

ベイジアン（ニューラルじゃない） ネットワークの構造学習の高速化

東京大学 宮城竜大

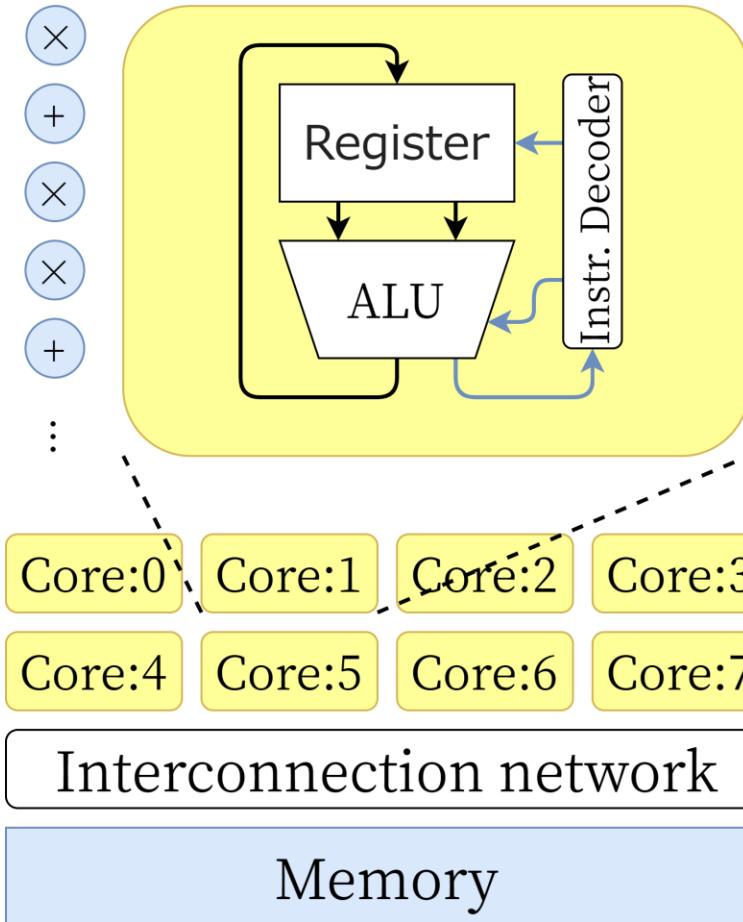
CV



- ・宮城竜大（みやぎりょうた）
 - ・2021年3月 京都大学 工学部情報学科卒業
 - ・2023年3月 “ 情報学研究科修了
 - ・2023年4月 東京大学 情報理工学系研究科
- ・自律移動ロボットにおける画像処理のFPGAオフロード
- ・SmartNICでの低遅延画像特徴点抽出
- ・複数FPGAによるストリーム計算の性能モデリング
- ・ベイジアン（ニューラルじゃない）ネットワークの構造学習の高速化
- ・...
- ・免責：HLSしか書いてない

問題に特化したアーキテクチャで高速化！

General-purpose architecture (Von Neumann architecture)

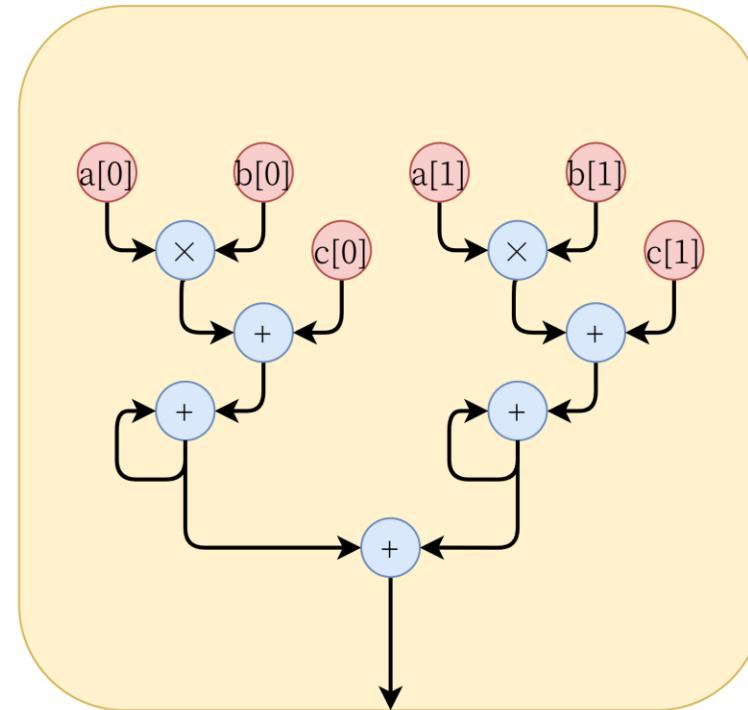


Complicated mechanism
(e.g. Branch Predication)

Inefficient data transfer through memory

Comm./sync.
costs among
cores/threads

Domain-specific architecture



Efficient domain-specific mechanism

Direct data transfer from ALUs to ALUs
(Dataflow)

Extensive spatial /temporal parallelism

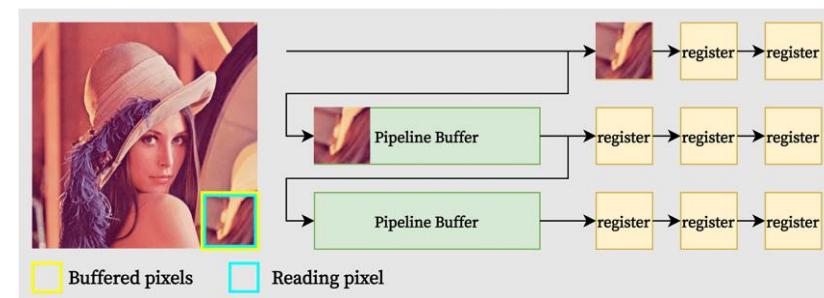
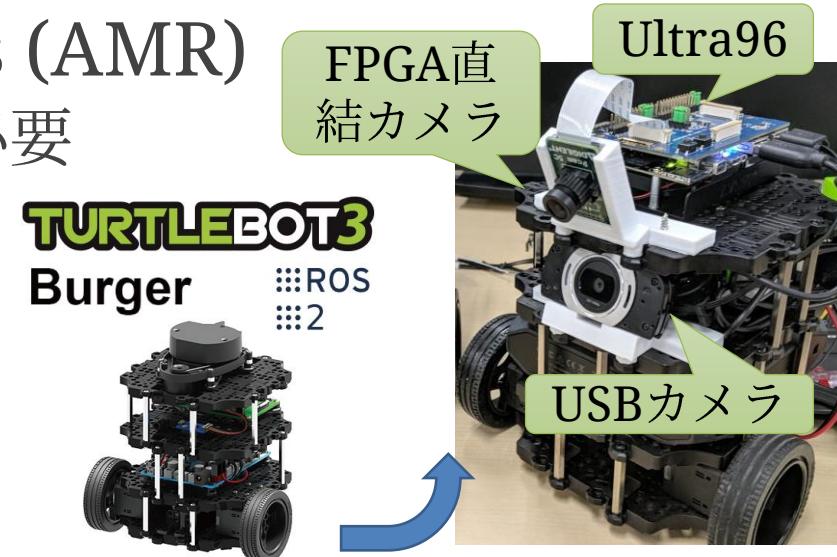
自律移動ロボットにおける物体検出の高速化

- 自律移動ロボット Autonomous mobile robots (AMR)

- 電力要件の下で低遅延で信頼性の高い意思決定が必要
- FPGA: 最適なトレードオフ (諸説あり)
 - 計算性能, 消費電力, 低レイテンシ

- FPGA Design Competition

- 学会付属のFPGAベースAMRロボコン
- 路面や信号などの物体認識と意思決定タスク
- Zynq Ultra96 + Turtlebot3
- 線形SVMによる赤信号の認識
 - FPGAで高速化



Miyagi, Ryota, et al. "Zytlebot: FPGA integrated ros-based autonomous mobile robot." 2021 International Conference on Field-Programmable Technology (ICFPT). IEEE, 2021.

FPGA搭載SmartNIC画像特徴点抽出@フィックスターズ

- SLAM (Simultaneous Localization and Mapping)

- 自己位置推定 + 地図作成を同時に使うタスク
- 自律移動ロボットや自動運転に必須
- ただし、それらの環境ではマシンパワーが足りない 🤯
- => サーバ側で低遅延に実行 😊

- SmartNIC (Network Interface Card)

- ネットワークインターフェース + FPGA
- TCP, UDP等ネットワークタスク + α をオフロード可能

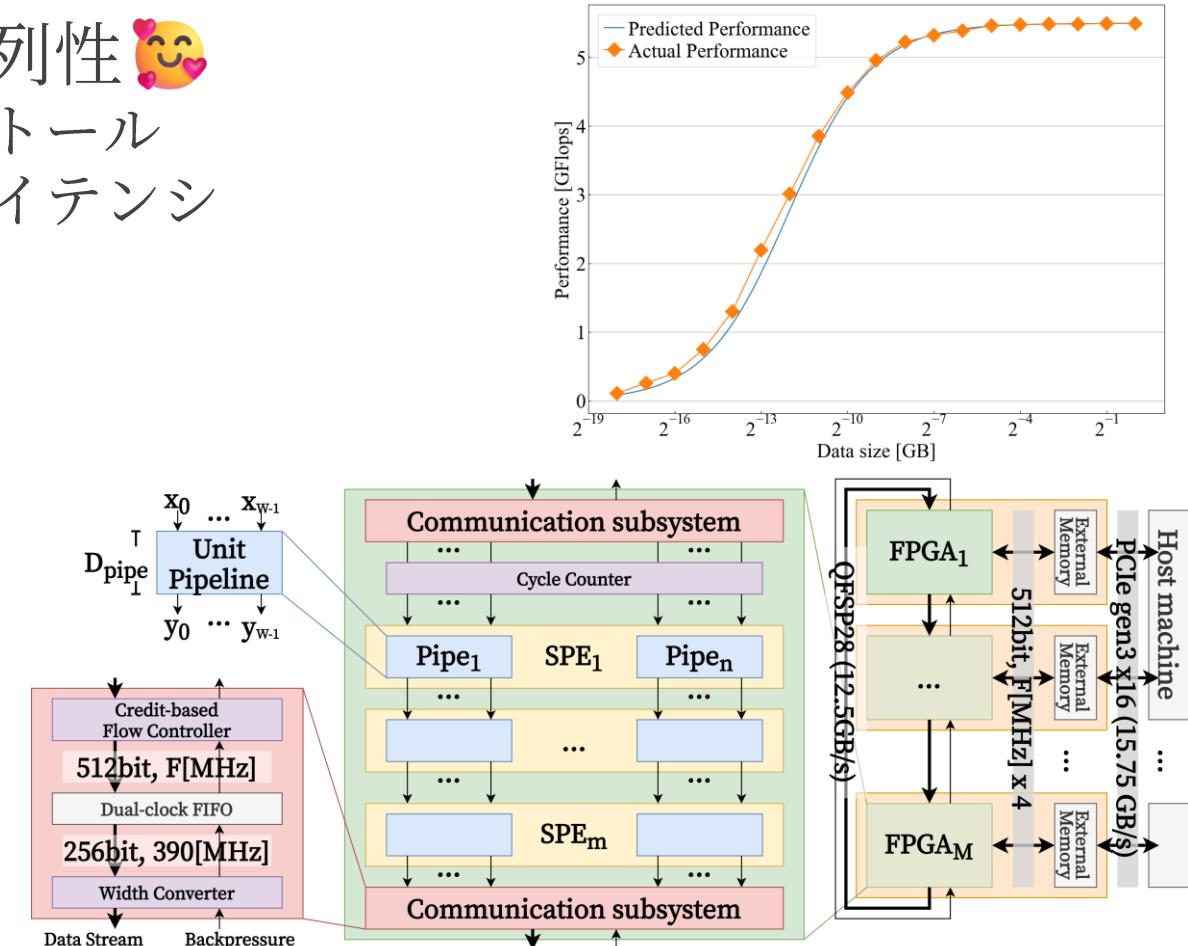
- 画像特徴点抽出 (SLAM処理の一部, 前処理)

- ホストCPUなしで、4K 画像140fps での高速処理 😊



複数FPGAデータフロー計算の性能モデリング

- 複数 FPGAを直列に接続して更なる並列性 
 - メモリ・ネットワーク帯域幅によるストール
 - パイプラインオーバーヘッド, 通信レイテンシ
- 理研FPGAクラスタESSPER
 - 16枚のIntel PAC D5005
 - FPGA間専用ネットワーク
 - AFUShell SoC
 - DMA, Crossbar,
 - Inter-FPGA network interface with credit-based flow control
- 1D リングの場合の性能モデリング
 - 実際の計算特性をうまく補足

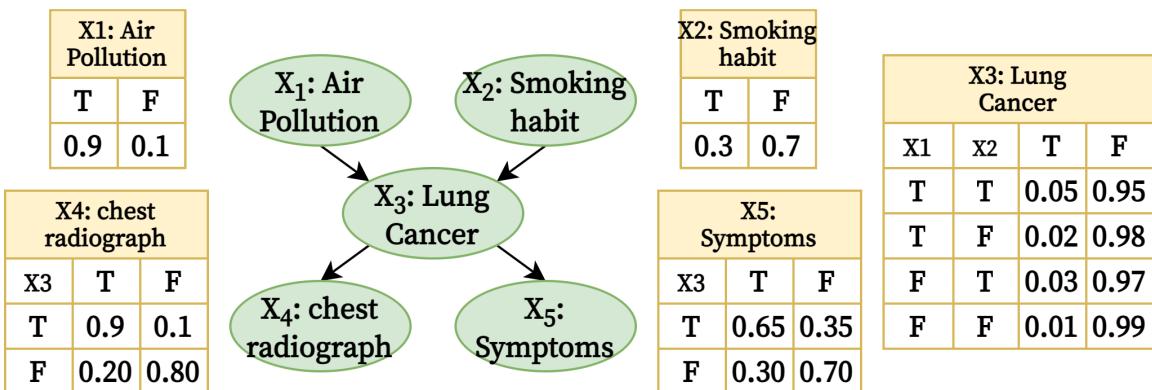


Miyagi, Ryota, et al. "Performance Modeling and Scalability Analysis of Stream Computing in ESSPER FPGA Clusters." 2023 International Conference on Field Programmable Technology (ICFPT). IEEE, 2023.

ベイジアン（ニューラルじゃない）ネットワーク

• Bayesian Network

- ベイズの定理 + ネットワーク
- 確率変数間の依存関係をネットワークで表す確率モデル

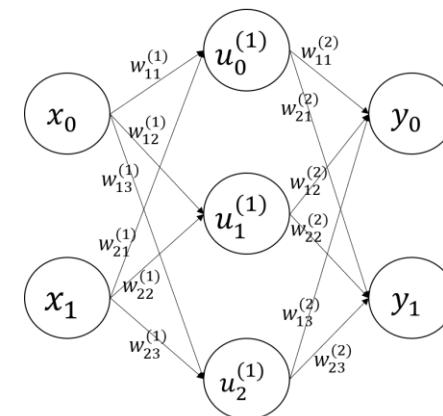


- 😊 混同されて門前払いされがち

• Bayesian Neural Network

- ベイズ的なアプローチによるNN
- NNのパラメータに確率分布を割り当て、不確実性を自然に定量化

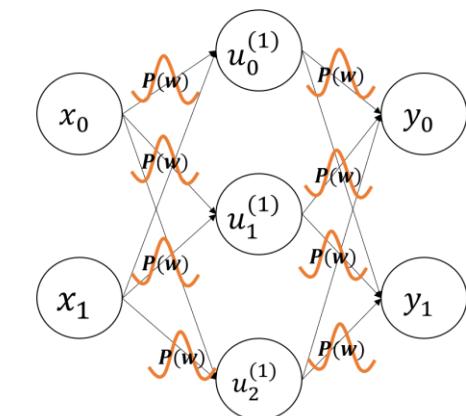
通常のニューラルネットワーク



$$\mathbf{y} = f(\mathbf{w}, \mathbf{x})$$

パラメータ \mathbf{w} は定数
入力 \mathbf{x} が決まれば \mathbf{y} は確定

ベイジアンニューラルネットワーク



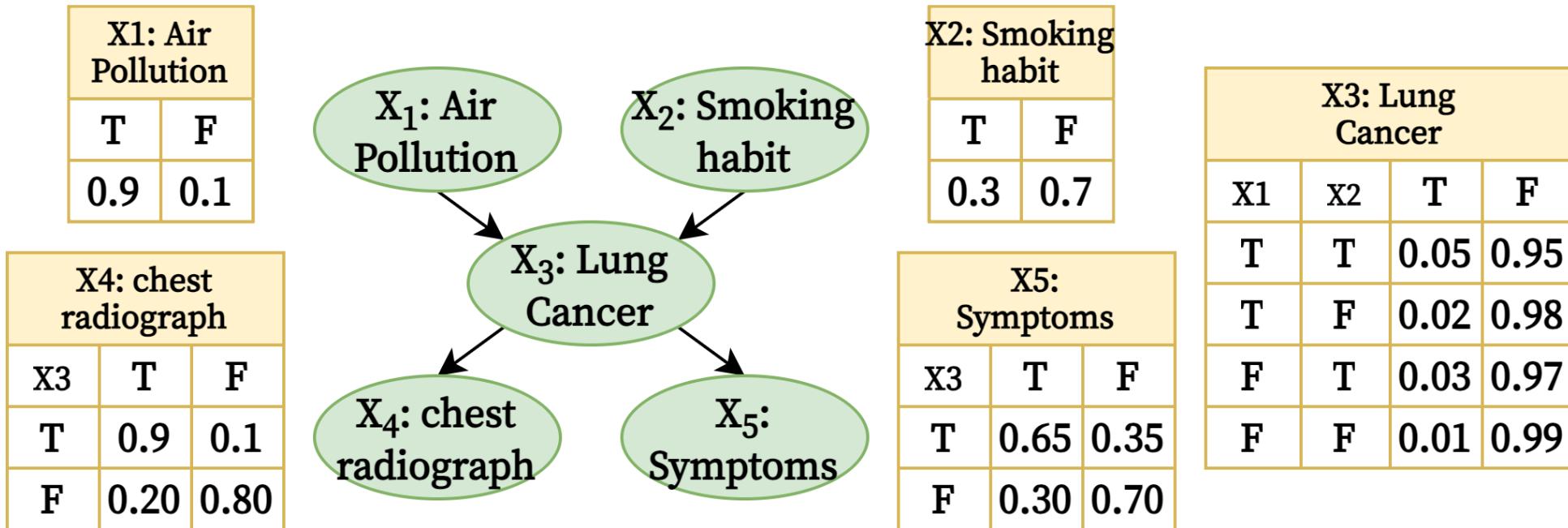
$$\mathbf{P}(\mathbf{y}) = f(\mathbf{P}(\mathbf{w}), \mathbf{x})$$

パラメータ \mathbf{w} を確率変数と考える
入力 \mathbf{x} で演算するたびに \mathbf{Y} も一定の分布に広がる

出典: https://qiita.com/qiita_kuru/items/8d20986b51c8e57e51b5

ベイジアンネットワーク(Bayesian Network, BN)

- 確率変数間の条件付き独立関係をDAGに符号化した確率モデル
 - 通常は離散, NNや決定木に対しても競合的な推論精度 😍 (AAAI2024)
- 確率変数集合 $\{X_1, X_2, \dots, X_n\}$ 上のBNは以下の (G, Θ) で定義される
 - 各変数 X_1, X_2, \dots, X_n に対応するノード集合によって構成されるDAG構造 G
 - 条件付き確率パラメータ集合 $\Theta = \{ p(X | \text{Pa}_X) \}$, Pa_X : 変数 X の親変数集合



ベイジアンネットワークのうれしさ

- DAG構造に符号化された条件付き独立性を仮定すると以下が成り立つ

$$p(X_1, X_2, \dots, X_n) = \prod_{i=1}^n p(X_i \mid \mathbf{Pa}_{X_i})$$

- 同時確率分布がコンパクトな条件付き確率の積に因数分解される
- (左辺) 離散確率変数の同時確率分布を直接推定すると...? 🤔
 - 😇 指数的な数のパラメータが必要 (X_1, X_2, \dots, X_n がすべて2値の場合 2^n)
 - 😇 各パラメータの妥当な推定値を得るために必要なサンプルの数も膨大
- (右辺) 因数分解が既知だとすると...? 🤔
 - 😊 各条件付き確率は局所的な変数集合に注目、少ないパラメータで記述可
 - 😊 比較的少数・疎なサンプル集合からも頑健な推定が可能
- i.e., BNは条件付き独立性を利用した同時確率分布の圧縮表現

ベイジアンネットワークの学習

$$p(G, \Theta | D) = p(G | D) p(\Theta | G, D)$$

- 構造学習(Structure Learning)

- データ D からDAG構造 G を学習, ただし, DAG/因数分解の候補の数は指数的

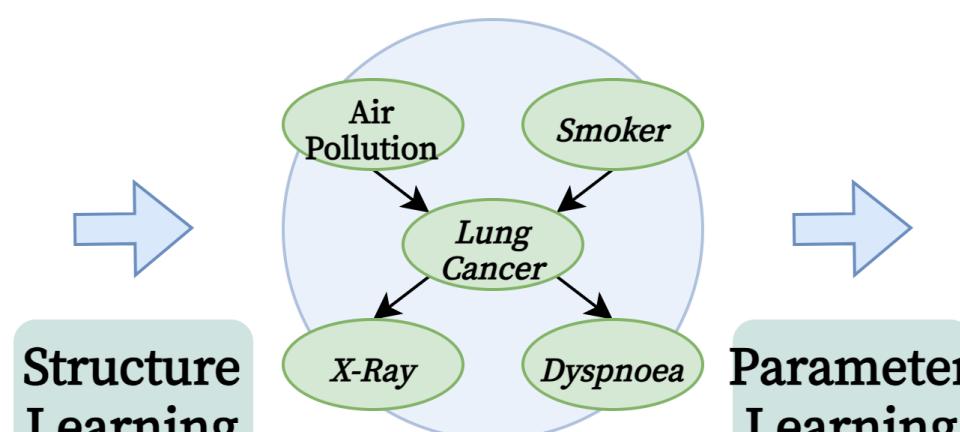
- パラメータ学習(Parameter Learning)

- データ D とDAG構造 G からパラメータ Θ を学習

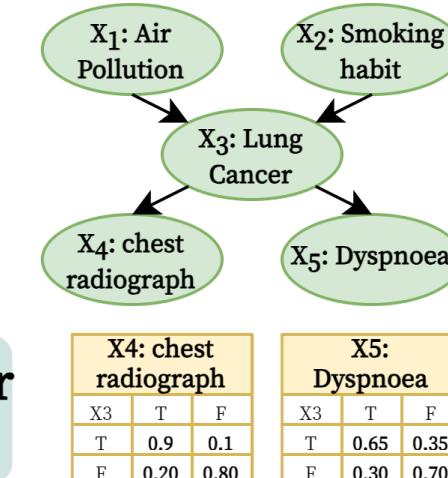
- 利活用のためには, 構造学習の高速化が必要不可欠

Records	Random Variables				
	Air Pollution	Smoker	Lung Cancer	X-Ray	Dyspnoea
Alex	True	True	False	True	False
Bob	False	True	True	False	True
Charlie	True	False	False	True	False
:	:	:	:	:	:
Zack	True	False	True	False	True

Structure Learning



Parameter Learning



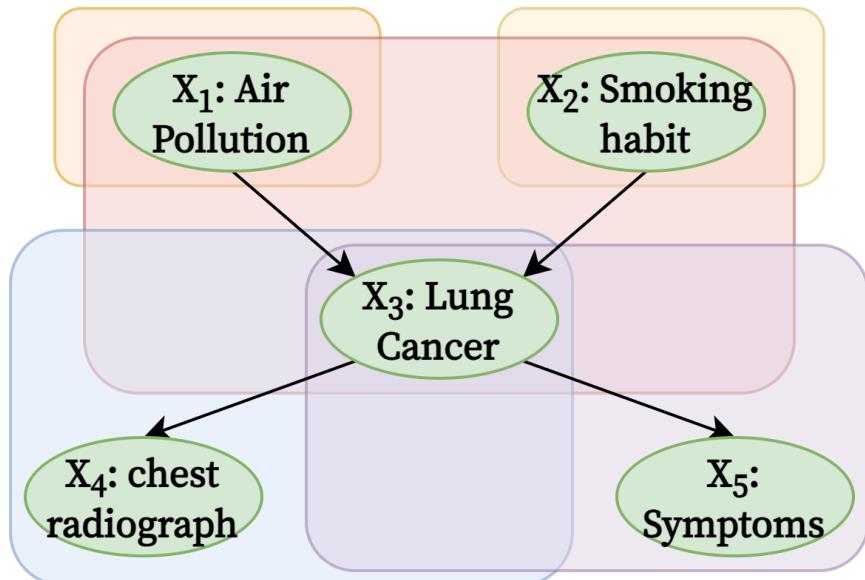
X1: Air Pollution	
T	F
0.9	0.1

X2: Smoking habit	
T	F
0.3	0.7

X3: Lung Cancer			
X1	X2	T	F
X1	X2	0.05	0.95
T	F	0.02	0.98
F	T	0.03	0.97
F	F	0.01	0.99

Score-based構造学習

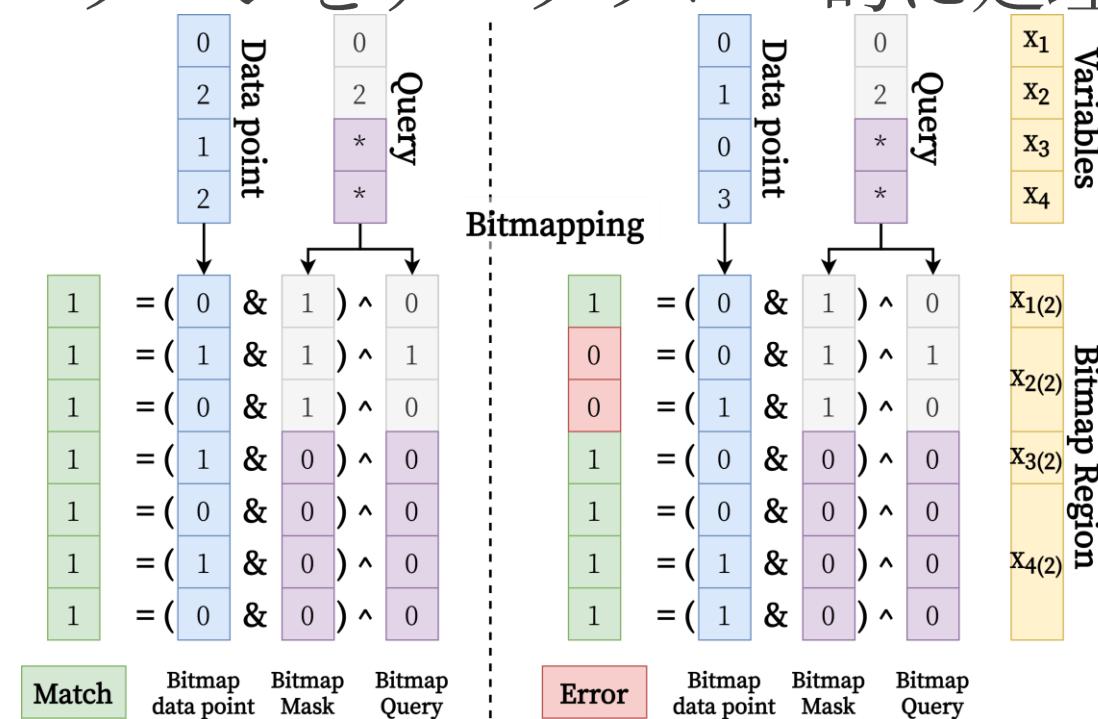
- =データ D からDAG構造 G /因数分解を学習（尤度最大化）
- (DAG構造/因数分解の尤度) = \prod (局所構造/各因数の尤度)
- 局所構造/各因数の候補数は指数的 😊
 - 実際には親変数の数を制限するが、それでも多い、高速化しましょう

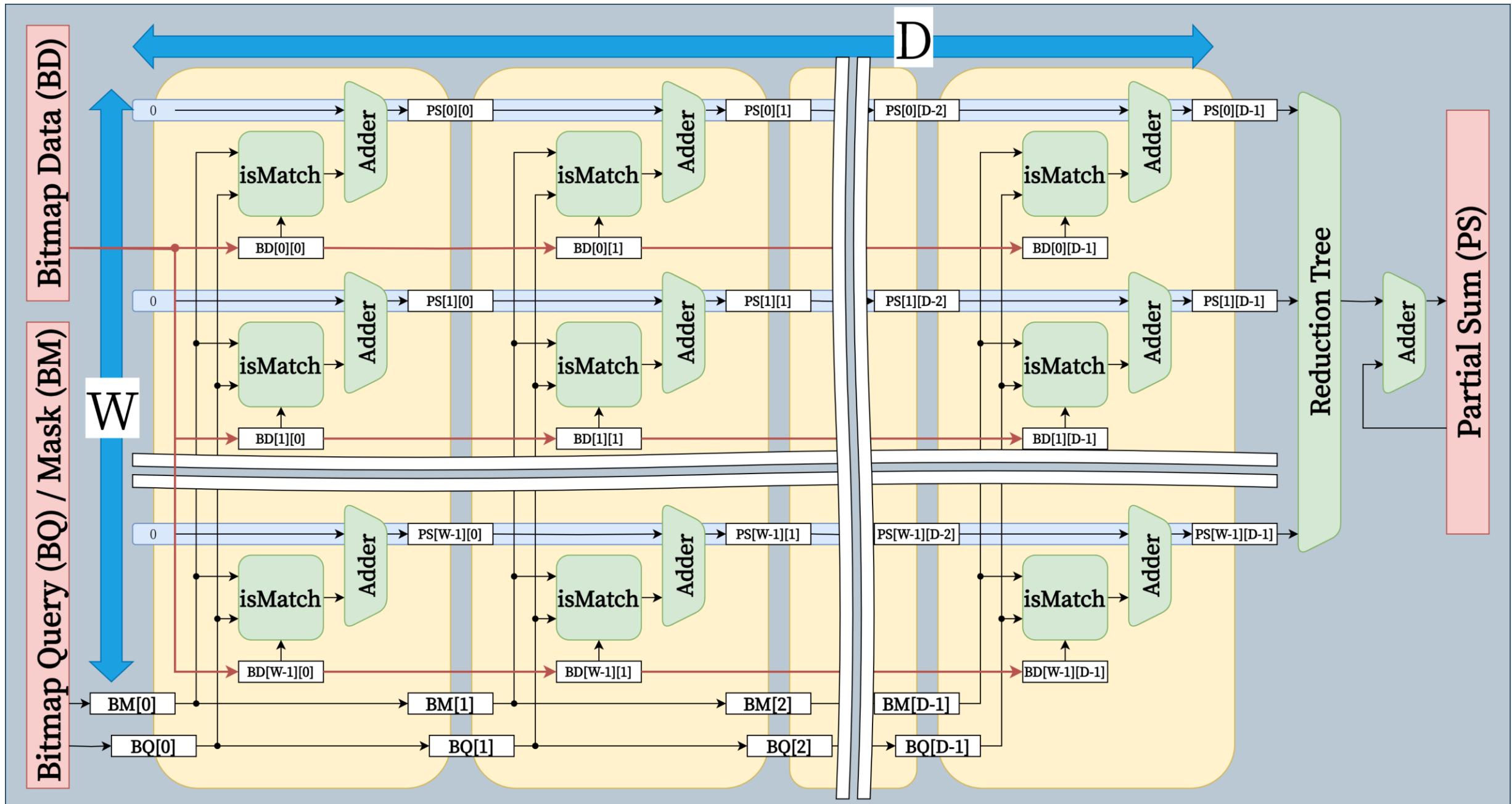


$$p(\mathcal{S} | G) = + \text{LS}(\mathcal{S}, X_1, \{\}) \\ + \text{LS}(\mathcal{S}, X_2, \{\}) \\ + \text{LS}(\mathcal{S}, X_3, \{X_1, X_2\}) \\ + \text{LS}(\mathcal{S}, X_4, \{X_3\}) \\ + \text{LS}(\mathcal{S}, X_5, \{X_3\})$$

局所構造学習の高速化

- 局所構造/各因数の尤度は「特定パターンのデータポイント数に依存」
 - Smoker=True, LungCancer=False, others=**
 - データ数は固定（数千から数万），パターン数は指数的
- そこで、データはキャッシュして、パターンをデータフロー的に処理
- あらかじめBitmapしてマッチング
- ランダムアクセスが排除
- Bit演算のみ、膨大に可能
- 問題の差異を統一的に扱える





FPGA card

Device Memory

Register

Module

Pipeline Stage

Add-reduce Lane

速度比較

- 局所スコアをすべて求める計算時間を測定
- 実験環境
 - FPGA Machine
 - Xilinx Alveo U50
 - Vitis v2020.2.
 - GPU Machine
 - NVIDIA TITAN RTX
 - CUDA 12.1
 - 少し古いが同時期（2018年）
- 実験条件
 - (1) CPU (DP的に賢く)
 - (2) CPU + GPU
 - 各パターンをスレッドごとに
 - (3) CPU + FPGA
- 😍 FPGAが一番早い

m	N	GPU Machine							FPGA Machine						
		(1) ADtree		(2) GPU					total	(3) FPGA				agg.	total
		Bmp	H2C	Kernel	C2H	Agg	Bmp	H2C		Kernel	C2H				
ALARM (Medium Network)	3	2048	0.95	0.18	0.01	0.05	< 0.01	0.13	0.38	-	-	-	-	-	-
		4096	1.54	0.17	0.01	0.07	< 0.01	0.13	0.39	0.13	0.01	0.02	< 0.01	0.23	0.41
		8192	3.23	0.17	0.01	0.12	< 0.01	0.13	0.45	0.13	0.01	0.04	< 0.01	0.21	0.39
		16384	7.82	0.17	0.01	0.24	< 0.01	0.13	0.56	0.13	0.01	0.08	< 0.01	0.22	0.44
		32768	54.01	0.17	0.01	0.49	< 0.01	0.12	0.80	0.13	0.01	0.17	< 0.01	0.22	0.53
		65536	162.72	0.17	0.01	0.99	< 0.01	0.12	1.30	0.13	0.01	0.33	< 0.01	0.21	0.68
	4	2048	10.00	2.77	0.26	0.58	0.09	2.36	6.07	-	-	-	-	-	-
		4096	14.91	2.77	0.26	1.16	0.09	2.43	6.71	2.40	0.21	0.39	0.02	3.76	6.78
		8192	27.49	2.75	0.26	2.33	0.09	2.44	7.87	2.40	0.21	0.78	0.02	3.76	7.16
		16384	60.08	2.76	0.26	4.69	0.09	2.39	10.19	2.41	0.21	1.56	0.02	3.85	8.04
		32768	378.54	2.75	0.26	9.45	0.09	2.32	14.88	2.39	0.21	3.12	0.02	3.82	9.55
		65536	1171.28	2.76	0.26	19.22	0.09	2.45	24.79	2.40	0.21	6.24	0.02	3.87	12.74
HAIFINDER (Large Network)	2	1024	0.44	0.06	0.01	0.02	< 0.01	0.05	0.14	0.13	< 0.01	< 0.01	< 0.01	0.10	0.26
		2048	0.54	0.05	0.01	0.03	< 0.01	0.05	0.15	0.11	< 0.01	0.02	< 0.01	0.09	0.23
		4096	0.81	0.06	0.01	0.07	< 0.01	0.05	0.18	0.11	< 0.01	0.03	< 0.01	0.09	0.24
		8192	1.42	0.05	0.01	0.11	< 0.01	0.05	0.22	0.11	< 0.01	0.06	< 0.01	0.09	0.28
		16384	7.64	0.05	0.01	0.17	< 0.01	0.05	0.28	0.11	< 0.01	0.13	< 0.01	0.09	0.34
		32768	30.18	0.05	0.01	0.35	< 0.01	0.05	0.46	0.11	< 0.01	0.25	< 0.01	0.09	0.46
		65536	85.14	0.05	0.01	0.70	< 0.01	0.04	0.81	0.11	< 0.01	0.51	< 0.01	0.10	0.73
	3	1024	12.32	2.86	0.53	0.57	0.10	2.80	6.87	5.96	0.47	0.41	0.02	4.14	11.00
		2048	16.15	2.89	0.53	1.14	0.10	2.82	7.49	5.97	0.47	0.83	0.02	4.23	11.52
		4096	21.74	2.87	0.52	2.30	0.10	2.80	8.60	5.98	0.47	1.66	0.02	4.21	12.33
		8192	32.79	2.86	0.53	4.60	0.10	2.71	10.80	5.97	0.47	3.31	0.02	4.15	13.91
		16384	101.76	2.87	0.53	9.27	0.10	2.65	15.42	5.95	0.46	6.62	0.02	4.07	17.12
		32768	365.82	2.86	0.53	18.81	0.10	2.54	24.85	5.96	0.47	13.25	0.02	4.02	23.72
		65536	1187.24	2.87	0.53	37.78	0.10	2.46	43.75	5.95	0.47	26.49	0.02	4.00	36.93

性能モデリング

- FPGAの場合はキャッシュできる
- データポイントの数が増えれば
- もっとスケールする

- スケーラビリティの評価のため
- 性能モデルを作成して検証
- 十分な数のデータがあれば、
- リソース量に応じた性能向上

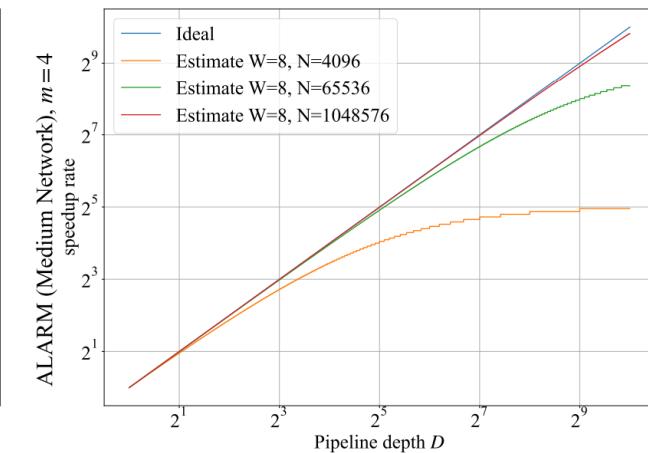
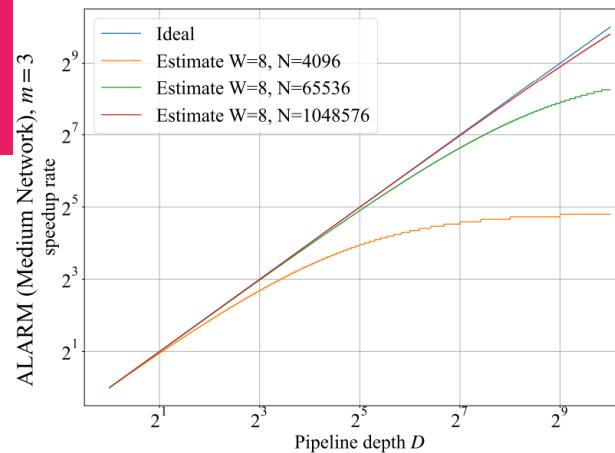


Fig. 13. Relative performance improvements with increasing computational resource using the ALARM Benchmark Bayesian Network. Notably, with larger datasets, performance enhancements align closely with the ideal scaling scenario. Note that these comparisons reflect the overall acceleration rate of local score computation, encapsulating both the data bitmap creation and transfer processes

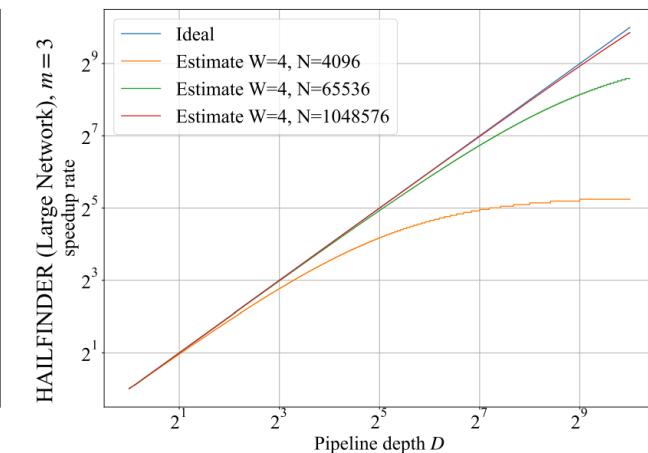
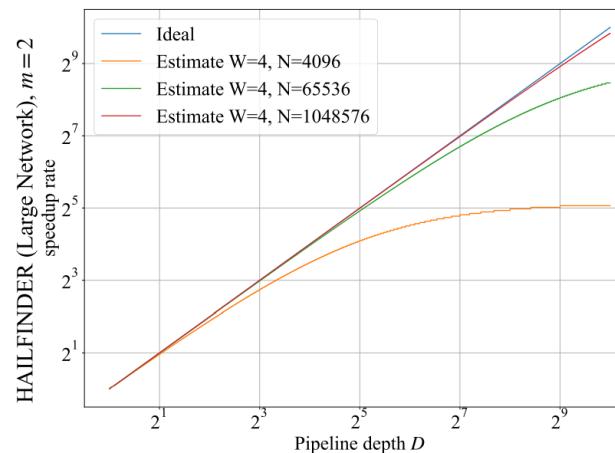


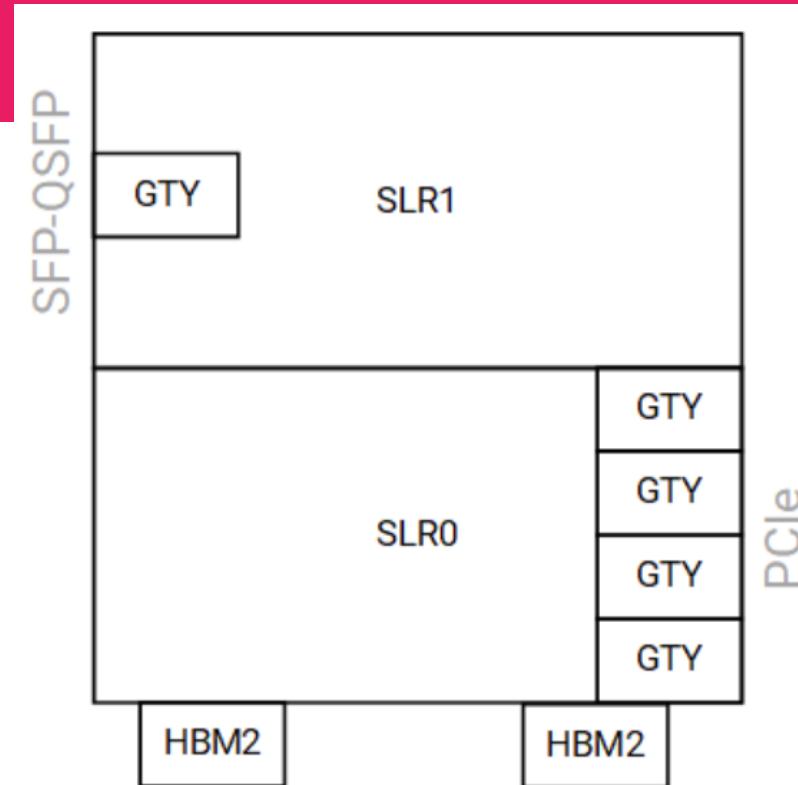
Fig. 14. Relative performance improvements with increasing computational resource using the HILFINDER Benchmark Bayesian Network. Notably, with larger datasets, performance enhancements align closely with the ideal scaling scenario. Note that these comparisons reflect the overall acceleration rate of local score computation, encapsulating both the data bitmap creation and transfer processes

実はFPGA実装はまだ数倍早くできるはず

- 実はVitis 2020.2 + U50は修士の時の京大のサーバでの結果
- ボトルネックは通信帯域幅ではなくLUT数, メモリ帯域幅には余裕あり
- U200の方がLUT数が多いのでそっちでやる...?
 - Vitis 2022.2(?) + U200 だと, **PS[index] += _ps** がHLSでII=1にならない
 - Vitis HLSのバージョンの問題?
 - HBM2じゃないせい?
 - そこで, 東大の研究室に余っていたU50とU200を差し替える
 - U50認識せず, ファームウェアの書き換えも不能, 詰み
 - のちにVitis 2022.2(?) + U280 (HBM2)の環境でもだめ, 多分バージョンの問題
 - 諦めて, 自分で書いてみる
 - PS[index] をburst readするカーネルとburst writeするカーネルを分けてパイプで接続
 - 配置配線に失敗, 同じ数のモジュールが乗らない..., リソースの最適化の関係?
- ブラックボックス, ベンダーロック, HLSの限界, とにかく技量不足

実はFPGA実装はまだ数倍早くできるはず

- 実はU50のSLR0しか使えていない
 - SLRs (Super Logic Regions) , U50は2つ,
 - それぞれにカーネルを用意し, パイプで接続
 - HBM2 => SLR0 => SLR1 => SLR0 => HBM2
 - 接続にはSLL(Super Long Line)リソース消費
- SLLが足りず配置配線に失敗する 😊
 - 実質的に使用可能なリソース量は半分に... 😰
- 😵 FPGA何もわからない..., 使いこなせない...



	D · W	LUT	LUTRAM	FF	BRAM	[MHz]
64-bit	4096	300,198 (34.50%)	11,443 (2.85%)	427,874 (24.59%)	190.5 (14.17%)	221.5
128-bit	1024	201,648 (23.18%)	11,421 (2.84%)	276,783 (15.91%)	192.5 (14.32%)	233.1
Available	-	870,016 (100.00%)	402,016 (100.00%)	1,740,032 (100.00%)	1,344 (100.00%)	-

Thank you