データフロー主導型カスタム計算機システム開発基盤と自動チューニング技術

第10回 自動チューニング技術の現状と応用に関するシンポジウム(ATTA2018)

佐藤 幸紀

豊橋技術科学大学　大学院工学研究科
情報・知能工学系　准教授
（兼）JST さきがけ　研究者

TOYOHASHI UNIVERSITY OF TECHNOLOGY

研究の背景

Society 5.0 社会の実現に向けて

Image: Creative Commons 3.0

https://www.insightssuccess.com/

自動運転

知的ロボット

https://www.icd.cs.tut.ac.jp

人工知能技術をフィールドに展開

実社会への応用には性能や経済性も重要

深層学習処理においては現状の
CPU/GPU技術だけでは性能が足りない

Society5.0/人工知能社会を実現する上では
「システムレベルの性能不足」
を解決していくことが必要

“CPUs to GPUs to FPGAs and, finally, on ASICs”

深層学習処理専用ハードウェア

Google TPU

(https://cloud.google.com/)

特定の応用ドメインへのカスタム化における課題

①高い性能効率
②生産性・開発コストの問題
③アルゴリズム進化への追従性（柔軟性）

TOYOHASHI UNIVERSITY OF TECHNOLOGY

AI技術の最適化、カスタム化、社会実装@計算機システム性能工学研究室

実社会へ向けたAI技術の展開

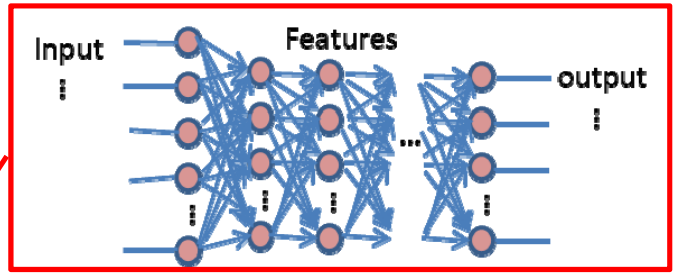
https://www.icd.cs.tut.ac.jp
Intelligent robot

https://www.insightssuccess.com/
Self-driving car

AI技術を使ったシステム最適化

Open ecosystem

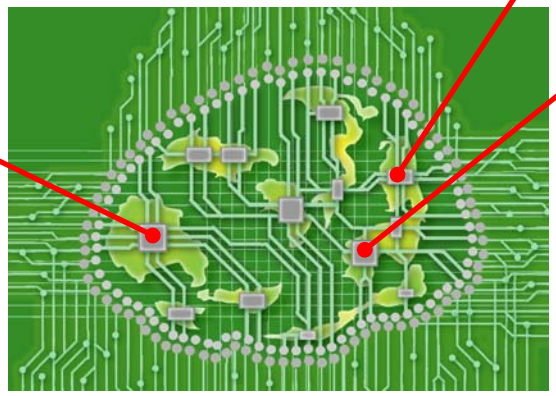
AIを中心とした計算パラダイムの創出

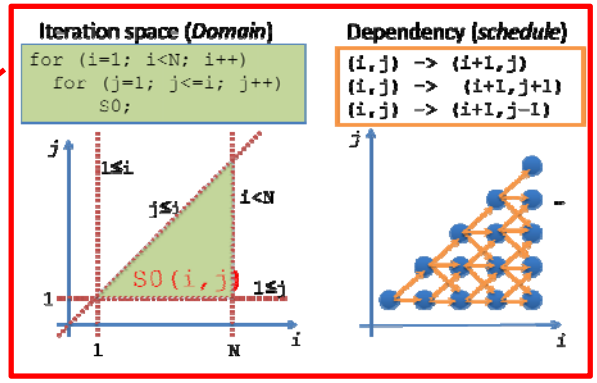


Multi-layer perceptron neural network

AI技術特化型カスタム計算機

MISD stytle Dataflow pipeline on FPGA (Maxeler Galava card)

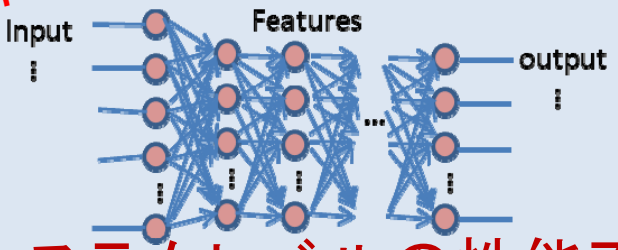




性能工学に向けたループネストのモデル化とPolyhedral最適化

AI技術と計算機システム性能工学

現状のCPU/GPU技術だけではまだまだ性能が足りない



「システムレベルの性能不足」を解決していくことが必要

我々のアプローチ

”Systems performance engineering”

=

MISD-stytle Dataflow custom pipeline on FPGA (Maxeler)

Domain specific hardware

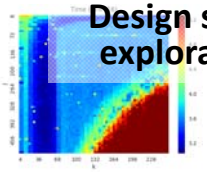
\+

Domain specific software

System optimizations technique using AI/ML

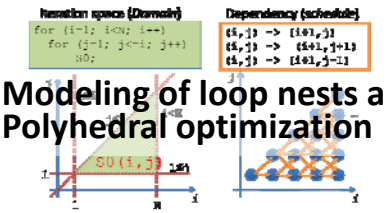
Analysis, Profile | **Modeling** | **Estimation, Inference** | **Action**

Performance profiler



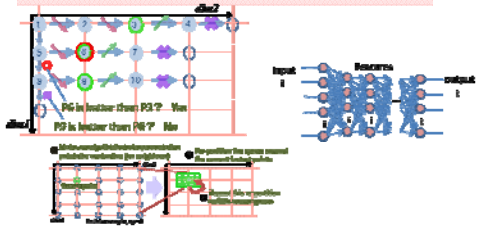
Design space exploration

Auto-tuning framework

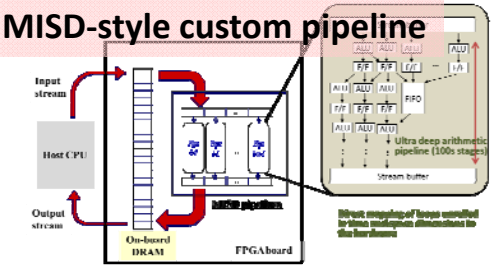


Modeling of loop nests and Polyhedral optimization

High-level optimization



MISD-style custom pipeline



データフロー主導による
カスタム計算機システム開発基盤の体系化

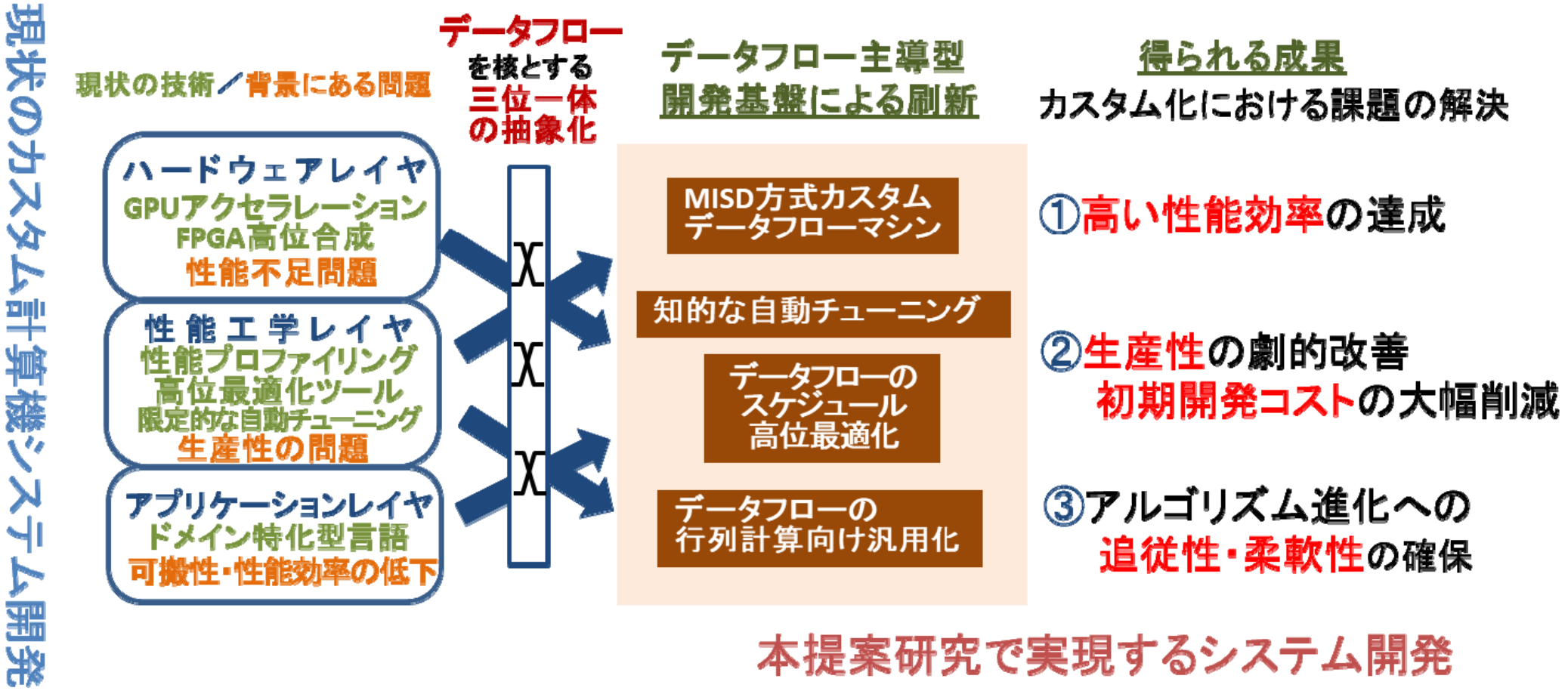
（2018-2021年度）
研究領域「Society5.0を支える革新的コンピューティング技術の創出」
（研究総括：井上　弘士、H30年度発足）

佐藤 幸紀

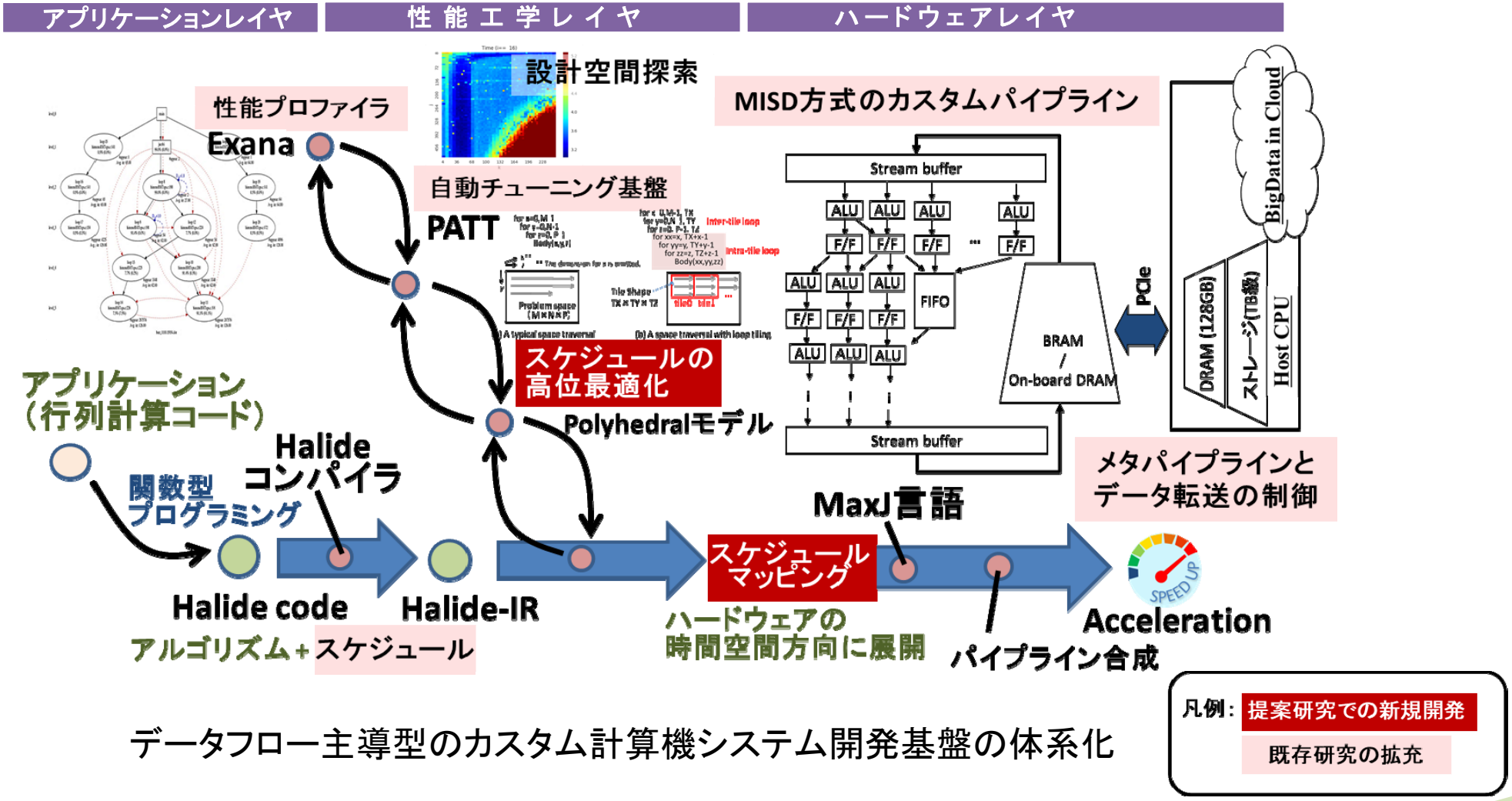
豊橋技術科学大学　大学院工学研究科
情報・知能工学系　准教授
（兼）JST さきがけ　研究者

TOYOHASHI
UNIVERSITY OF TECHNOLOGY

提案研究の目的

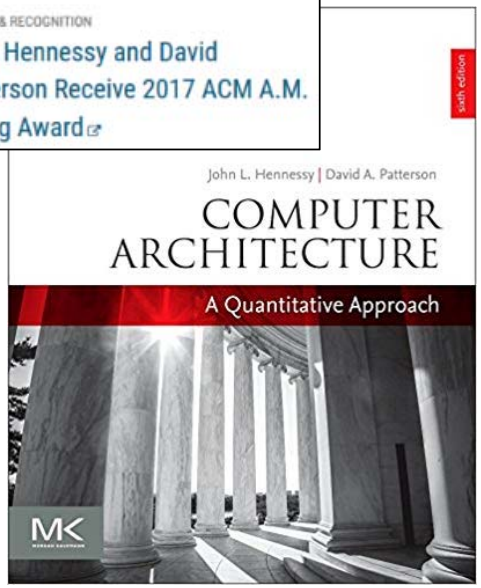

現状のカスタム計算機システム開発
現状の技術／背景にある問題
データフロー
を核とする
三位一体
の抽象化
データフロー主導型
開発基盤による刷新
得られる成果
カスタム化における課題の解決
ハードウェアレイヤ
GPUアクセラレーション
FPGA高位合成
性能不足問題
性能工学レイヤ
性能プロファイリング
高位最適化ツール
限定的な自動チューニング
生産性の問題
アプリケーションレイヤ
ドメイン特化型言語
可搬性・性能効率の低下
MISD方式カスタム
データフローマシン
知的な自動チューニング
データフローの
スケジュール
高位最適化
データフローの
行列計算向け汎用化
①高い性能効率の達成
②生産性の劇的改善
初期開発コストの大幅削減
③アルゴリズム進化への
追従性・柔軟性の確保
本提案研究で実現するシステム開発

研究構想



データフロー主導型のカスタム計算機システム開発基盤の体系化

計算機アーキテクチャと性能工学



Computer Architecture, Sixth Edition:
A Quantitative Approach
J. L. Hennessy / D. A. Patterson著

計算機アーキテクチャ

- 応用の現場（フィールド）が要求する性能が達成可能な計算原理の探求
  - 並列性、局所性、分割、同期、負荷均衡
- 定量的アプローチが重要
  コンピュータの性能・コスト・電力を定義し、計測し、要約する
- フィールドに展開する際に必要なコストを最大限下げる工学的手法も重要
  性能・コスト・電力のトレードオフに対して

定量的理解、分析、推定やプロファイリング技術が鍵　＝　性能工学

近年、ドメイン特化型のアクセラレータを混載したヘテロジーニアスコンピューティング環境が近年注目されている

- 性能不足問題が顕著化する中、性能電力比の改善が見込まれるため
- システム開発における生産性という工学的側面の研究も極めて重要
  - フィールドのアプリは年々多様化が進みコード規模増加の一途をたどる
  - 複雑なコードほど、ドメイン特化のための初期開発コストが膨大に
  - 多種多様なドメイン特化型の障壁であり、自動化や効率化が熱望される

＝　性能工学

ACM SIGPLAN主催の人工知能や経験的手法による並列システム向けソフトウェア工学に関するワークショップ

採録論文はACM Digital Libraryにて出版

2018年は11/6にボストンで開催

計算機の性能問題に対するアプローチ

計算機の性能工学に関する研究が重要であるという機運が国際的にも高まる

- ACM/SPEC主催 International Conference on Performance Engineering (ICPE)は2018年で8回目
- 国際会議において該当分野のワークショップが立ち上がる （ACM/IEEE SC、ACM SPLASH)

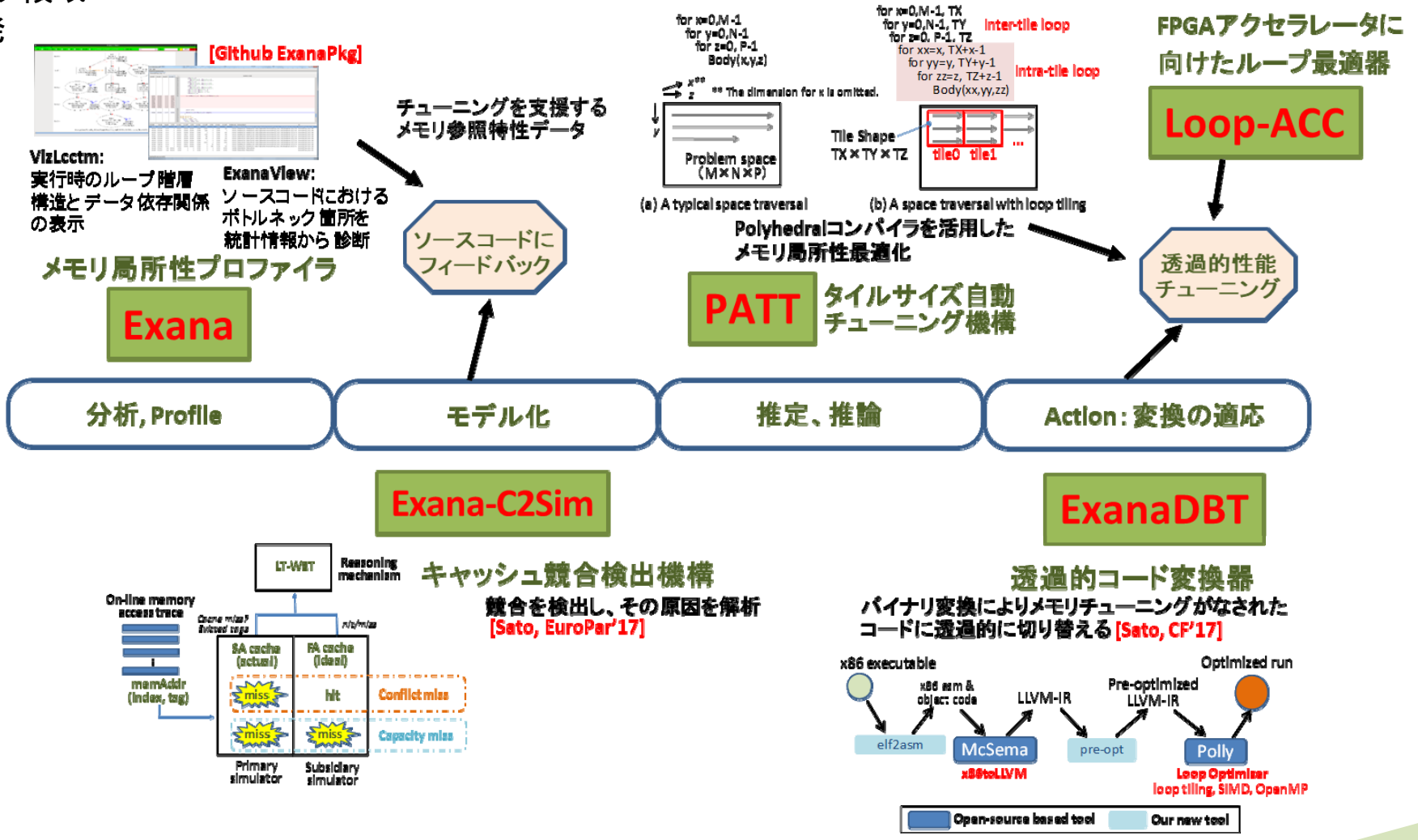
黎明期から積極的にコミュニティに参加
今後も、プレゼンスを高め、研究力を強化したい

本ワークショップを2015年開催分から共同で企画し、組織委員として運営に参加

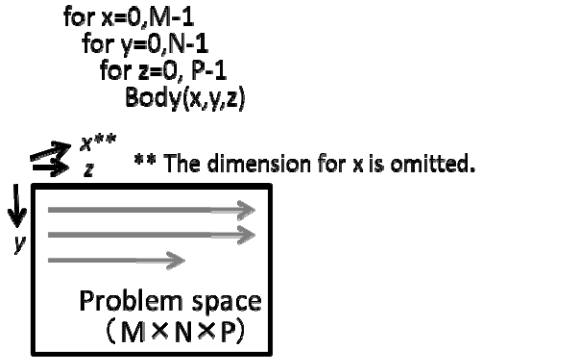
TOYOHASHI UNIVERSITY OF TECHNOLOGY

透過的メモリ局所性チューニングのためのツール群

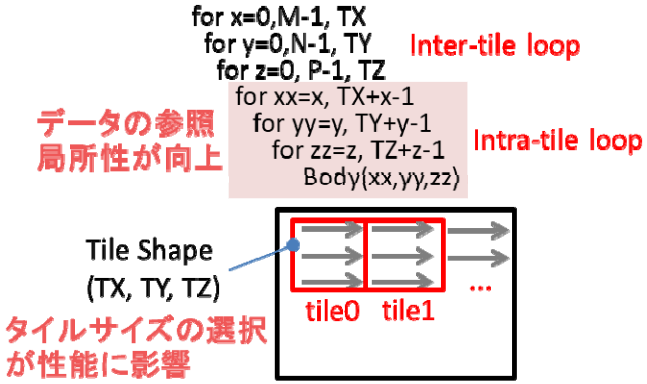
JST CRESTポストペタ領域
の支援を受け開発
(2012FY-2017FY)



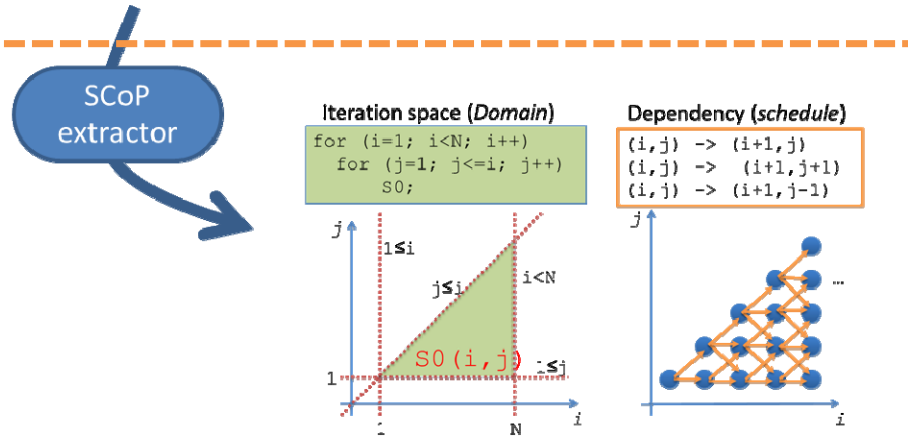
Polyhedralモデルによる高位最適化



(a) データ空間の走査とオリジナルのループ階層



(b) タイリングされたループ階層構造

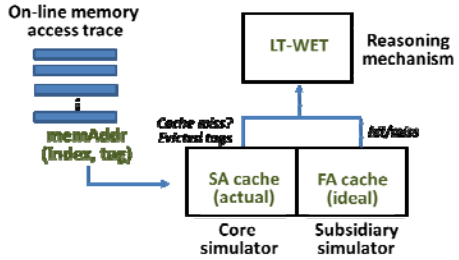


ループ構造のモデル化の例(SCoP形式)

Exana-C2Sim: Memory hierarchy performance simulator

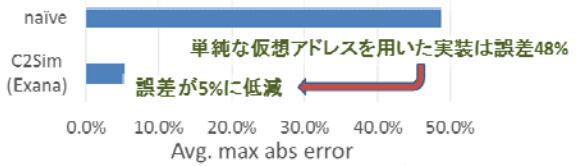
目的:
　キャッシュやメモリ階層に関するチューニングの半自動化
提案する手法:
　キャッシュライン競合に着目し、その発生の正確な検出、原因箇所の特定、及び、チューニング戦略の提示

- **正確な競合検出法:** FAキャッシュとSAキャッシュを同時にシミュレーションすることにより検出



競合検出法の概要

物理アドレス変換やL3キャッシュのスライス構造をモデル化



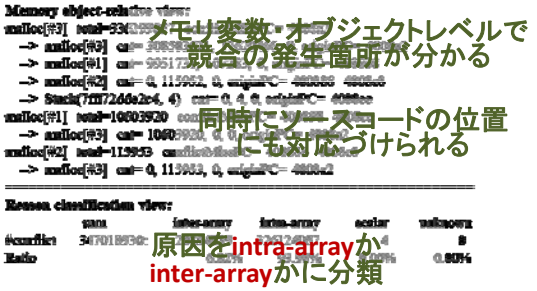
高度なキャッシュモデリングの効果

シミュレーション精度を大幅に改善

**ハードウェアプリフェッチ(HwPF)はオフとして計測

	Conflicts [%]		
	L1	L2	L3
doitgen	87.68%	98.44%	0.00%
3D-FDTD (1)	37.45%	0.00%	0.00%
Himeno (2)	92.94%	0.00%	1.76%

(1) FDTD: 128x128x64, timestep=50, # of mediums (prescribed by array 'id')=10
(2) Himeno Benchmark: OpenMP, C_Dynamic, size=S



プロファイル結果としてライン競合発生割合と原因解析を出力

ライン競合プロファイル結果を用いたチューニングフロー

	Tuning strategy	When
Opt.1	Intra-array padding insertion	To resolve intra-array conflicts
Opt.2	Use of hugetlbfs (2MB page)	To resolve conflicts in L2/L3 cache
Opt.3	Inter-array padding insertion	To resolve inter-array conflicts

doitgenのチューニング結果

	Speedup
Original	1.00
Opt.1	1.19
Opt.1+Opt.2	1.21
Opt.2	1.02

3D-FDTDのチューニング結果

	Speedup
Original	1.00
Opt.3	1.32

Himenoベンチのチューニング結果

	HW PF	Speedup (**)
1 thread	off	1.62
	on	1.75
16 threads	off	1.50
	on	1.70

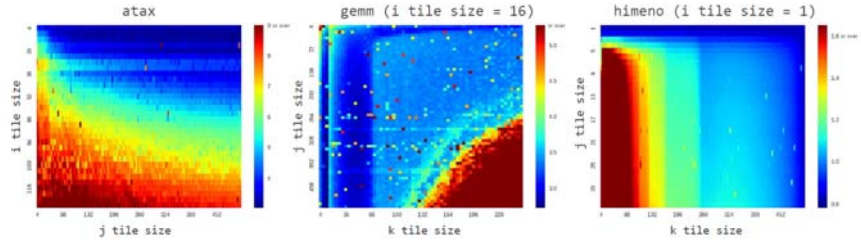
(**) Opt.3 is performed

C2Simのライン競合の原因検出の結果を用いて、生産的にチューニングするべき箇所を特定し、性能向上が得られることを実証

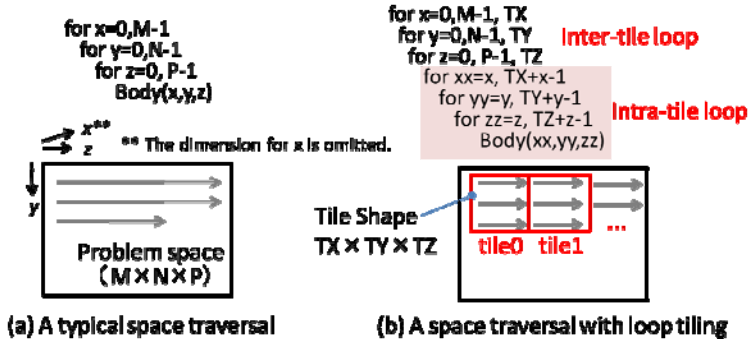
[Euro-Par2017] Yukinori Sato and Toshio Endo. An Accurate Simulator of Cache-line Conflicts to Exploit the Underlying Cache Performance. Proc. of the 22nd Int'l European Conf. on Parallel and Distributed Computing, Euro-Par 2017, pp. 119-133.

PATT: Polyhedral最適化に基づくループタイルサイズ自動チューニング機構

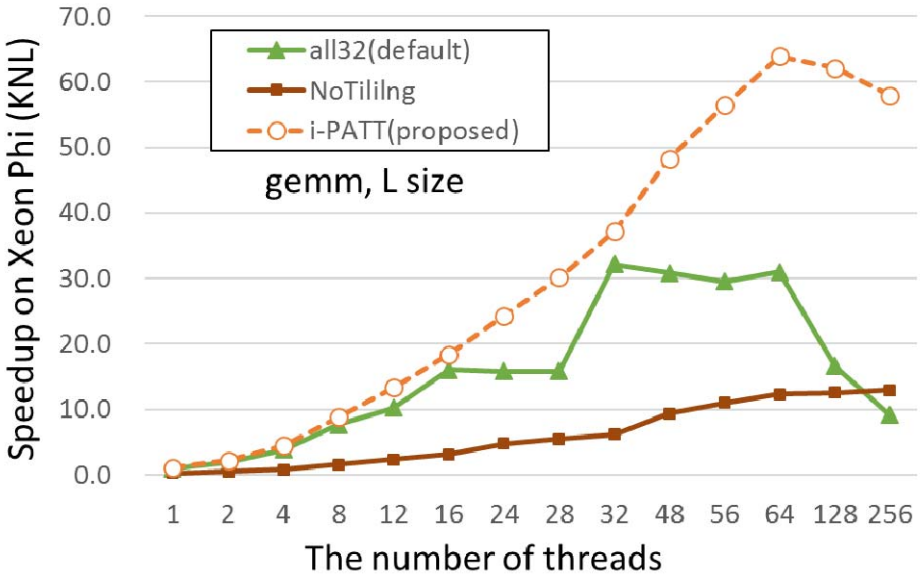
タイリングにおけるタイルサイズの選択は、アプリケーションや実行するプラットフォームなど様々な要因に依存するため未だ未解決な問題



タイルサイズにより性能が大きく変動する



PATT (Polyhedral compilation based AuTo Tile size optimizer):
オープンソースのPolyhedralコンパイラと数理最適化技術を組み合わせた自動チューニング機構によりタイルサイズ選択を自動化



予備評価にて、メニーコアCPU（Intel XeonPhi KNL）においてタイリングによりスケーラビリティが低下することを観測

メニーコアCPUのコア間のロードバランスを考慮したタイリングサイズ自動チューニングPATTを提案

PATTにおけるロードバランスのモデリング

ロードバランスを主体として性能モデルを構築

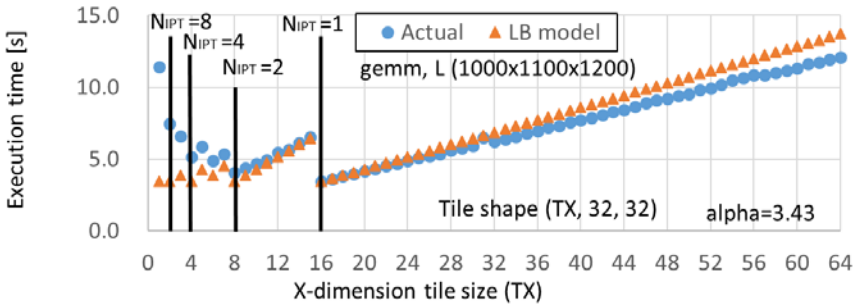
スレッドあたりの反復数N_{IPT}を算出

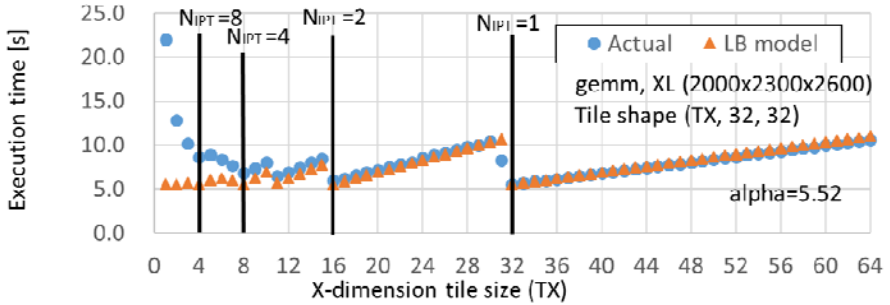
$$N_{IPT} = \lceil \frac{M}{TX \cdot p} \rceil$$

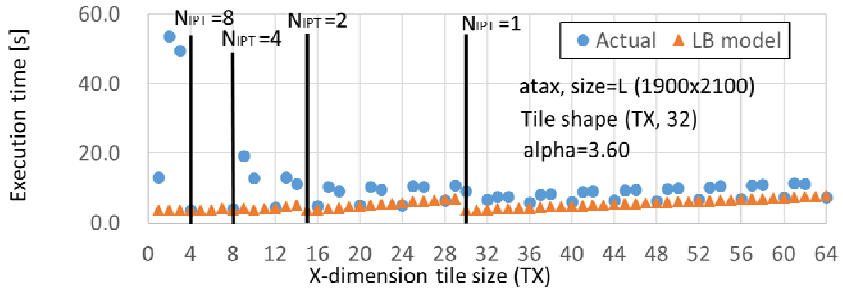
タイル実行時間$T_{LBmodel}$を、Inter-tile loopの重みをN_{IPT}とし、
Intra-tile loopの重みをTXとして推定

$$T_{LBmodel} = \alpha \cdot \lceil \frac{M}{TX \cdot p} \rceil \cdot TX$$

実測と性能モデルを比較（KNL64スレッド）





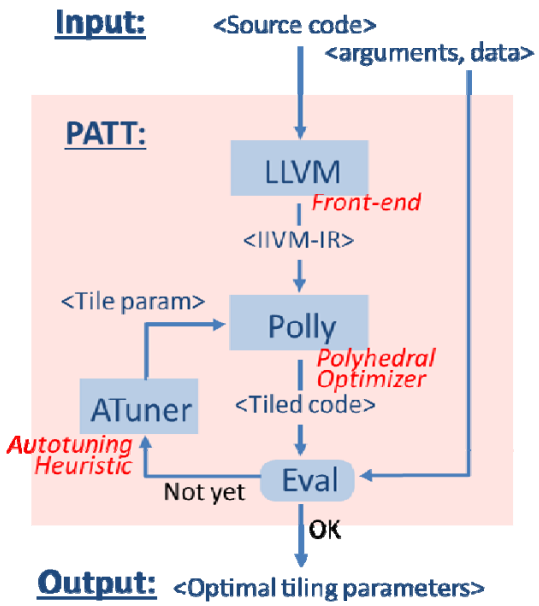


- タイリングの効果が考慮されていないがgemmではLBmodelとおおよそ一致
- しかし、ataxに見られるように、すべてのアプリを予測できるとも限らない

=> ロードバランスのモデリング ＋ キャッシュ挙動の計測と推定

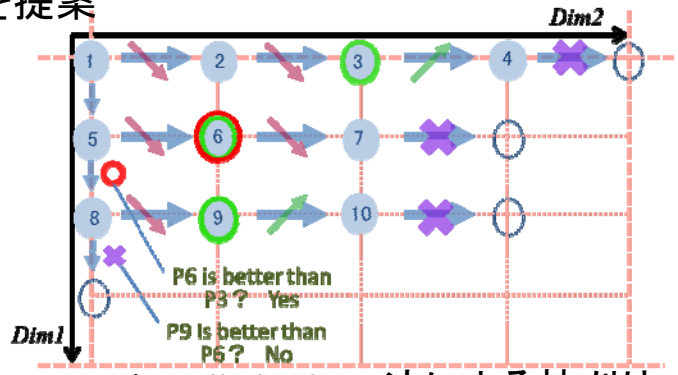
このロードバランスに関する知見を基にタイリングサイズ自動チューニング機構PATTを提案

PATTの概要と性能評価結果

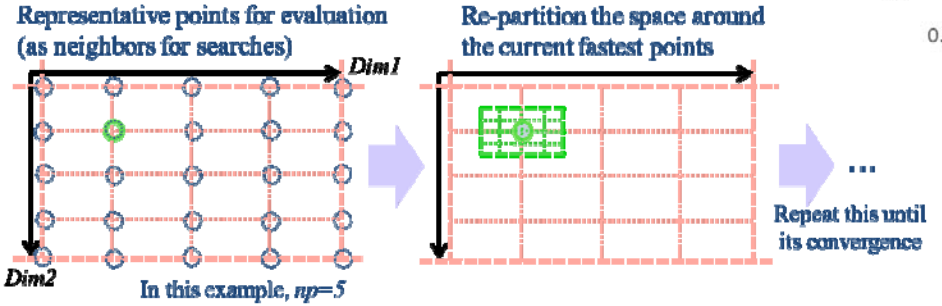


PATT： 反復的コンパイレーションに基づく最適化フレームワーク

PATTにおいてはロードバランスのモデリングとタイルサイズ選択に特化した空間探索最適化法を提案

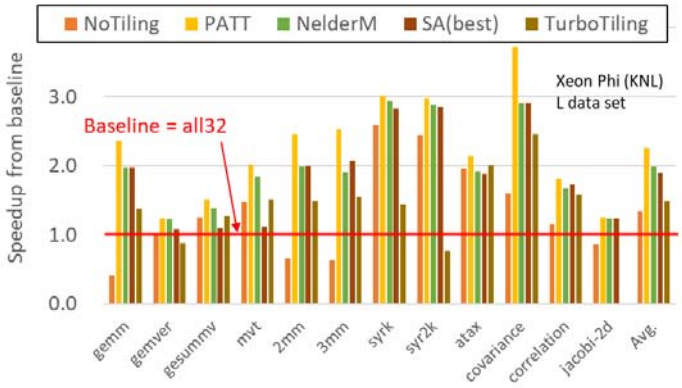


1) Hill climbing法による枝刈り



2) 適応型メッシュに基づく探索

汎用的な最適化手法（Simulated Annealing, NelderMead法）や既存の解析的な手法（TurboTiling)より高速なタイルサイズを発見



To be published in ACM TACO December 2018. Yukinori Sato, Tomoya Yuki, and Toshio Endo. “An Autotuning Framework for Scalable Execution of Tiled Code via Iterative Polyhedral Compilation”.

PATTのアルゴリズムの詳細

Algorithm 1 PATT_ATuner_main

Input: Problem space (M, N, P) or (N, P) for 2D, #threads (p)
  if Problem space is 3D **then**
    for all $i = 1, 2, 4, 8$ **do**
      $CTX_i = \lceil \frac{M}{i \cdot p} \rceil$
      for all $i = CTX_i - 1, CTX_i, CTX_i + 1$ **do**
        $t_i = tile\_eval(i, 32, 32)$
      end for
    end for
    select i such that it minimizes t_i
    $TX = i$
  else if Problem space is 2D **then**
    $TX = NONE$
  end if
  $TY, TZ = Custom\_ATuner\_for\_TSS(N, P)$
Output: Output tile size (TX, TY, TZ)

得られたロードバランスの知見を活用し、
最外の次元のタイルサイズTXを決める

Procedure 2 tile_eval

Input: tile size (i, j, k)
  $build\_exe(i, j, k)$
  $t_{exe} = measure\_exetime(i, j, k)$
Output: t_{exe}

LVM Pollyにおいて '–polly-tile-sizes=x,y,z' にてタイルサイズを指定しタイリング

Procedure 3 Custom_ATuner_for_TSS

Input: Problem size (N, P)
  $pos\_start[] = init_{dim1}, init_{dim2}, pos\_end[] = N, P$
  $np = 8$
  loop {/* Loop for AGS mechanism */}
    $step[] = calc\_step(pos\_start, pos\_end, np)$
    $dim1\_list, dim2\_list = calc\_AGS\_points(pos\_start, step, np)$
    for $q = 0$ to $np - 1$ **do** {/* Body of 2D hill climbing */}
      $j = dim1\_list[q]$
      for $r = 0$ to $np - 1$ **do**
        $k = dim2\_list[r]$
        $t_{jk} = tile\_eval(TX, j, k)$
        if $t_{jk} > t_{current}$ **then**
          break this loop [/* Pruning points for dim2 */]
        end if
        $t_{current} = t_{jk}; tile_{current} = (TX, j, r)$
      end for
      if $fastest\_time > t_{current}$ **then**
        $fastest\_time = t_{current}; fastest\_tile = tile_{current}$
      else
        break this loop [/* Pruning points for dim1 */]
      end if
    end for
    if all of s in $step[] = 0$ **then**
      return $fastest\_tile$
    end if
    $pos\_start, pos\_end = update\_pos()$
  end loop

適応型メッシュによる探索

Hill climbing法による
dim2方向枝刈り

Hill climbing法による
dim1方向枝刈り

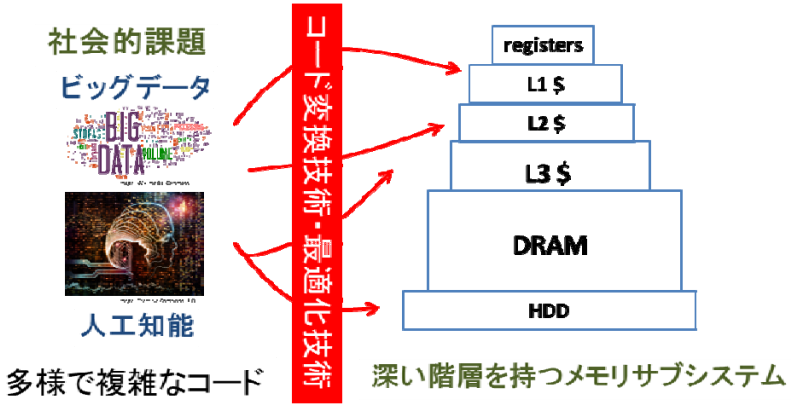
To be published in ACM TACO December 2018. Yukinori Sato, Tomoya Yuki, and Toshio Endo.
"An Autotuning Framework for Scalable Execution of Tiled Code via Iterative Polyhedral Compilation".

性能工学レイヤのこれまでの知見

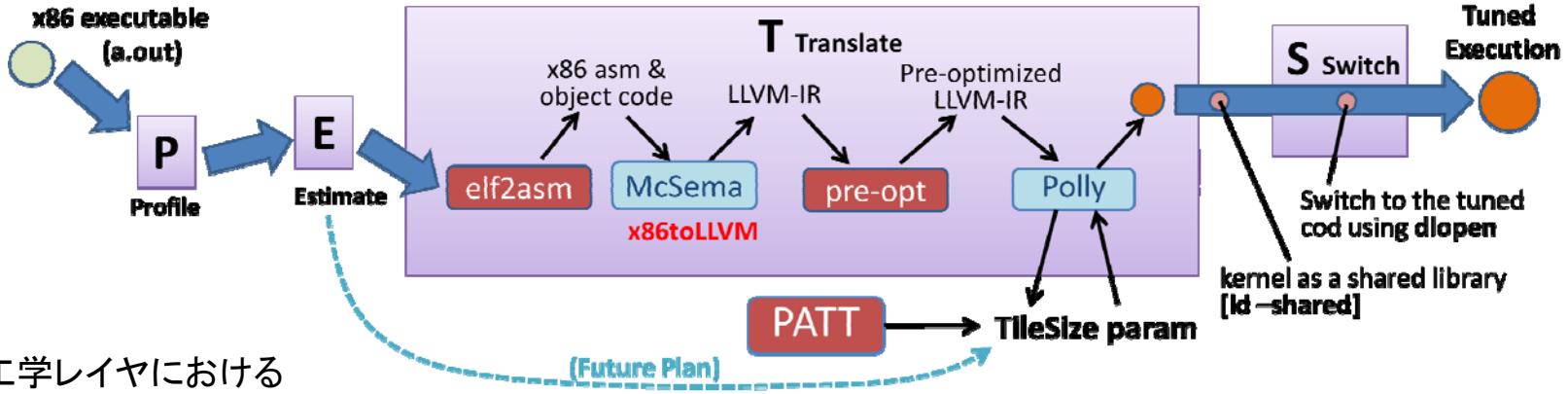
近年取り組んできたスパコン性能チューニングの研究

性能問題を解決する糸口

ソフトウェアのメモリ参照局所性を的確に
ハードウェアの特性にカスタマイズすること



[Sato, ACM CF'17]



性能工学レイヤにおける
ExanaDBT（バイナリ動的最適化）＋PATT（自動チューニング）による透過的性能チューニング

高位最適化ツールの入力としたLLVM-IRの質が悪く、アルゴリズムレベルにはない依存関係によりスケジュール最適化が想定していたように行えないという問題に悩まされる

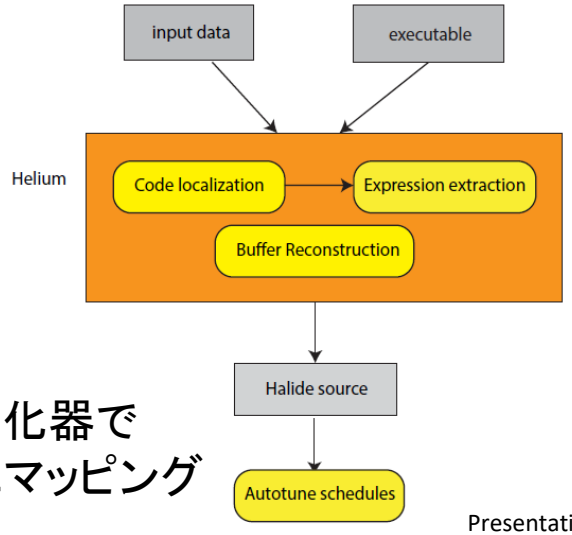
TOYOHASHI UNIVERSITY OF TECHNOLOGY

Halideスケジュールを起点とした高位最適化

MITの研究チームによるバイナリ動的最適化

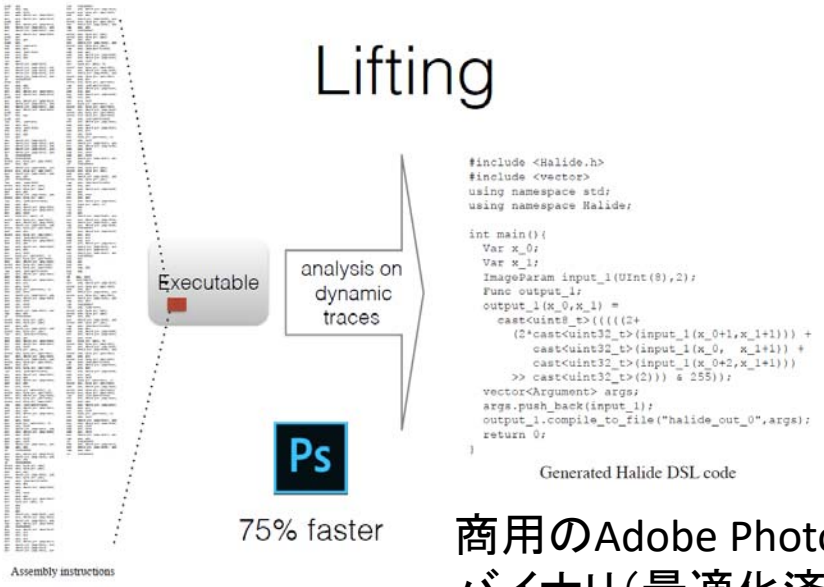
[PLDI 2015] Helium: Lifting High-Performance Stencil Kernels from Stripped x86 Binaries to Halide DSL Code

バイナリから画像処理のカーネル部分を
画像処理向けDSLであるHalideに復元



Halideの最適化器で
対象ハードにマッピング

動的な命令トレースからアルゴリズムを復元
DBTにはDynamoRIOを使う



Presentation materials in http://projects.csail.mit.edu/helium/

商用のAdobe Photoshopの
バイナリ（最適化済）を75%高速化

ExanaDBTの研究で苦しんだLLVM-IRにおけるアルゴリズムとスケジュールの
分離がHalideにより言語として解決されており、高位最適化と相性がよい

アプリケーションレイヤにおけるHalide

MITの研究チームが開発した高性能画像処理向けの言語 [http://halide-lang.org/]

- 関数型言語でアルゴリズムを記述
- 一時記憶、計算順序はスケジュールとしてアルゴリズムから分離して記述

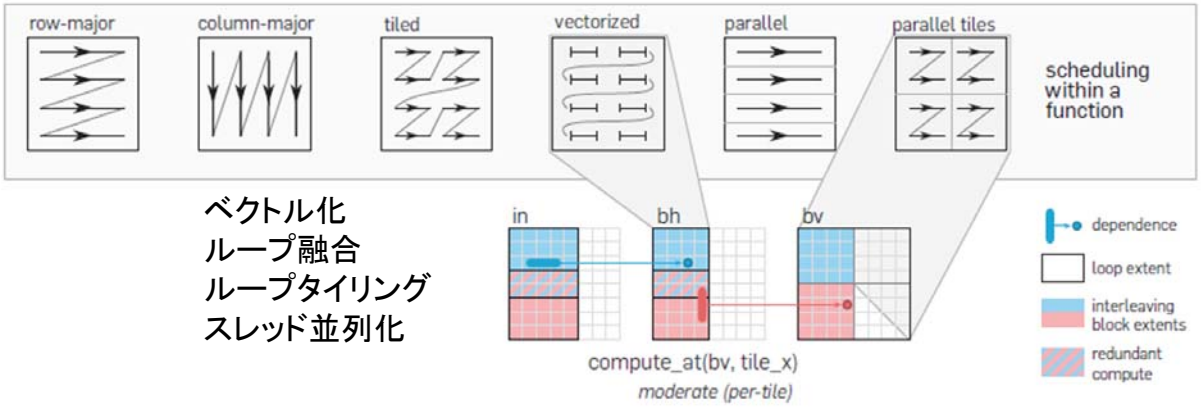
```
Func halide_blur(Func in) {
 Func bh, bv;
 Var x, y, xi, yi;

  // アルゴリズム
 bh(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;
 bv(x, y) = (bh(x, y-1) + bh(x, y) + bh(x, y+1))/3;
  // スケジュール
 bv.tile(x, y, xi, yi, 256, 32)
   .vectorize(xi, 8).parallel(y);
 bh.compute_at(bv, x).vectorize(x, 8);

 return bv;
}
```

3×3 box filter のコードの例

[CACM, January 2018]



ベクトル化
ループ融合
ループタイリング
スレッド並列化

コンパイラが対象ハードウェアに向けた最適化を行う
⇒ スケジュールの制約を満たす最も効率の良いコードを探求

- 構造格子、多次元配列に対する計算が可能で汎用的な行列計算に適応可能
- スケジュールはPolyhedralモデルにおけるSCoPに対応するため高位最適化が直接可能

Halideによる初期開発コストの削減

(a) Clean C++: 6.5ms per megapixel

```
void blur(const Image<uint16_t> &in, Image<uint16_t> &bv) {
 Image<uint16_t> bh(in.width(), in.height());

  for (int y = 0; y < in.height(); y++)
   for (int x = 0; x < in.width(); x++)
    bh(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;

  for (int y = 0; y < in.height(); y++)
   for (int x = 0; x < in.width (); x++)
    bv(x, y) = (bh(x, y-1) + bh(x, y) + bh(x, y+1))/3;
}
```

3×3 box filter のコードの例

ベクトル化
ループ融合
ループタイリング
スレッド並列化

20倍高速化

複雑な記述

CC BY-SA 3.0

(b) Fast C++ (for x86) : 0.30ms per megapixel

```
void fast_blur(const Image<uint16_t> &in, Image<uint16_t> &bv) {
 __m128i one_third = _mm_set1_epi16(21846);
 #pragma omp parallel for
 for (int yTile = 0; yTile < in.height(); yTile += 32) {
  __m128i a, b, c, sum, avg;
  __m128i bh[(256/8)*(32+2)];
  for (int xTile = 0; xTile < in.width(); xTile += 256) {
   __m128i *bhPtr = bh;
   for (int y = -1; y < 32+1; y++) {
    const uint16_t *inPtr = &(in(xTile, yTile+y));
    for (int x = 0; x < 256; x += 8) {
       a = _mm_loadu_si128((__m128i*)(inPtr - 1));
       b = _mm_loadu_si128((__m128i*)(inPtr + 1));
       c = _mm_load_si128 ((__m128i*)(inPtr));
     sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
     avg = _mm_mulhi_epi16(sum, one_third);
     _mm_store_si128(bhPtr++, avg);
     inPtr += 8;
   }}
   bhPtr = bh;
   for (int y = 0; y < 32; y++) {
    __m128i *outPtr = (__m128i *)(&(bv(xTile, yTile+y)));
    for (int x = 0; x < 256; x += 8) {
       a = _mm_load_si128(bhPtr + (256 * 2) / 8);
       b = _mm_load_si128(bhPtr + 256 / 8);
       c = _mm_load_si128(bhPtr++);
     sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
     avg = _mm_mulhi_epi16(sum, one_third);
     _mm_store_si128(outPtr++, avg);
}}}}}
```

x86向けの高速実装(c++)

20倍高速化
可読性も良好

CC BY-SA 3.0

(c) Halide : 0.29ms per megapixel

```
Func halide_blur(Func in) {
 Func bh, bv;
 Var x, y, xi, yi;

 // The algorithm
 bh(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;
 bv(x, y) = (bh(x, y-1) + bh(x, y) + bh(x, y+1))/3;

 // The schedule
 bv.tile(x, y, xi, yi, 256, 32)
   .vectorize(xi, 8).parallel(y);
 bh.compute_at(bv, x).vectorize(x, 8);

 return bv;
}
```

高位最適化器が
HWに合わせて
コードを生成

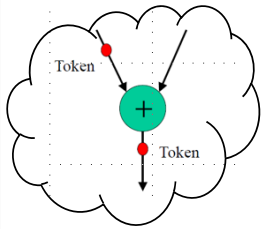
CC BY-SA 3.0

[CACM, January 2018]

Halideによる実装　関数型言語による簡潔なアルゴリズム表記
スケジュールの分割による移植性・性能可搬性の改善

データフロー方式のハードウェア

1980年代	2000年代	現在



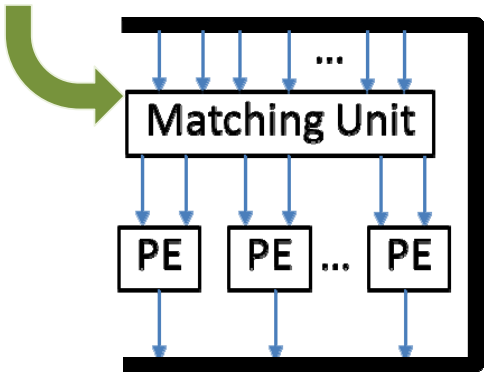
データフローグラフ

データフローブームの終焉

Out-of-order方式スーパースカラ

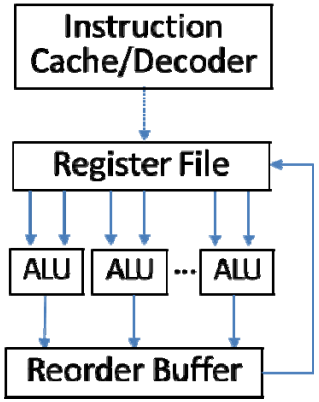
トランジスタ資源活用の探求とカスタム化

深層学習専用ハードの演算部

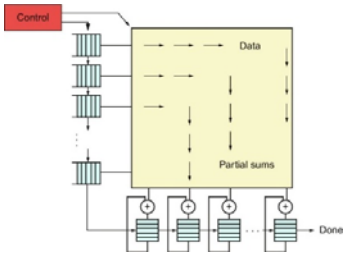


平木教授最終講義資料を元に作成

空間方向にのみ展開されたPEにおける並列処理

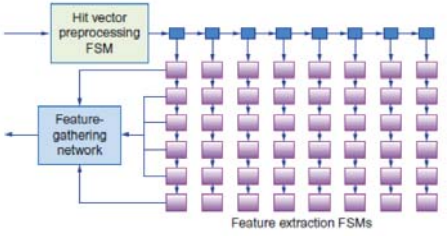


空間方向の並列性においてデータフローの概念を実現



Hennessy & Patterson. *"Computer Architecture, 6th Edition"*

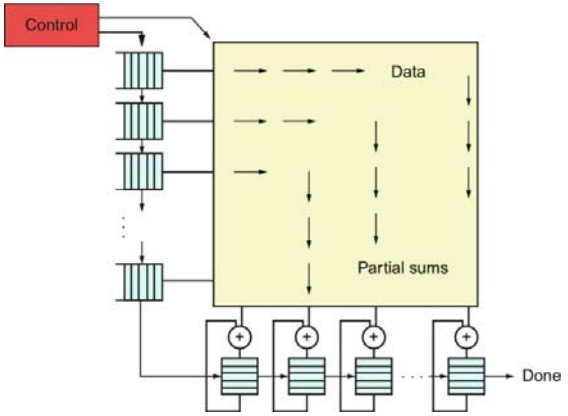
Systolic array in Google's TPU



Feature Extraction stage in Microsoft Catapult [ISCA 2014]

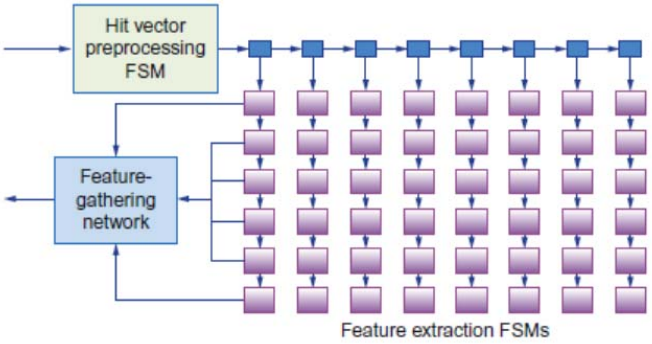
空間時間方向に展開されたFlynnの4分類のMISD方式

MISD方式カスタムパイプラインの高位最適化



Hennessy & Patterson. *"Computer Architecture, 6th Edition"*

Systolic array in Google's TPU



Feature Extraction stage in Microsoft Catapult [ISCA 2014]

性能工学レイヤ

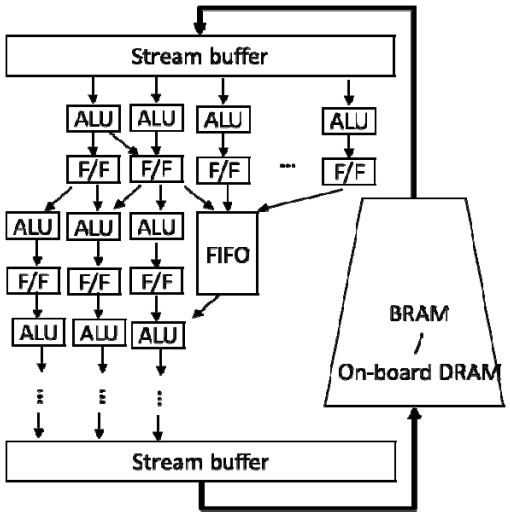
任意のアプリでシストリックアレイを柔軟に合成

MISDパイプラインの自動合成による汎用化

行列計算のドメインでの汎用化と生産性向上

Halide [PLDI 2013] アプリレイヤ

アプリケーション実行の本質を追求し最適化を試みる総合的なシステム志向研究

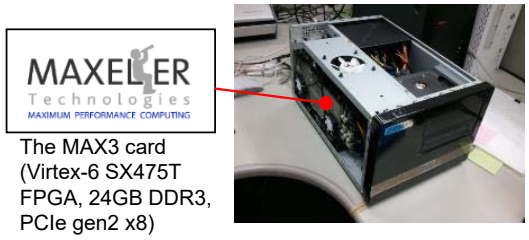


アプリ特化型のカスタムパイプラインを合成

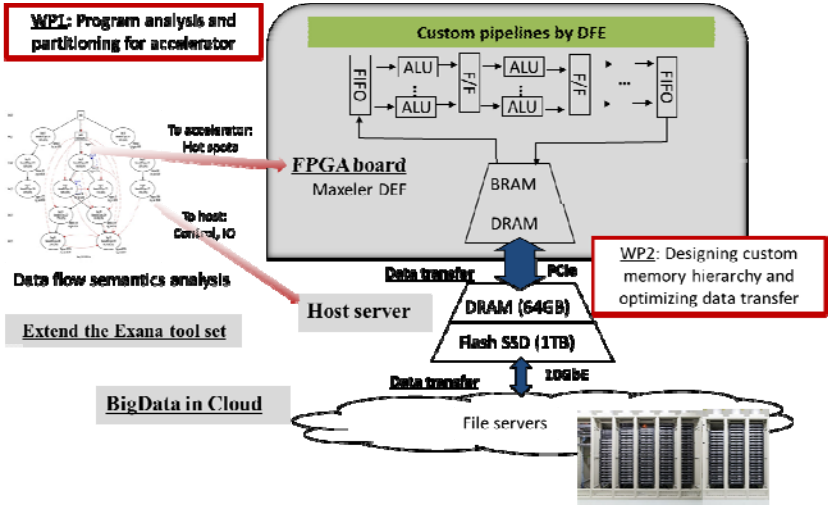
時間空間方向に展開したデータフローグラフを直接マッピング

M. Flynn, *"Maxeler uses a dataflow approach, which goes back to my old taxonomy, MISD, where the data flows through a fixed set or a fixed graph of instructions."*

MISD方式データフローカスタムパイプラインの合成

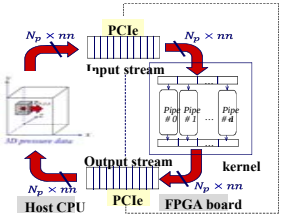


The MAX3 card
(Virtex-6 SX475T
FPGA, 24GB DDR3,
PCIe gen2 x8)

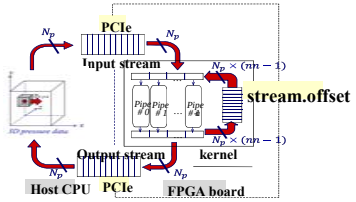


- MaxJ言語を用いたMISD回路の高位合成
  (Java-base meta programming)
- メモリ階層へのマッピングに対する最適化

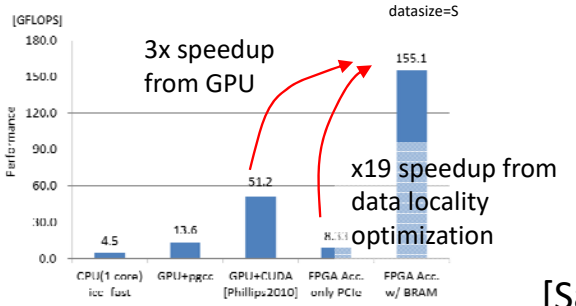
Himeno Benchmark (19-pt stencil)を用いたFPGAアクセラレータの評価



(1) Data transfer via PCIe

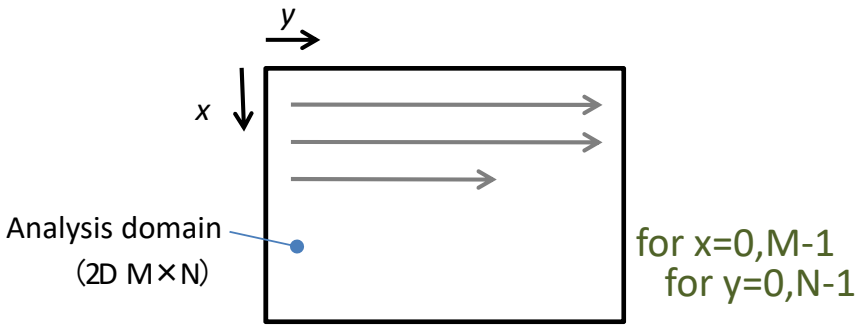


(2) Using local memory on FPGA



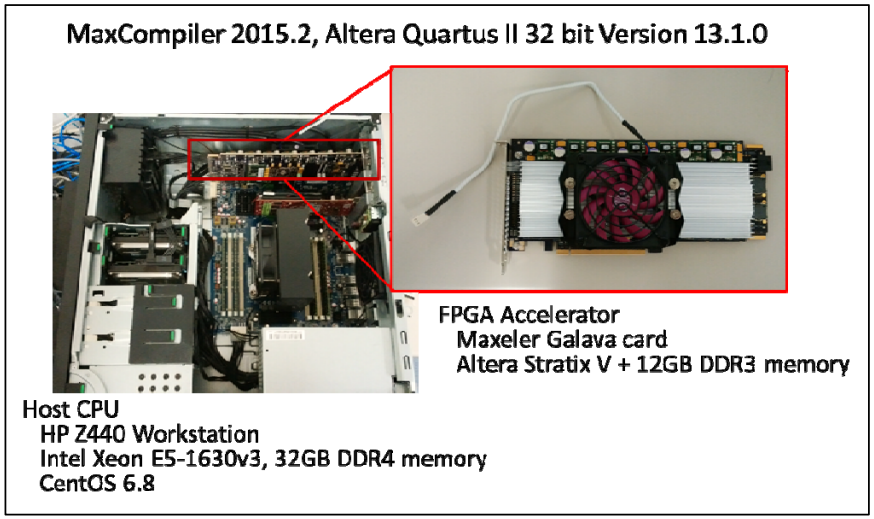
[Sato, ReConFig'12]

データフローパイプラインへのタイリングの適応

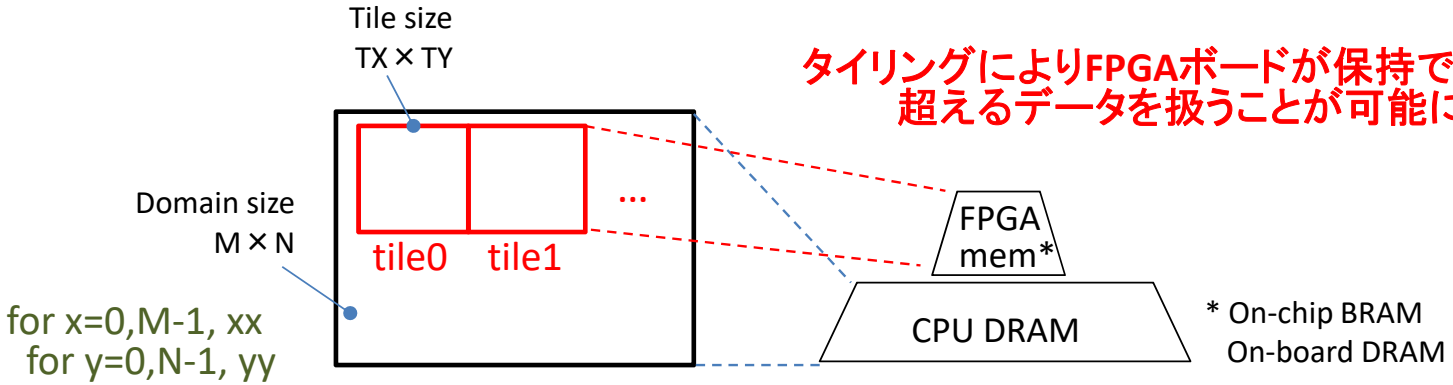


(a) 2次元の場合の一般的な空間走査(2重ループ)

解析空間がFPGAに搭載されるメモリサイズを超える場合、メモリ容量の不足から回路が合成できない



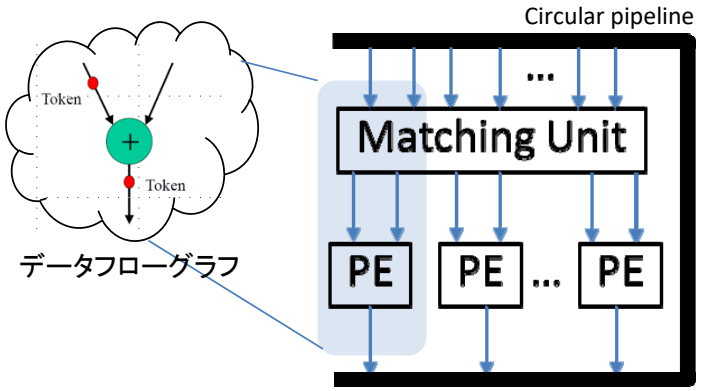
LLVM + Polly



タイリングによりFPGAボードが保持できる容量を超えるデータを扱うことが可能に！！

(b)ループタイリングを用いたヘテロなメモリ階層へのマッピング

80年代データフローからのMISD方式の進化



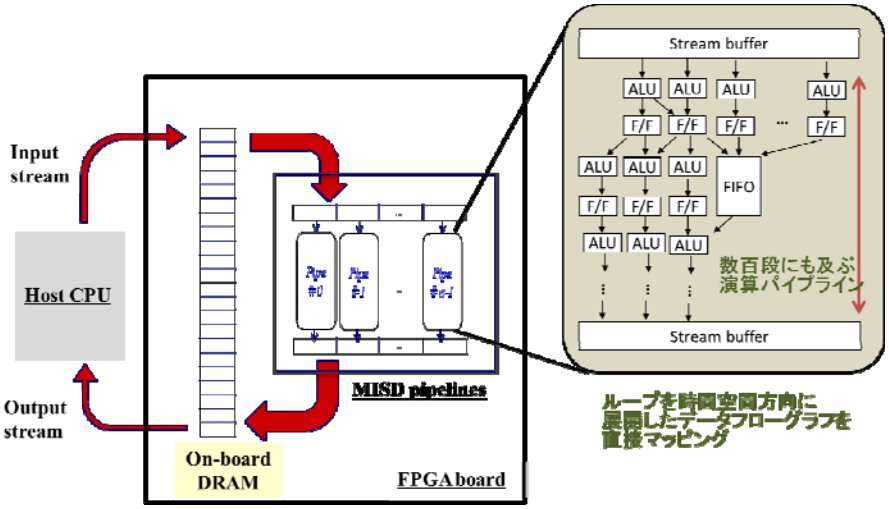
平木教授最終講義資料を元に作成

80年代データフロー

非同期のMatching logic
Token + firing => output

汎用を目指す（命令レベル並列性）
水平方向にリソースを展開
Circular pipeline architecture
Feedback-loopによるスケーラビリティの欠如

ソフトウェアを書くのが困難



MISD方式データフロー

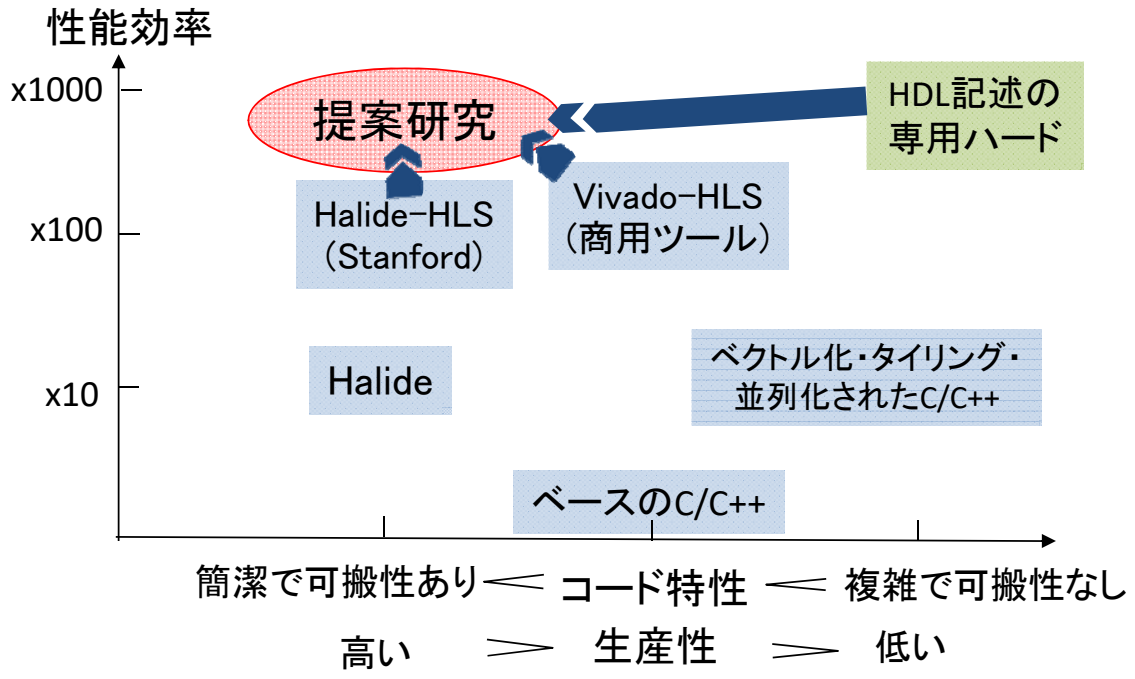
完全な同期式
ストリームデータの規則的な処理

ループを時間方向にも展開（2次元に）
分岐処理の排除（Predication）
Feedback-loopを極力なくした数百段にも及ぶパイプライン
固定パターンの静的処理が対象。パイプを並列にする

高位合成環境を用いたアルゴリズム記述によるプログラミング

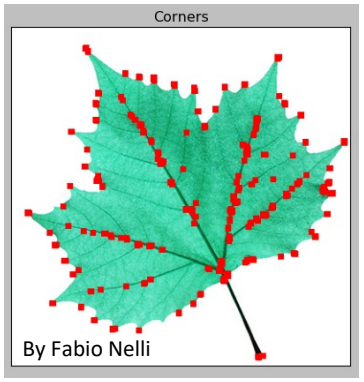
提案研究の独創性と優位性

データフロー方式の簡潔さと美しさに着目し、アプリ・ソフト・ハードにわたりデータフロー主導で首尾一貫して最適化していく点が独創的

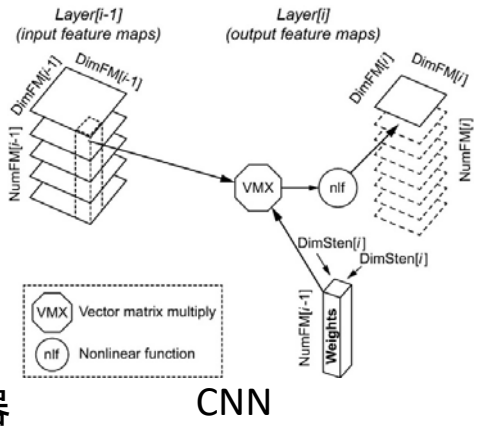


【ベンチマークによる比較】

自動運転や知的ロボットへの適応を見据え、画像処理や深層学習分野から代表的なものを取り上げ評価



Harrisのコーナー検出器



CNN

Halideのリポジトリで共有されているコードを利用

Vivado-HLS、Halide-HLSと比べたアプリ記述や最適化の過程の生産性、実行における性能効率を定量的に比較し、優位性を示す計画

まとめと将来展望

「データフロー計算原理」に基づきフルスタックで高位最適化するツール群の研究開発

性能工学レイヤ	Polyhedralモデルによるスケジュールの高位最適化 自動チューニング技術も非常に重要
アプリケーションレイヤ	関数型言語である*Halide*を用いて記述
ハードウェアレイヤ	*MISD* 方式のカスタムパイプラインを合成