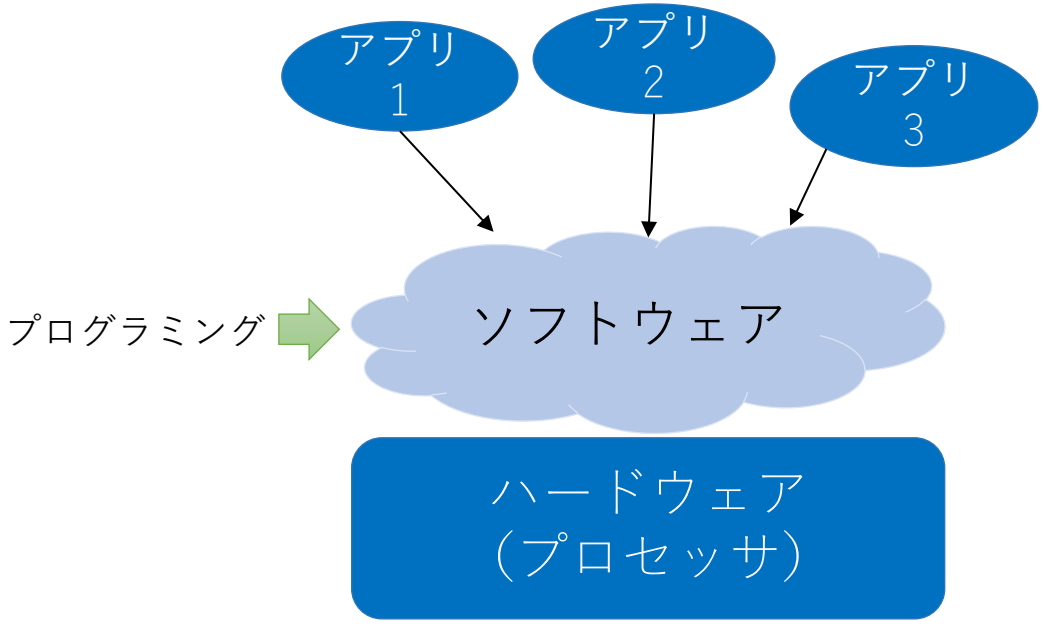
リコンフィギャラブルコンピューティングの課題と展望

長崎大学 情報データ科学部

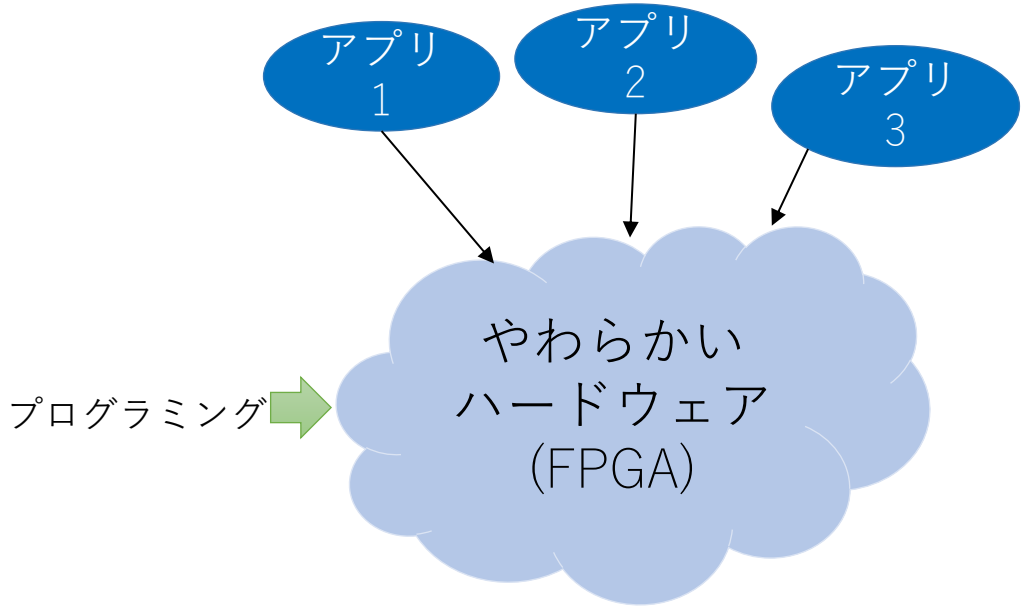
柴田　裕一郎

リコンフィギャラブルコンピューティング

- Reconfigurable = Re + configure + able (再構成可能な)
- FPGAなどのプログラマブルデバイスなハードウェアデバイスの柔軟性を積極的に利用
  - 専用コンピュータの性能
  - 汎用コンピュータの柔軟性



従来のコンピューティング



リコンフィギャラブル コンピューティング

研究分野としても認知

中区分60：情報科学、情報工学およびその関連分野

小区分	内容の例
60010	〔情報学基礎論関連〕 離散構造、数理論理学、計算理論、プログラム理論、計算量理論、アルゴリズム理論、情報理論、符号理論、暗号理論、学習理論、など
60020	〔数理情報学関連〕 最適化理論、数理システム理論、システム制御理論、システム分析、システム方法論、システムモデリング、システムシミュレーション、組合せ最適化、待ち行列論、数理ファイナンス、など
60030	〔統計科学関連〕 統計学、データサイエンス、モデル化、統計的推測、多変量解析、時系列解析、統計的品質管理、応用統計学、など
60040	〔計算機システム関連〕 計算機アーキテクチャ、回路とシステム、ＬＳＩ設計、ＬＳＩテスト、リコンフィギャラブルシステム、ディペンダブルアーキテクチャ、低消費電力技術、ハードウェア・ソフトウェア協調設計、組込みシステム、など
60050	〔ソフトウェア関連〕 プログラミング言語、プログラミング方法論、オペレーティングシステム、並列分散処理、ソフトウェア工学、仮想化技術、クラウドコンピューティング、ソフトウェアディペンダビリティ、ソフトウェアセキュリティ、など
60060	〔情報ネットワーク関連〕 ネットワークアーキテクチャ、ネットワークプロトコル、インターネット、モバイルネットワーク、パーベイシブコンピューティング、センサーネットワーク、ＩｏＴ、トラフィックエンジニアリング、ネットワーク管理、サービス構築基盤技術、など
60070	〔情報セキュリティ関連〕 暗号、耐タンパー技術、認証、バイオメトリクス、アクセス制御、マルウェア対策、サービス妨害攻撃対策、プライバシー保護、ディジタルフォレンジクス、セキュリティ評価認証、など
60080	〔データベース関連〕 データモデル、データベースシステム、マルチメディアデータベース、情報検索、コンテンツ管理、メタデータ、ビッグデータ、地理情報システム、など
60090	〔高性能計算関連〕 並列処理、分散処理、クラウドコンピューティング、数値解析、可視化、コンピュータグラフィクス、高性能計算アプリケーション、など

電子情報通信学会（IEICE）
リコンフィギャラブルシステム研究専門委員会（RECONF）

最終更新日：2023年1月31日

皆様の発表や聴講を歓迎しております．ぜひご参加ください．
聴講には電子予稿集ダウンロードキー用の参加費が必要です．オンライン開催の場合は早めにご登録・お支払いいただきますようお願いいたします．電子化の詳細はこちらをご覧ください．
開催会によっては企業展示を歓迎しております．詳細についてはお気軽にお問合せください．

電子情報通信学会（http://www.ieice.org）内 RECONF 関連記事の検索はこちら

検索

研究会開催情報
（これまでの開催に関しては「こちら（研究会 開催スケジュール一覧）」を参照ください．）

2023年6月研究会

【開催日程】　2023年6月8日(木)- 6月9日(金) (予定)
【開催場所】　高知工科大学永国寺キャンパス＋オンライン（Zoom）
【補足説明】　発表申込へのリンク
【補足説明】　学生は無料で聴講可能です。開催プログラムのページにある事前登録フォームにて登録をお願いいたします。

研究会の目的および趣旨

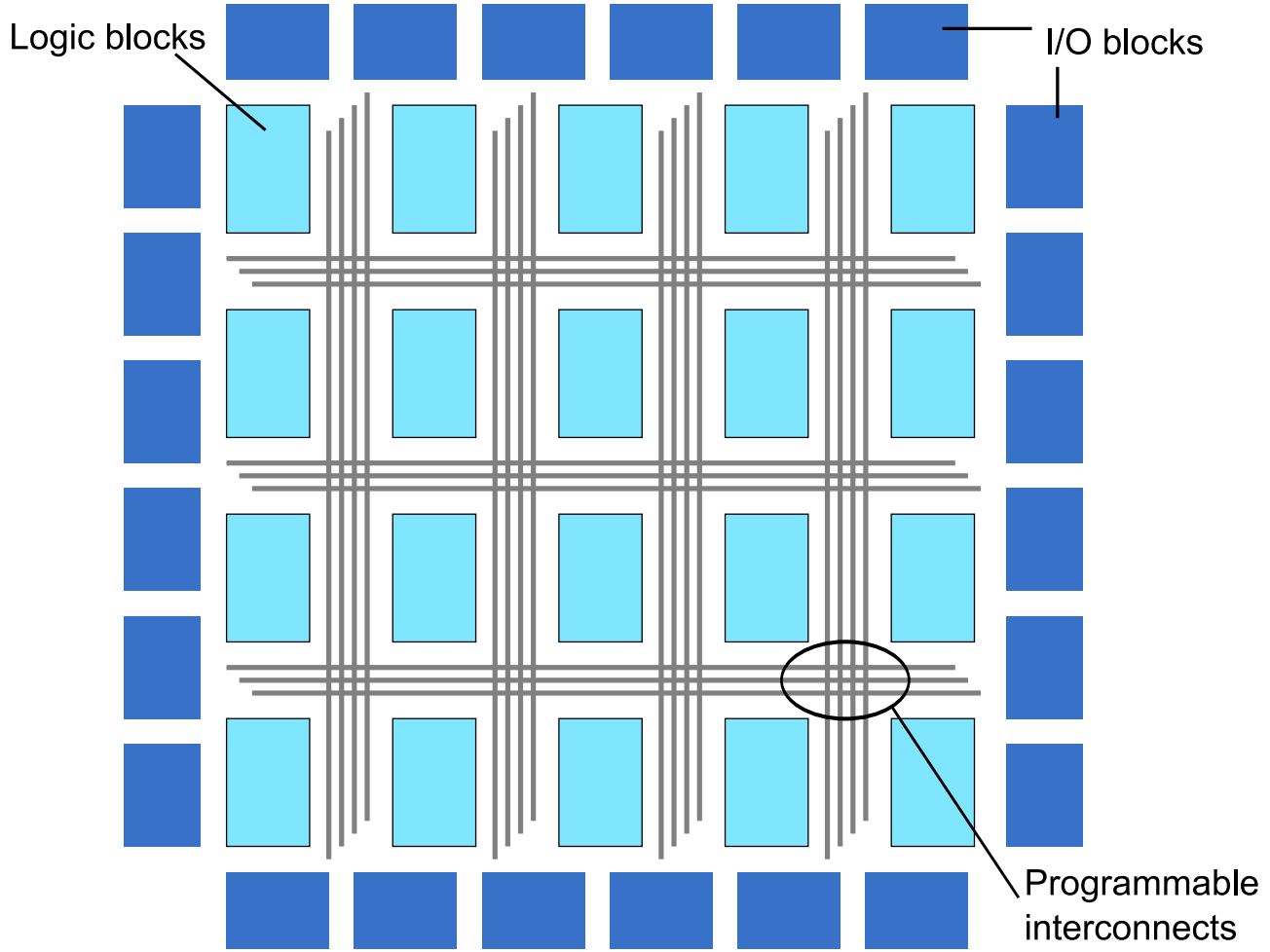
リコンフィギャラブルシステムは，問題の解法アルゴリズムをハードウェア化してユーザが書き換え可能なデバイス上で直接実行することにより，高い性能と柔軟性を実現するシステムです．従来のコンピュータにおいて蓄積されたソフトウェア技術，ハードウェア技術の単純な延長上にあるだけでなく，それを含有する大きな枠組みを形作っています．よってリコンフィギャラブルシステムの実現には，アーキテクチャ，デバイス，設計技術， CAD，システム技術，並列処理，アプリケーションなど，基礎と実用の両領域を含む多面的な研究を推進していくことが重要となります．本研究会は，定期的に研究会を開催することにより，関連する研究者に広く技術交換の機会を提供し，産業界と大学等の研究機関の交流の機会を増やすことにより，関連する技術の発展を促進することを図るため，2003年，電子情報通信学会コンピュータシステム研究専門委員会所属第2種研究会として発足しました．2005年からは第1種研究会としての活動をスタートしています．

研究会で対象としている主な分野（主な分野であり限定するものではありません）

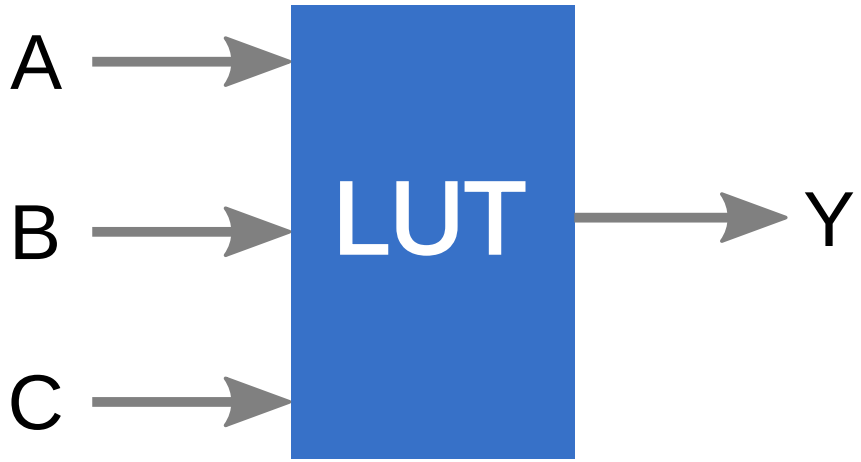
(日) リコンフィギャラブルシステム関連技術（アーキテクチャ，デバイス，アルゴリズム，設計技術，開発環境）　FPGA／PLD関連技術（デバイス，回路，設計技術，省電

FPGA

- Field Programmable Gate Array
- 手元で自由にプログラムできるLSI
  - 多数の論理ブロック間をスイッチを介した配線で接続
  - LUT (Look-up Table) をベースとした論理ブロックが一般的

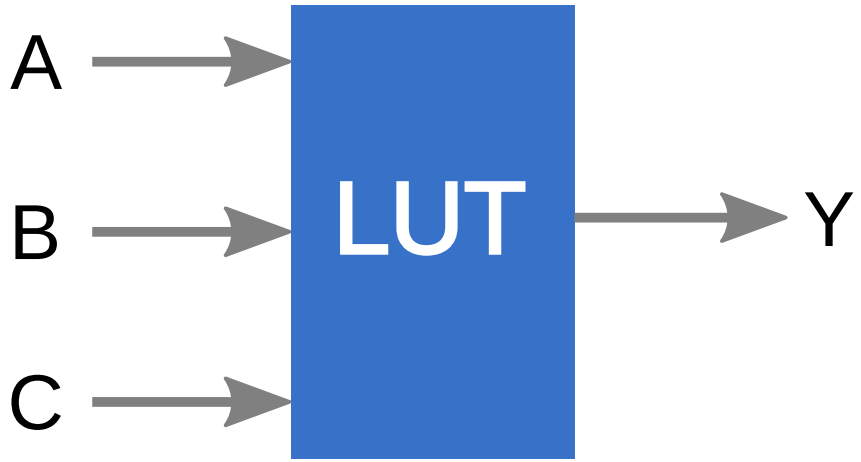


3入力LUTの例



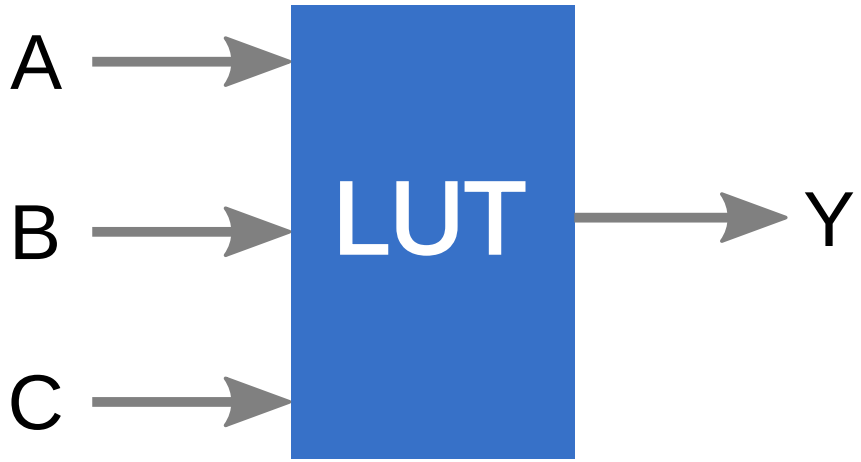
A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

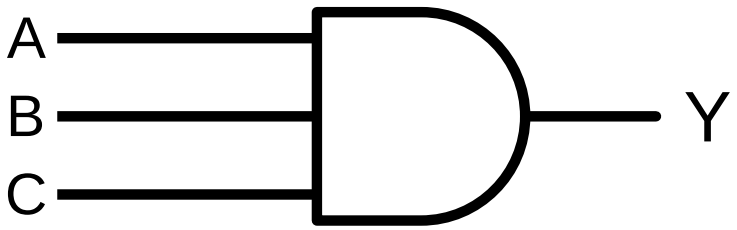
3入力LUTの例



A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

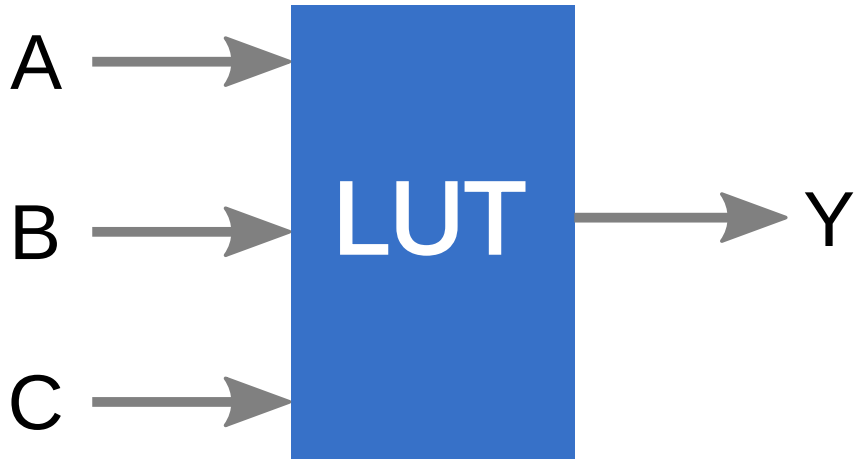
3入力LUTの例





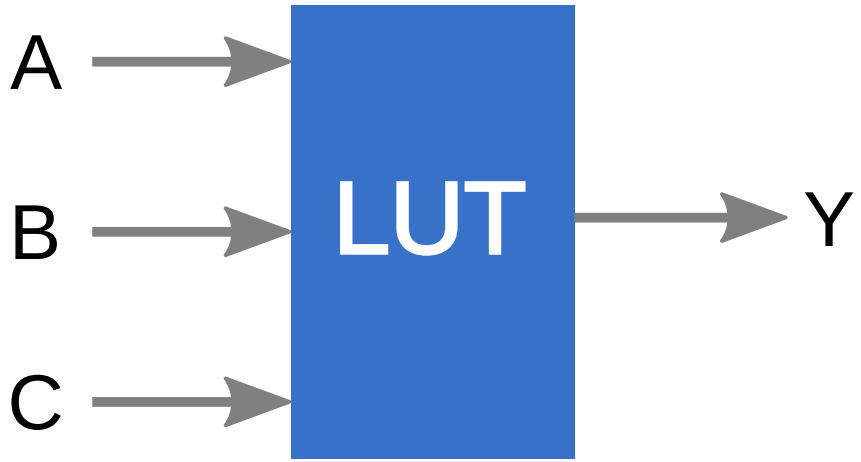
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

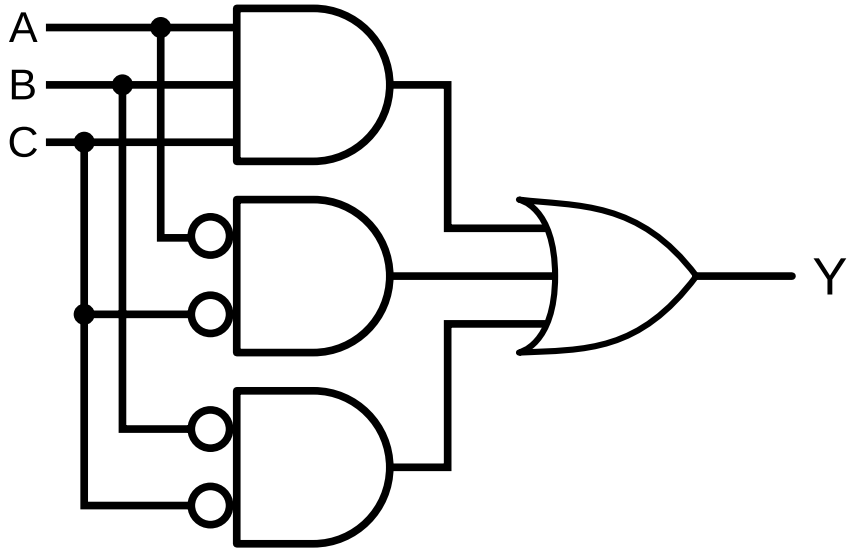
3入力LUTの例



A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

3入力LUTの例

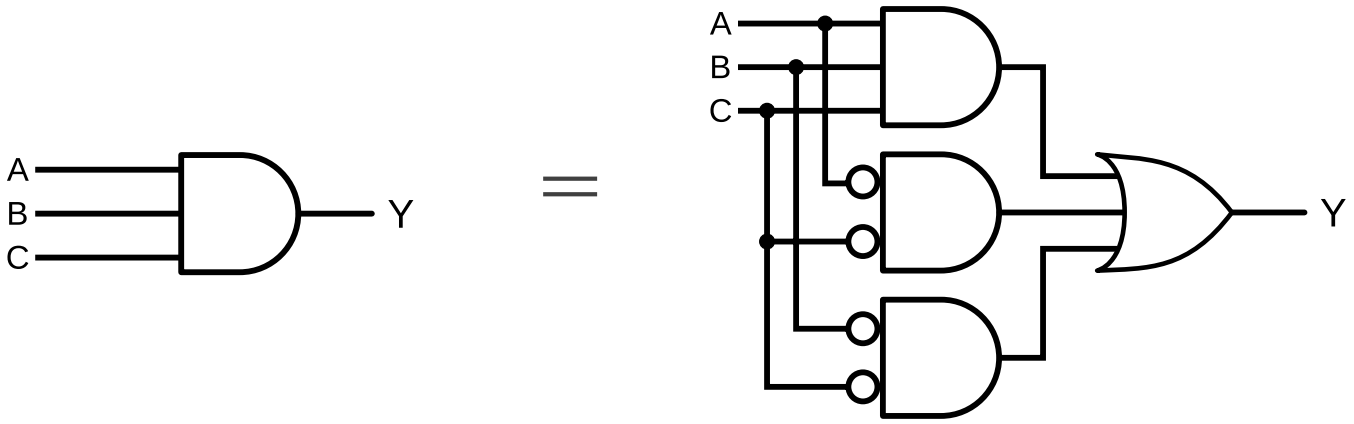




A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

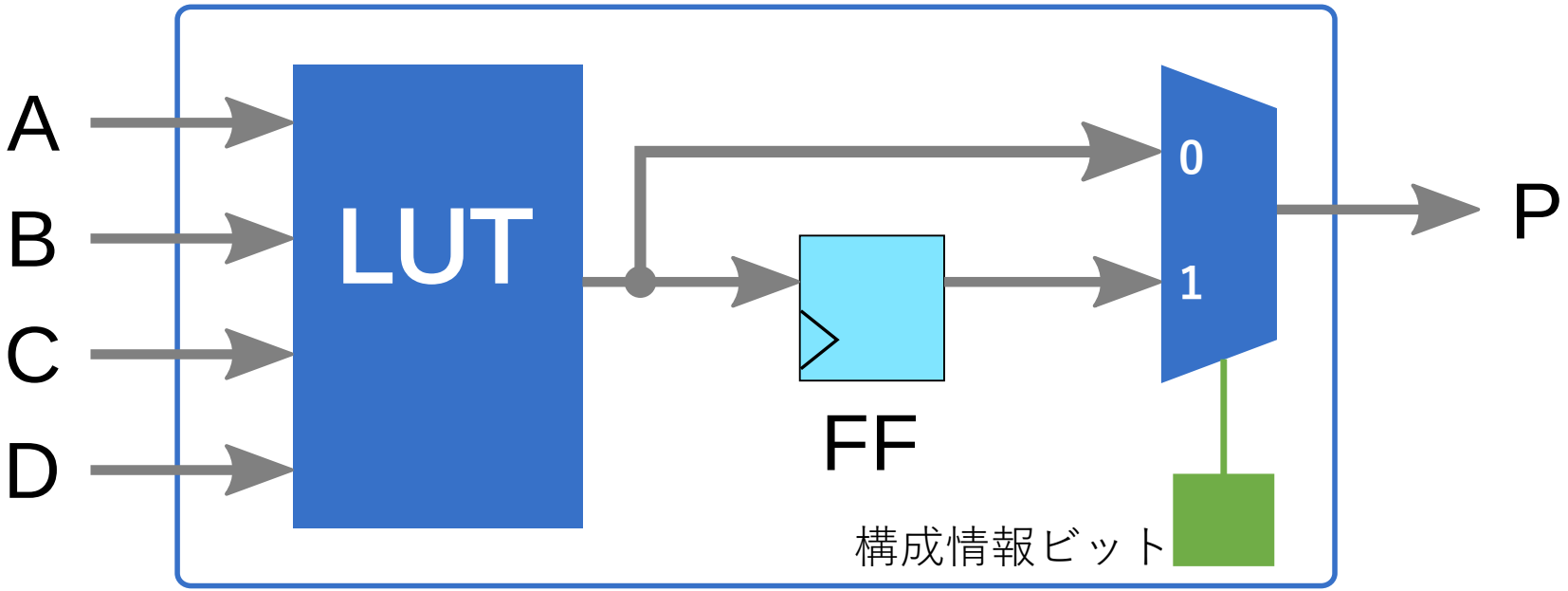
n入力LUT

- 入力数 n 以下の任意の組合せ回路を実現
  - 遅延時間も規模も同じ



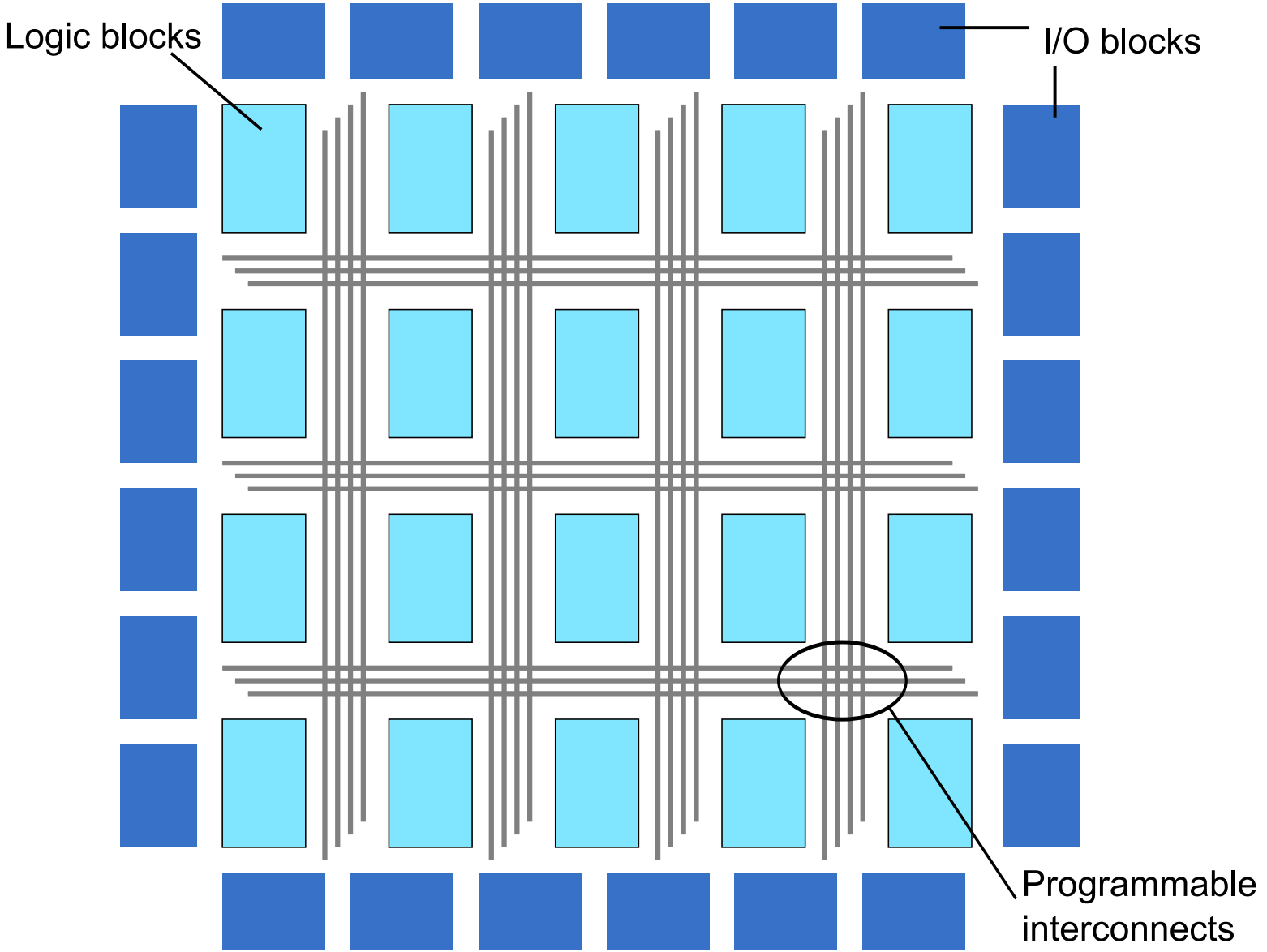
- 入力数が n を超えるとLUTを多段に接続
  - 実際のFPGAでは n は4～6

論理ブロックの基本構造

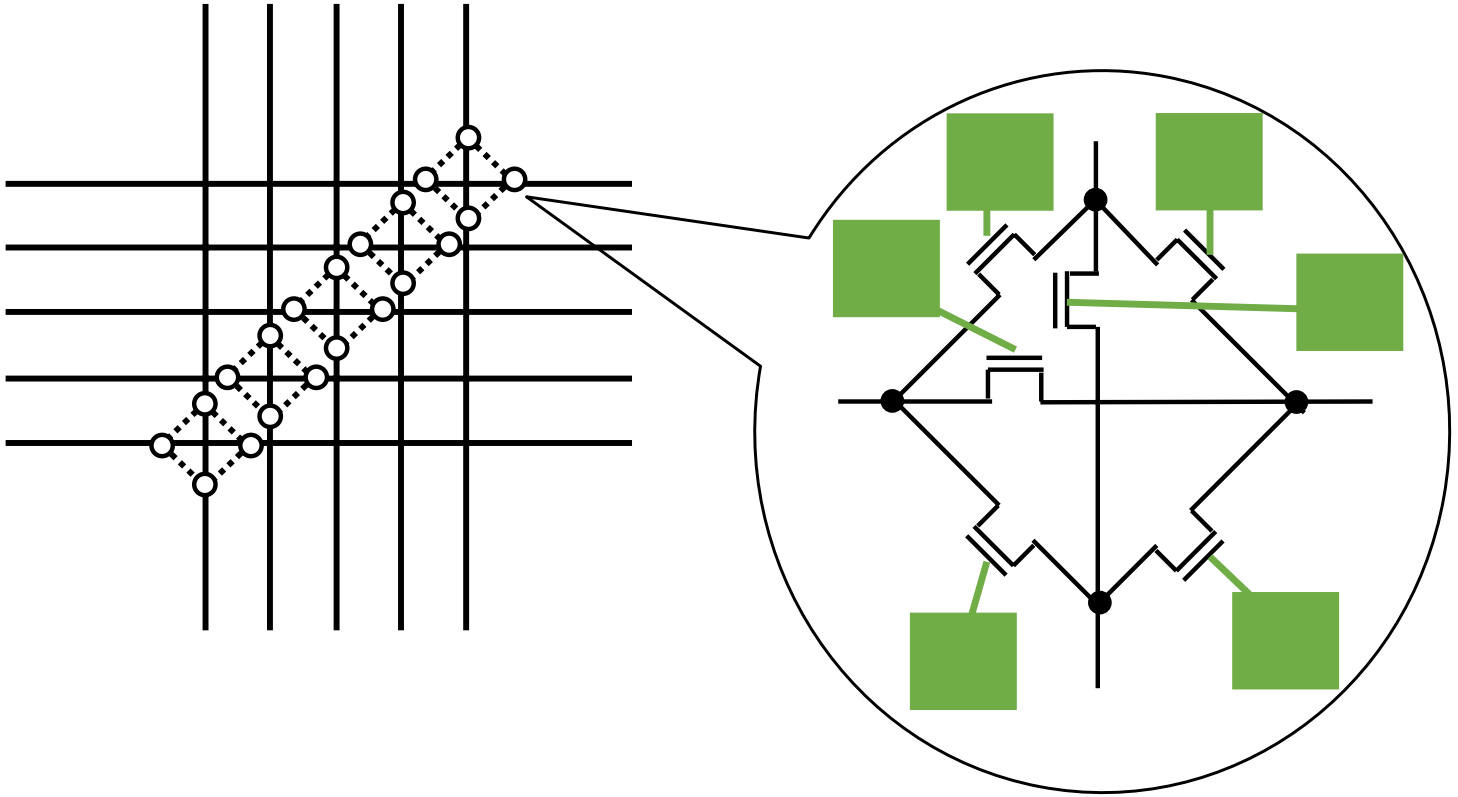


- LUT (Look-up table) で真理値表を実現
  - 内容を書き換えれば異なる論理に
- FF (Flip-flop) で順序回路を実現

FPGAの基本構造

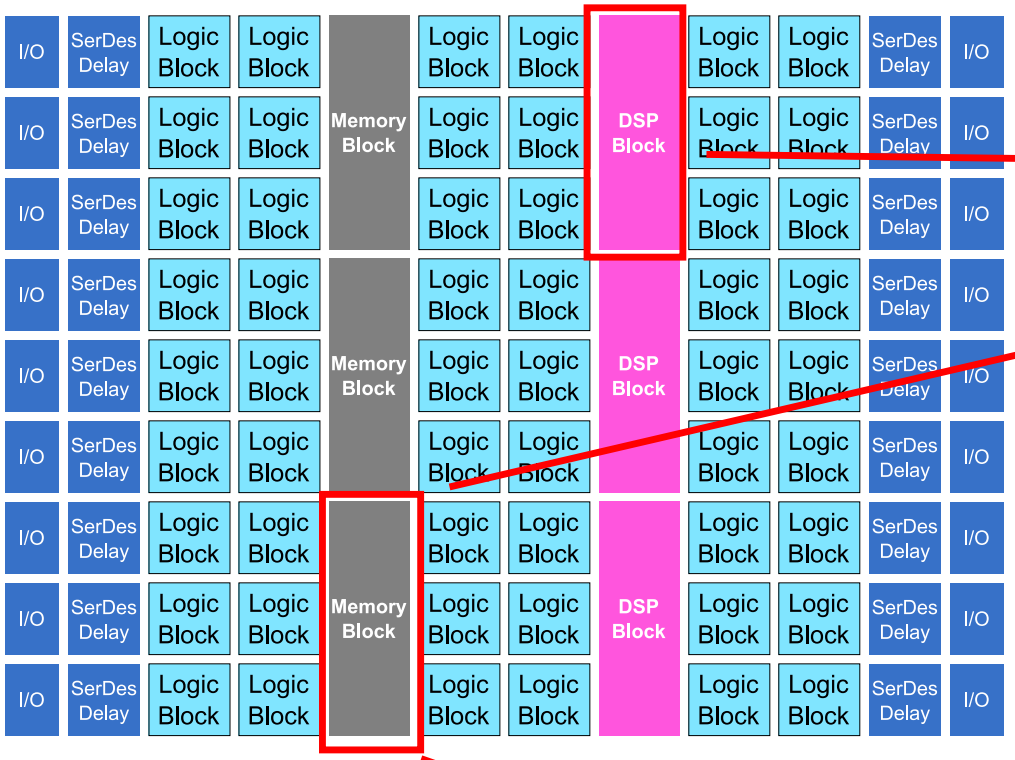


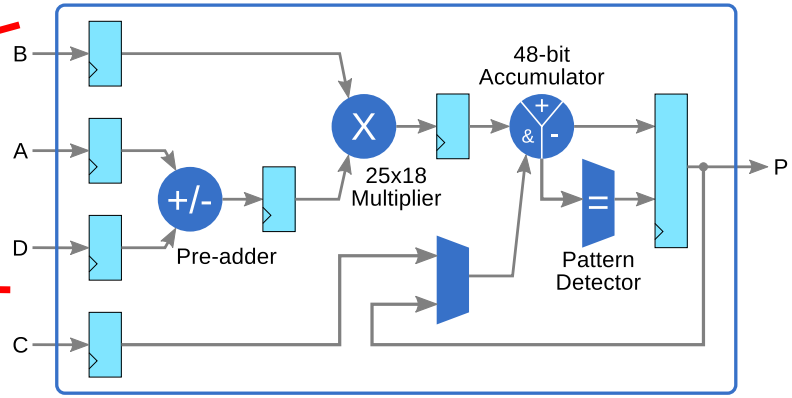
スイッチボックスの構成例

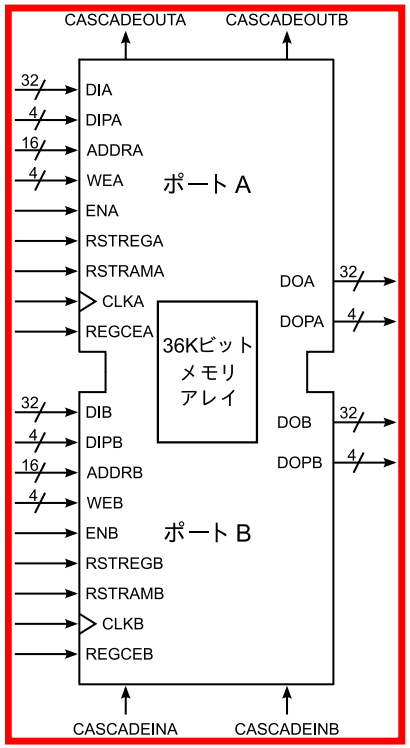


- スイッチを何段も通過するためゲート遅延に対する配線遅延の割合が大きい
- チップ面積に占める配線リソースの割合も大きい

近年のFPGAの構造

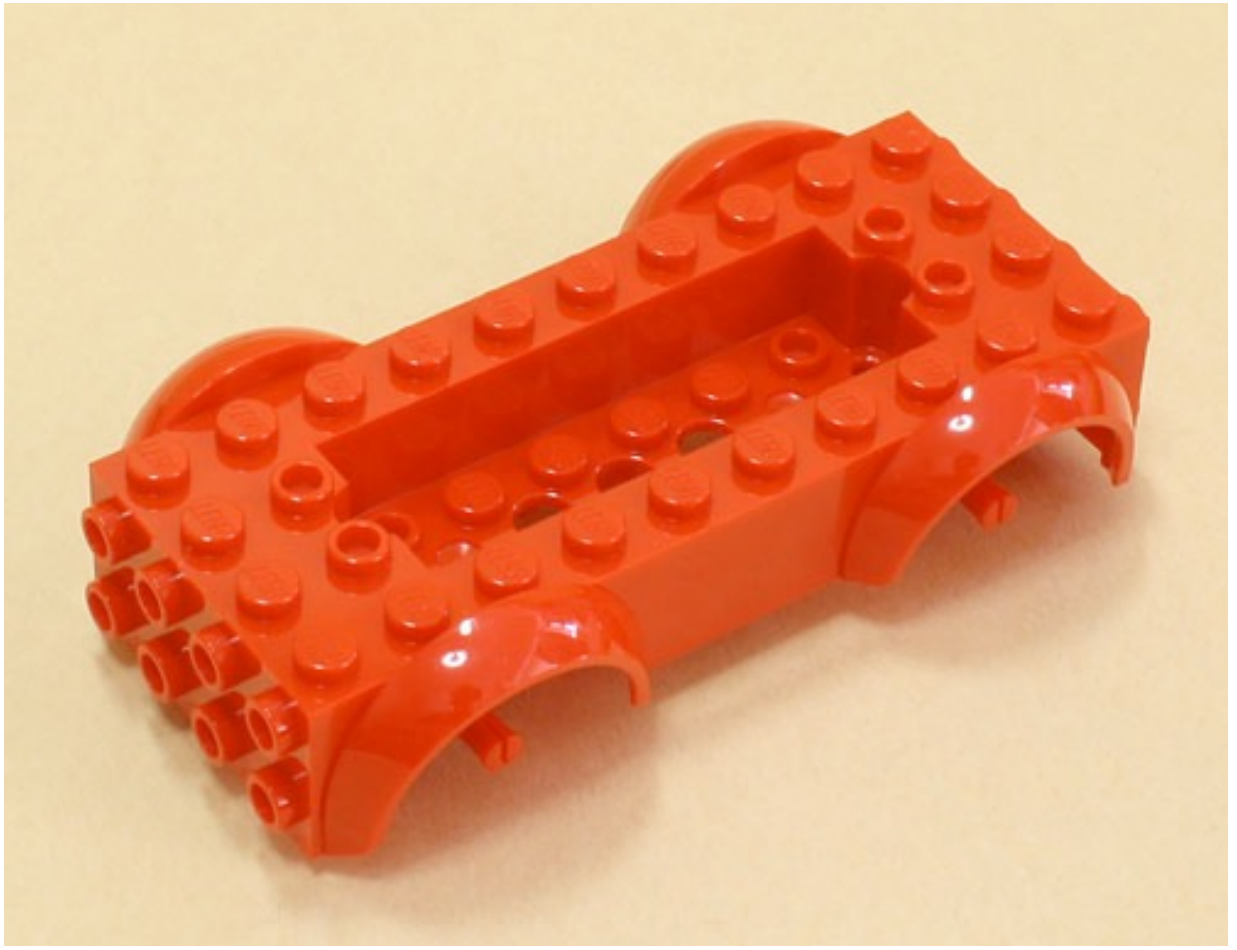






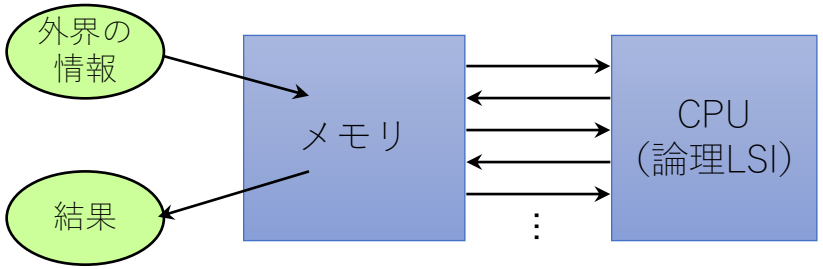
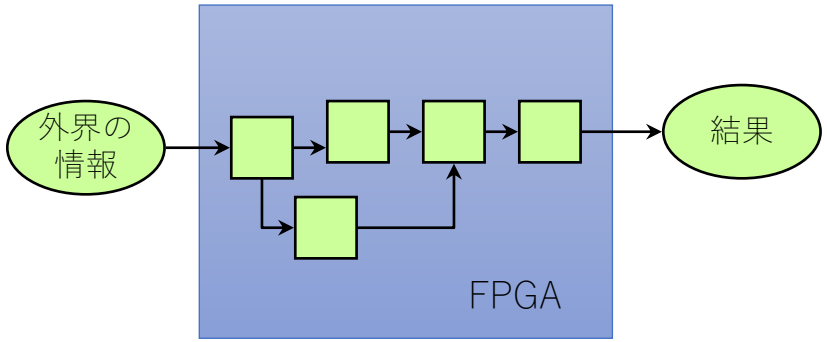
- プロセッサの混載
- 浮動小数点演算器を搭載したものもある

LEGOグロック今昔



http://brickstation.net

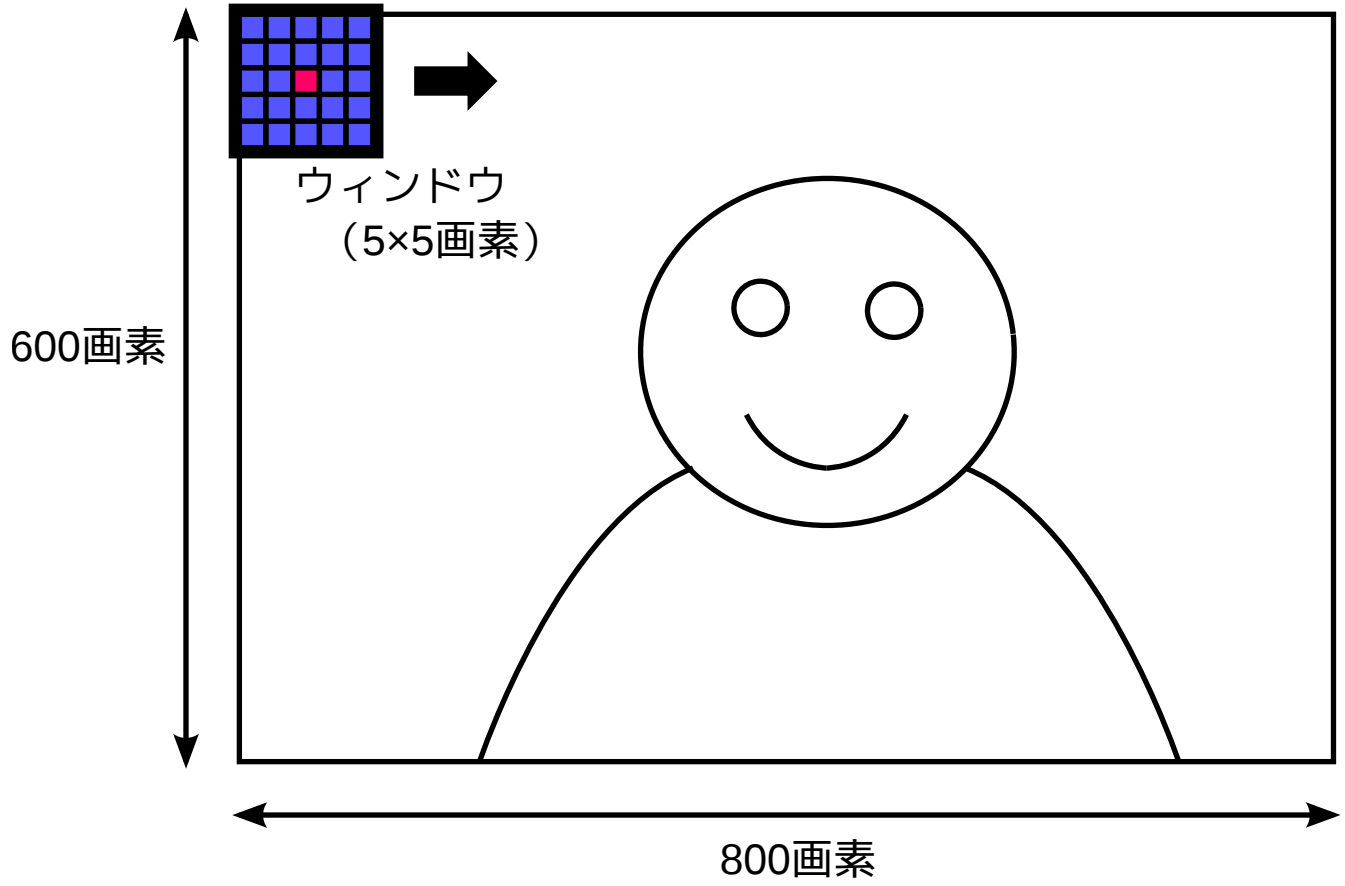
ストリーム処理

従来の計算手法（往復処理）	ストリーム処理
ソフトウェアによる処理 従来のハードウェアの使い方	アプリケーションに特化した 新しいハードウェアの使い方
	
外界の情報を一旦メモリへ取り込んで メモリとCPU間でデータを 往復させて処理を進める	外界の情報をそのまま 必要な処理の中を流す
逐次処理 二分されたブロック間での転送	パイプライン処理 → 高速 近接間でのデータ転送 → 低消費電力

アルゴリズムのハードウェア化

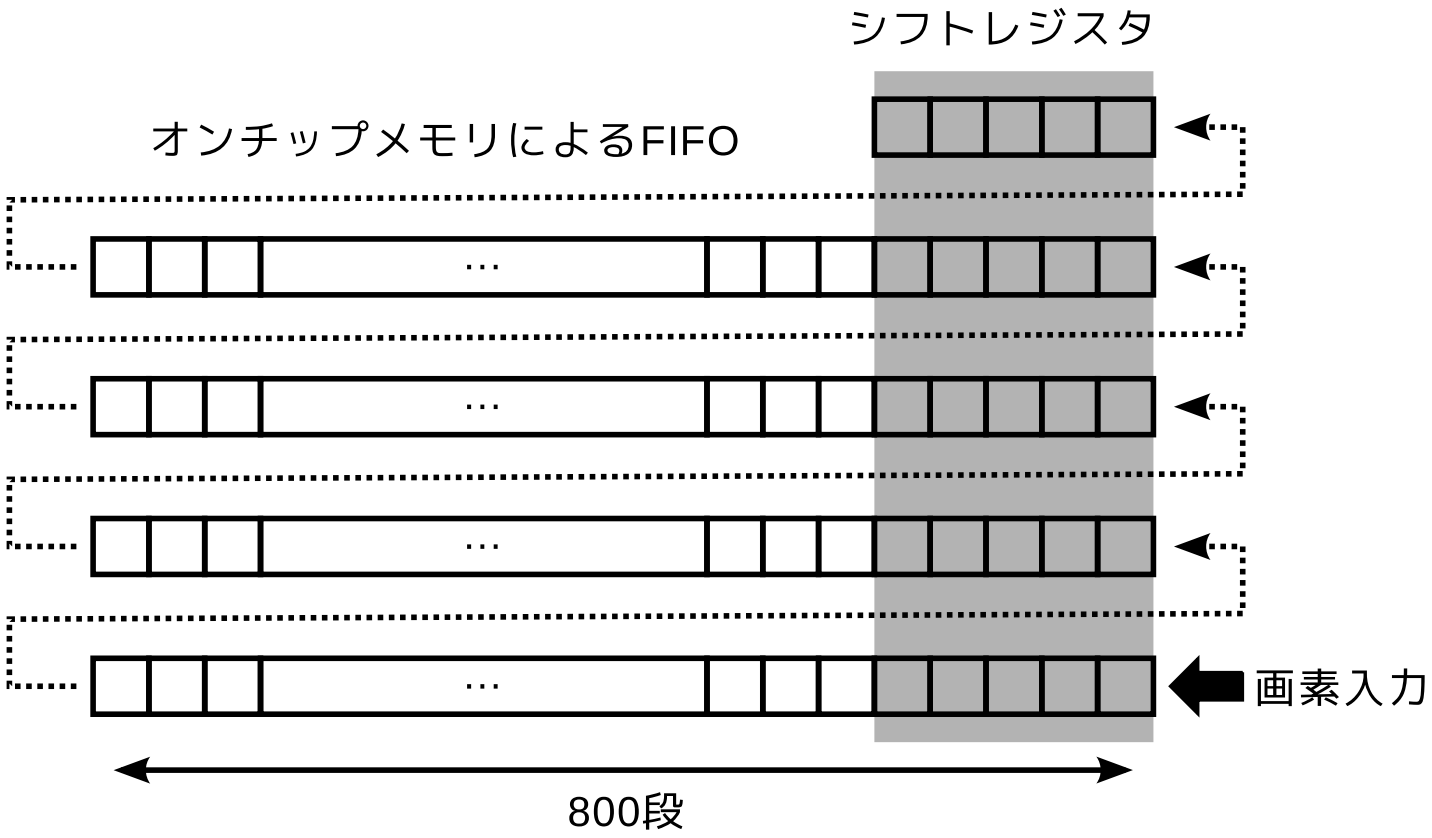
- ハードはソフトよりも速い？
  - かなり怪しい
  - そもそもソフトはハードで動いている
  - 汎用プロセッサのクロック周波数はFPGAよりも1桁高い
  - 「ソフトは逐次実行」されているわけではない
- それでもメリットはある？
  - 処理に応じた演算とメモリアクセス機構の実現による演算密度向上
  - 容量は小さいが高いバンド幅をもつ内部メモリの活用
  - 入出力処理と演算処理の密な結合
  - 高い電力効率
- 高位合成 (High Level Synthesis)
  - プログラミング言語によるハードウェア設計
  - ハードウェア構造を意識せずに良いハードウェアを設計できる？

例：画像処理における畳み込み演算



- ナイーブにやると1回の畳み込みあたり25要素の読み出しが必要
- 並列性は高いがメモリアクセスがボトルネックに

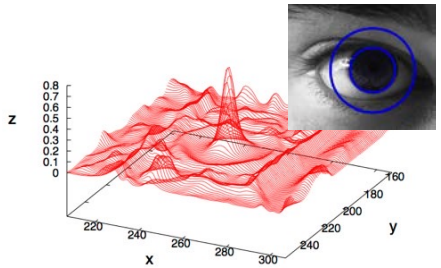
ラインバッファアーキテクチャ



- 1要素を読み出すたびに1回の畳み込みが行える
- カメラデバイスからの入力と演算をパイプライン化できる
- 多くの科学技術計算（ステンシル計算）にも応用できる

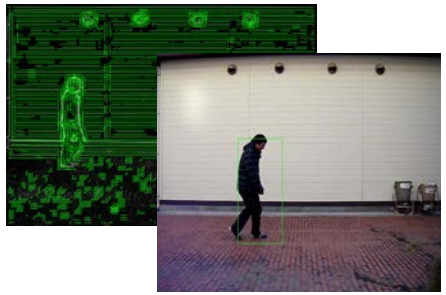
リアルタイム画像処理への応用

ステレオテンプレートマッチング
ISFPGA2008

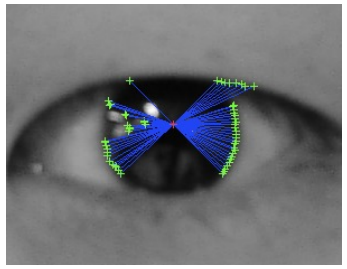


分離度フィルタ
JSST2008

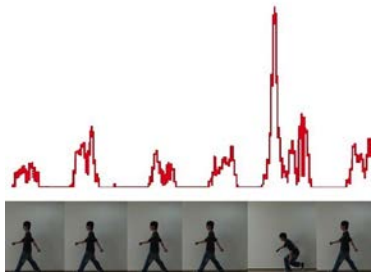
FASTコーナ検出
FPL2011



HOG・AdaBoost物体認識
FPT2011

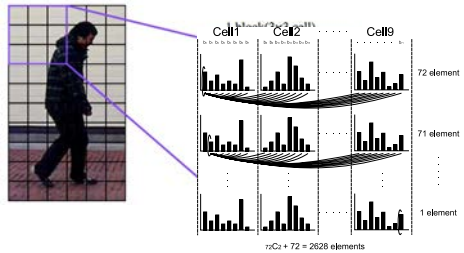


Starbust・RANSAC回帰
FPL2012



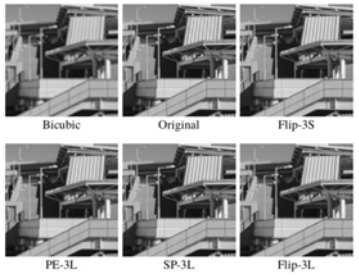
CHLAC特徴抽出・異常動作検出
ARC2014

AutoEncoderストリーム圧縮
RECONF2015

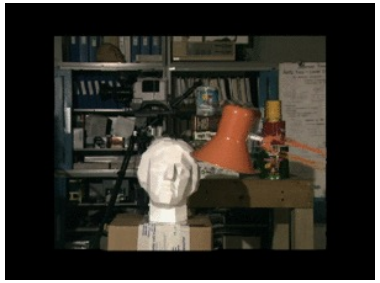


FIND特徴抽出
CoolChips2016

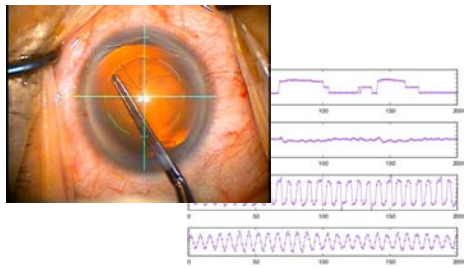
粒子フィルタ・物体追跡
CANDAR2016



CNN・超解像
FPT2016



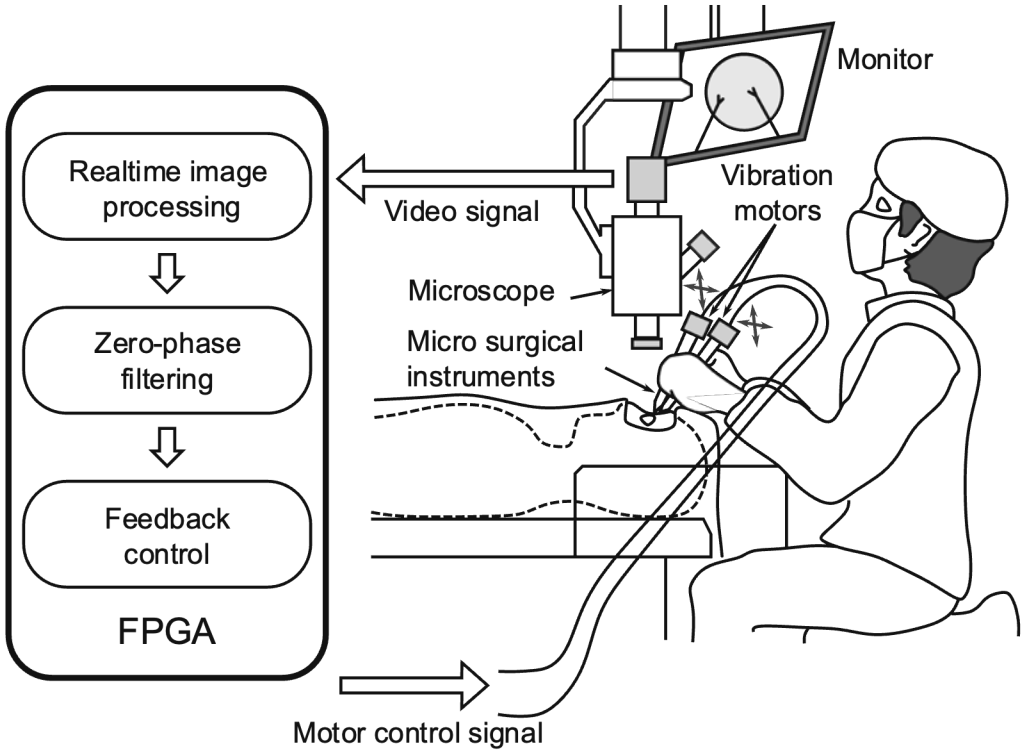
ステレオグラフカット
HEART2017

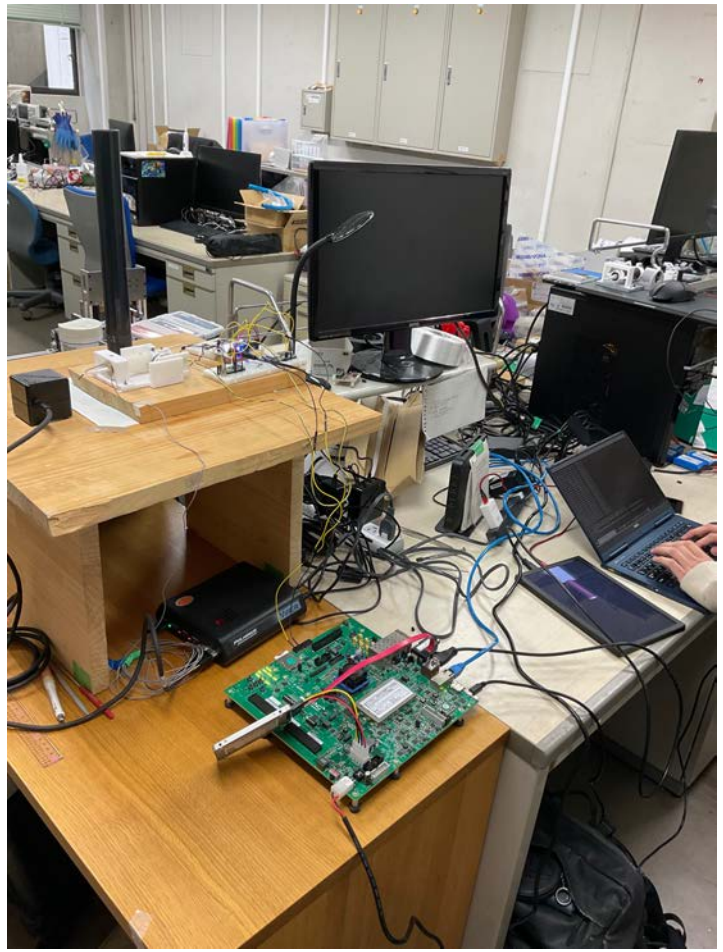


オプティカルフロー・振動推定
DASIP2018

高速画像処理を基軸とした振動抑制と医療応用

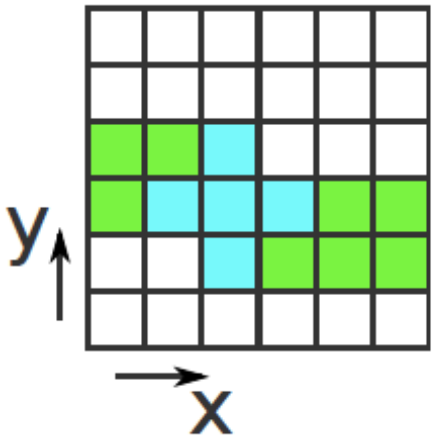
- 長崎大学病院眼科・中央大学との共同研究
- FPGAによる画像処理により振動情報をリアルタイムに抽出
- 専用の加振装置でアクティブ制振

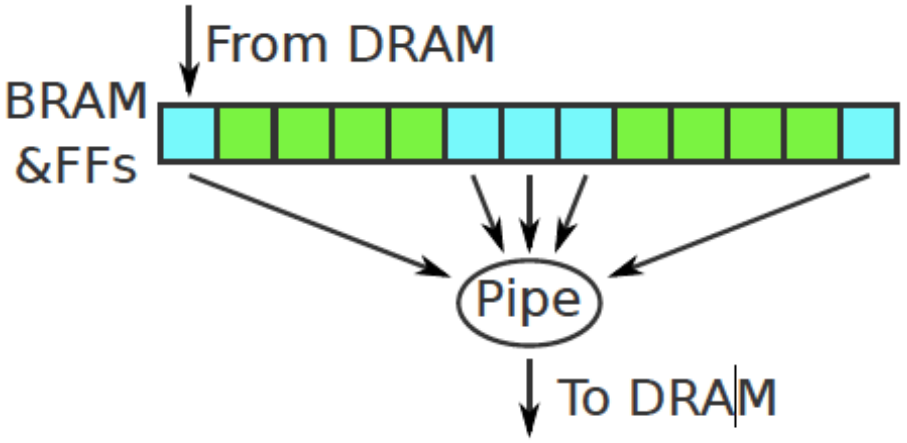




FPGAアクセラレータによるステンシル計算

- 格子要素の更新式をパイプライン演算器化
  - 格子要素をストリームとして流す
- 長い論理的FIFO
  - BRAM、LUT、FFの組み合わせ
  - 内部メモリの高いバンド幅を活用



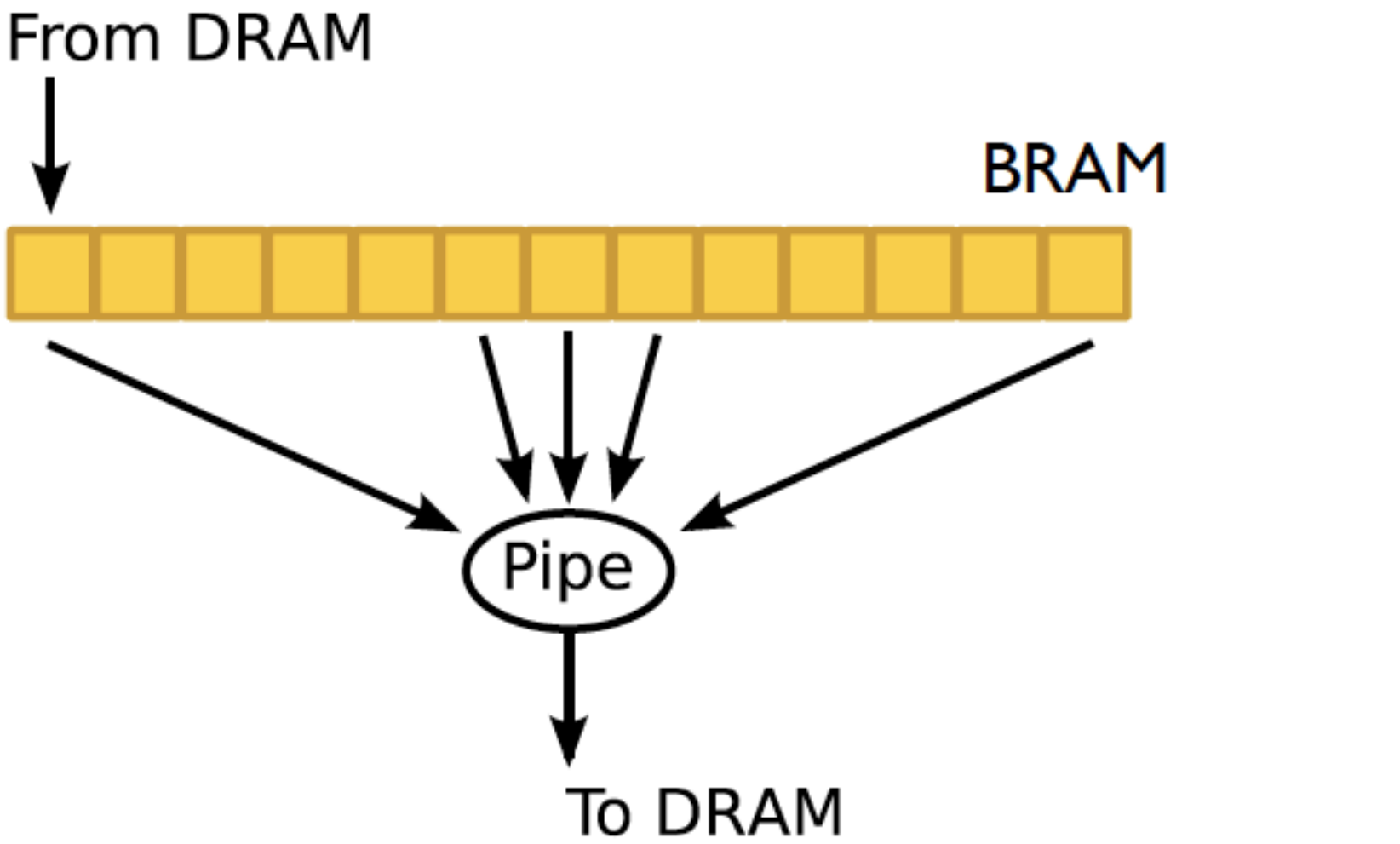


高位合成言語による熱拡散シミュレーション記述例

```
public class HeatKernel extends FDKernel {
    private Stencil getStencil() {
        double rollOn[][] = { { 0.0, -2, 1} };
        double rollOff[][] = { { 1, -2, 0.0 } };
        double interior [] = { 1, -2, 1 };
        return asymmetricStencil( -1, 1, rollOn, interior, rollOff, 2.0 );
    }
    public HeatKernel(FDKernelParameters param, HeatEngineParameters eparam) {
        super(param);
        int n_timestep = eparam.getNumStep();
        Stencil stencil = getStencil();
        FDVar input = io.waveFieldInput("in_T", 1.0, n_timestep);
        FDVar alpha = io.earthModelInput("alpha", 10, 0);
        FDVar result = constant.fdvar(0.0);
        optimization.pushDSPFactor( 1.0 );
        for ( int t= 0; t<n_timestep; ++t ) {
            FDVar laplacian = alpha * convolve( input, ConvolveAxes.XYZ, stencil);
            result = input + laplacian;
            result = result.cast(-5, 5);
            input = result;
        }
        optimization.popDSPFactor();
        io.hostOutput("receiver", result);// Receiver output to host
        io.waveFieldOutput("out_T", result);// Wavefield output
    }
}
```

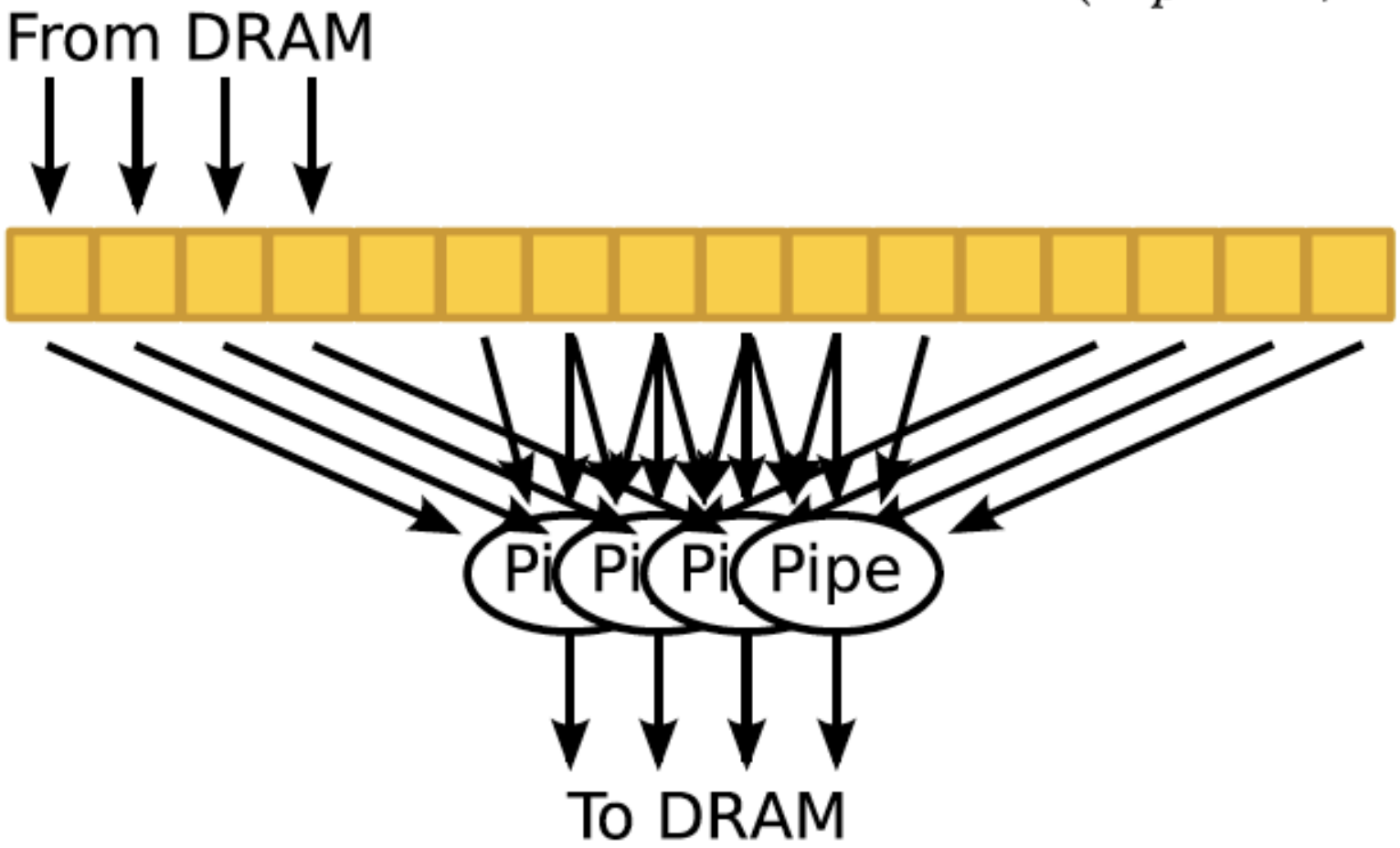
基本のパイプライン構成

$$(N_p = 1, N_s = 1)$$



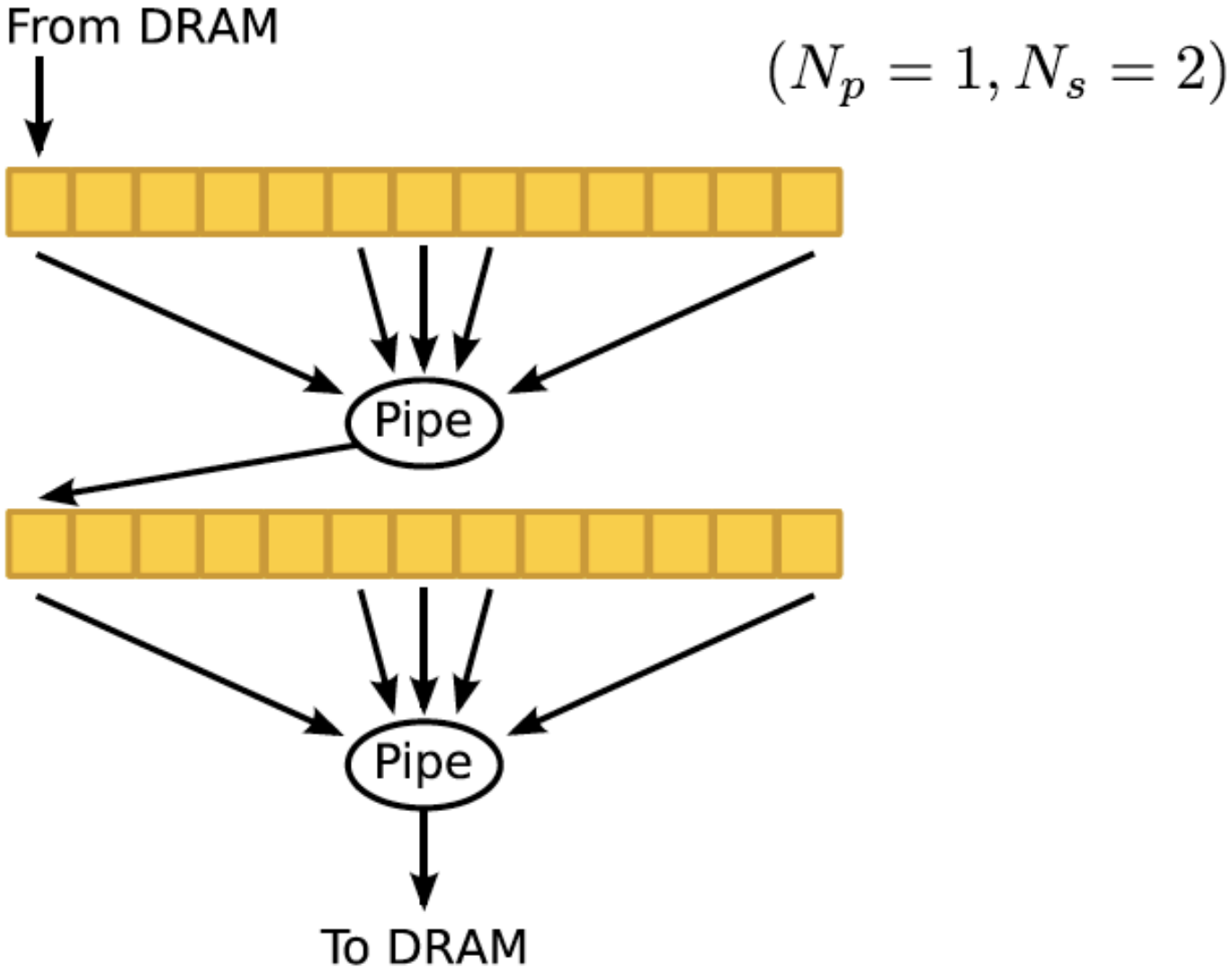
マルチパイプライン（x4）

$(N_p = 4, N_s = 1)$



必要な外部メモリバンド幅は 4 倍になる

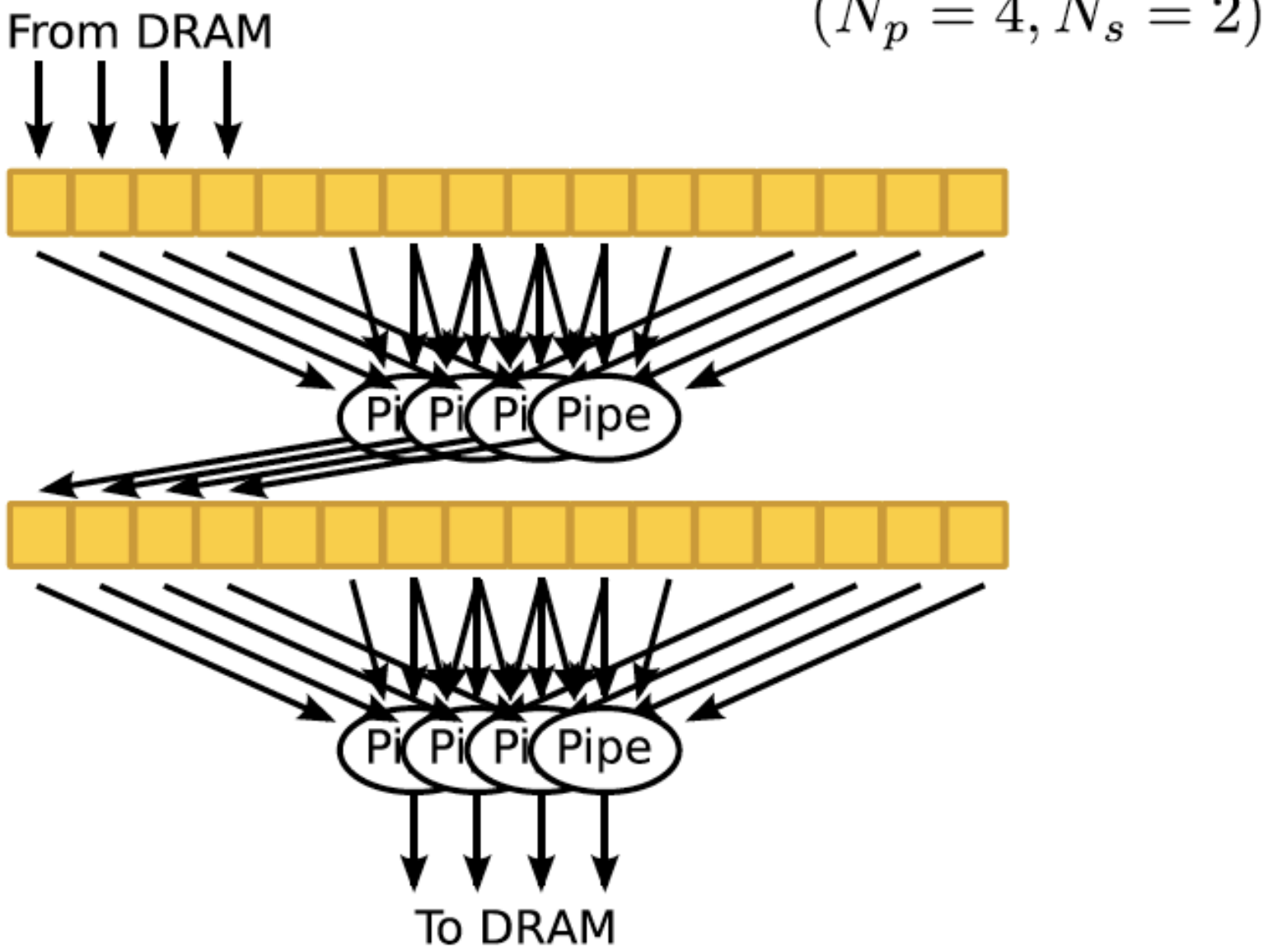
マルチステップ (x2)



必要なBRAMが2倍になり袖領域も拡大する

マルチパイプライン・マルチステップ併用 (x8)

$(N_p = 4, N_s = 2)$



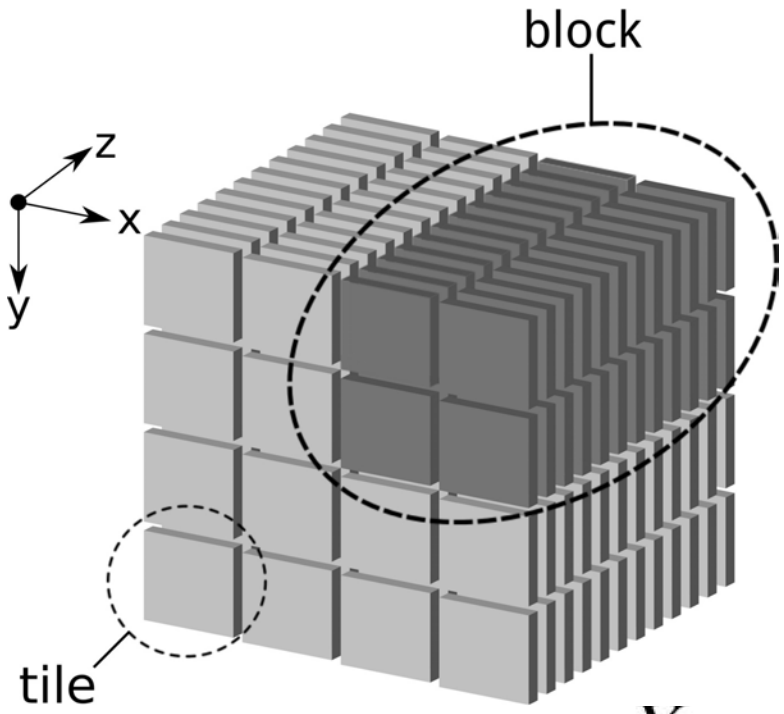
高位合成と設計空間探索

- 高位合成言語で簡単に記述できても、ユーザが決定すべきパラメータは多い
  - マルチパイプライン
  - マルチステップ
- どんなハードウェア生成されるのか？
  - 性能？　ハードウェア規模？
  - 実際に合成・配置配線して試すのは時間がかかる
- ユーザがカスタマイズできる範囲で、最適なパラメータを簡単に求めたい
  - 性能・資源量モデルの構築
  - 効率の良い設計空間探索

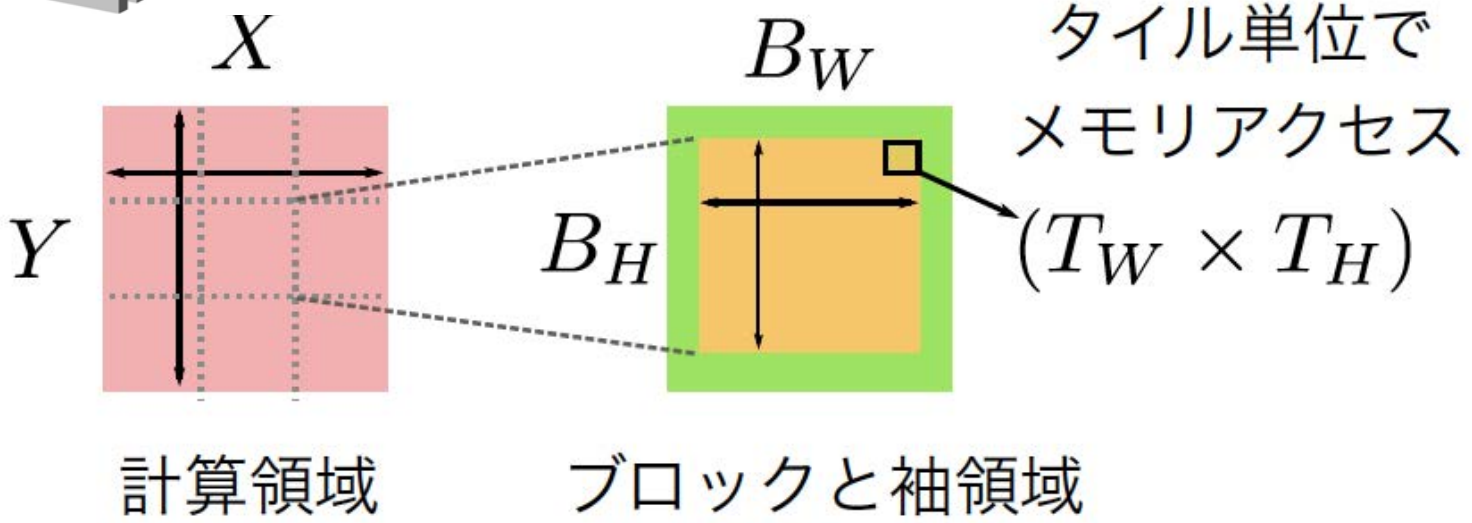
性能・資源量モデル

- 資源量
  - DPS数: Np * Nsに比例
  - BRAM数: Nsにほぼ比例
- 演算性能
  - DSP数に比例
  - メモリバンド幅の上限
- ステンシルの形状
  - メモリアクセス効率
- DRAMメモリバンド幅
  - FPGAシステムにより決定

計算領域の分割



- 計算領域全体をX-Y平面で3次元ブロックに分割
- メモリアクセスはタイル(2次元配列)単位で行う



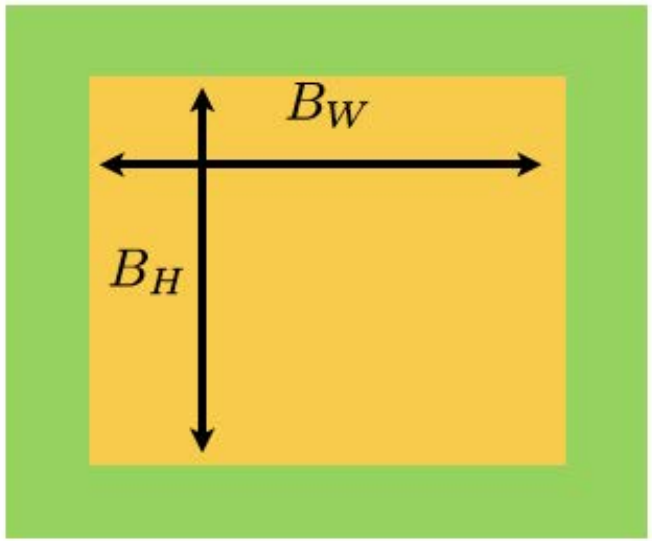
性能モデル：演算性能

$$G_{\text{read}} = \left(B_W + 2\left\lceil \frac{S_d N_s}{T_W} \right\rceil T_W\right) \times \left(B_H + 2\left\lceil \frac{S_d N_s}{T_H} \right\rceil T_H\right) \times Z$$

$$G_{\text{write}} = B_W \times B_H \times Z$$

$$\eta = \frac{G_{\text{write}}}{G_{\text{read}}}$$

$$P_{\text{compute}} = \eta N_p N_s F_s \ [\text{elements/sec}]$$



ブロックのX-Y平面

G_{read}	：参照要素数
G_{write}	：出力要素数
F_s	：周波数

性能モデル：メモリ律速性能

$$T_{\mathrm{mem}} = 8 \times 2 \times F_{\mathrm{mem}} N_{\mathrm{mem}} \,[\mathrm{Byte/sec}]$$

$$P_{\mathrm{mem}} = G_{write} \times \frac{T_{\mathrm{mem}}}{B_{\mathrm{mem}}} \times N_s \,[\mathrm{elements/sec}]$$

T_{mem} ：ピークメモリバンド幅

F_{mem} ：メモリ周波数

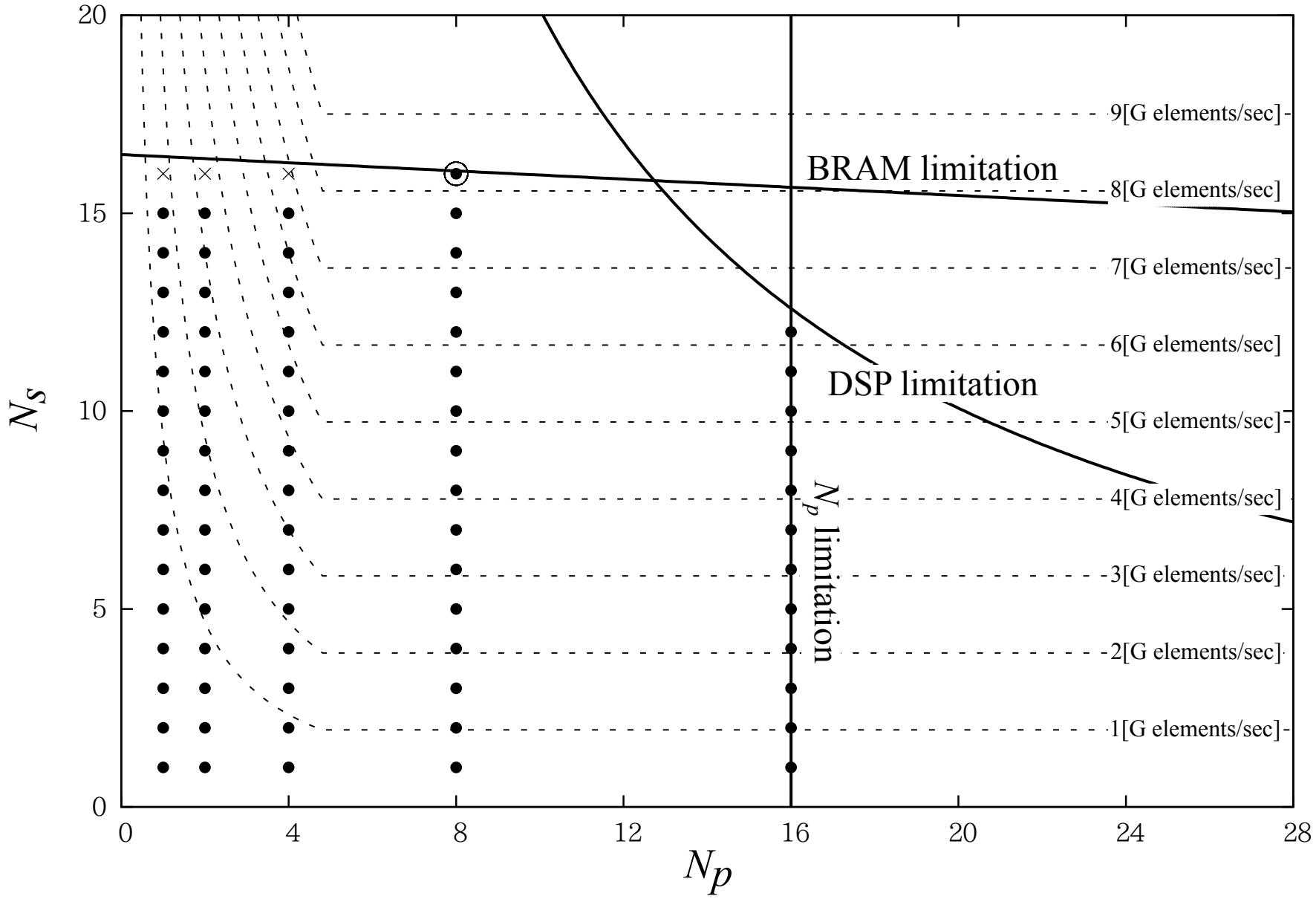
B_{mem} :ブロック当たりに必要なデータ量

P_{mem} ：メモリバンド幅によるピーク性能

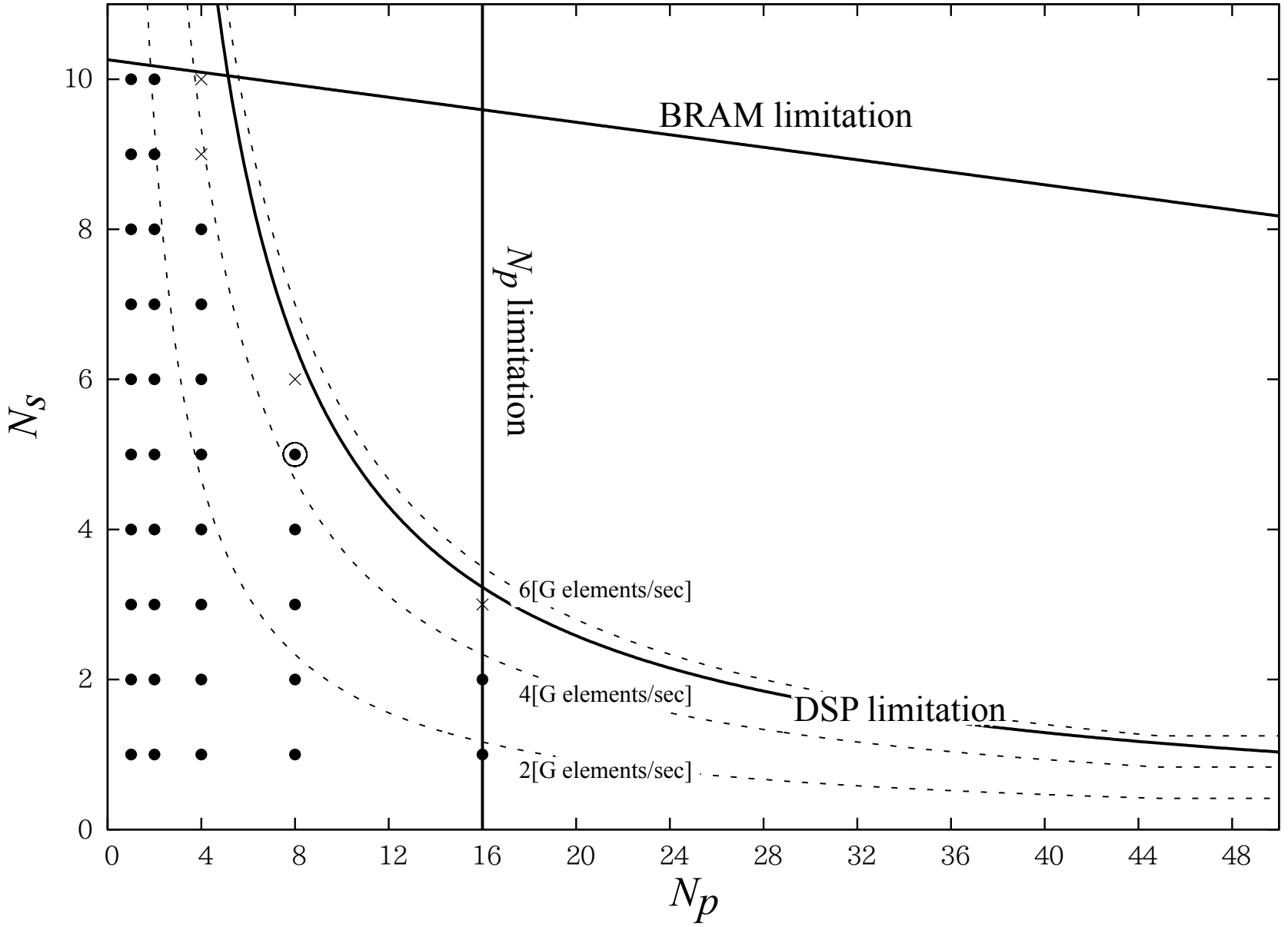
3次元熱拡散シミュレーション

- 陽解法に $\frac{\partial T}{\partial t}(x,y,z,t) = \alpha(x,y,z)\nabla^2 T(x,y,z,t)$
- 計算領域
  512 x 512 x 512
- 演算精度
  - 単精度浮動小数点数
  - 32ビット固定小数点数
- クロック周波数
  - パイプライン: 150 MHz
  - メモリ: 303 MHz

資源・性能モデルと合成結果（fixed）



資源・性能モデルと合成結果（float）

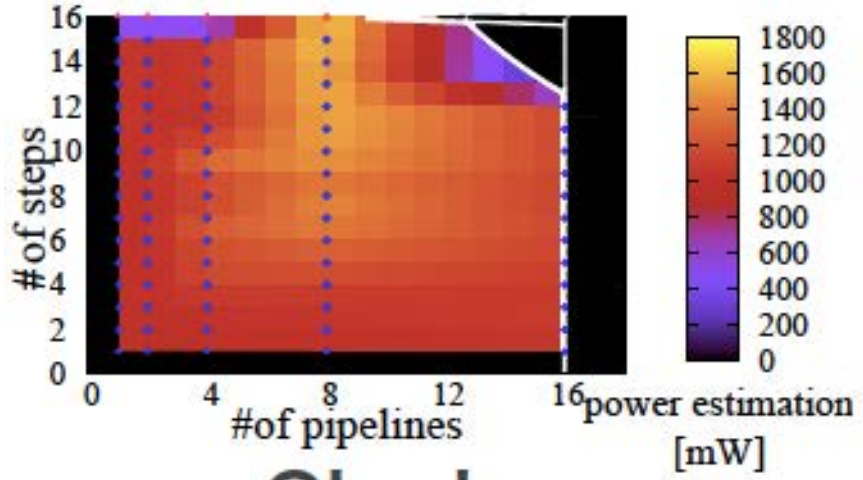


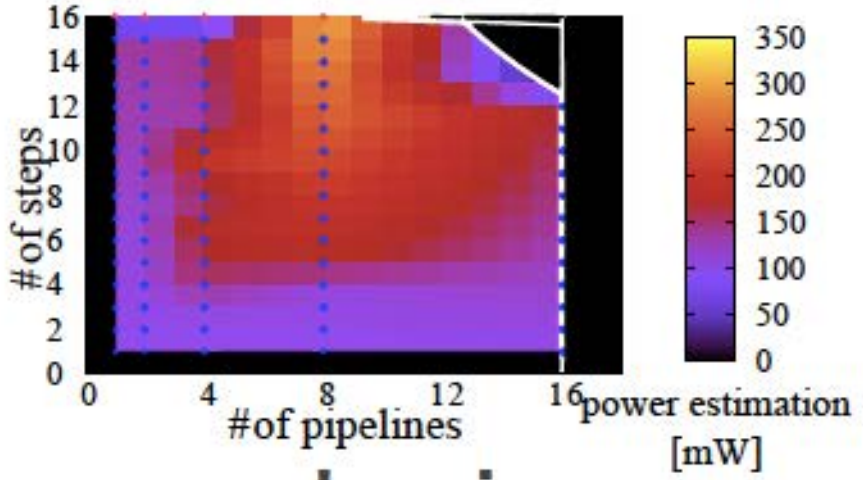
性能と消費エネルギー

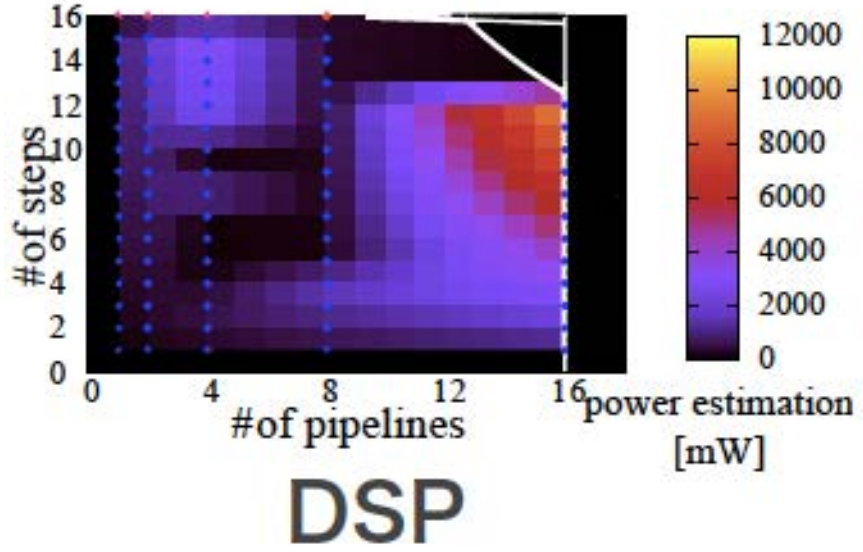
- 性能を最適化するための設計空間探索法は目処がついた

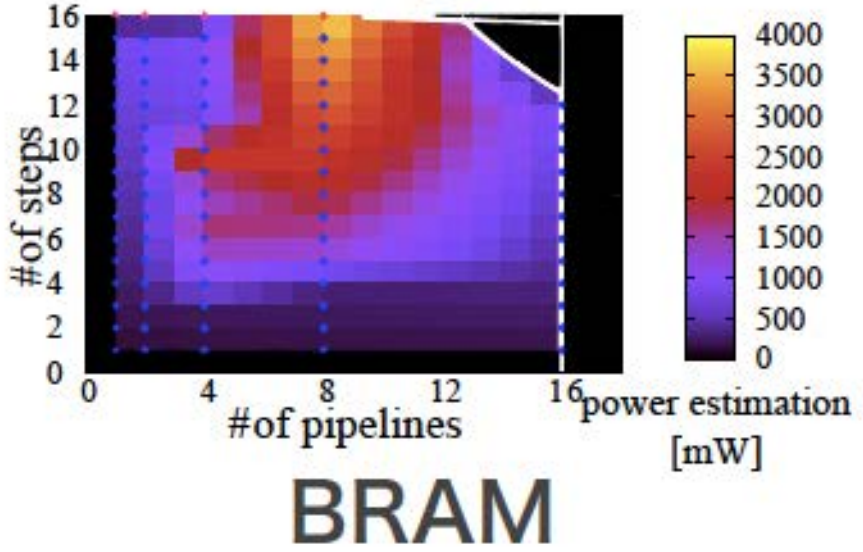
- エネルギー効率（単位エネルギーあたり性能）を最適化するには？
  - 指針を得るために，FPGAアクセラレータの消費電力をプロファイリング
  - XPower Analyzer
  - 熱伝導シミュレーション（fixed）

リソースごとの消費電力



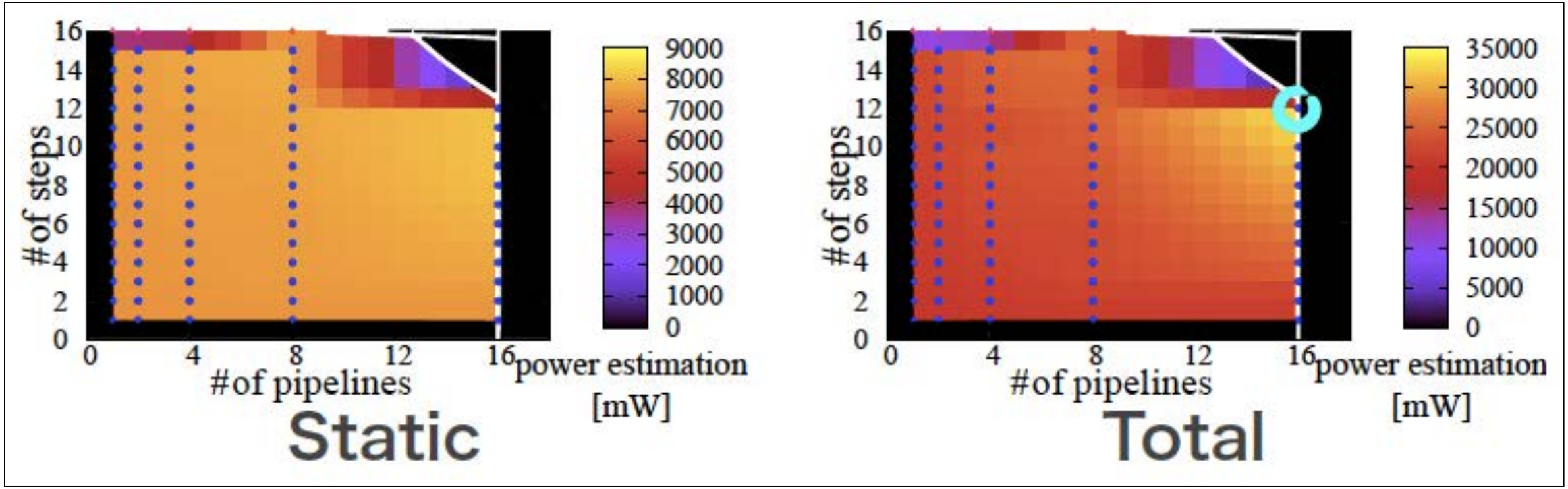






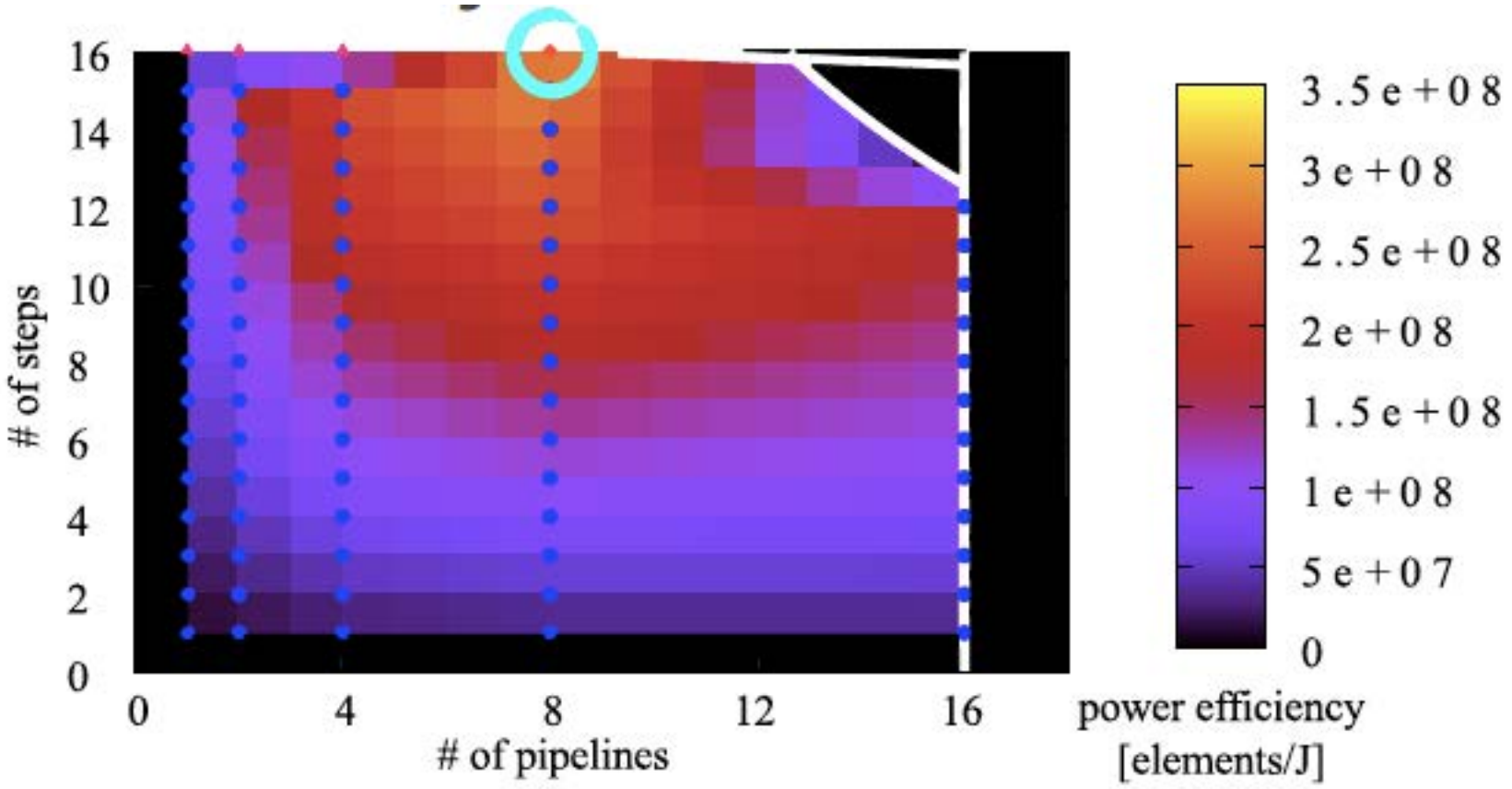
DSPによる消費が支配的

静的電力とトータルの消費電力



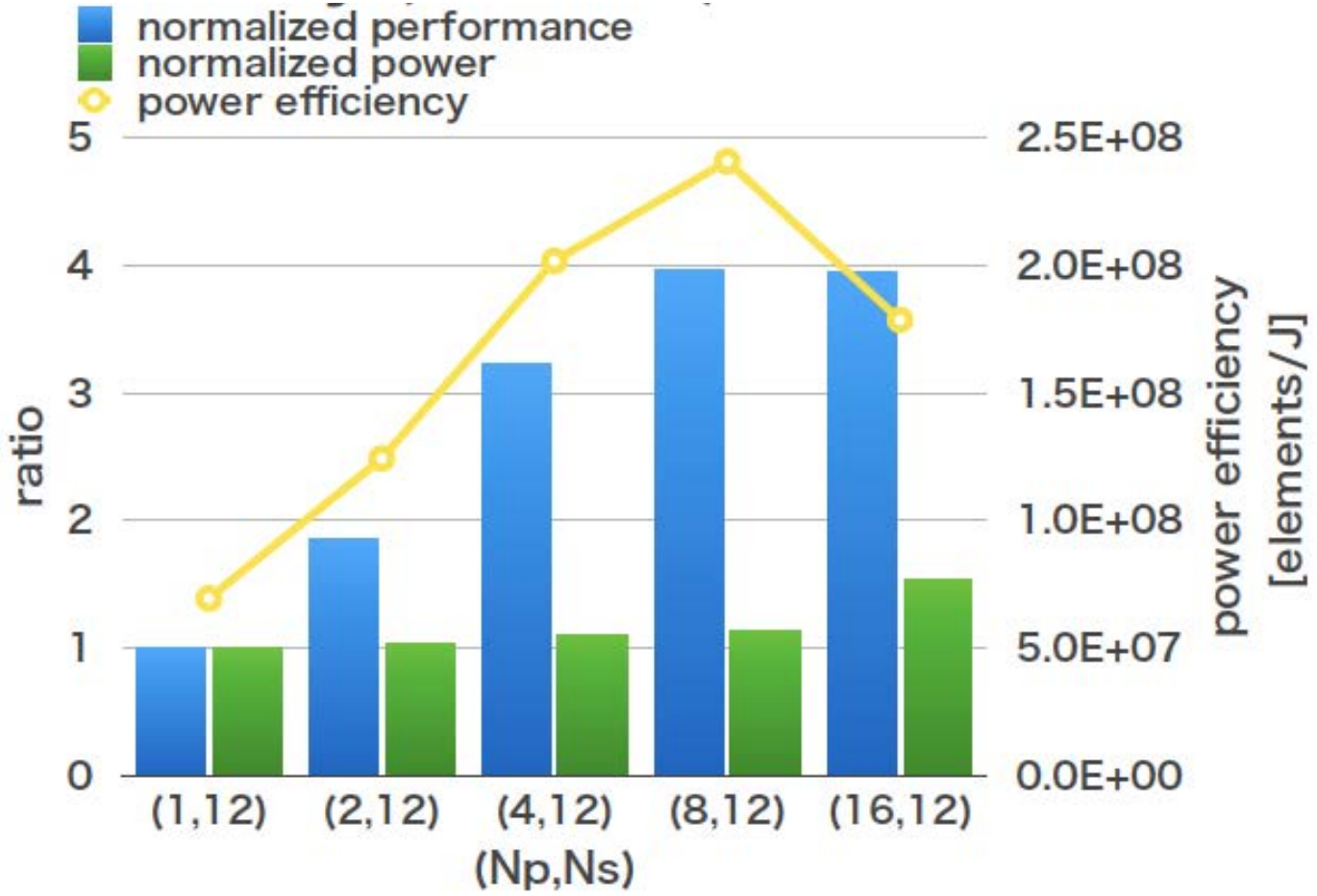
$(N_p, N_s) = (16, 12)$ のとき最大

エネルギー効率
（1Jあたりの計算要素数）



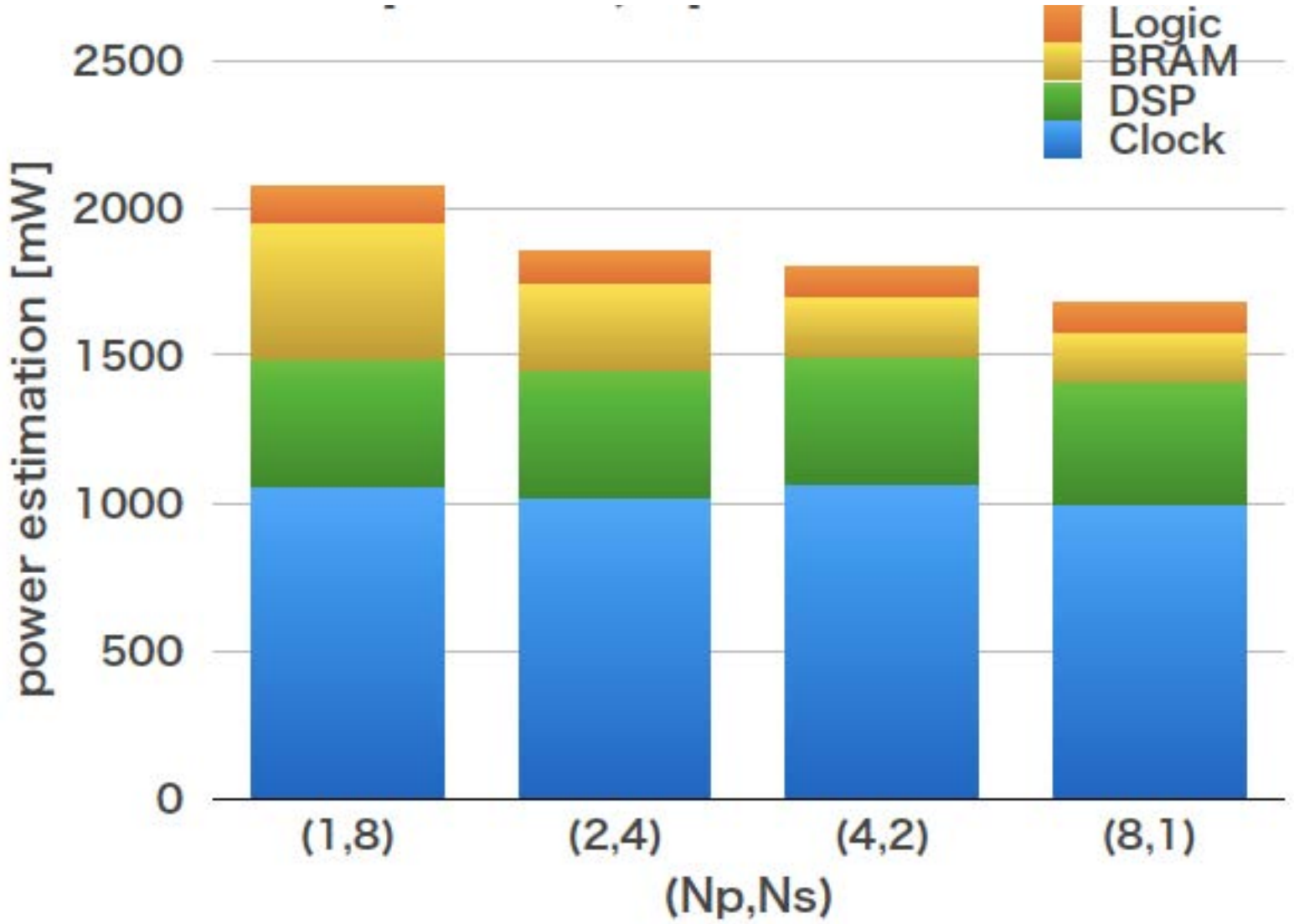
- $(N_p, N_s) = (8, 16)$ で効率最大
- 性能の最適化と矛盾しない

Ns = 12 の場合



メモリバンド幅が十分であれば
パイプライン数の増加に伴って効率も向上

Np x Ns = 8 の場合



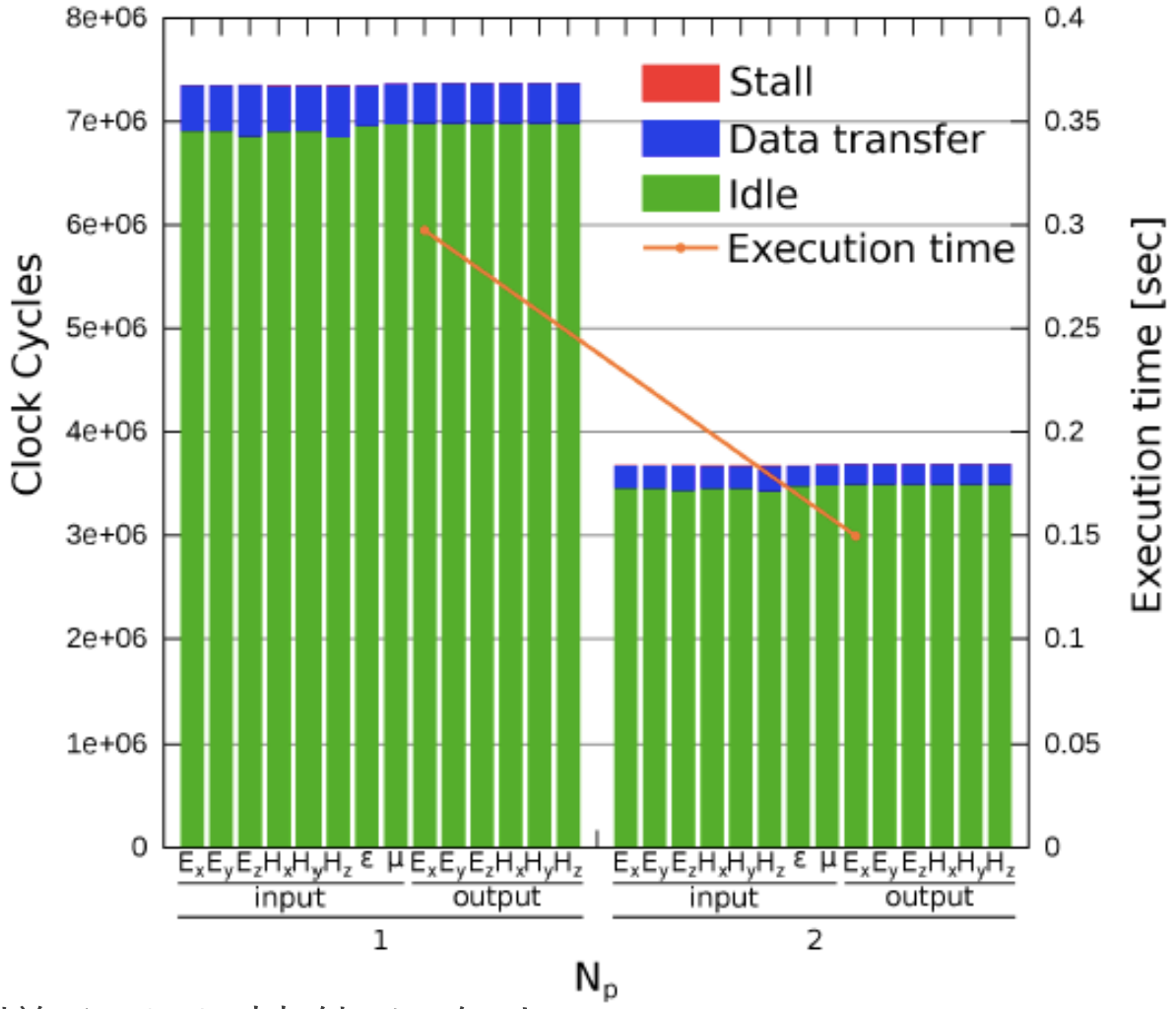
同じ性能なら Ns より Np を増やす方が有利（BRAM）

FPGAアクセラレータ特有の最適化手法

- アプリケーション記述と生成されるハードウェアの関係をさらに明らかにしたい

- プロファイラ機能つきステンシル計算用フレームワークを実装
  - 高位合成言語のクラスライブラリとして実装
  - ハードウェア稼働率やメモリアクセス性能をプロファイリング
  - データストリーム制御回路に着目
  - FDTD法による電磁界シミュレーションで評価

FDTDシミュレーションのプロファイル結果



- Np を増やすと性能も向上
- ストリームコントローラもアイドル率が高く性能向上の余地

パイプライン数とハードウェア規模

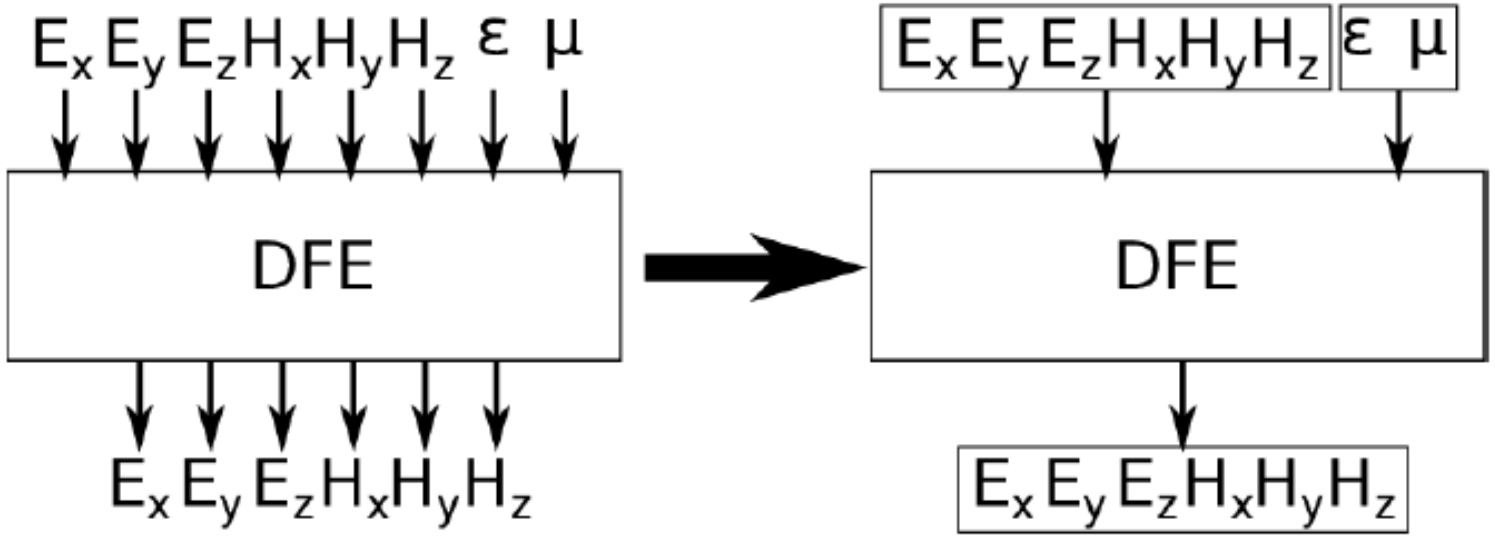
- Np = 4の場合

N_p	LUTs	FFs	BRAMs	DSPs
4	165070(55.47%)	224807(37.77%)	727(68.87%)	384(19.05%)

  - 資源容量にはまだ余裕があるが，タイミング制約を満たさず合成不可
  - ストリーム増加に伴いDRAMコントローラが複雑化？
  - ストリーム数を減らすのが効果的？

AoS化によるストリーム数削減

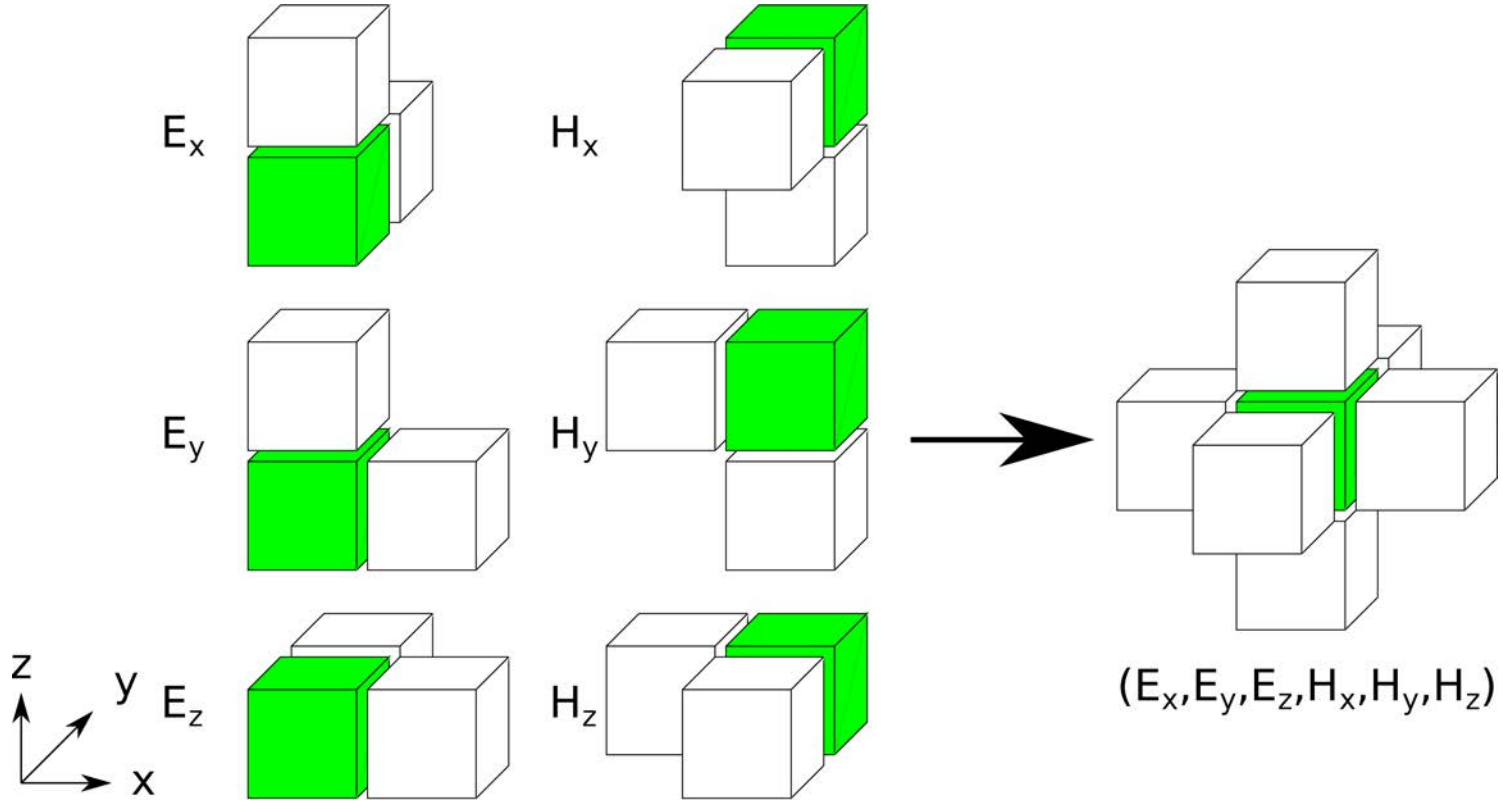
- ストリームされるデータをSoA（配列の構造体）からAoS（構造体の配列）に変更



- ただし、不要なメモリアクセスが生じる

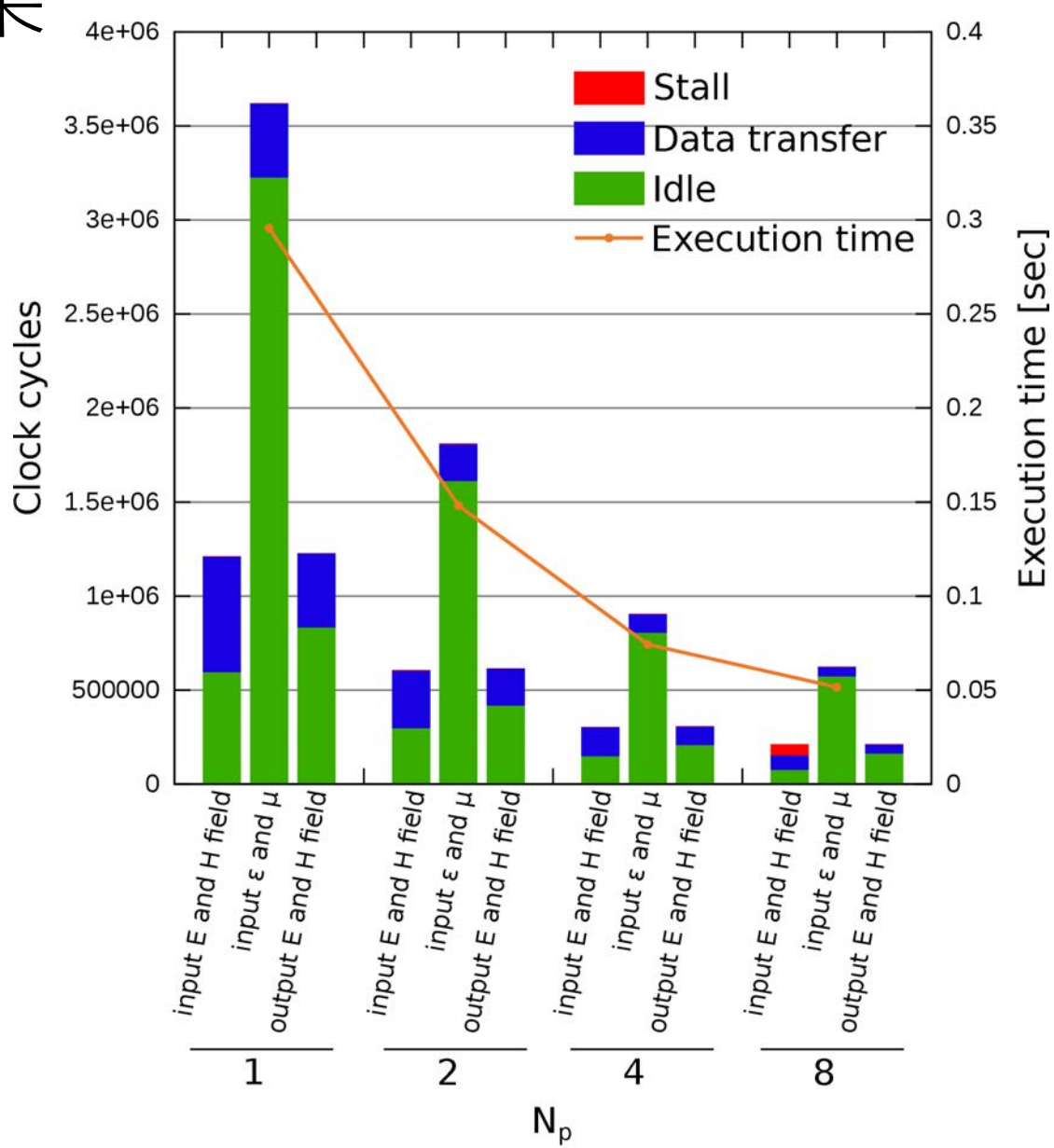
AoS化によるメモリアクセスオーバヘッド

- FDTD法ではストリームごとにステンシル形状が異なる
- AoS化することでステンシルが共通化
- メモリアクセスが約1.3倍増加



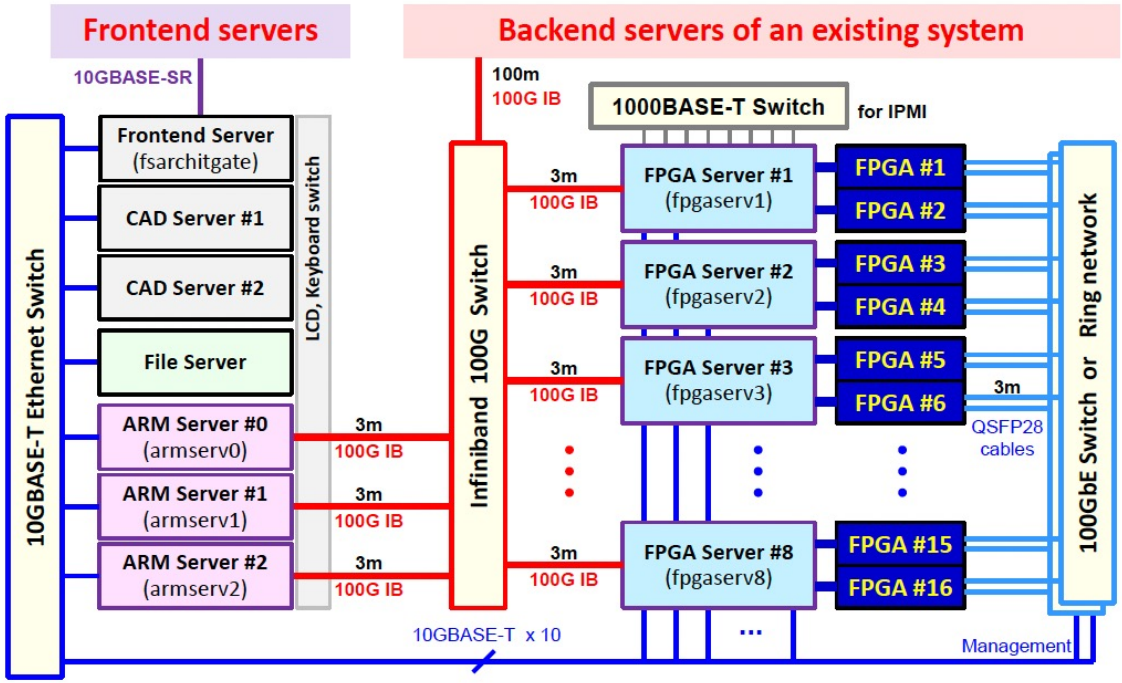
ストリームAoS化の効果

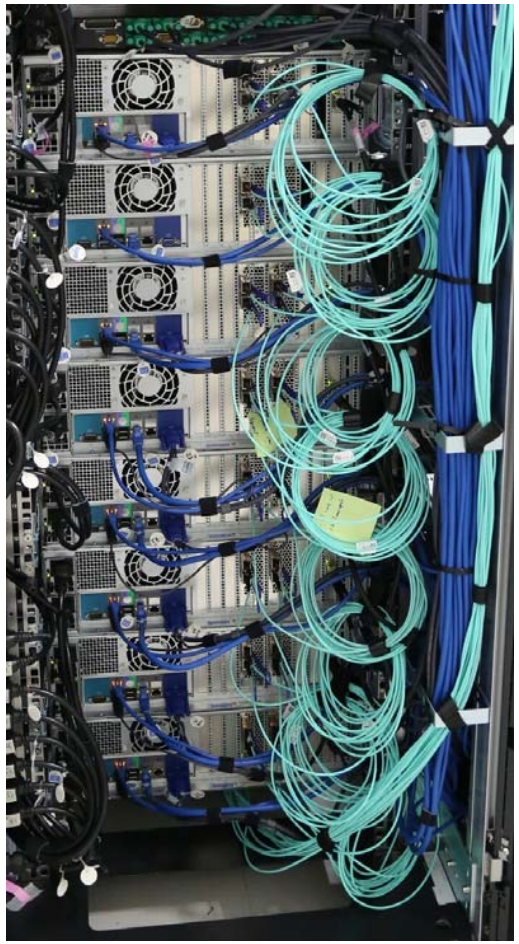
- Np=8まで実装可能になり
  約3倍の性能向上



スーパーコンピュータ富岳の高付加価値検討

- 理化学研究所との共同プロジェクト
- 既存プロセッサアーキテクチャの性能向上鈍化
  - 問題特化型のハードウェア機構が有望
- FPGAクラスタ実験試作機ESSPER
  - 富岳に接続し理研と共同で研究を実施

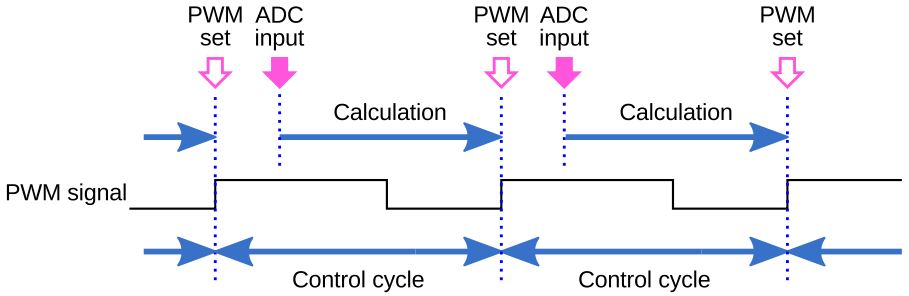


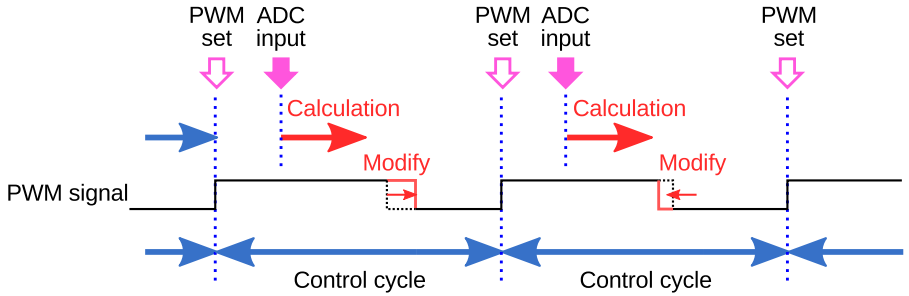


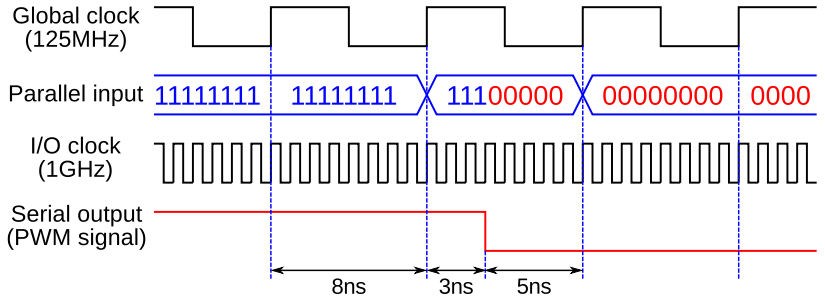
電力変換制御への応用

- リアルタイム信号処理
- 高速信号解析と並列制御演算
- ダイレクトな高分解能PWM制御

NEDOグリーンITプロジェクト
直流給電用DC/DCコンバータのFPGA制御







おわりに

- やわらかいハードウェアによるコンピューティング
  - コンピュータアーキテクチャに柔軟性を導入
  - 論理の世界（アルゴリズム）と物理の世界（コンピューティングデバイス）の最適なインタフェースを模索
- プログラマブルデバイスを計算の高速化に用いることは当然になった
  - ただし既存のアルゴリズムがそのまま有効とは限らない
  - 計算処理と入出力処理との密な連携が鍵
- 高位合成技術がアプリケーション生産性を飛躍的に向上
  - ただしソフトウェアとは異なる記述法も必要
  - 現状ではハードウェア設計を楽に行う技術
  - 設計空間探索のためのアーキテクチャモデリングも重要
- Domain Specific Architectureの先の汎用的な枠組みを模索
  - 動的再構成の有効利用については研究の余地