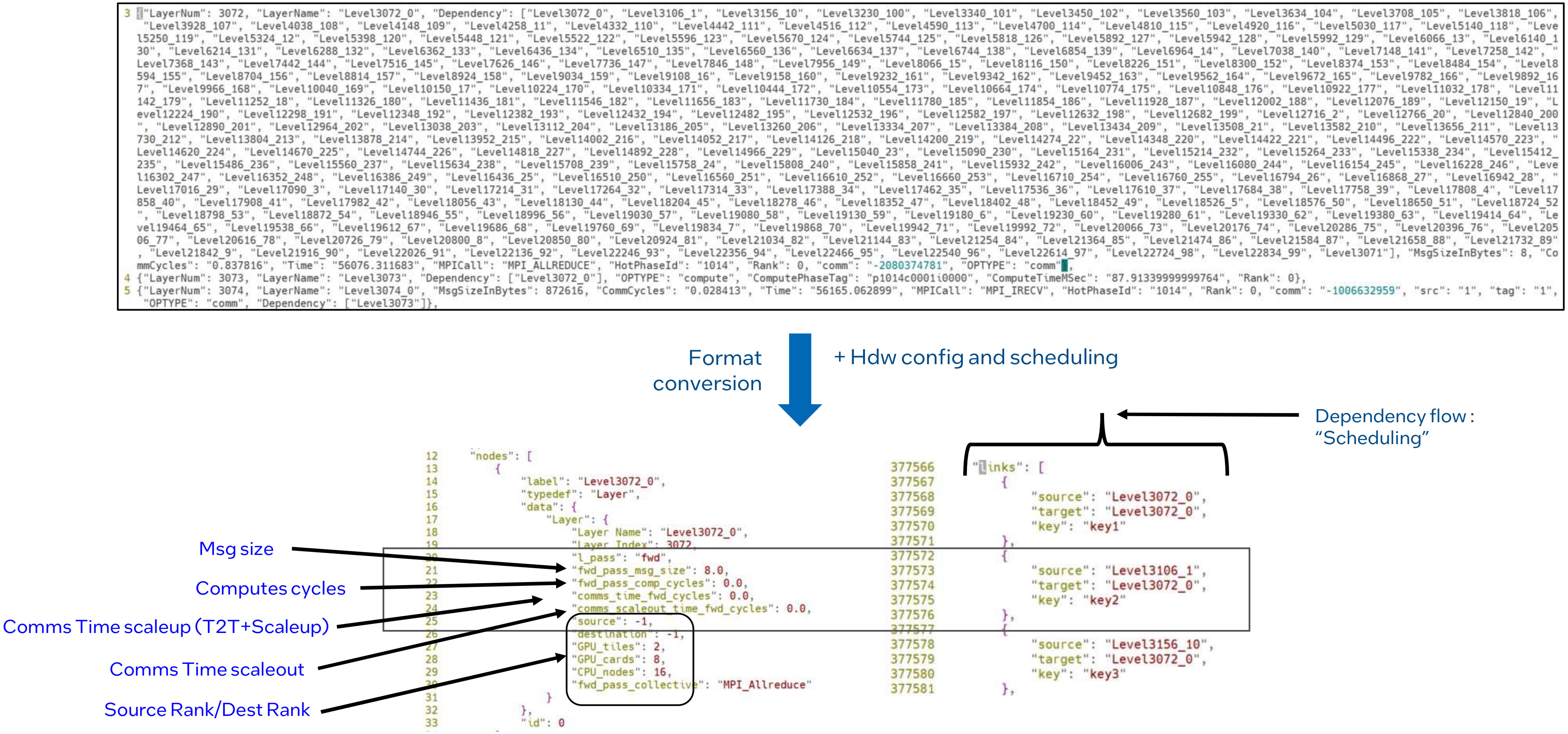
Identify GEMM, convolution per layers (mostly size and shapes)

Extrapolate to a given number of end points using
- Analytical collective models
- Topology aware tools (SST, cycle acc., …)

Extrapolate them
Roofline extrapolation,
(pseudo) Cycle Accurate simulation

Merge communications and compute timing
- Summation with or without overlap
- Reuse original scheduling (if DAG exist)

Graph-based methods



Traces format Standardization

- Using Python model APIs : Custom dev for the full workload
  - What we do so far. Some advantages but many disadvantages
- Using ONNX model format (OpenNeural Network exchange)
- Using Chakra Execution trace

ONNX exchanges models between various frameworks,
Chakra's exchanges execution traces between different teams.

Choosing the right "API" is extremely important to scale the methods to various application and model

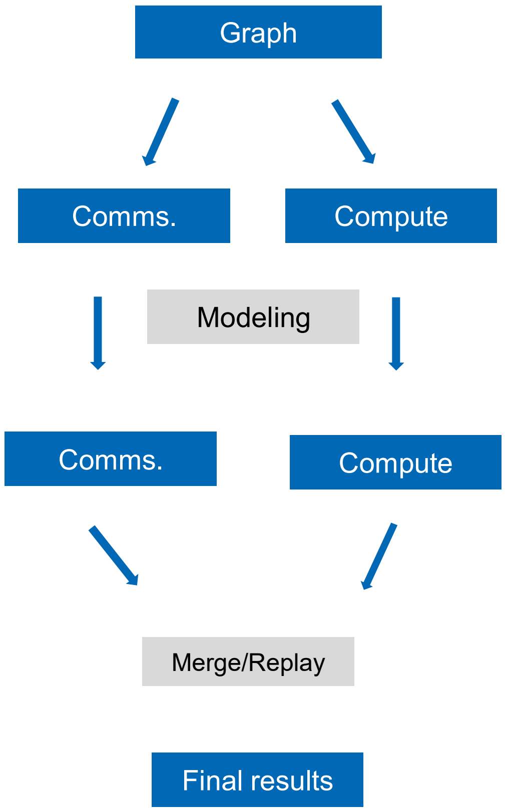
**see Sridharan et al. 2023. "Chakra.: Advancing Performance Benchmarking and Co-design using Standardized Execution Traces."*

HPC and AI cases

Profile volume of communication
Who is talking to who, imbalance
Aggregated (APS, vampyr) or full tracing (SIMGRID, SCOREP)

Extrapolate to a given number of end points using
- BW/lat scaling for aggregated comms
- Topology aware tools (SST, SIMGRID, cycle acc., …)

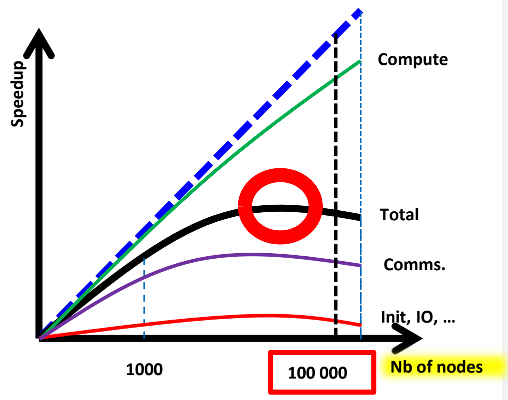
Merge communications and compute timing
- Summation with or without overlap
- Reuse original scheduling (if DAG exist)



Profile the compute (sampling, hdw counters)
- Aggregated
- Per function
- Tracing

Extrapolate them with
Roofline extrapolation,
(pseudo) Cycle Accurate simulation

Amdahl’s law inflection point



Modeling when and why the scalability is failing down remains a huge challenge, for both strong and weak scaling. Not to mention IO.

We need it for Interconnection/Topologies whatif analysis and pathfindings.

With analytical modeling and graph-based AI methods, we can mimic the communication and compute overlap.
Doable with HPL, 3dFD, 3dFFT, LQCD, AI LLM (i.e deterministic applications “easy” to formalize).

For more complex applications, it remains impossible to profile on O(x) ranks and extrapolate to O(10x) ranks (i.e extrapolation of scalability).

The only way would be a full tracing (time independent) of those O(10x) ranks to capture the real communications patterns and compute.

Too expensive with SIMGRID. Same with SCORE-P. Traces quickly becomes far too large

Our current tentative is with SCORE-P using a trace compressor (phases detection) that works with regular application.

This remains “WIP”. Not to mention the Ultimate: Modeling productions runs with several apps running at same time.

SST features for network simulation

Ember – State machine model to generate network traffic:

- Features:
  - Provides a collection of HPC comm patterns and MPI Collectives ("motifs")
- COMET* extensions:
  - OneCCL Allreduce Algos (includes topo-aware pipelined Allreduce)
  - Enable overlap compute and communication events for AI Modeling

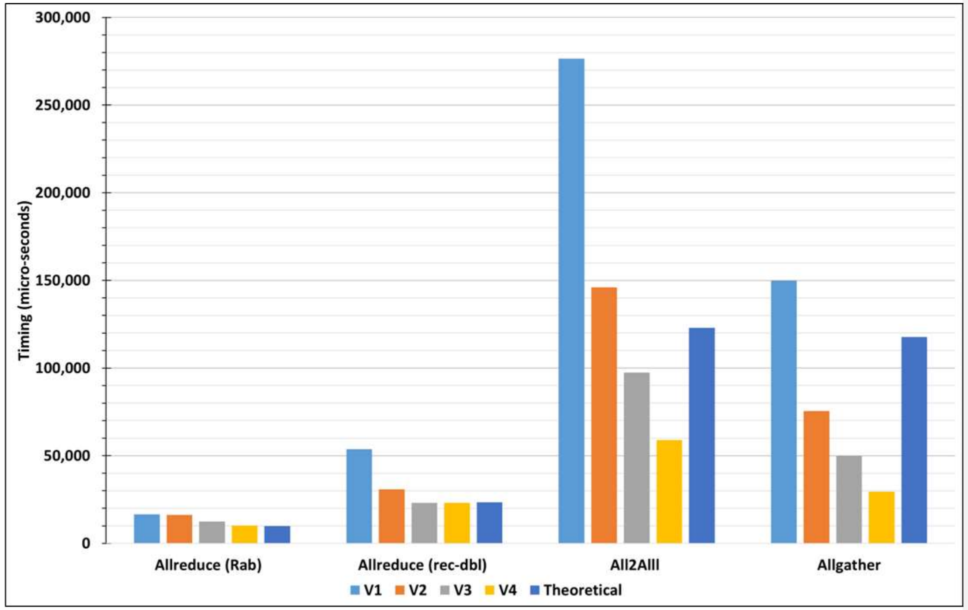
Firefly – Interface b/w network driver (Ember) and router (Merlin):

- Features:
  - Provides detailed NIC model and packetization and byte movement engine
  - Implementation of MPI communication protocols
- Limitations:
  - No Non-blocking Collectives

Merlin – Low-level, high-radix router component:

- Features:
  - Models flit-level movement, physical routing and delivery of packets
  - Provides readily available topologies with static and dynamic routing (congestion-aware)
- COMET extensions:
  - Included additional topologies: Hierarchical networks, Polarfly/Polarstar,
- Limitations:
  - No collective/switch offload
  - IP specific Congestion control routing schemes

*COMET – Intel extension of SST.

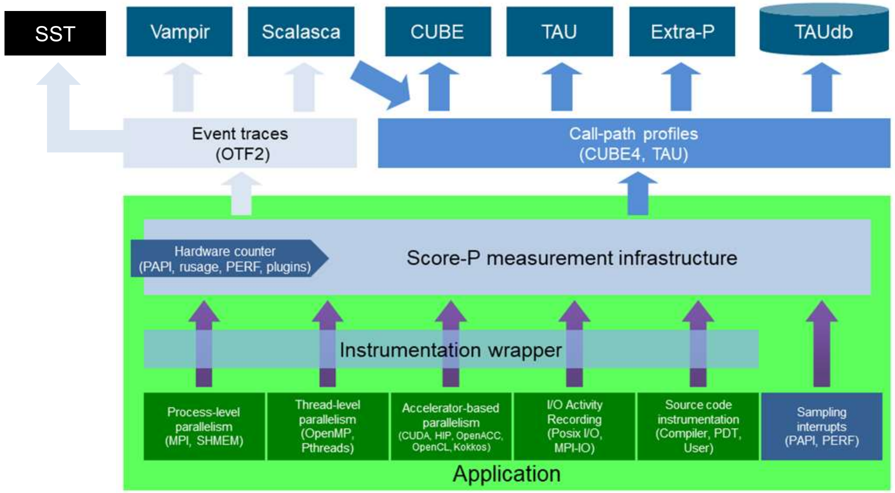


"Application-less" patterns modeling for various (realistic) topologies compared to analytical (theoretical) model.

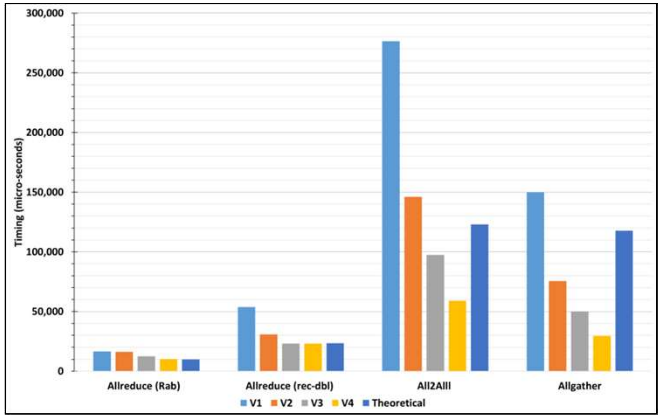
This is just a scale-up example with 24 endpoints in a heterogeneous node.

Analytical collective models are often too optimistic , and this effect keep growing with scale-out.

SST various usages

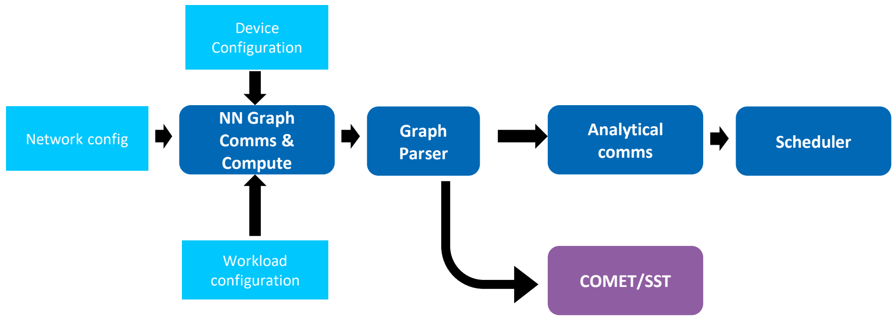


SST as a SCORE-P backend for Communications and compute replay for HPC and AI "measurements-based" methods



Application-less patterns modeling for various topologies

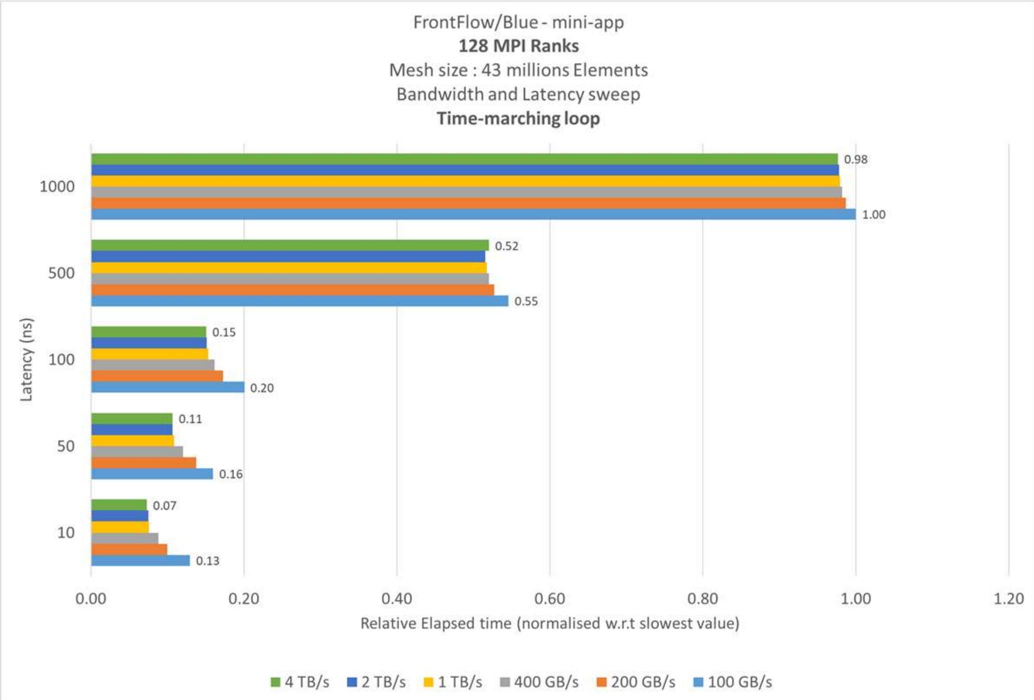
- Extract relevant information from traces
  - System Counters (perf)
  - Communication Phases: P2P blocks , Collective Operations
- Combine trace events from different independent ranks
- Merging P2P operations into MPI Phases
- Detecting Compute Phases (gaps)
- Detecting Cyclic Phases / Loops -> Reduce Tracefile size and modeling time/complexity
- Feeding application communication phases into SST

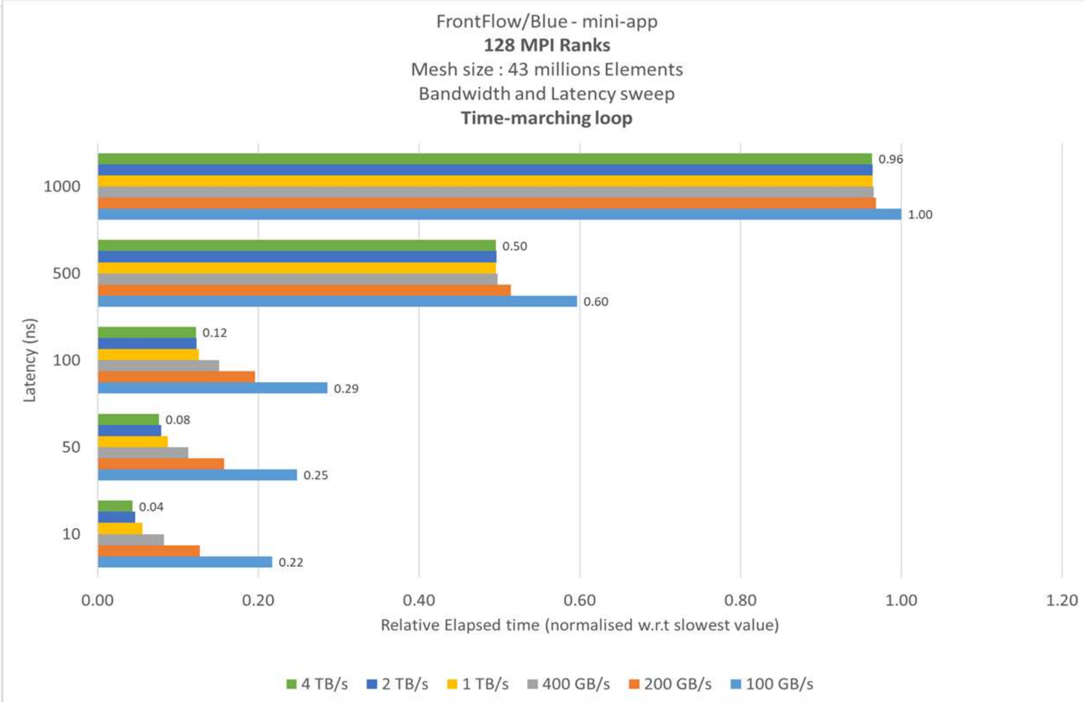


As an alternative to Comms analytical modeling for AI-NN apps.

*COMET – Intel extension of SST.

SIMGRID modeling. Topo, latency and BW on real applications





SIMGRID modeling. Normalized elapsed time for varying bandwidth/latency parameters.
Left-panel (flat topology) – Right-panel (2-level topology). Colors represent various BW [0.1 : 4 TB/s] and the series denotes various latencies

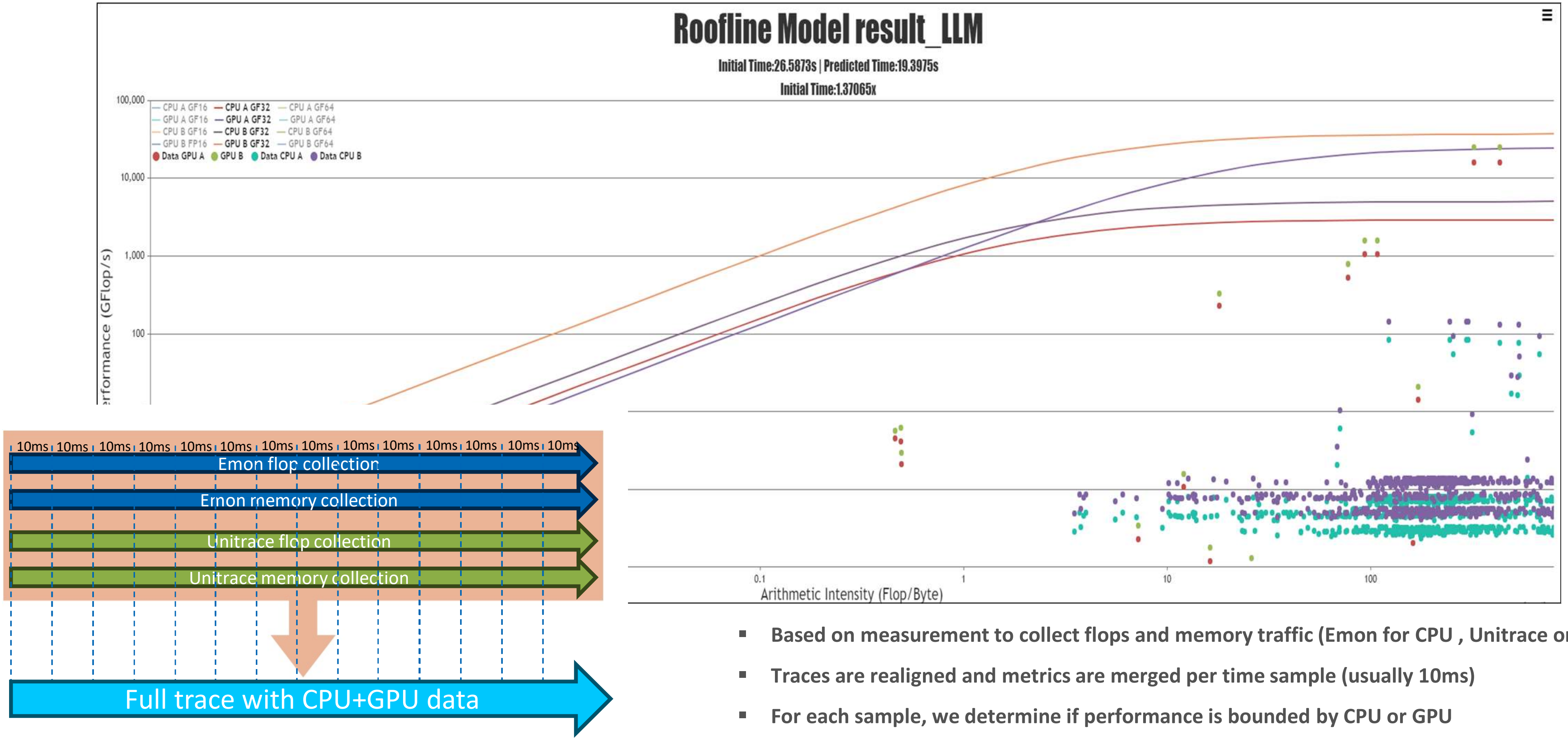
Advantages : Full real application and test cases. No instrumentation. SIMGRID is an active opensource project.

As for SST, topologies can be added. Can be coupled with compute models. Handle comms & comp dependencies.

Limitations: Can scale up to O(3) nodes but getting slow after that.

Compute phases extrapolation from measurements

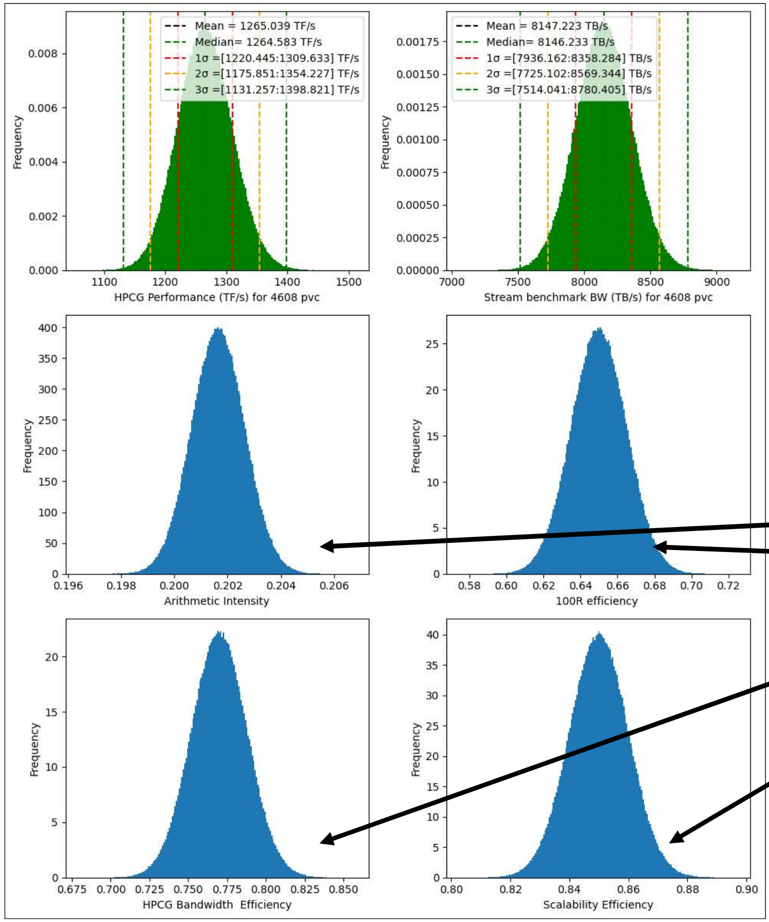
A CPU+GPU performance extrapolations with a (extended) Roofline Model



- Based on measurement to collect flops and memory traffic (Emon for CPU , Unitrace on GPU)
- Traces are realigned and metrics are merged per time sample (usually 10ms)
- For each sample, we determine if performance is bounded by CPU or GPU
- Efficiency is extracted on current platforms and projected on new platforms

Analytical modeling

Analytical HPCG-UQ



Analytical HPCG model

HPCG = 0.201 (F/B HPCG) → **Hand counted**

* Peak_bw* eff_vs peak → **100%READ BW**
* efficiency HPCG → **What HPCG can extract**
* efficiency comms → **How does HPCG scale**

As those few parameters are uncertain, just add distributions

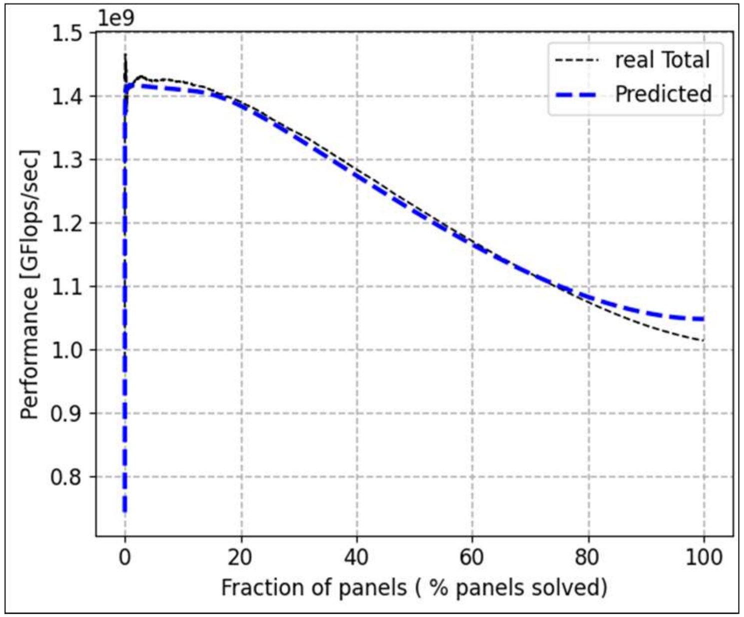
```
perf_dist = dev_num*
arith_int_dist
• peak_bw * eff_100R_dist
• *eff_HPCG_dist
• * eff_scala_dist
```

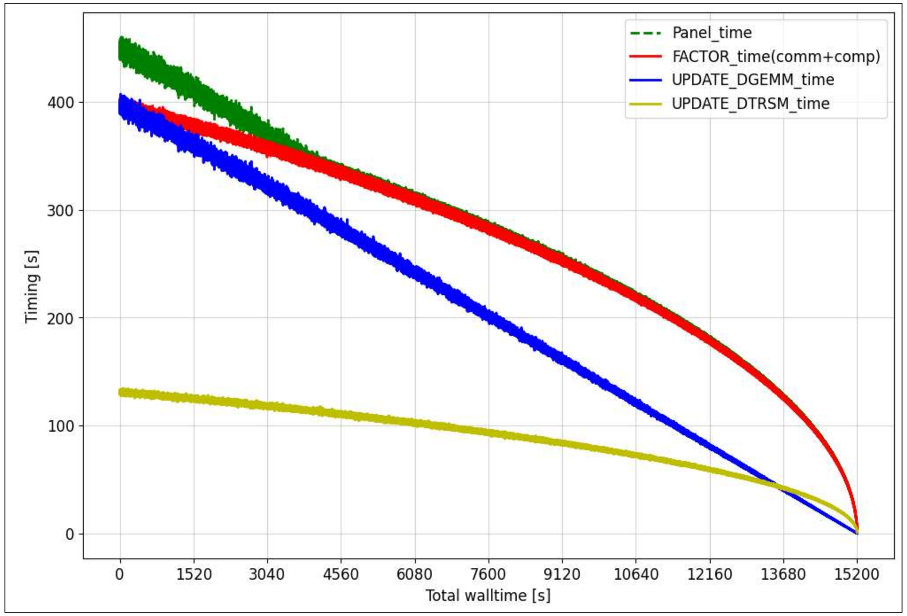
Initial distribution depends on *a priori* knowledge

Even with a perfect model, using single hdw/sftw parameters gives a single data point, which cannot be accurate, or be very lucky !

HPCG-UQ modeling on 4608 PVC [mean 1264.58 TF/s], vs HPCG Measurements done on 768 Aurora nodes [1259.94 TF/s] .
In Green are the HPCG and Stream PDF.In Blue are the uncertainties on input parameters. (Phil)

HPL Analytical modeling @ 1.012 EF/s



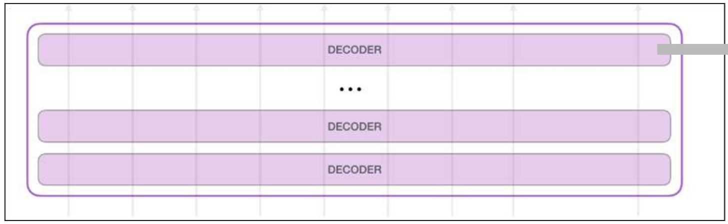


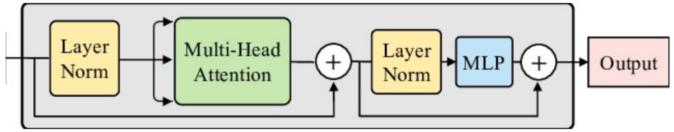
HPL can be approximated with XLS very quicky but the Analytical model can simulate compute and communications panel after panel.
(with random noise to mimic machine noises, CPU and GPU efficiency, dimensions, BWs)

Limitations : No topology. Need to mimic the real implementation. Hard to model all parameters behavior (P, Q, NB variations)

SIMGRID can do it too but with limited number of nodes.

Analytic LLM





Structure of transformer-based LLMs (megatron- GPT, Llama...)

Transformer based LLMs: are built using transformer decoder blocks. With each block comprising two sub-components:

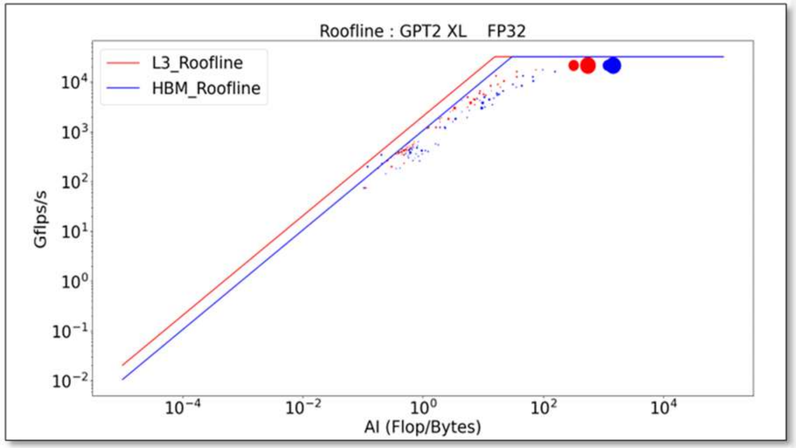
- Multi-head Attention
- MLP (Multi-layer perception)

We calculate the number of floating-point operation for each sub-components:

- Hidden size, The number of attention heads, the sequence size
- The training batch size, micro batch size
- Number of transformer blocks

	Analytical model	PVC RUN	Error
Num flop (Tflop)	3433	3037,334798	13%
Elapsed time (second)	153	160,8311873	-5%
Throughput (tokens/second)	2342,48366	2228,423516	5%

Lamma training 1,46B : Comparison of the analytical model and the results of the runs on 1 tile PVC



Advantages : The result is accurate, which can be attributed to the compute-bound nature of this case test

Limitations : Handling various parallelisms (projection at scale) and memory impact (size, bw, lat). WIP in the community

Conclusion

Conclusions

- There is no single tool to model all hardware and applications details at every scale
- The definition of the objectives is the most important starting point
  - Clear Definition of input / output, i.e what questions we want to solve
    - a new u-arch / a SOC / a system platform , switch, IO, memory tiers, power, TCO
  - Definition of the "application" scale, probably the easiest
    - a few instructions / a kernel /a proxy application / a full application
- Accuracy and speed remains orthogonal
  - There are orders of magnitude between each level.
  - Somehow difficult to merge granularity but propagating uncertainties is mandatory.
  - Full tracing remains very difficult.
- Collection tools are critical. Investing in opensource community and standardization is crucial.
- Modeling Scientific computing and AI applications is different, but AI remains a HPC application

Notices and Disclaimers

Statements in this document that refer to future plans or expectations are forward-looking statements. These statements are based on current expectations and involve many risks and uncertainties that could cause actual results to differ materially from those expressed or implied in such statements. For more information on the factors that could cause actual results to differ materially, see our most recent earnings release and SEC filings at www.intc.com.

All product plans and roadmaps are subject to change without notice.

Performance varies by use, configuration and other factors. Learn more on the Performance Index site. Intel technologies may require enabled hardware, software or service activation.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.