『メニーコア/ネットワークオンチップの基礎と組込みシステムへの応用』(前編)

鯉渕 道紘
国立情報学研究所

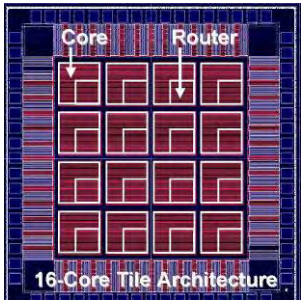
自己紹介 (計算機システムNW)

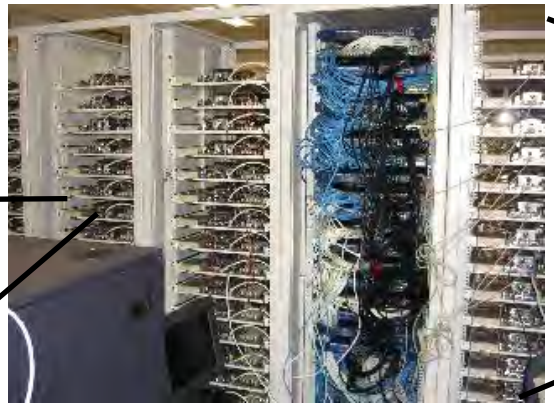
2005年　国立情報学研究所入所(鯉渕研立ち上げ), 現在に至る(構成:計3名)

http://research.nii.ac.jp/~koibuchi/

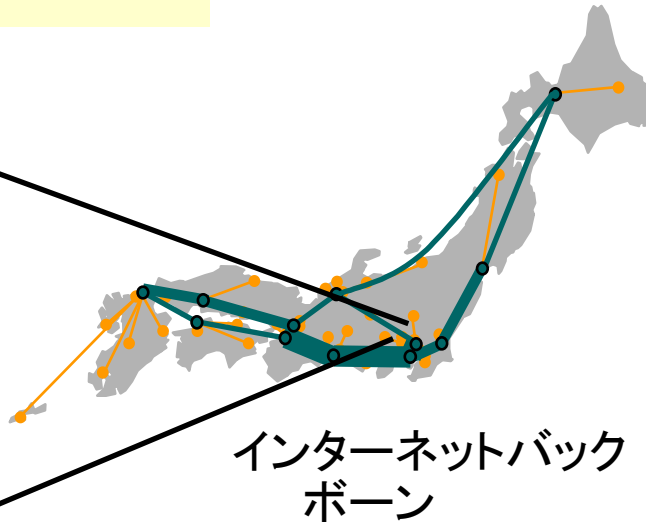
- メニーコア・ネットワークオンチップ
- スパコン　/　データセンターのインターコネクト
- インターネットバックボーンアーキテクチャ

メニーコアプロセッサ内NW





HPC向けインターコネクト



インターネットバックボーン

内容

- ネットワークオンチップとは？
- トポロジ
- スイッチング技術とルーティング
- ルータアーキテクチャ
- まとめ

メニーコア

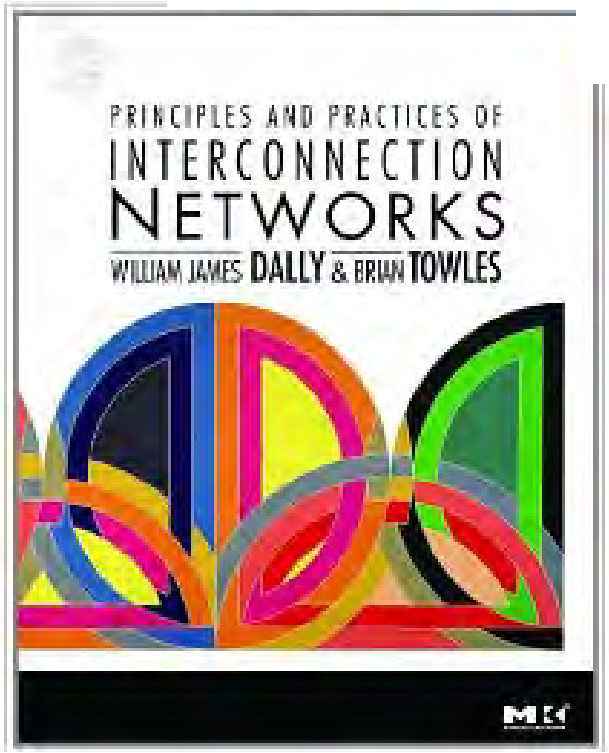
大体メニーコアへ

- マスマーケット：Intel, AMD クアッドコアCPUs
- カスタムSystems-on-Chip (SoCs)

“Number of cores will double every 18 months“
- Prof. A. Agarwal, MIT, founder of Tilera Corp.

教科書

Appendix: 相互結合網
(訳:鯉渕、松谷)
100ページ超, 2008



CS系大学院生向け

NoC 研究成果の発表場所

- 設計　DAC　/　DATE　（6枚）　数件
  - *www.dac.com*, *www.date-conference.com/*
- 計算機アーキテクチャ　ISCA / MICRO / HPCA(12枚）　数件
- NOCS　(10枚）　20件以上
  - *www.nocsymposium.org/*

本チュートリアル後には、大体これらの論文が読めます

IEEE MCSoC（9/26-28@東京）

NoC/SoC領域の一流研究者が来日

- Ran Ginosar, Technion, Israel, The Plural Architecture: Shared Memory Many-cores with Hardware Scheduling
- Jose Flich, Universidad Politécnica de Valencia, Spain, Many-core System Designs through Effective Routing Support and Reconfigurability
- Jiang Xu, Hong Kong University of Science and Technology, China, Network-on-Chip Benchmarks Based on Real MPSoC Applications
- Peter A. Beerel. USC, Practical Advances and Applications of Asynchronous Design
- Michael McCool, Intel, Embedded and Mobile Software Development for Intel SoCs

http://www.mcsoc-forum.org/

8/25が早期登録期限（学生：35,000円）

NoCとは？

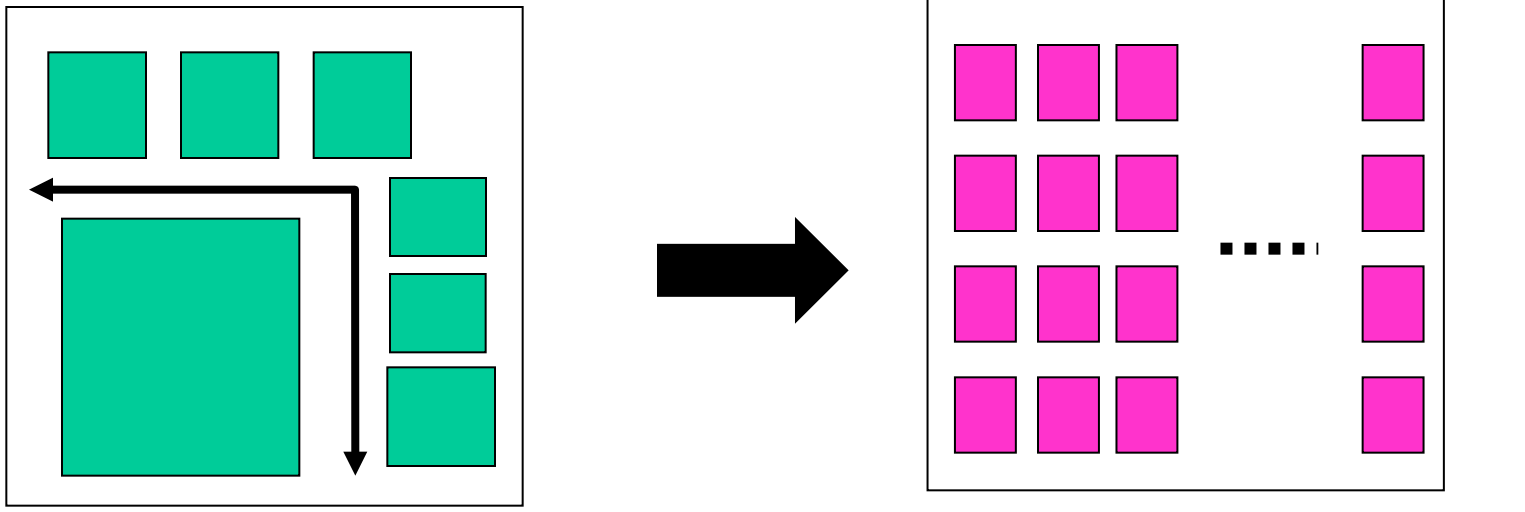
- チップ内ネットワーク＝
  Network-on-Chip (NoC) =
  On-Chip Network(OCN)=
  On-Chip Interconnection Network(OCIN)
- 広義
  - チップ内のモジュール間ネットワークすべて
    - 古典的バス:
      - 単位時間あたり1データ転送, poor performance scalability
    - 専用配線
      - 全対全通信. poor area scalability
    - パケットネットワーク
      - 複数データ転送: high scalability
- 狭義
  - スイッチ（ルータ）ベースのパケットネットワーク

NoCパラダイムシフト

- Bus からP2P パケット転送へ

  Route packets, not wires – Bill Dally, 2000

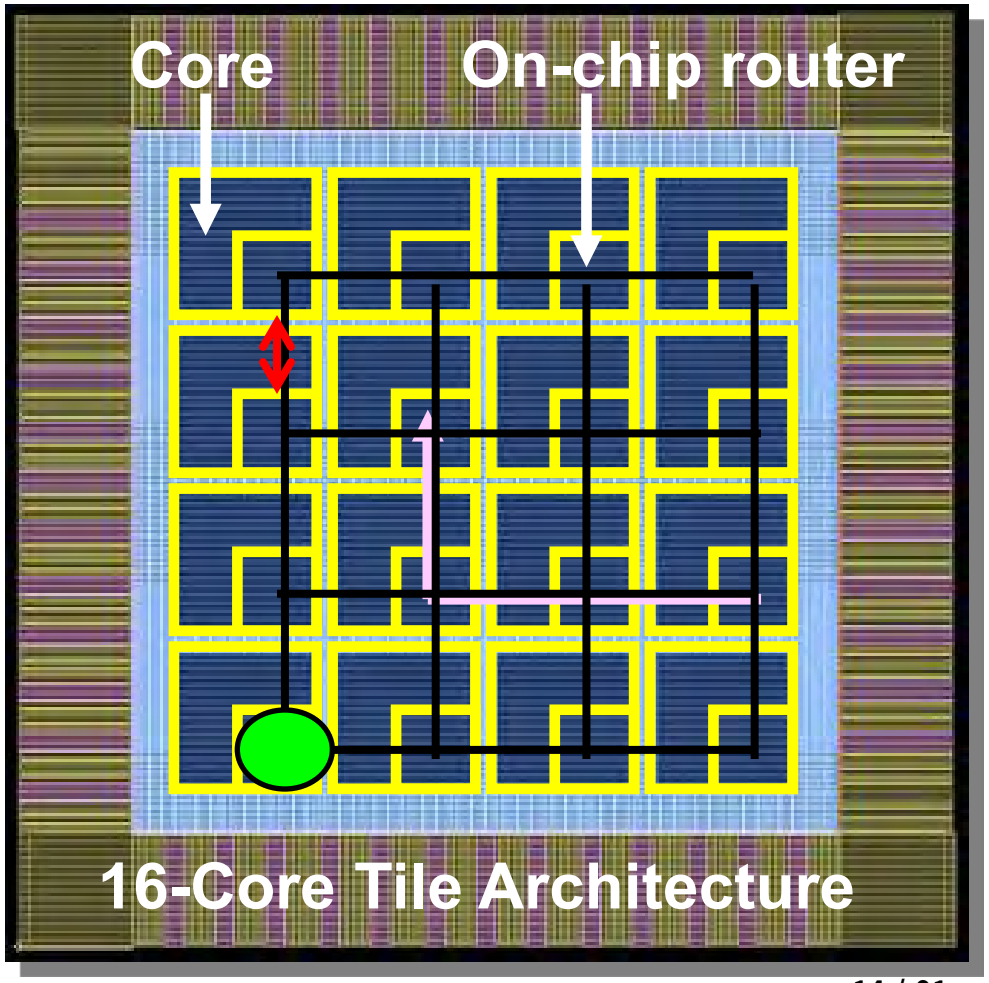
- バス構造の限界
  - コア数の増加
  - 配線遅延の増加



NoCの例

- トポロジ
  - 2次元メッシュ
- スイッチング技術
- ルーティング
- アービトレーション

（ルータアーキテクチャ）

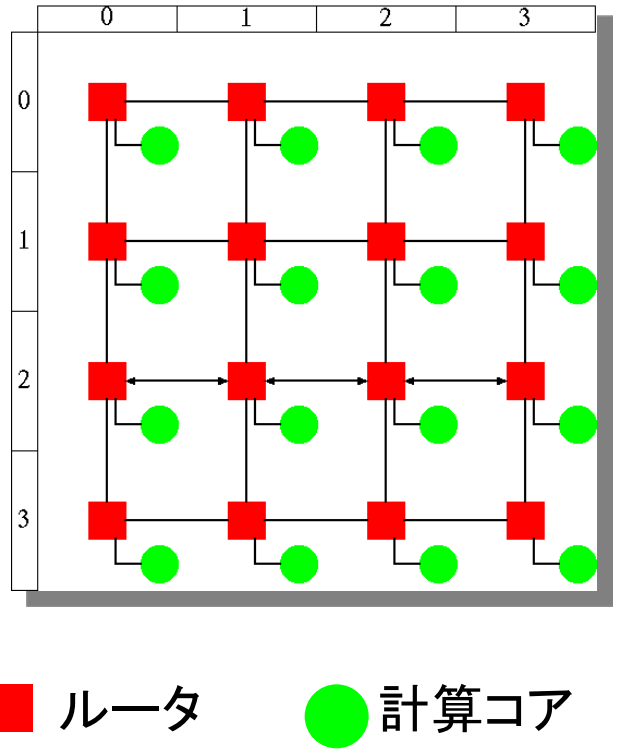


内容

- ネットワークオンチップとは？
- トポロジ
- スイッチング技術とルーティング
- ルータアーキテクチャ
- まとめ

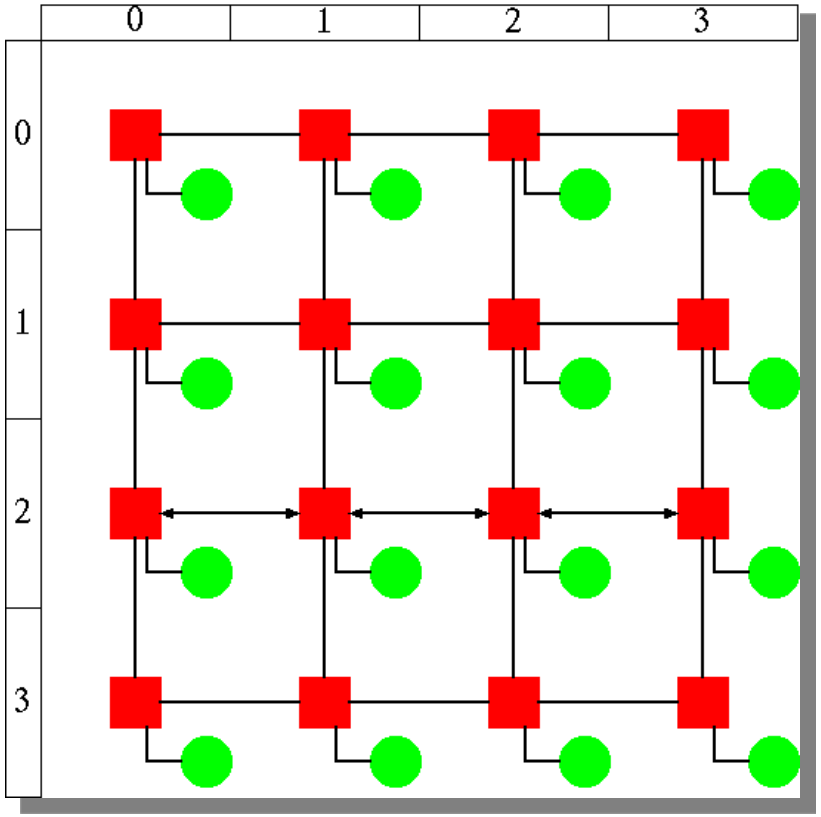
トポロジ

- 規則的な2次元レイアウト
  - メッシュ、トーラス、ツリーが基本
- アプリの通信特性と合わせて総合的に選択
  - bisectionバンド幅
  - スループット
  - 平均ホップ数、直径
  - ルータの個数
  - ハードウェア量
  - 最長・総リンク長
  - 電力、エネルギー

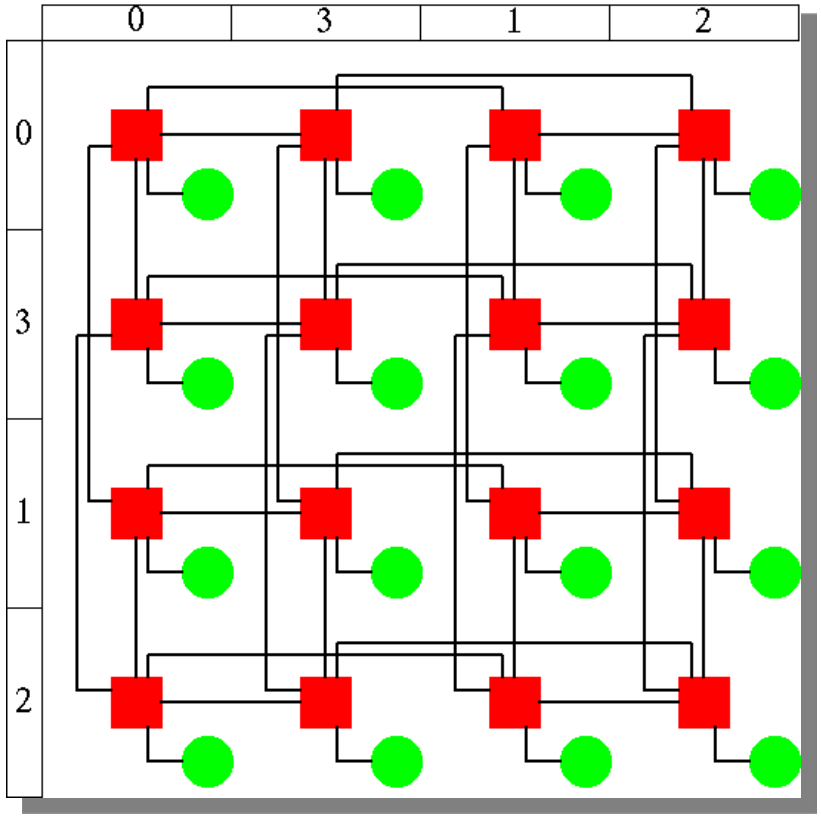


2次元トポロジ: Mesh & Torus

- 2-D Mesh
  - RAW [Taylor, Micro'02]
  - Intel's 80-tile [Vangal, ISSCC'07]



- 2-D Folded Torus [Dally, DAC'01]
  - 16コアの場合、メッシュの2倍の帯域



ルータ　計算コア

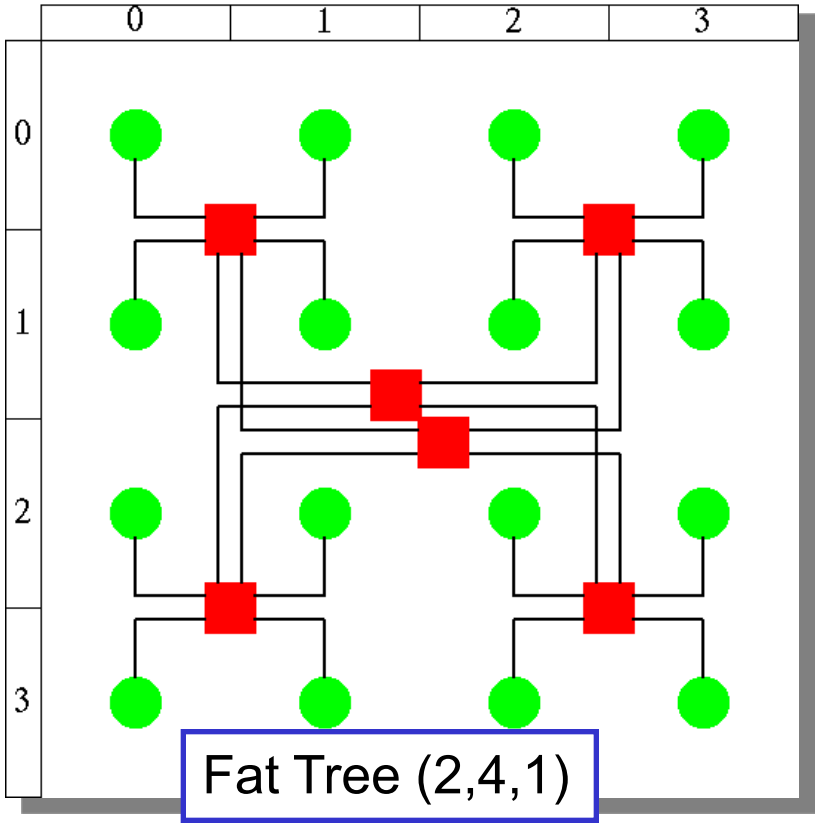
2次元トポロジ: Fat Tree

- Fat Tree (p, q, c)
  - p: 上位リンクの数
  - q: 下位リンクの数
  - c: コアのポート数

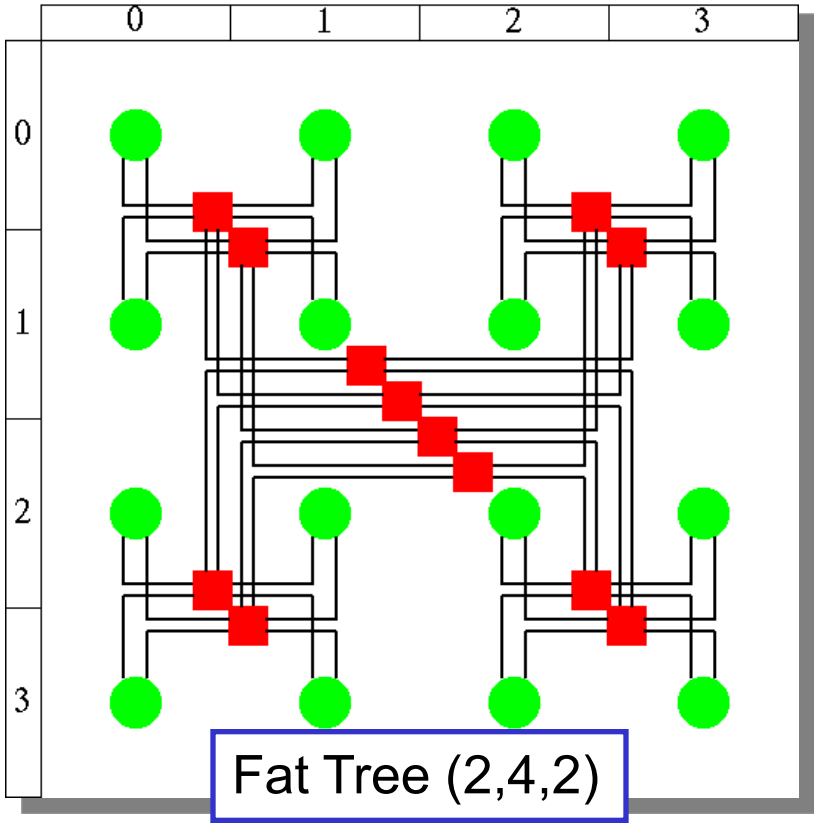
SPIN [Andriahantenaina, DATE'03]

SCORE [Caspi, FPL'00]

ACM [Furtek, FPL'04]



Fat Tree (2,4,1)



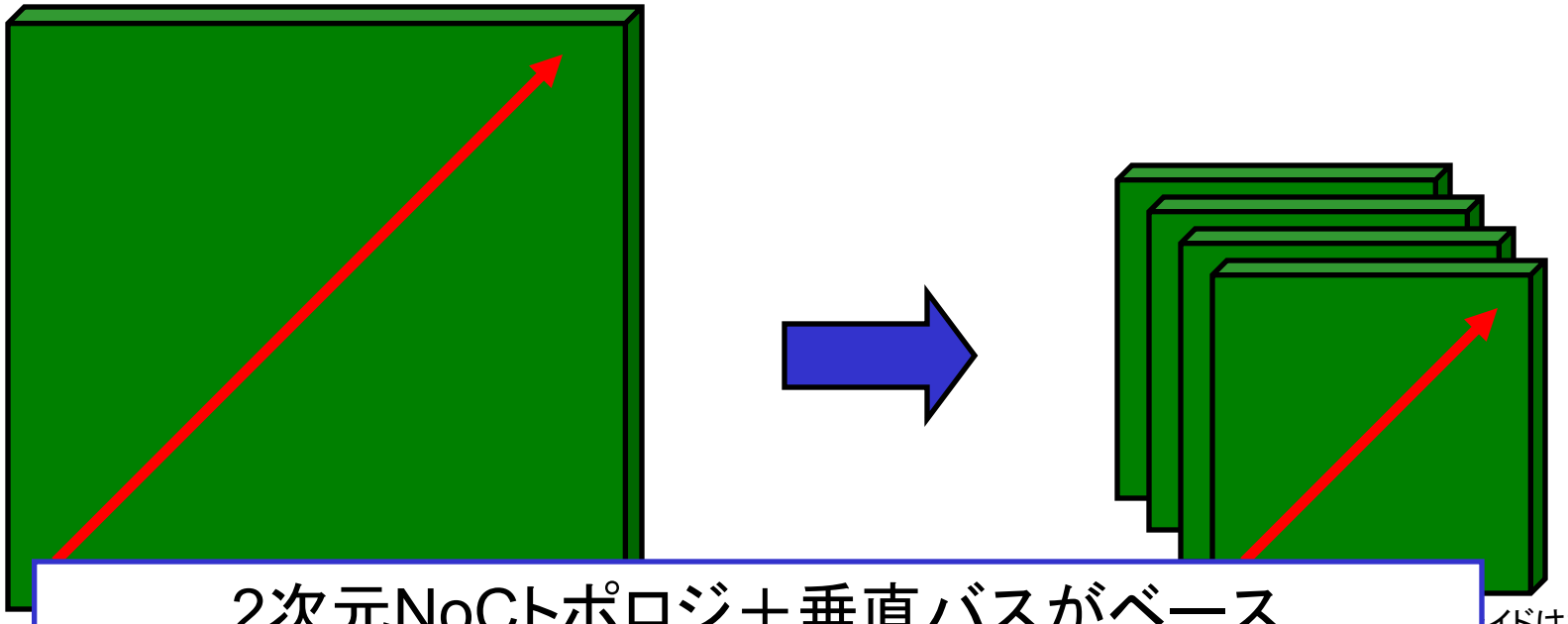
Fat Tree (2,4,2)

ルータ　計算コア

2D NoC vs. 3D NoC

- 2D NoCs
  - Long wires
  - Wire delay
- 3D NoCs
- Vertical link
  - Very short (10-50um)

Long horizontal wires in 2D NoCs can be replaced by very short vertical links in 3D NoCs



2次元NoCトポロジ＋垂直バスがベース

トポロジの展望

- 省電力技術の適用しやすさ
  - Power Gating / DVFS
- 直径、平均ホップ数は、遅延にはさほど影響しないが、パケットの転送電力と比例する。
- 驚くようなトポロジは出尽くした？

既存のNoC

	Topology; Data width	Switching; VCs	Flow control	Routing algorithm
MIT Raw (dynamic network)	2-D mesh; 32-bit	wormhole; no VC	credit based	XY DOR
UPMC/LIP6 SPIN Micro Network	Fat Tree; 32-bit	wormhole; no VC	credit based	up*/down* routing
UMass Amherst aSOC arch	2-D mesh	PCS; no VC	timeslot based	shortest-path
Sun UltraSparc T1 (CPX bus)	crossbar; 128-bit	N/A	handshaking	N/A
Sony, Toshiba, IBM Cell BE EIB	ring; 128-bit	PCS; no VC	credit based	shortest-path
UT Austin TRIPS (operand NW)	2-D mesh; 109-bit	N/A ?; no VC	on/off	YX DOR
UT Austin TRIPS (OCN)	2-D mesh; 128-bit	wormhole; 4VCs	credit based	YX DOR
Intel SCC architecture	2-D torus; 32-bit	wormhole; no VC	stall/go	XY,YX DOR; odd-even TM
Intel Teraflops NoC	2-D mesh; 32-bit	wormhole; 2lane	on/off	source routing (e.g. DOR)
Tilera TILE64 iMesh (dynamic NW)	2-D mesh; 32-bit	wormhole; no VC	credit based	XY DOR

トポロジ：2-Dメッシュが基本

内容

- ネットワークオンチップとは？
- トポロジ
- スイッチング技術とルーティング
- ルータアーキテクチャ
- まとめ

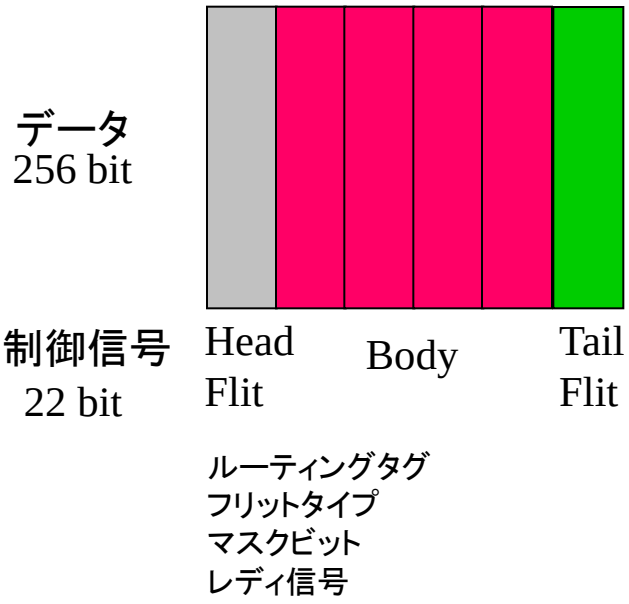
スイッチング技術

複数のルータを経由してパケットを配送

- スイッチング技術
  - ルータにおけるパケット処理をいつ開始するか？
  - Store-and-Forward
    - パケット全体が到着してから
  - ワームホールスイッチング
    - パケットのヘッダが到着し次第
      - カットスルーとコンセプトは一緒

パケット構造

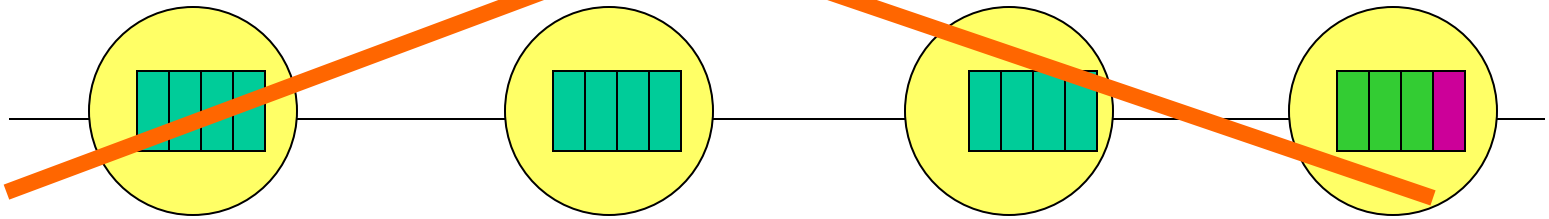
パケットは1単位時間で送信可能なサイズのデータ（フリット）に分割



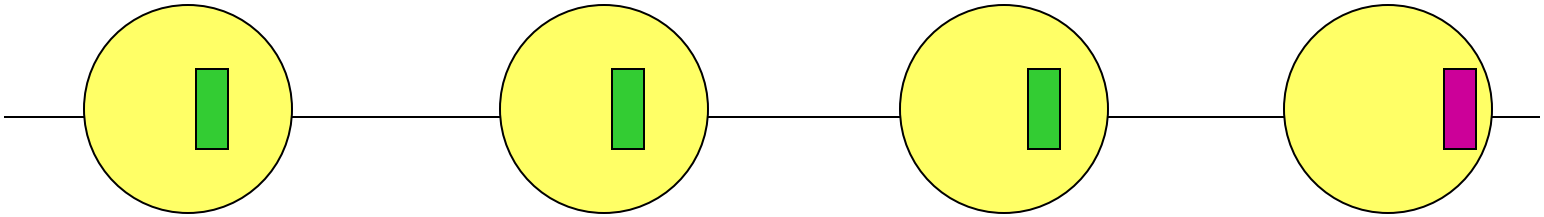
[Dally,2001]

スイッチング技術

- Store-and-forward 方式
  - ソフトウェアによる制御が可能
  - パケット単位の移動 → レイテンシ大



- ワームホール方式
  - 専用のスイッチが必要
  - フリット単位の移動 → レイテンシ小



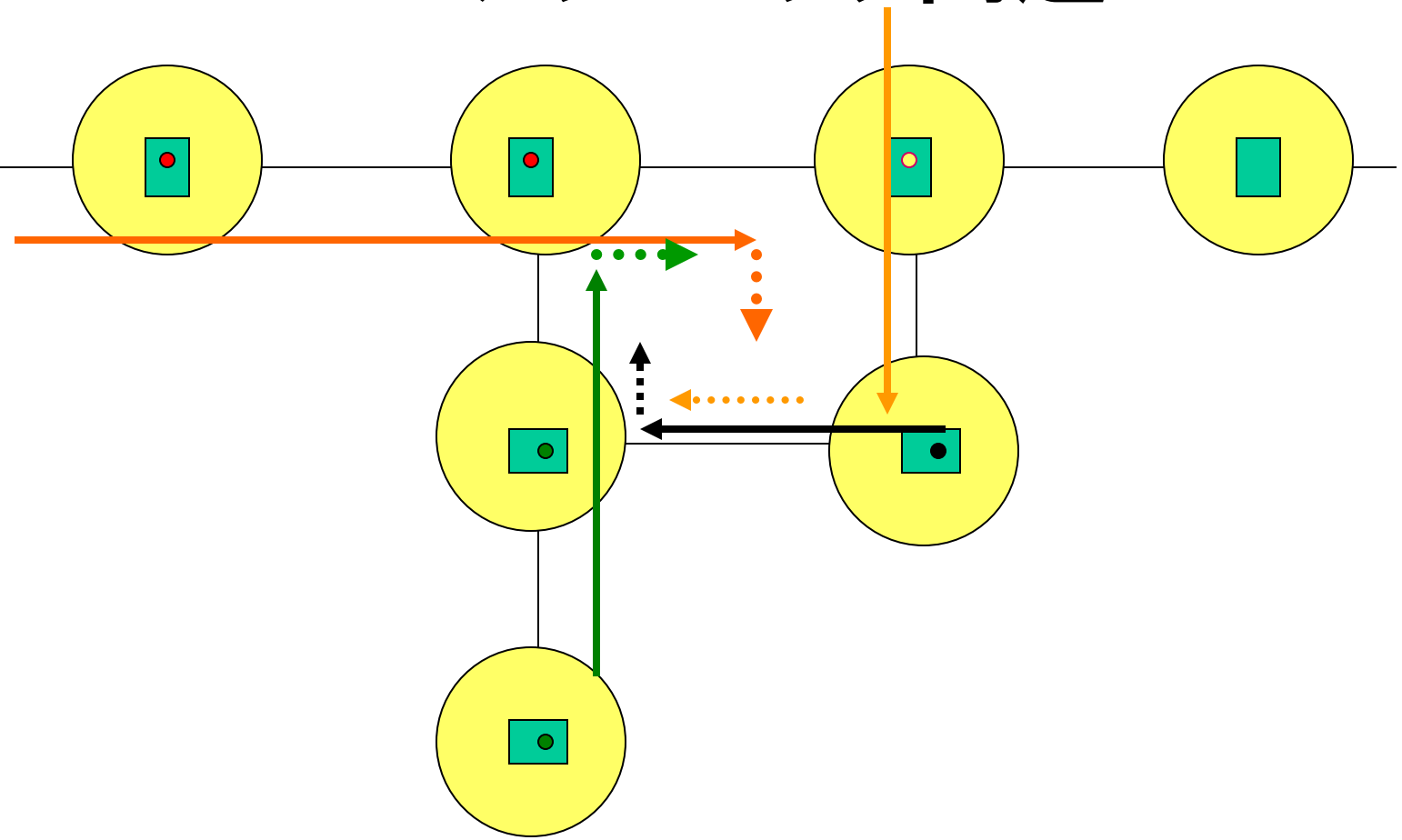
ルータ

既存のNoC

	Topology; Data width	Switching; VCs	Flow control	Routing algorithm
MIT Raw (dynamic network)	2-D mesh; 32-bit	wormhole; no VC	credit based	XY DOR
UPMC/LIP6 SPIN Micro Network	Fat Tree; 32-bit	wormhole; no VC	credit based	up*/down* routing
UMass Amherst aSOC arch	2-D mesh	PCS; no VC	timeslot based	shortest-path
Sun UltraSparc T1 (CPX bus)	crossbar; 128-bit	N/A	handshaking	N/A
Sony, Toshiba, IBM Cell BE EIB	ring; 128-bit	PCS; no VC	credit based	shortest-path
UT Austin TRIPS (operand NW)	2-D mesh; 109-bit	N/A no VC	on/off	YX DOR
UT Austin TRIPS (OCN)	2-D mesh; 128-bit	wormhole; 4VCs	credit based	YX DOR
Intel SCC architecture	2-D torus; 32-bit	wormhole; no VC	stall/go	XY,YX DOR; odd-even TM
Intel Teraflops NoC	2-D mesh; 32-bit	wormhole; 2lane	on/off	source routing (e.g. DOR)
Tilera TILE64 iMesh (dynamic NW)	2-D mesh; 32-bit	wormhole; no VC	credit based	XY DOR

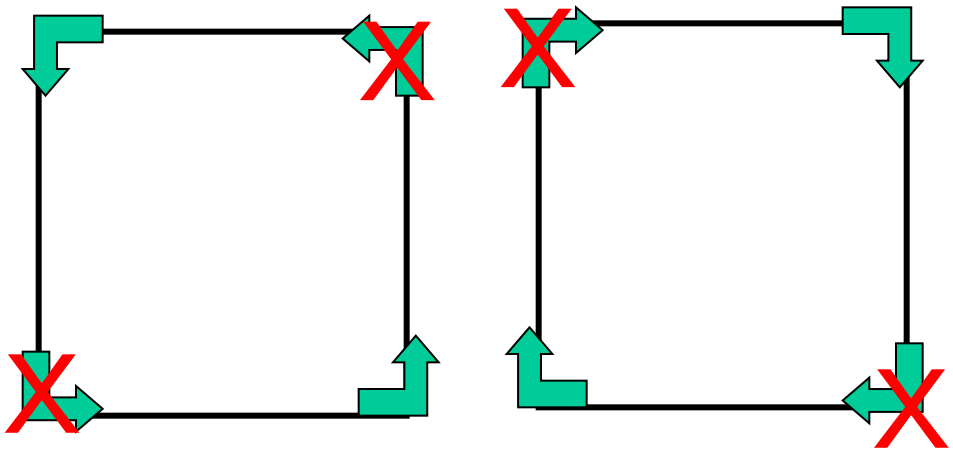
スイッチング技術：ワームホール方式が多数！！

ワームホール方式における デッドロック問題

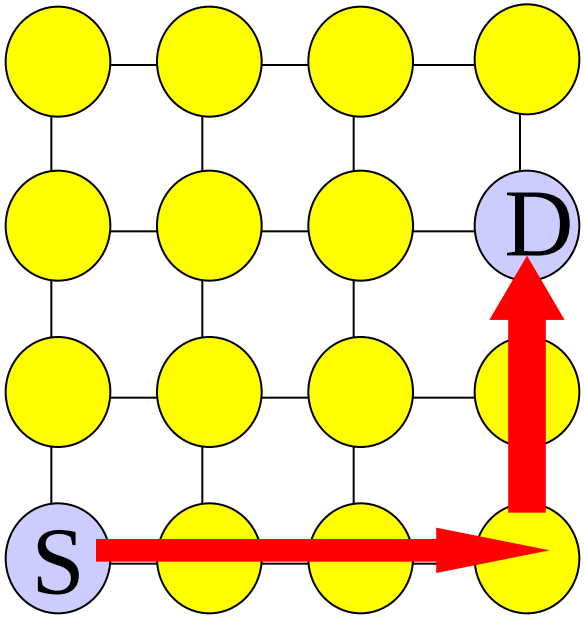


- デッドロックを生じないにルーティングが必要

次元順ルーティング(Dally,1987)

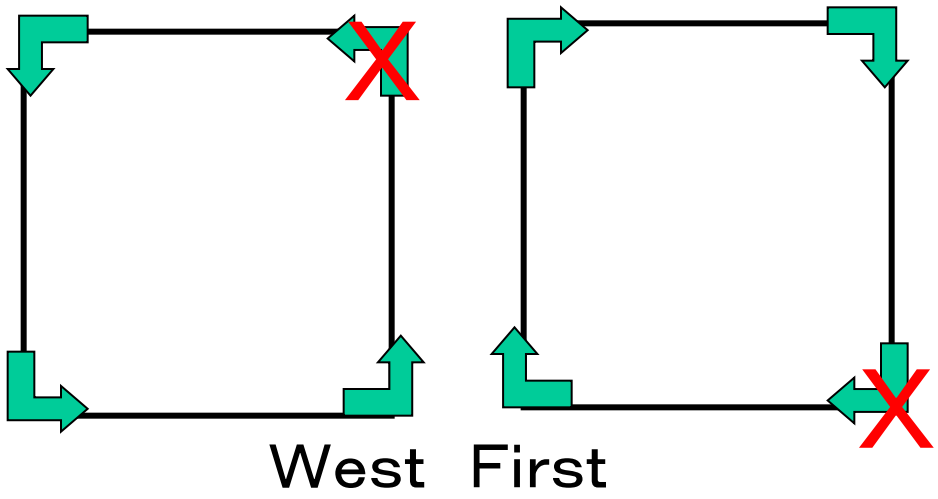


この方角への
パケット転送禁止

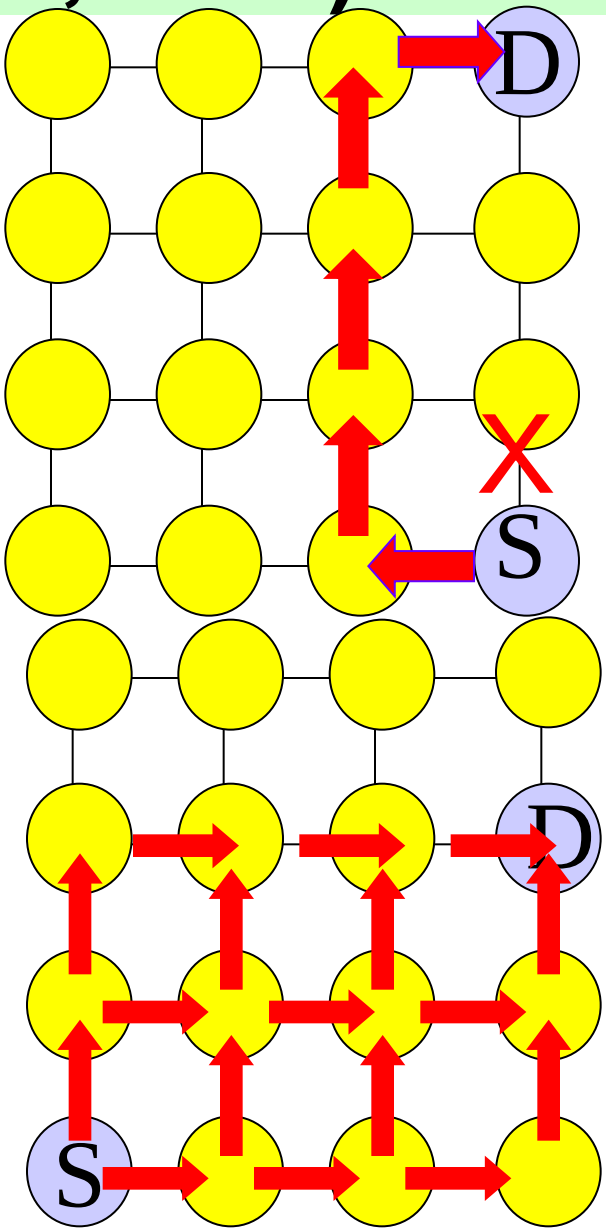


- X方向に必要ホップ数移動した後、Y方向に移動
- 各ノード間に経路1個

Turnモデル(Glassら,1992)



- 高スループット
- 複数経路
- 故障個所を迂回可能



ルーティング技術の展望

並列計算機のルーティングから発展

- 最適化問題
  - トラヒックパターンを事前解析
  - DVFS ＋低エネルギールーティング
  - ダークシリコン/パワーゲーティング用ルーティング
- 直観的なアプローチが多い

既存のNoC

	Topology; Data width	Switching; VCs	Flow control	Routing algorithm
MIT Raw (dynamic network)	2-D mesh; 32-bit	wormhole; no VC	credit based	XY DOR
UPMC/LIP6 SPIN Micro Network	Fat Tree; 32-bit	wormhole; no VC	credit based	up*/down* routing
UMass Amherst aSOC arch	2-D mesh	PCS; no VC	timeslot based	shortest-path
Sun UltraSparc T1 (CPX bus)	crossbar; 128-bit	N/A	handshaking	N/A
Sony, Toshiba, IBM Cell BE EIB	ring; 128-bit	PCS; no VC	credit based	shortest-path
UT Austin TRIPS (operand NW)	2-D mesh; 109-bit	N/A ?; no VC	on/off	YX DOR
UT Austin TRIPS (OCN)	2-D mesh; 128-bit	wormhole; 4VCs	credit based	YX DOR
Intel SCC architecture	2-D torus; 32-bit	wormhole; no VC	stall/go	XY,YX DOR; odd-even TM
Intel Teraflops NoC	2-D mesh; 32-bit	wormhole; 2lane	on/off	source routing (e.g. DOR)
Tilera TILE64 iMesh (dynamic NW)	2-D mesh; 32-bit	wormhole; no VC	credit based	XY DOR

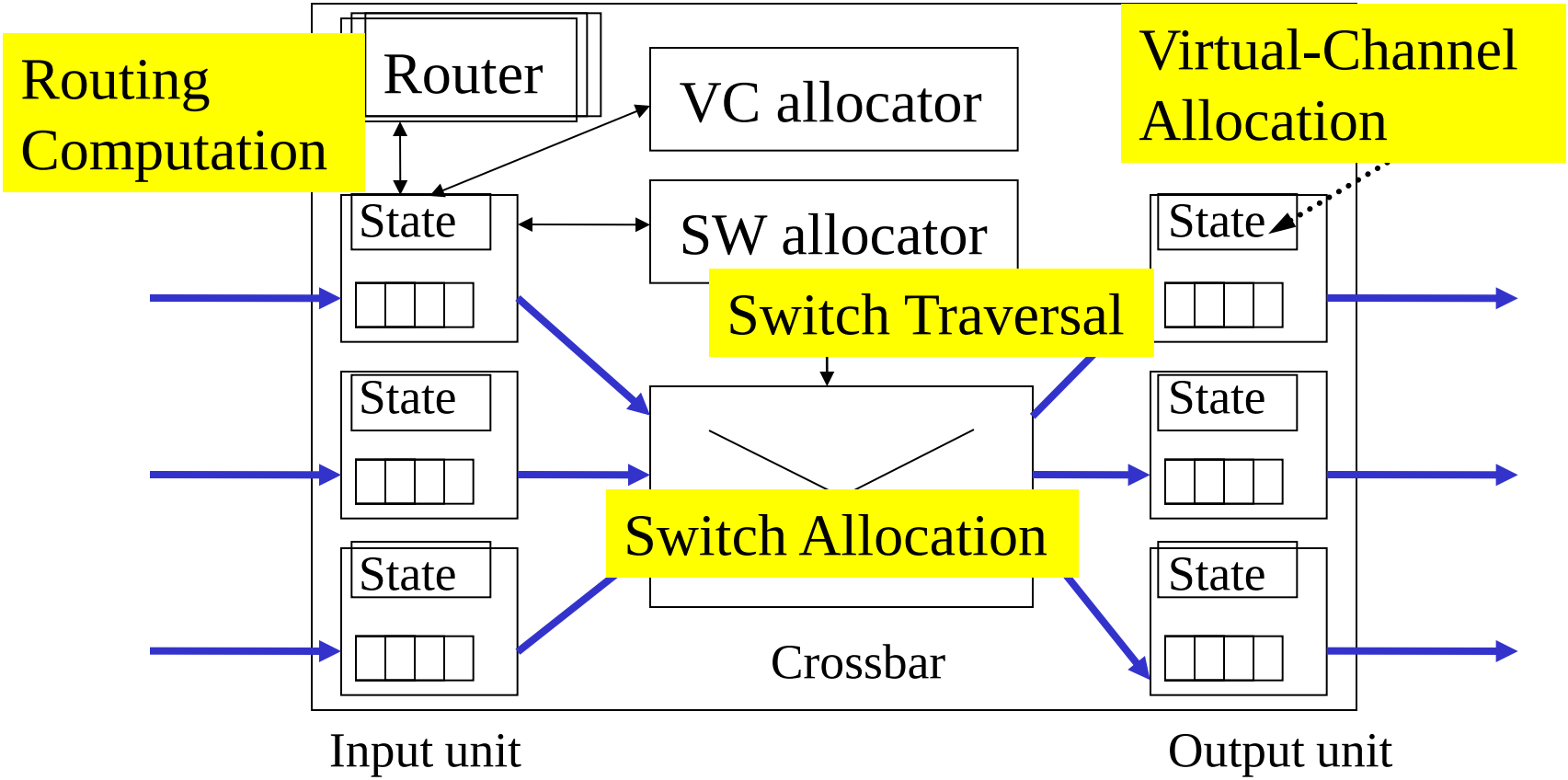
トポロジ：2-Dメッシュ　スイッチング技術：ワームホール方式（VC無）

ルーティング：次元順ルーティング(DOR)が多数！！

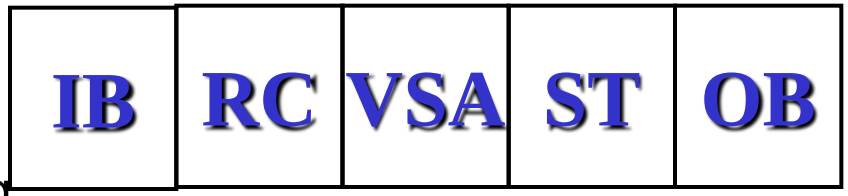
内容

- ネットワークオンチップとは？
- トポロジ
- スイッチング技術とルーティング
- ルータアーキテクチャ
- まとめ

NoCルータの動作



パイプライン処理



IB: Input buffering

時間

実際には各ステージを更に細分化し、数段のパイプライン処理

ルータアーキテクチャの展望

驚くような技術革新が起きた

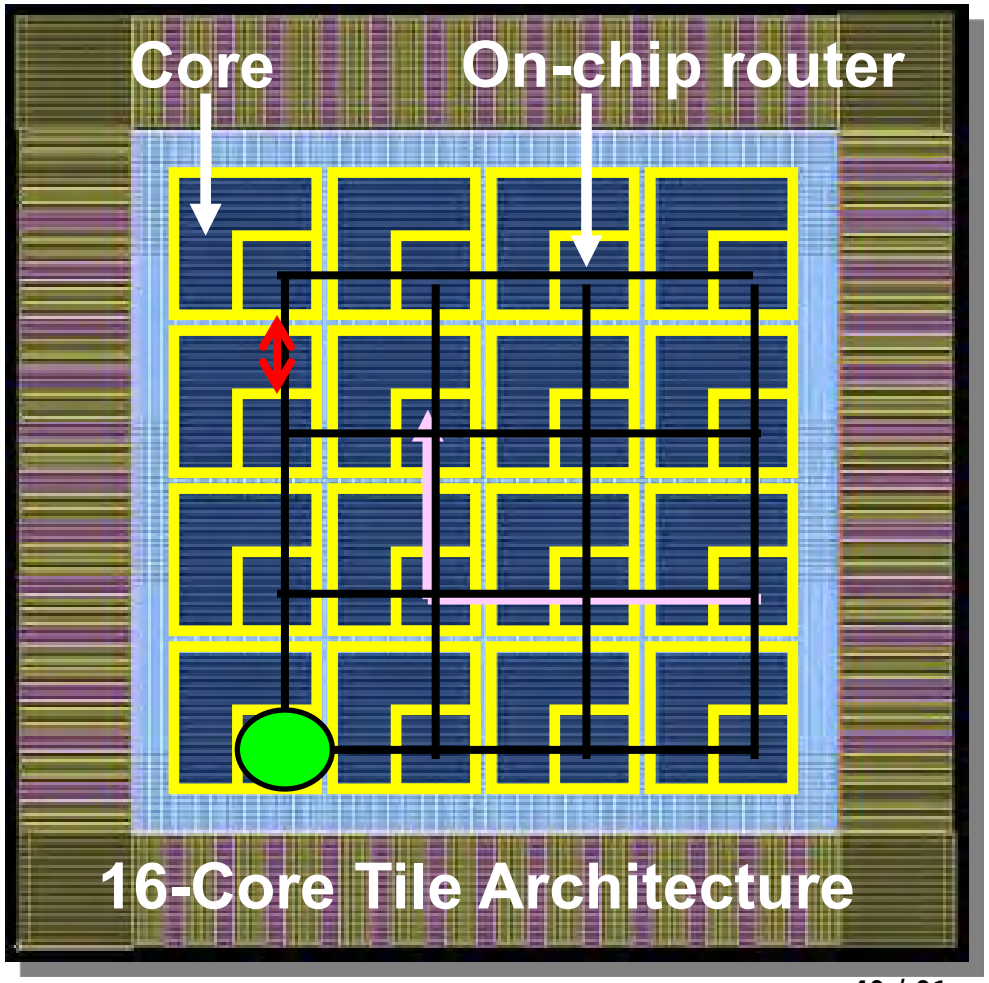
プロセッサのパイプライン技術の応用が多い

- 省電力化
  - Power Gating / DVFS ルータ
- 低遅延化
  - Speculative ルータ
  - 予測ルータ

まとめ

メニーコアNoCは以下を基準に、要求に応じた最適設計

- トポロジ
  - 2次元メッシュ
- スイッチング技術
  - ワームホール方式
- ルーティング
  - 次元順ルーティング
- ルータアーキテクチャ
  - 4サイクルルータ



既存のチップ内ネットワーク(おさらい)

	Topology; Data width	Switching; VCs	Routing algorithm
MIT Raw (dynamic network)	2-D mesh; 32-bit	wormhole; no VC	XY DOR
UPMC/LIP6 SPIN Micro Network	Fat Tree; 32-bit	wormhole; no VC	up*/down* routing
UMass Amherst aSOC arch	2-D mesh	PCS; no VC	shortest-path
Sun UltraSparc T1 (CPX bus)	crossbar; 128-bit	N/A	N/A
Sony, Toshiba, IBM Cell BE EIB	ring; 128-bit	PCS; no VC	shortest-path
UT Austin TRIPS (operand NW)	2-D mesh; 109-bit	N/A ナ; no VC	YX DOR
UT Austin TRIPS (OCN)	2-D mesh; 128-bit	wormhole; 4VCs	YX DOR
Intel SCC architecture	2-D torus; 32-bit	wormhole; no VC	XY,YX DOR; odd-even TM
Intel Teraflops NoC	2-D mesh; 32-bit	wormhole; 2lane	source routing (e.g. DOR)
Tilera TILE64 iMesh (dynamic NW)	2-D mesh; 32-bit	wormhole; no VC	XY DOR

+R 未来を生みだす人になる。
立命館大学
RITSUMEIKAN

メニーコア/ネットワークオンチップの基礎と組込みシステムへの応用（後編）

～SMYLEプロジェクトを例に～

冨山宏之

立命館大学　理工学部
http://www-ja.tomiyama-lab.org/

SWEST15（2013年8月23日）

謝辞

- SMYLEプロジェクトは、NEDOからの支援により実施されました。
- 以下の方々にスライドを提供して頂きました。
  - 井上 こうじ　（九州大学）
  - 近藤 正章　（電気通信大学）

講演の流れ

- SMYLEプロジェクトの概要
- SMYLErefアーキテクチャ
- SMYLE OpenCL

SMYLEプロジェクトの概要

SMYLEプロジェクト

～Scalable ManY-core for Low-Energy computing～

- NEDOグリーンITプロジェクト
  「低消費電力メニーコア用アーキテクチャとコンパイラ技術」
- 研究期間2010年12月～2013年2月
- 実施者
  - 九州大学
  - 立命館大学
  - 電気通信大学
  - 株式会社フィックスターズ
  - 株式会社トプスシステムズ
- 再委託先
  - 東京農工大学
  - 電子情報技術産業協会
  - イーソル株式会社
  - キャッツ株式会社

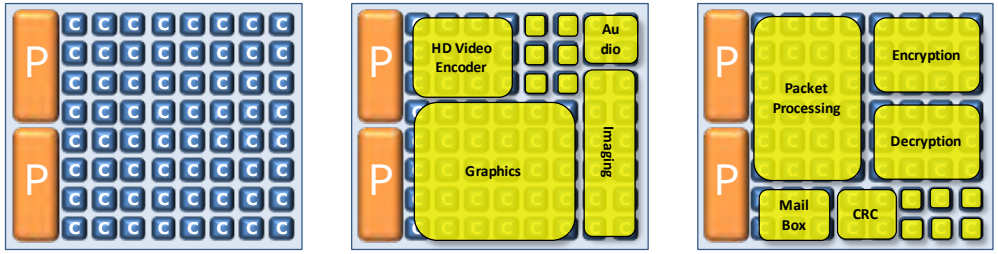
分担とゴール

	九大＋立命館＋電通大＋農工大	トプスシステムズ	フィックスターズ	JEITA+eSOL+CATS
どのようなメニーコアを作れば良いのか?	メニーコアSoCとその実行環境	ビデオマイニング向けメニーコア		市場調査
どのようにソフトウェアを開発すれば良いのか?		ビデオマイニング用ソフトウェア	プログラム開発環境	動作環境／開発環境調査
どのようにメニーコア研究開発を進めれば良いのか?	FPGA評価環境や機能シミュレータ		アクセラレータ用ベンチマーク	プロジェクト提案

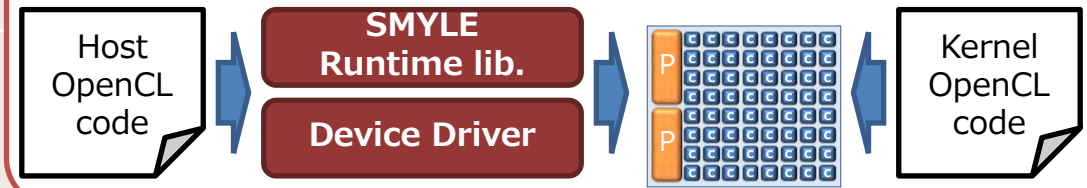
分担とゴール

	九大＋立命館＋電通大＋農工大
どのようなメニーコアを作れば良いのか?	メニーコアSoCとその実行環境
どのようにソフトウェアを開発すれば良いのか?	
どのようにメニーコア研究開発を進めれば良いのか?	FPGA評価環境や機能シミュレータ

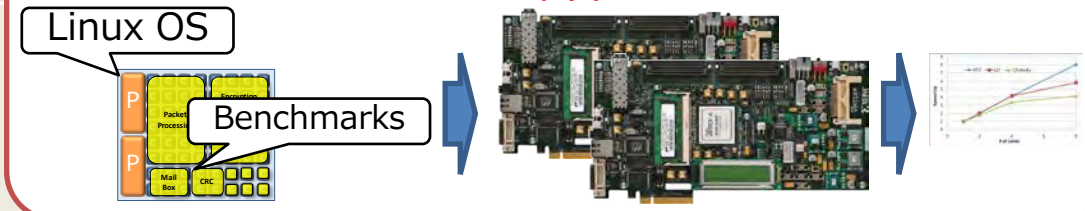
SMYLEref: 専用HWをメニーコア（SW処理）で置換え→汎用SoCの実現を目指す!



SMYLEref用OpenCL実行環境を構築する!



FPGAを用いた性能評価環境を構築する!



(可能な限り)設計データやツール群を無償で公開

分担とゴール

	九大＋立命館＋電通大＋農工大	トプスシステムズ
どのようなメニーコアを作れば良いのか?	メニーコアSoCとその実行環境	ビデオマイニング向けメニーコア
どのようにソフトウェアを開発すれば良いのか?		ビデオマイニング用ソフトウェア
どのようにメニーコア研究開発を進めれば良いのか?	FPGA評価環境や機能シミュレータ	

SMYLEvideo: ビデオマイニング向けメニーコアを開発する!

評価ボード開発：SIFTのSWリアルタイム処理を実現する!

評価ボードデモを開発しIPコアビジネスへと展開

分担とゴール

CLtrump: 並列化コード作成支援ソフトウェアを開発する!

PEMAP: 移植後の性能見積もり支援ソフトウェアを開発する!

BEMAP: OpenCL Benchmark（4分野8種）を開発する！

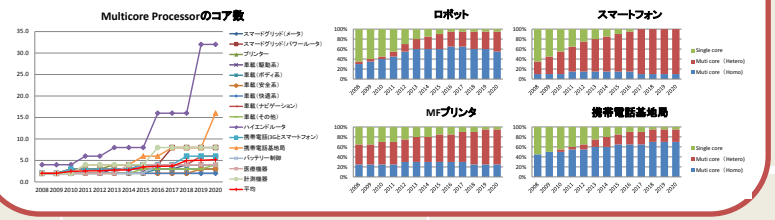
	フィックスターズ	JEITA+eSOL+CATS
どのようなメニーコアを作れば良いのか?		市場調査
どのようにソフトウェアを開発すれば良いのか?	プログラム開発環境	動作環境／開発環境調査
どのようにメニーコア研究開発を進めれば良いのか?	アクセラレータ用ベンチマーク	プロジェクト提案

ベンチマークやツール群を無償で公開

分担とゴール

		JEITA+eSOL+CATS
どのようなメニーコアを作れば良いのか?	九大＋立命館＋電通大＋農工大	市場調査
どのようにソフトウェアを開発すれば良いのか?	メニーコアSoC＋その実行環境	動作環境／開発環境調査
どのようにメニーコア研究開発を進めれば良いのか?	FPGA評価環境＋機能シミュレータ	プロジェクト提案

マルチ／メニーコア市場を予測する!



メニーコアに関する動作環境やシステム開発環境を調査する!

日本の半導体産業の競争力強化に繋がる提言をまとめる!

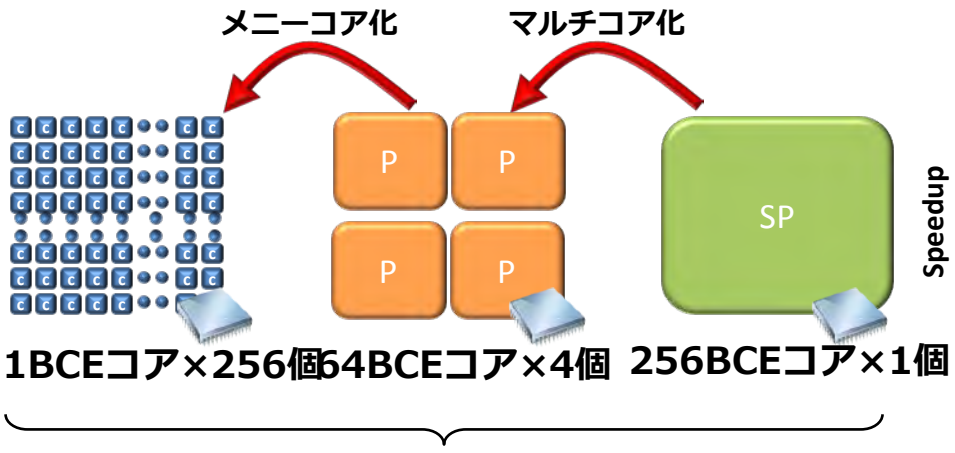
調査結果や検討結果を公開

SMYLErefアーキテクチャ

メニーコアの本質とは?

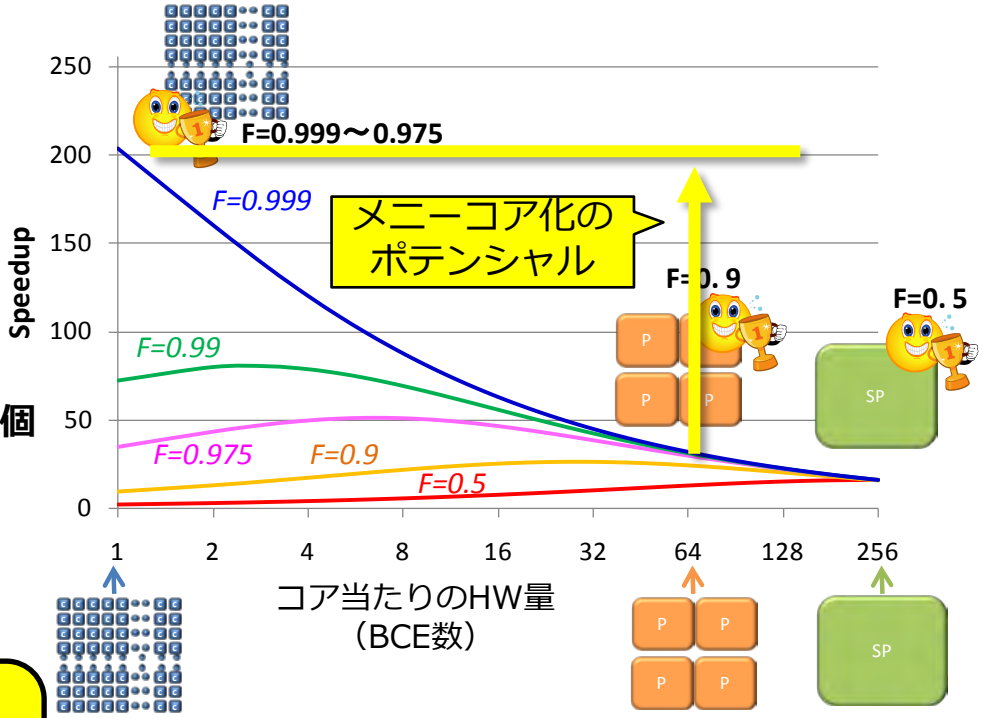
低性能，小面積，低消費電力なコアを大量に搭載

オンチップ並列処理により劇的な性能向上を実現



ハードウェア量は同じ

1BCE（Base Core Equivalent）は最小コア「c」のHW量



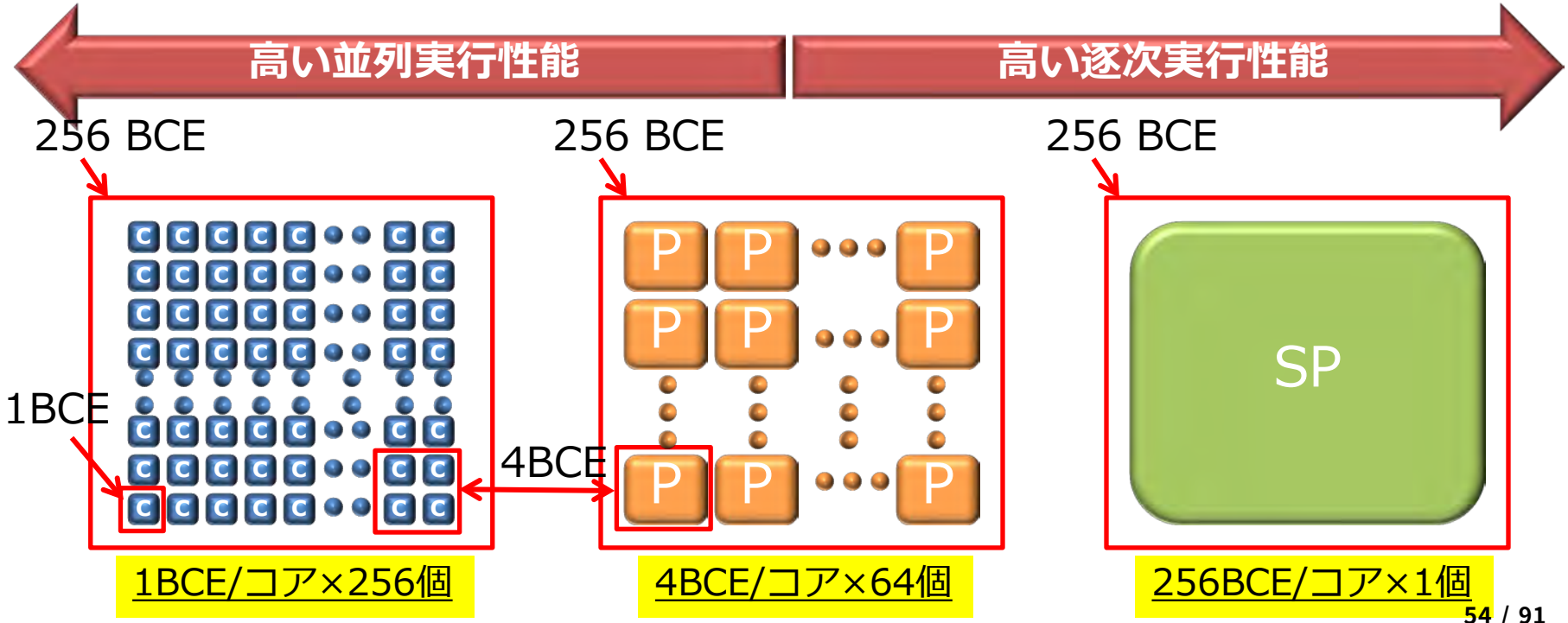
- ポラックの法則：R-BCEコアの逐次性能比＝Rの平方根
- メモリによる影響等は無視
- アムダールの法則に基づく性能見積り（Hill, HPCA'08 ）

「並列処理」を手段として
大量トランジスタを徹底活用!

Mark D. Hill, HPCA'08 Keynote, http://pages.cs.wisc.edu/~markhill/includes/publications.html#year2008

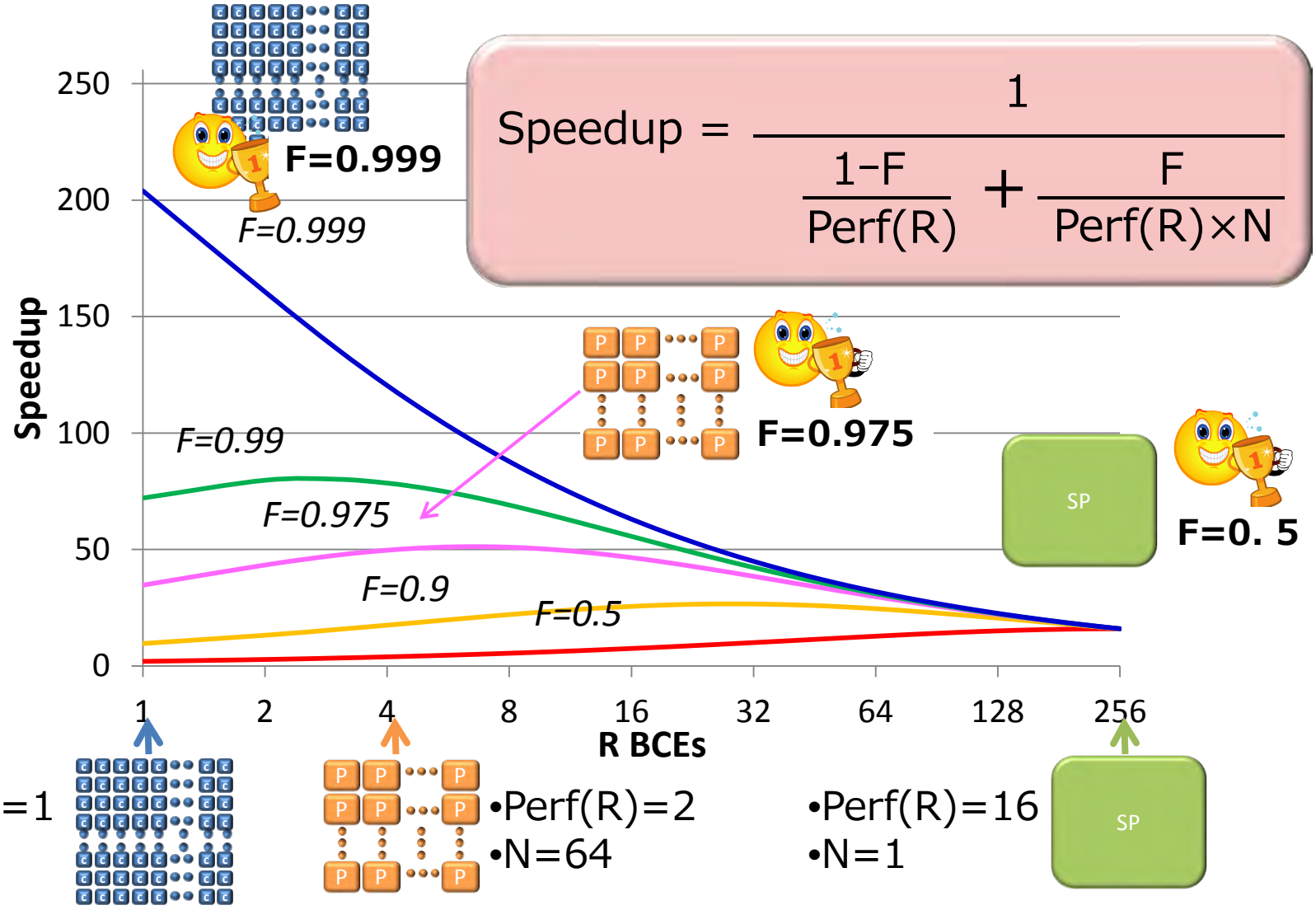
Many 対 Multi 対 Single

- 1BCE（Base Core Equivalent）は最小コア「c」の実装に要するHW量
- R-BCEでの逐次実行性能＝Perf(R)＝square root(R)
- メモリによる影響等は無視（理想状態を想定）



Mark D. Hill, HPCA'08 Keynote, http://pages.cs.wisc.edu/~markhill/includes/publications.html#year2008

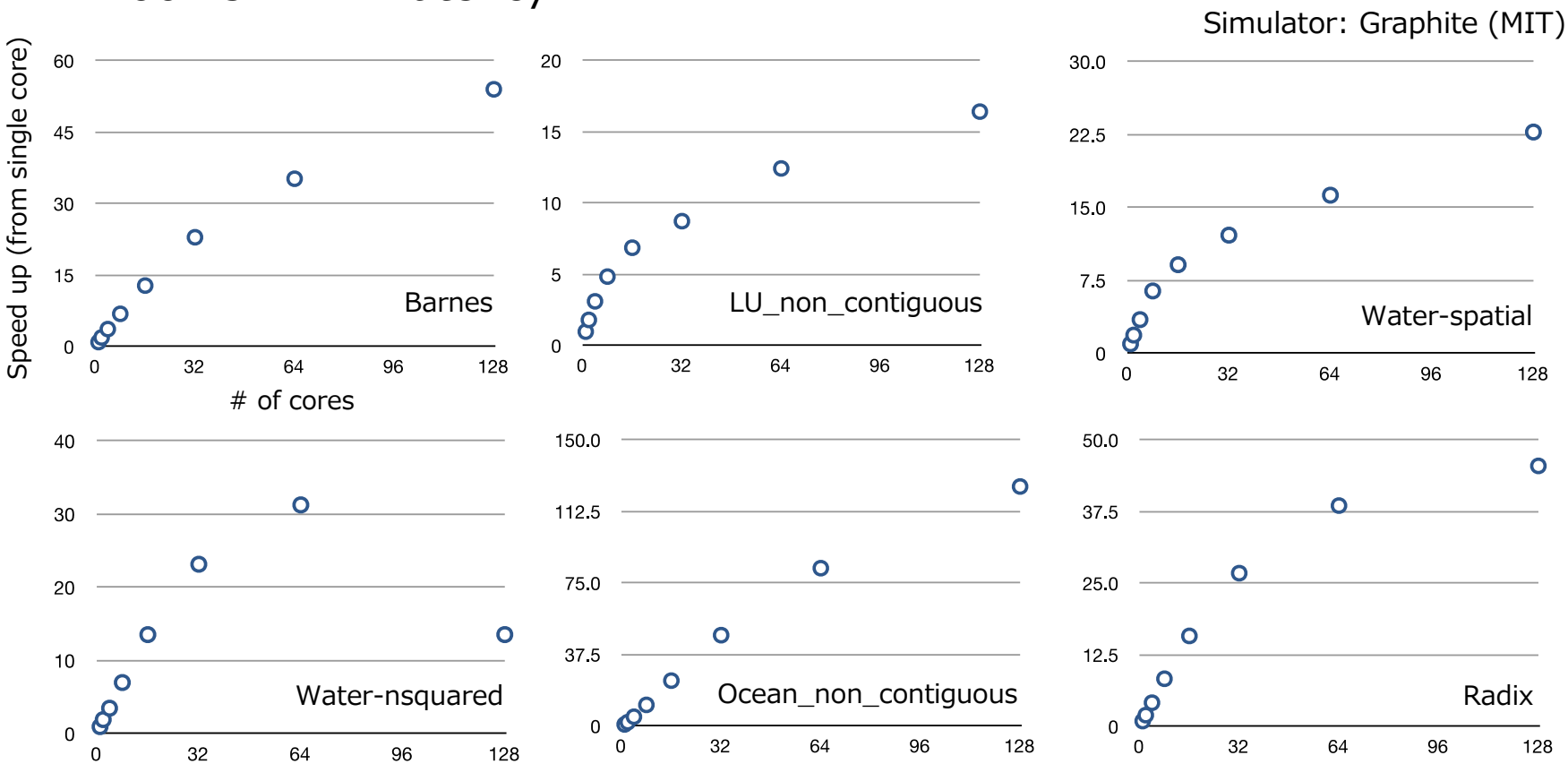
Many 対 Multi 対 Single



Mark D. Hill, HPCA'08 Keynote, http://pages.cs.wisc.edu/~markhill/includes/publications.html#year2008

単一アプリ性能はスケールするのか?
～科学技術計算アプリでさえも・・・～

- In-Order Core@1GHz (1～128コア)　w/ private 32KB L1 & 512KB L2
- 2D Mesh NoC (no contention)
- 100 ns DRAM latency



ベンチマーク：SPLASH-2

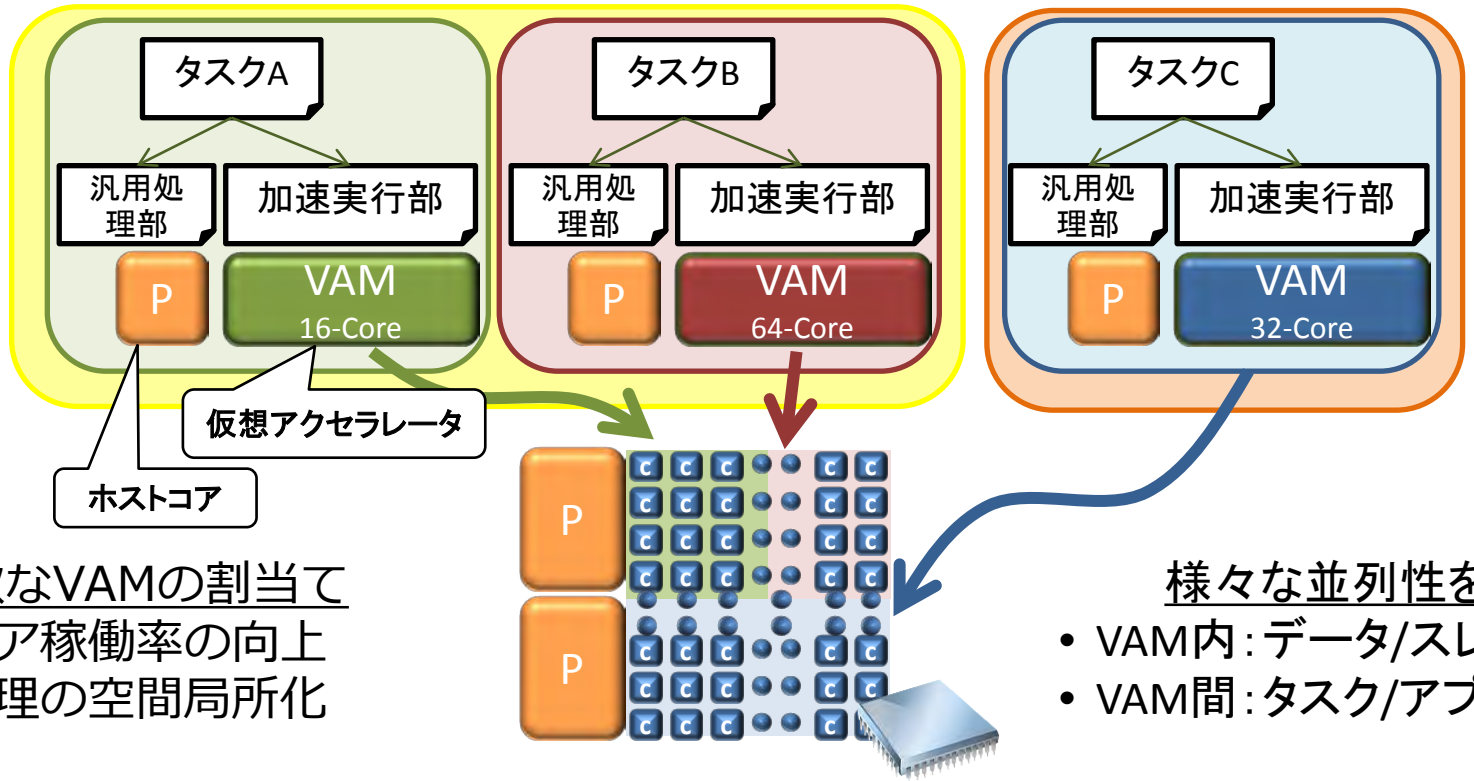
メニーコアプロセッサの課題

- スケーラビリティが低い
  - アプリケーションプログラムが持つ並列性の不足
  - メモリアクセスのボトルネック
  - バリア同期のオーバーヘッド

- 組み込みシステム向けメニーコア・プロセッサの要件
  - 効率的な並列処理の実現による高性能・低消費電力化
  - 様々な並列性（データ / スレッド / タスク / アプリレベル）の活用
  - メモリアクセスボトルネックを緩和するキャッシュメモリ制御
  - 高速バリア同期のサポート

SMYLErefの基本コンセプト

- 柔軟な仮想アクセラレータ実行プラットフォーム：VAM
  - VAM: Virtual Accelerator on Many-core
  - 小規模コアを多数用いてメニーコアを構成



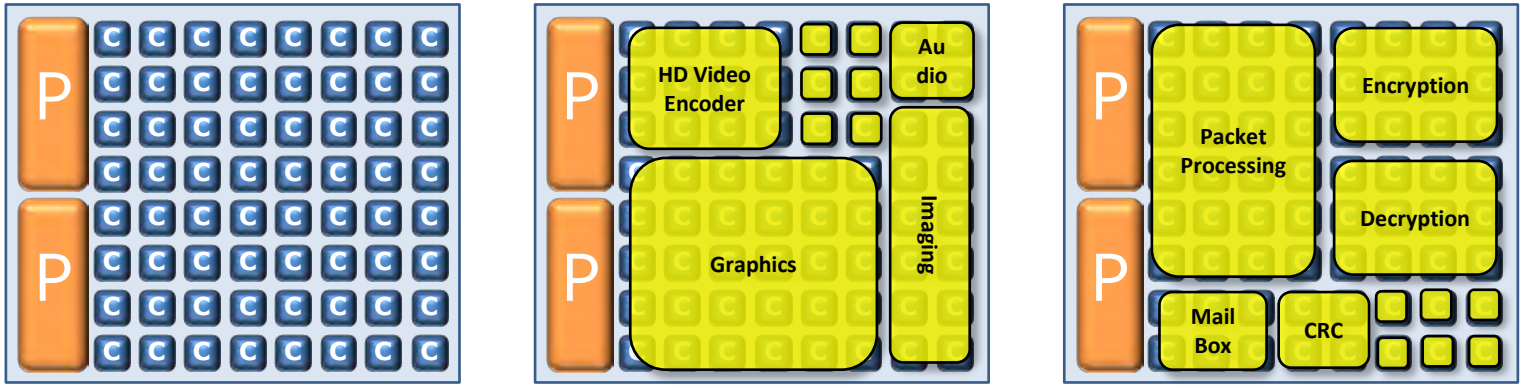
柔軟なVAMの割当て
- コア稼働率の向上
- 処理の空間局所化

様々な並列性を活用
- VAM内：データ/スレッド並列
- VAM間：タスク/アプリ並列

SMYLErefの狙い

～マルチスレッド・マルチタスク(アプリ)実行でスケーラビリティを確保～

- 汎用性重視型メニーコア
  - マルチコア性能<<メニーコア性能<専用エンジン性能
  - 様々なアプリケーションを「比較的」効率良く実行
  - SoCに搭載された複数専用エンジンをメニーコア(SW処理)で置換え

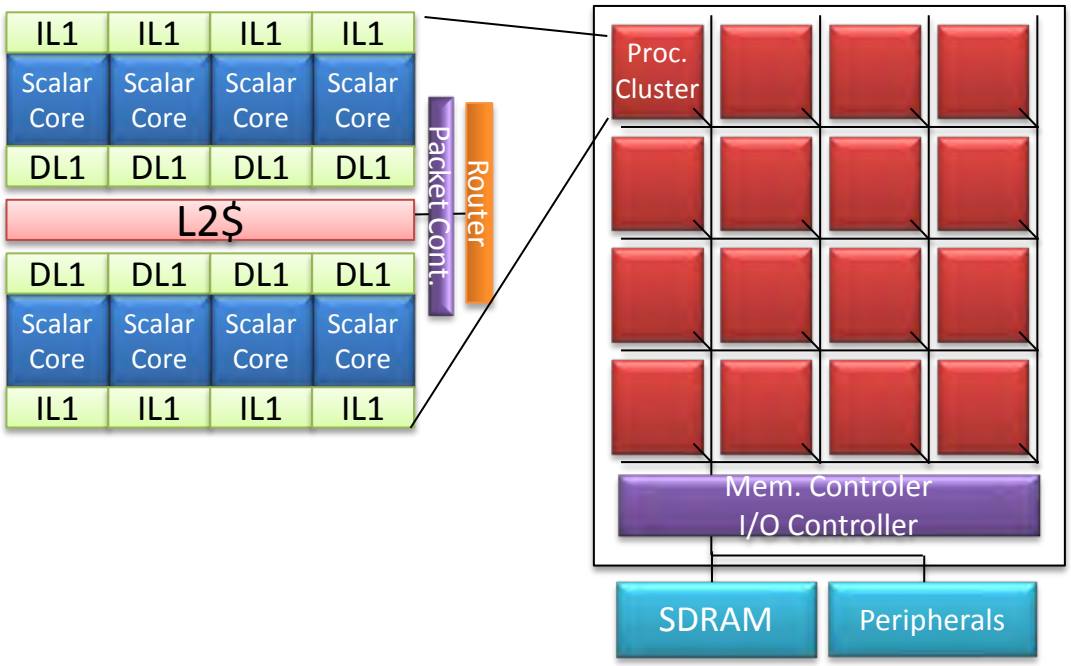


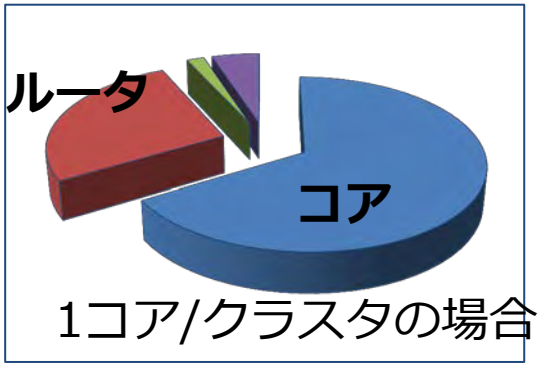
•メニーコア・ドメインはホモジニアス構成
•再構成可能性によりヘテロジニティを実現（例：メモリ構成）
•データ/スレッド並列処理＋タスク並列処理＋アプリ並列処理

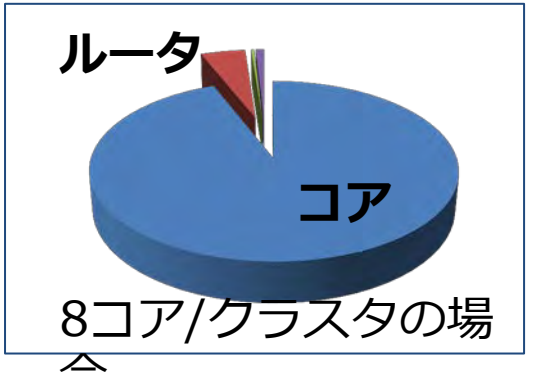
メニーコアアーキテクチャSMYLEref

- SMYLEref: VAM実装のリファレンスアーキテクチャ
  - 数個のプロセッサコアをバスで結合したクラスタ構成
  - クラスタを2次元メッシュのオンチップネットワーク(NoC)で結合

→ クラスタ化によりルータのコスト減（より多くのトランジスタを演算に利用）

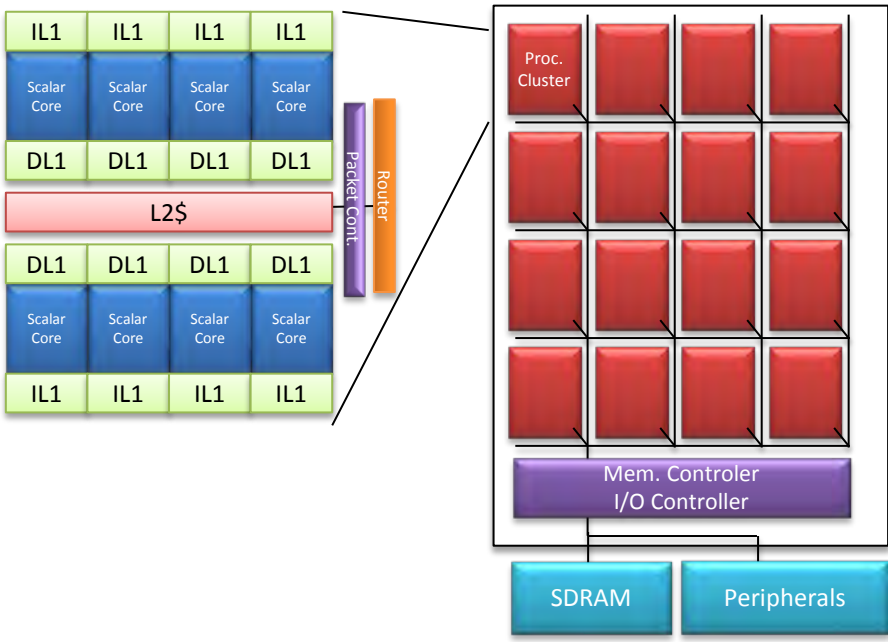






*SMYLEref*の構成

- プロセッサコア
  - 「革新的電源制御による次世代超低電力高性能システムLSIの研究(代表：東大 中村宏教授)」で開発されたGeyserコア
  - MIPS R3000ベース、LSI実装の実績あり
- クラスタの構成
  - Processor Cluster
    - 複数個のコア
    - 分散共有L2$
    - ルータがCluster-Busを通して接続
  - Peripheral Cluster
    - DRAMコントローラ
    - I/Oコントローラ

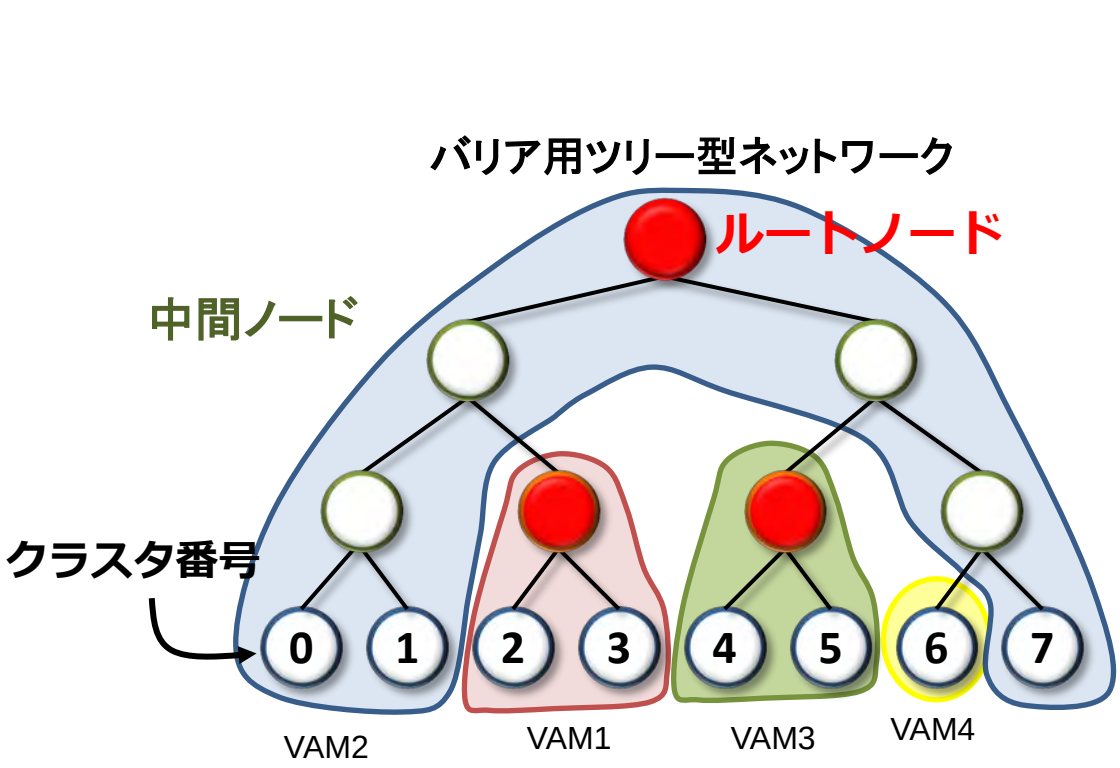


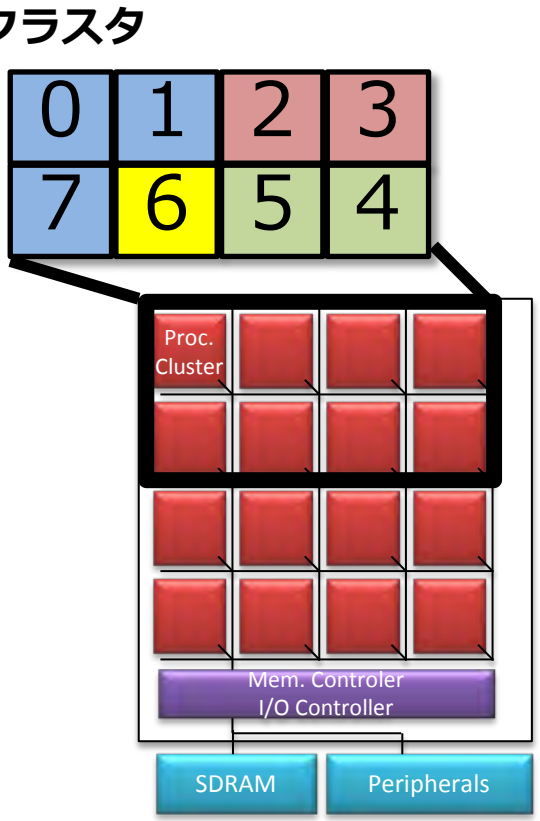
VAM向けの拡張

- 再構成可能L1キャッシュ
  - コア毎にL1キャッシュメモリをキャッシュとして、もしくはSPMとして利用
  - VAMの特性に応じてコンパイラにより決定
- 分散共有L2キャッシュの拡張
  - 基本構成：L2キャッシュは全クラスタで共有
  - 拡張構成：VAM毎にL2キャッシュグループを構成
    - ソフトウェアで設定するアドレス変換機構によりL2キャッシュ位置指定
    - VAM間でのL2キャッシュ競合の回避
- グループ・ハードウェアバリア
  - 各VAMに割り当てられたコアグループを対象にした高速ハードウェアバリアをサポート

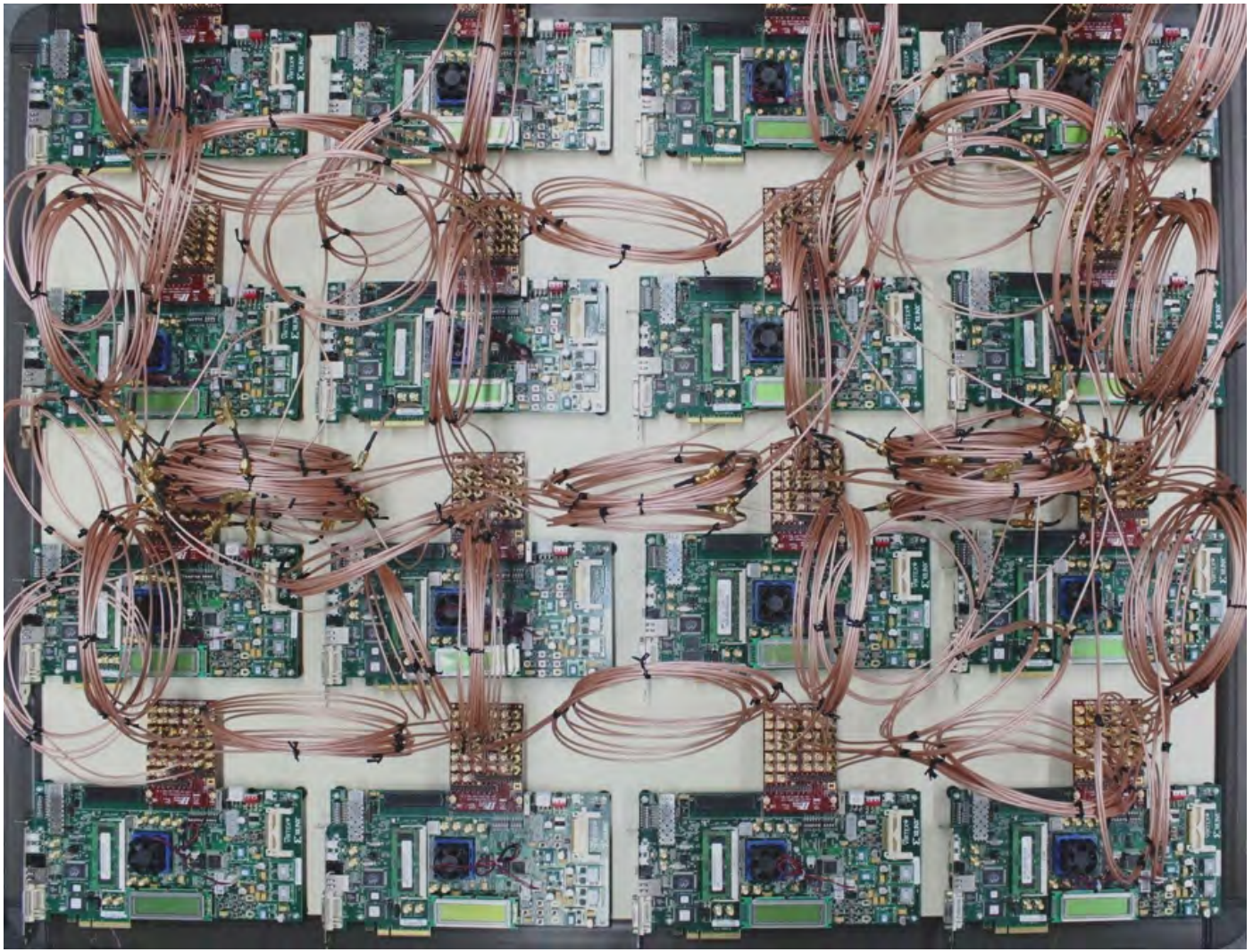
グループハードウェアバリア

- バリア同期専用のツリー型ネットワーク
  - VAM内での高速バリア同期の実現
  - 複数VAM間でバリア同期が可能





評価環境の外観



MPEGデコードのデモ

- SMYLEref評価環境でのMPEG2デコードのデモ
  - MediaBenchのmpeg2decをpthreadで並列化
  - 8cores x 2-clusters, コア周波数は40MHz

1 core　　16 cores

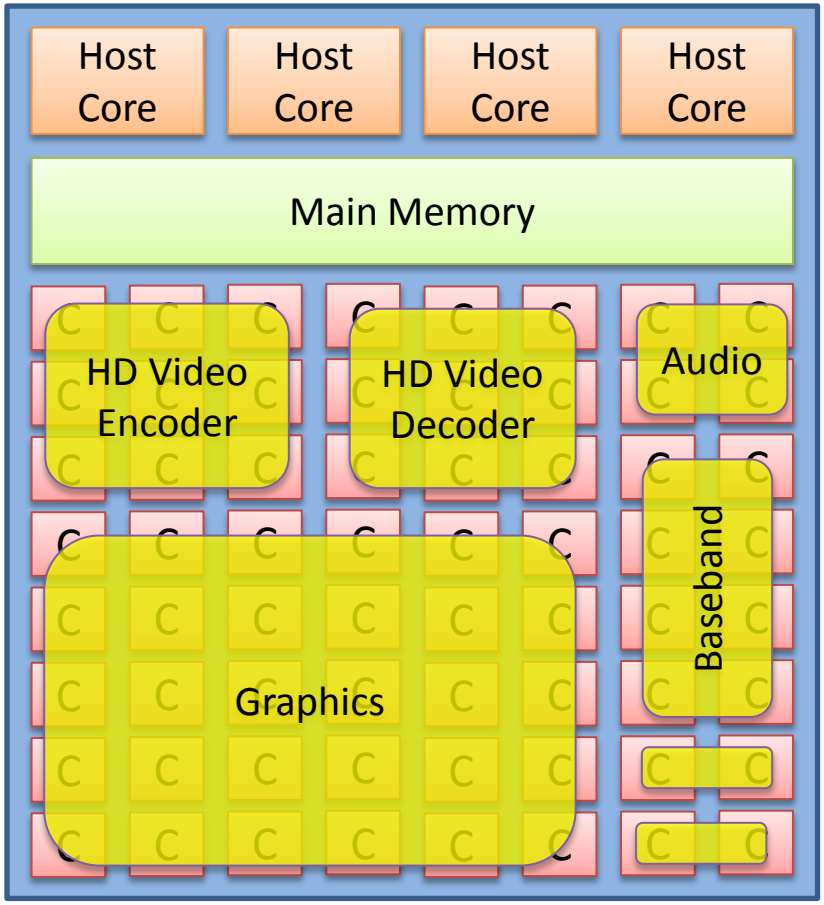
SMYLE OpenCL環境

SMYLE OpenCLの目的

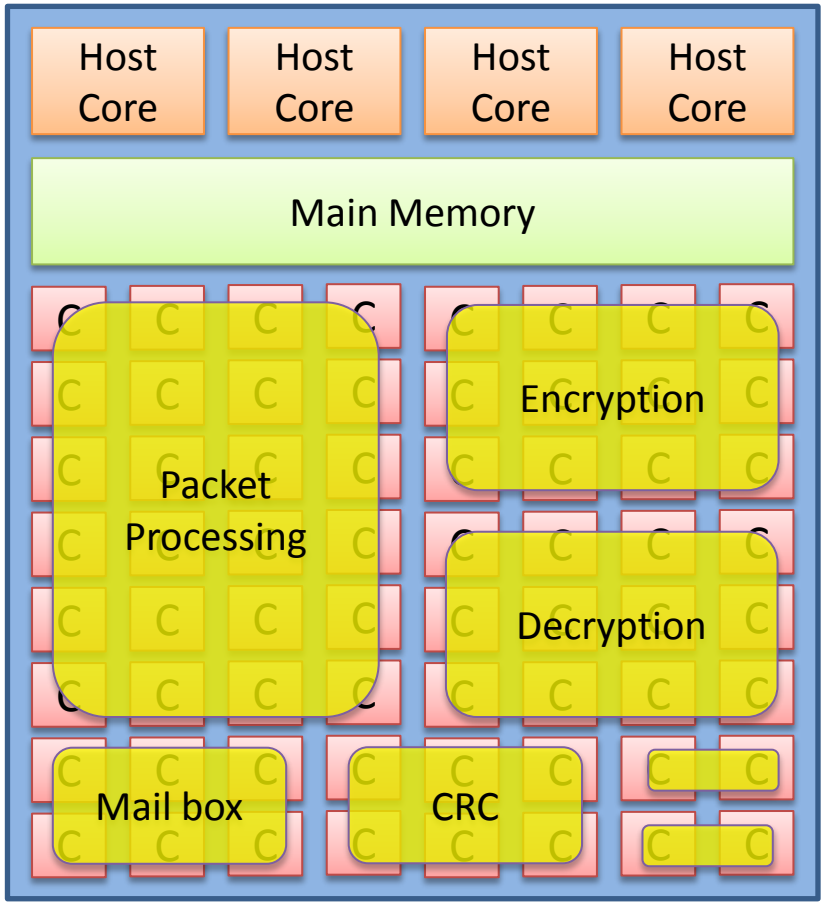
- メニーコアの組込みシステム向けSoCへの応用
  - ASIC型SoCから、ASSP型SoCへ
    - ASIC: Application Specific Integrated Circuit
    - ASSP: Application Specific Standard Product
  - 多品種少量生産から、少品種大量生産へ
- 課題
  - リアルタイム性の保証
  - 高性能（スループット）
  - 低消費電力

ソフトウェアによる専用化

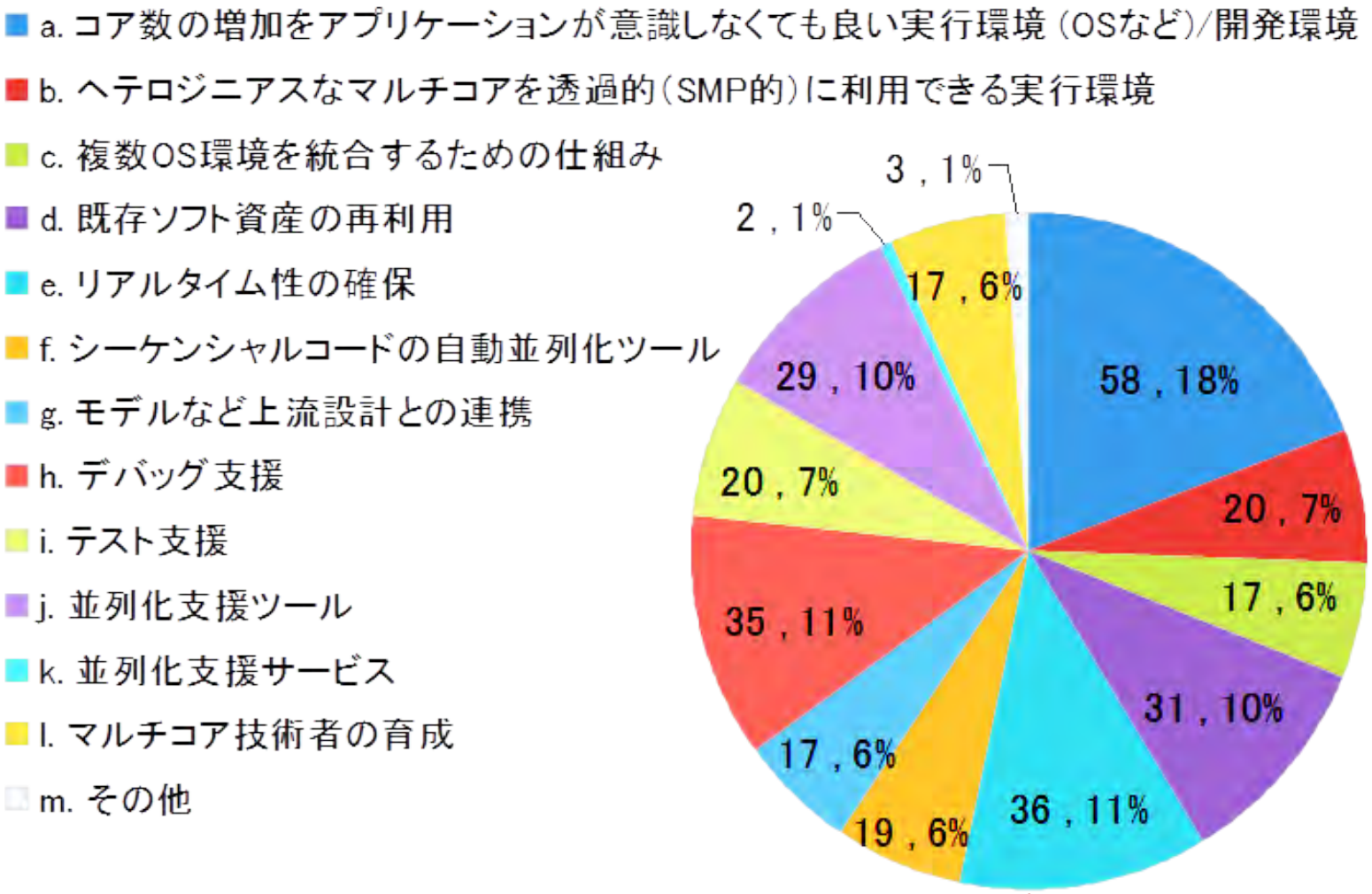
◆タブレット用SoC



◆ネットワーク機器用SoC



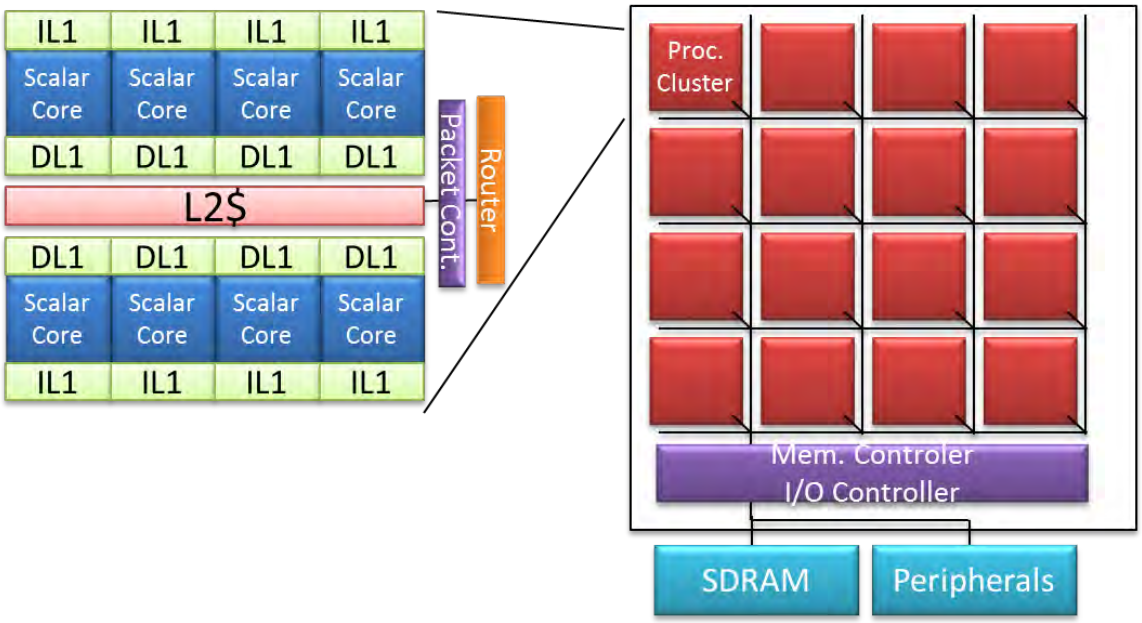
マルチ/メニーコアのソフトウェアの課題



出典：JEITAアンケート（2011）

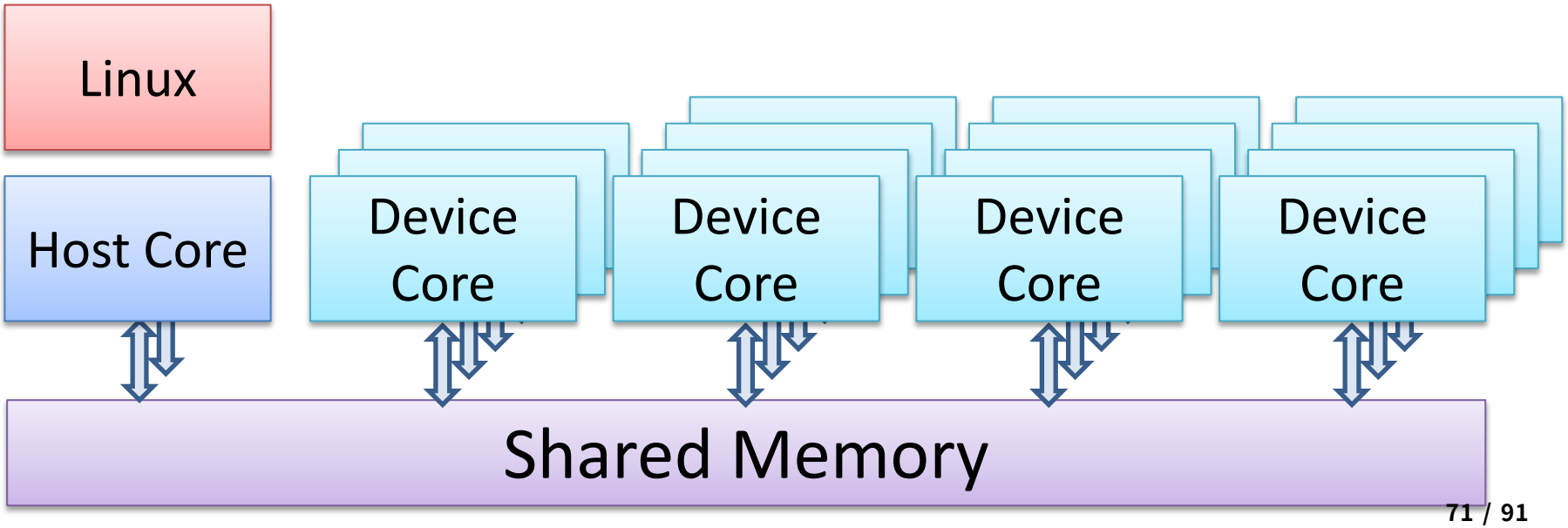
SMYLErefアーキテクチャ

- 九州大学と電気通信大学が開発
- クラスタの2Dメッシュ
  - 8コア/クラスタ
  - 各コアはMIPSベース
- 共有メモリ、プライベートL1キャッシュ、共有L2キャッシュ
- 128コアのFPGAプロトタイプ



SMYLErefのプログラミングモデル

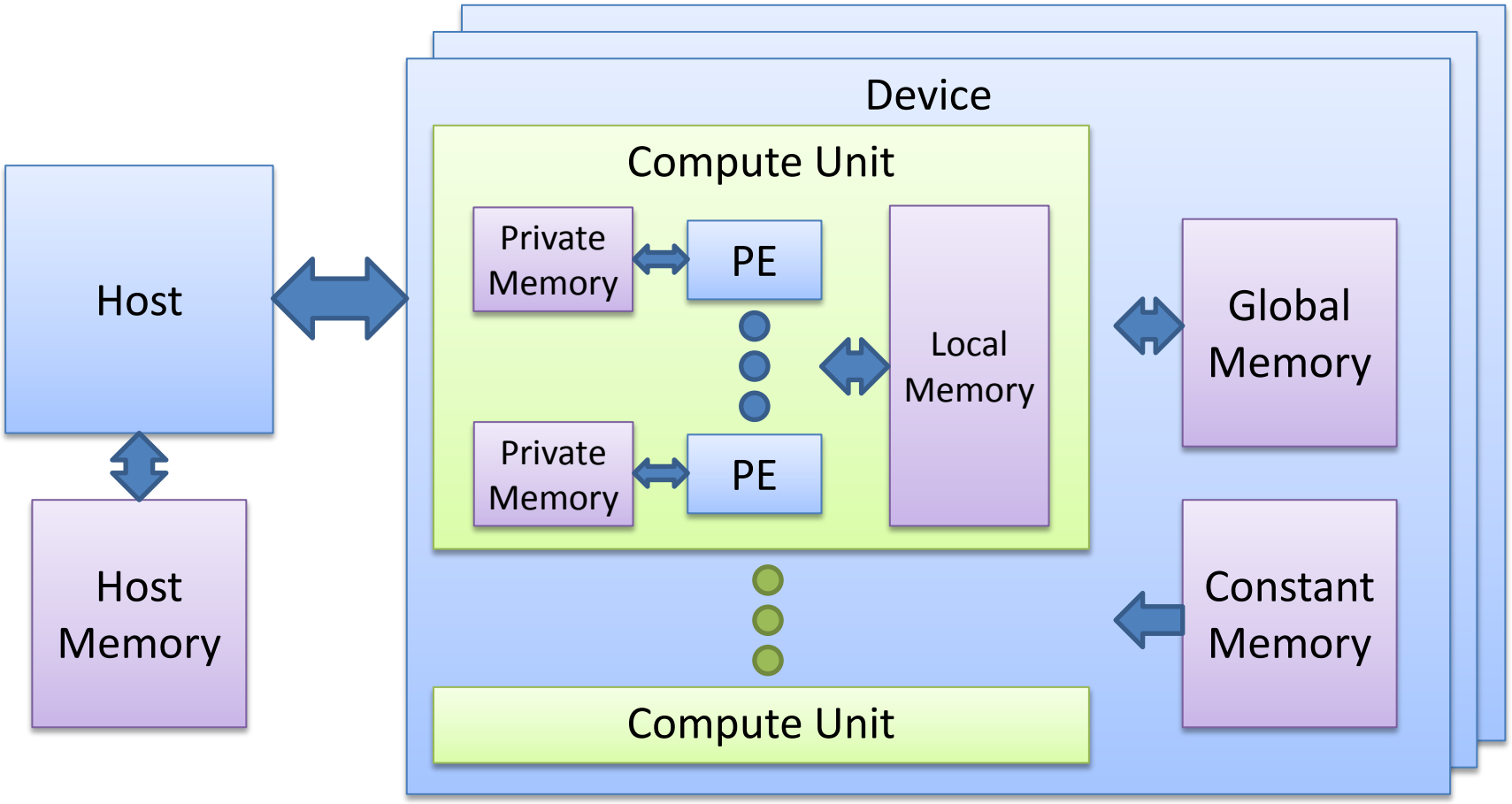
- 1コアをホストとして使用
  - Linux、仮想アドレス
- 残りのコアをデバイス（アクセラレータ）として使用
  - 物理アドレス
- 共有メモリ、ソフトウェア透過なキャッシュ



並列プログラミング

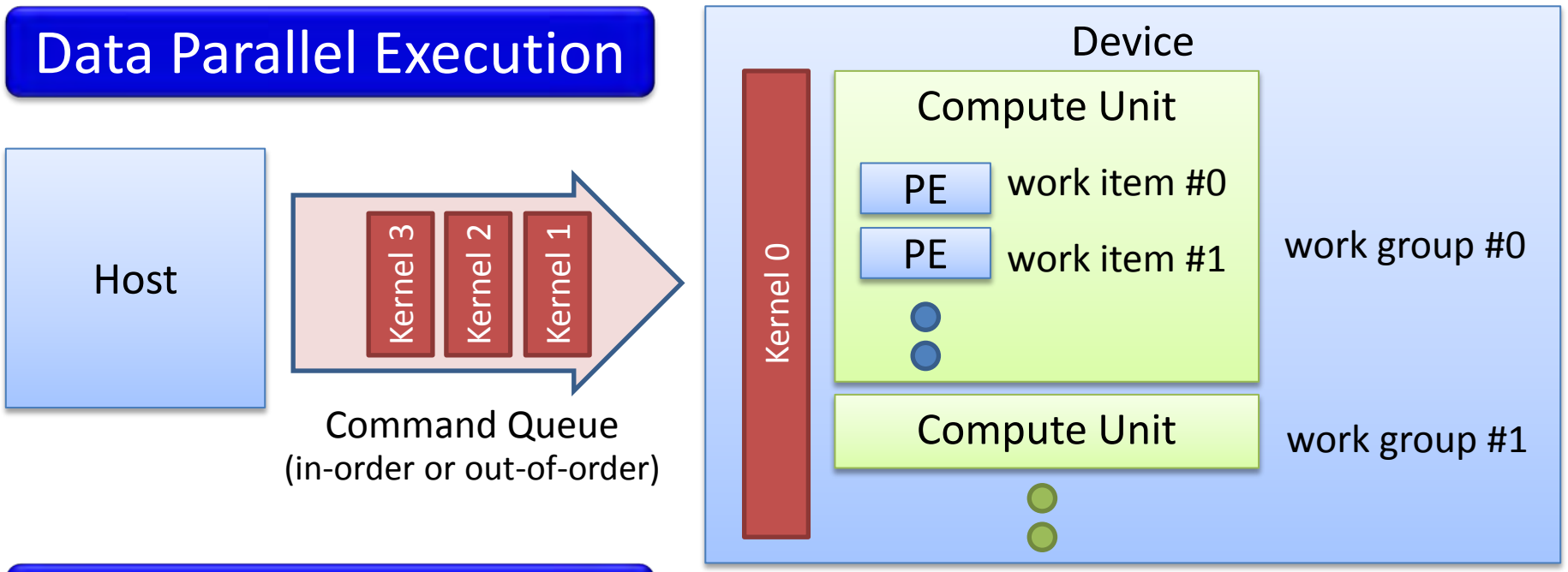
- メニーコアを活かすためには、並列プログラミング言語/フレームワークが必要
  - OpenMP, MPI, OpenCL, Intel Threading Building Blocks, Nvidia CUDA, etc
- OpenCLを採用
  - オープンで、ロイヤルティフリーな仕様
    - Specification 1.0 released in 2008
  - C言語ベース
  - ヘテロジニアスなプラットフォームに対応
  - プラットフォーム非依存
    - Intelマルチコア、Nvidia GPU、AMD GPU、Cell B.E.など
  - データ並列実行とタスク並列実行の両方に対応

OpenCLアーキテクチャモデル

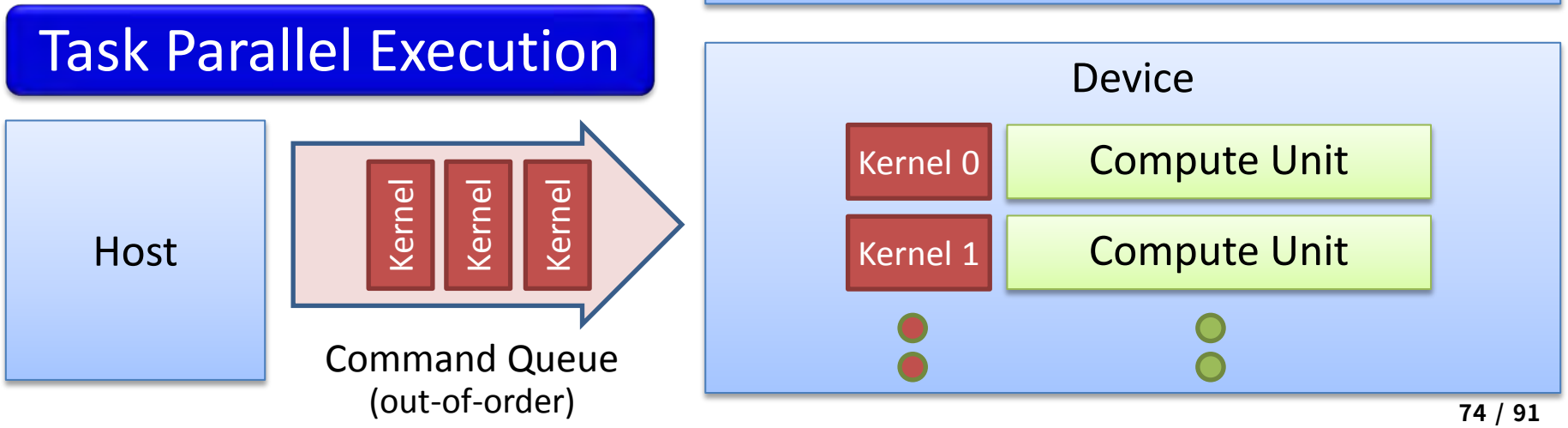


OpenCLプログラミングモデル

Data Parallel Execution

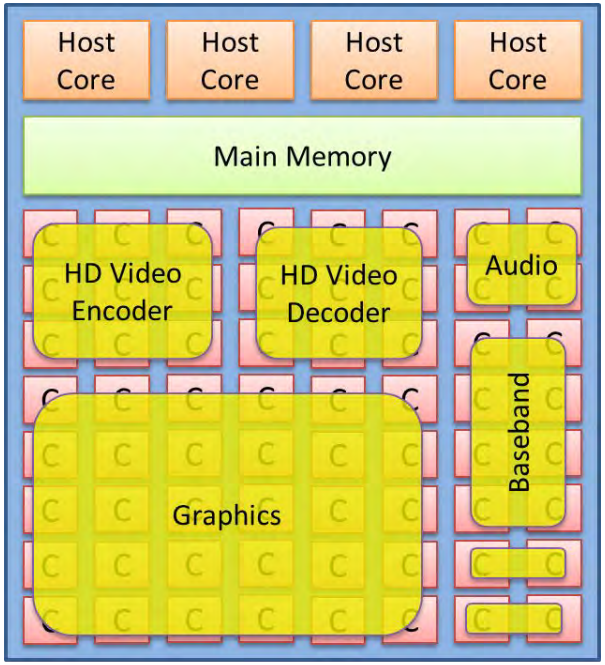


Task Parallel Execution

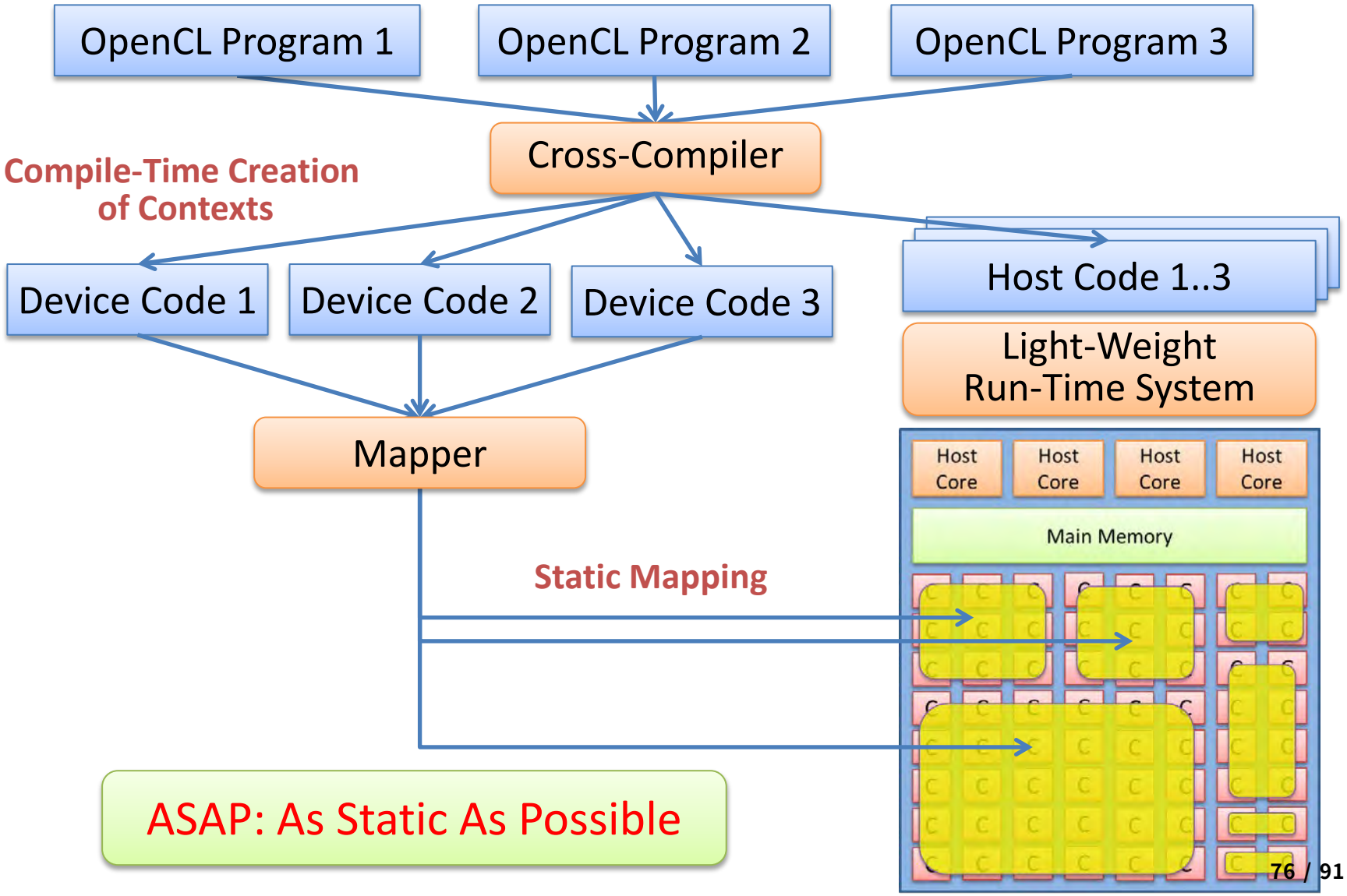


既存のGPU向けOpenCL環境の問題点

- 複数のOpenCLアプリケーションの並列実行が不可
  - 唯一のOpenCLプログラムがデバイス全体を占有
- リアルタイム性の保証が困難
  - 実行時の性能オーバーヘッドが大きい
    - コンテキストの生成、カーネルのディスパッチなど
  - オーバーヘッドの予測性が悪い

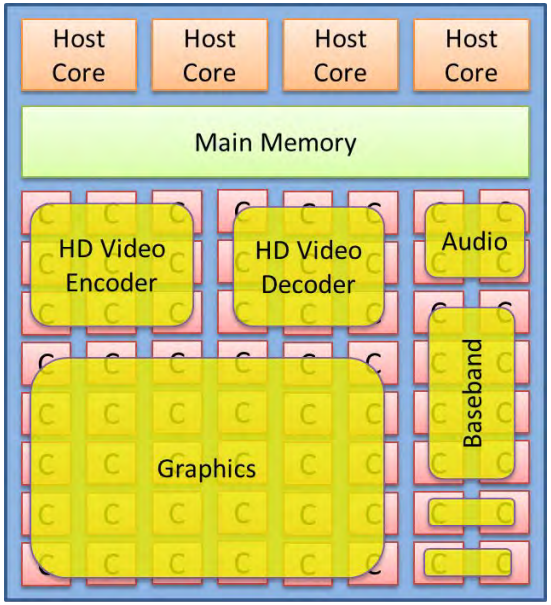


SMYLE OpenCL環境



SMYLE OpenCLの特徴

- 小さな起動時オーバーヘッド
  - 静的なコンテキストやオブジェクトの生成
  - 静的なタスクマッピング
- 様々なレベルにおける並列化
  - 複数のOpenCLアプリの並列実行
  - 個々のOpenCLアプリ内の
    - タスク並列実行
    - データ並列実行



典型的なOpenCLの実行の流れ

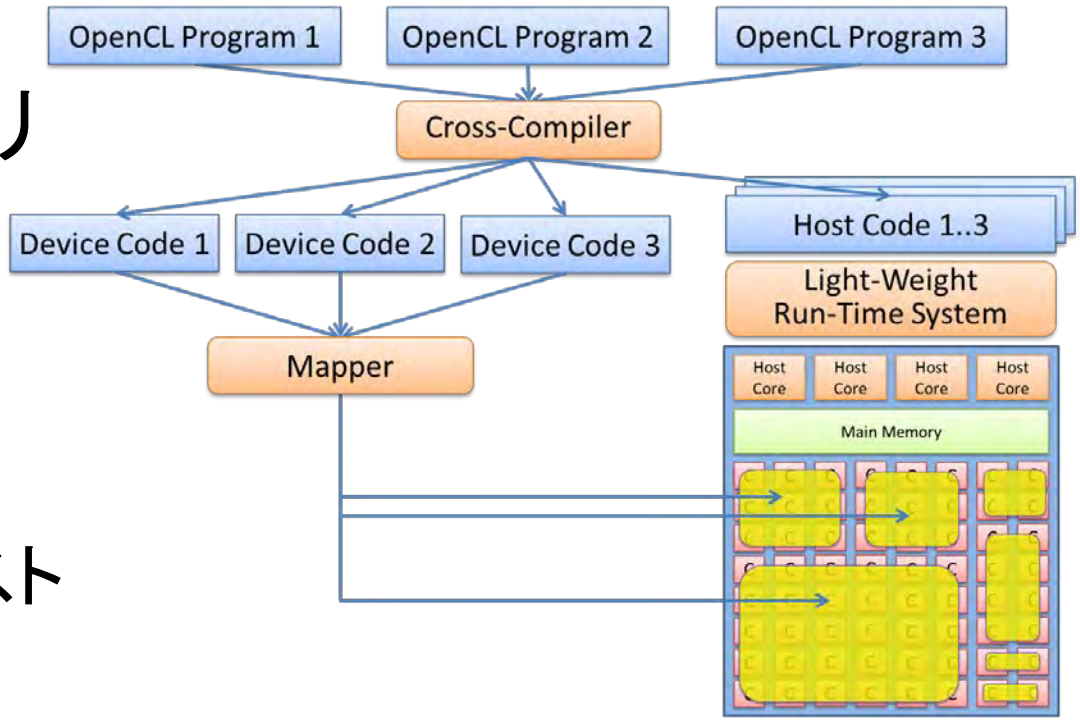
1. デバイス情報を取得、デバイスを確保
2. コンテキストを生成
3. コマンドキューやメモリバッファを生成し、コンテキストに格納
4. カーネルをビルドし、コンテキストに格納
5. 入力データをメモリバッファに格納
6. カーネルを実行
7. 実行結果をメモリバッファから読み出し
8. コマンドキュー、メモリバッファ、コンテキストなどを解放

SMYLE OpenCLの実行の流れ

1. ~~デバイス情報を取得、デバイスを確保~~
2. ~~コンテキストを生成~~
3. ~~コマンドキューやメモリバッファを生成し、コンテキストに格納~~
4. ~~カーネルをビルドし、コンテキストに格納~~
5. 入力データをメモリバッファに格納
6. カーネルを実行
7. 実行結果をメモリバッファから読み出し
8. ~~コマンドキュー、メモリバッファ、コンテキストなどを解放~~

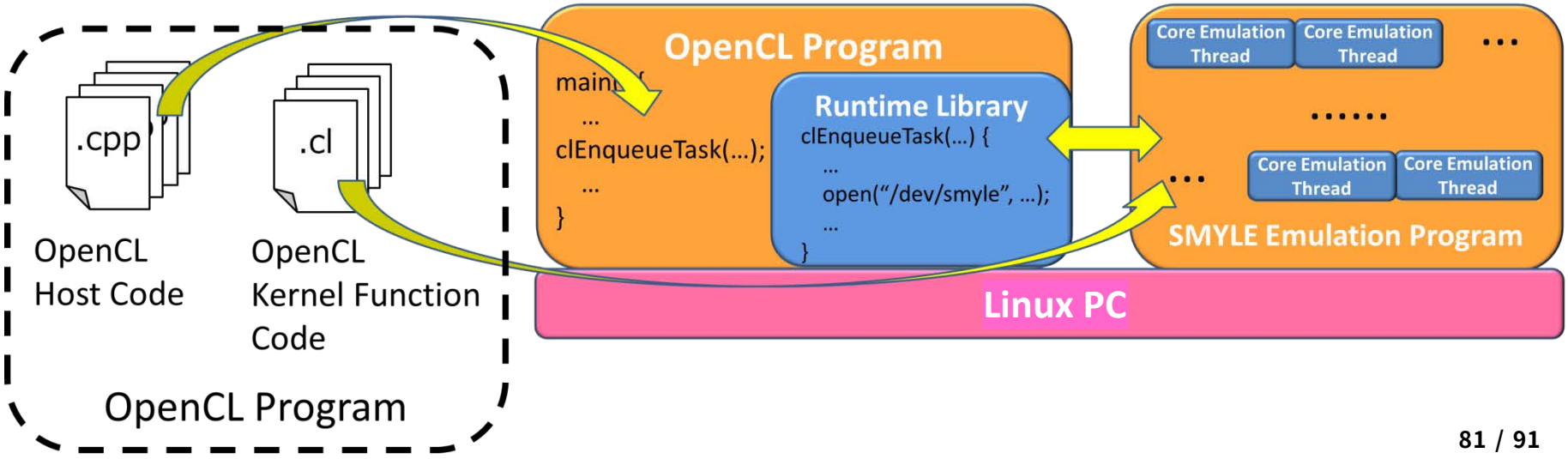
SMYLE OpenCL Toolkit

- クロスコンパイラ
  - GCCを使用
- ランタイムライブラリ
  - ホスト側
  - デバイス側
- マッパ
  - シングルコンテキスト
  - マルチコンテキスト
- 機能シミュレータ
  - LinuxベースPC上で動作

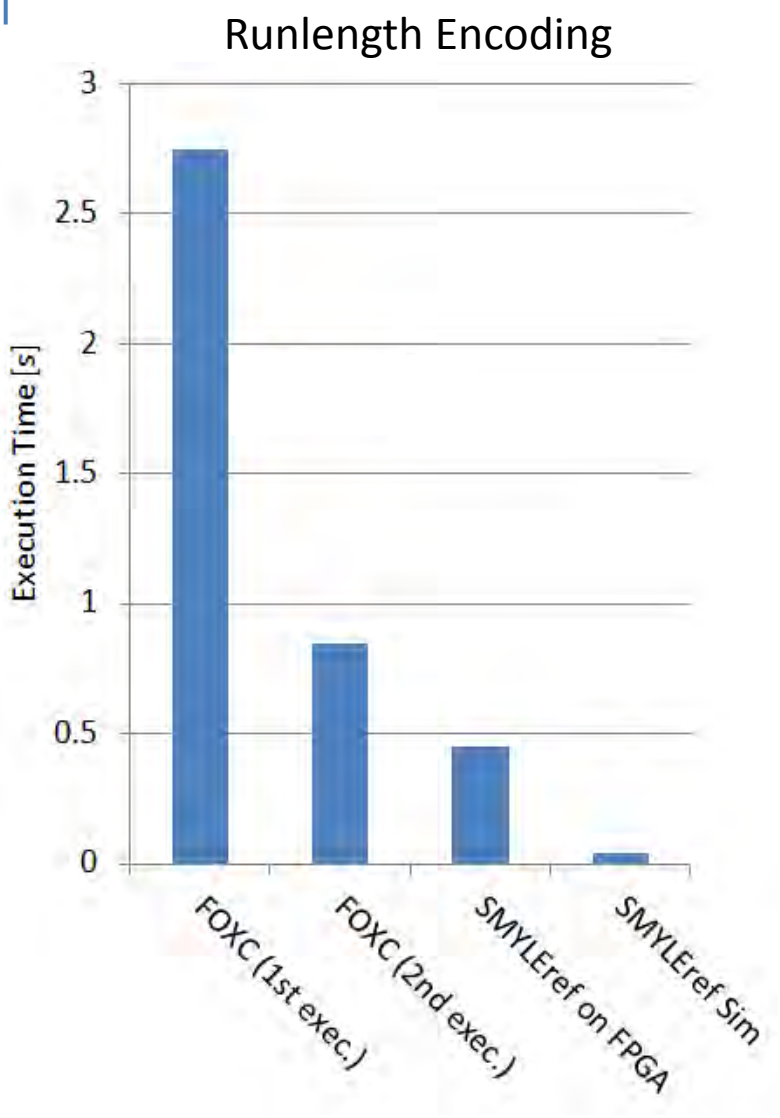


SMYLEref機能シミュレータ

- OpenCLプログラムのデバッグ用
  - 各コアをpthreadがシミュレーション
  - 性能や消費電力の見積もりは不可能
- 通常のOpenCL環境として使用可能



実行時オーバーヘッドの評価

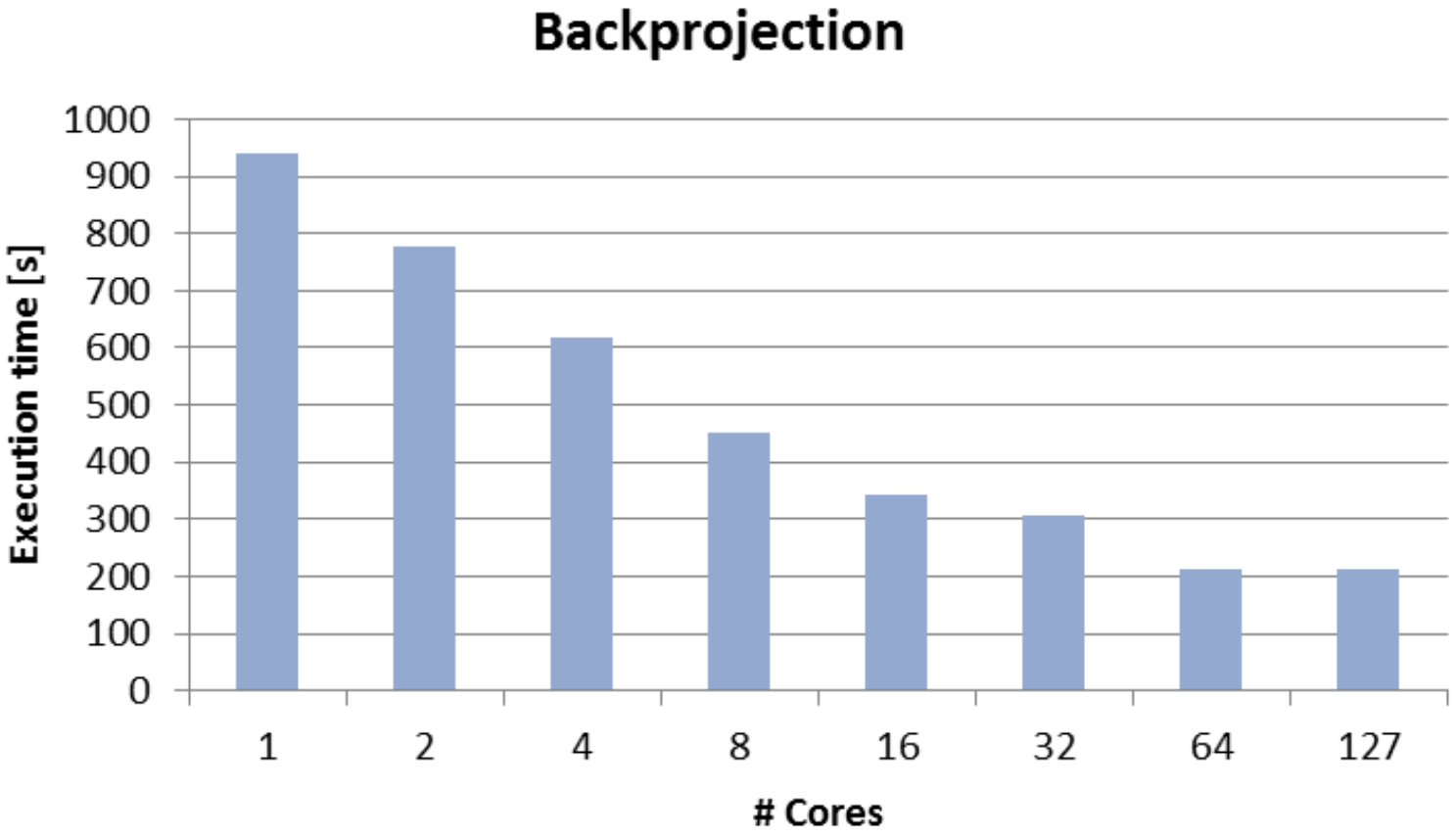


- FOXC
  - フィックスターズ社が開発したインテルマルチコア向けOpenCL
  - Core i7, 2.80GHz, 2 cores (4 threads)
  - 1回目の起動はキャッシュミス等のため低速
- SMYLEref on FPGA
  - 4 single-issue MIPS-based cores
    - 1 host and 3 device cores
  - Virtex 6, 10 MHz
- SMYLEref シミュレータ
  - Core i7, 2.80GHz, 2 cores (4 threads)

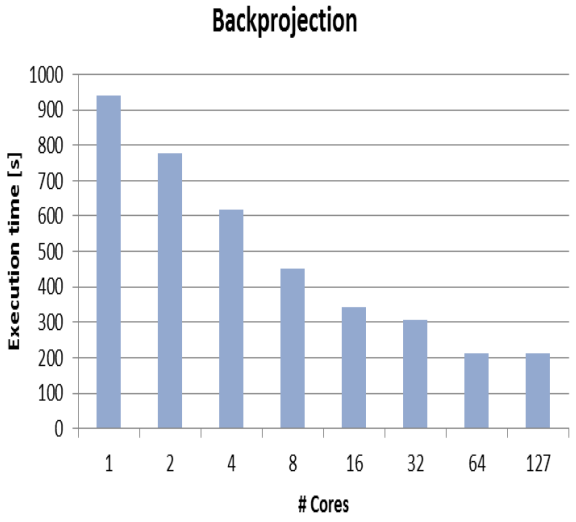
SMYLE OpenCL on 10MHz SMYLEref is faster than 2.8GHz Core i7

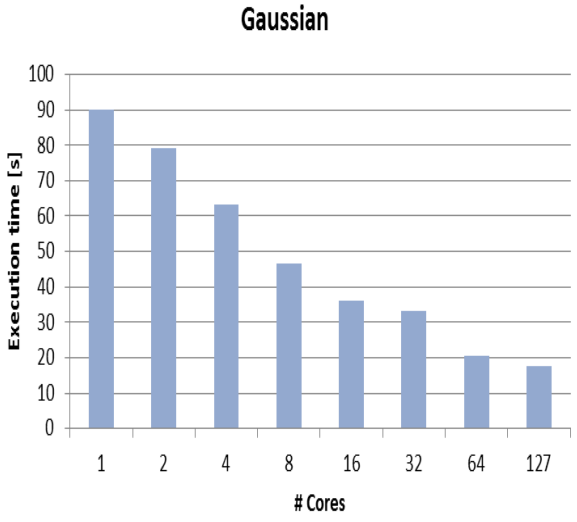
128コアSMYLEref/FPGA上での実行

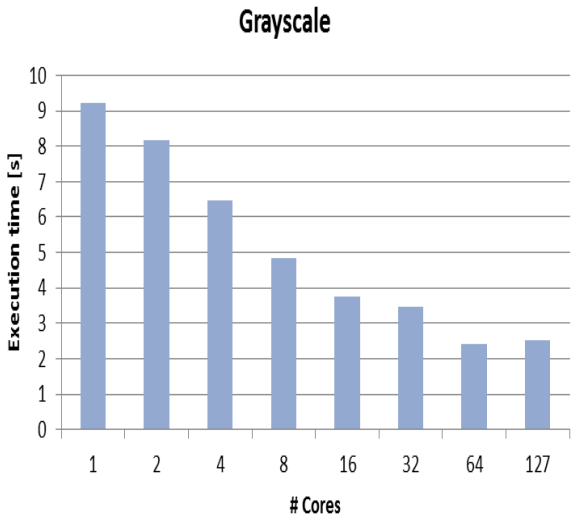
- Device cores: 1 to 127
- Core clock: 10 MHz

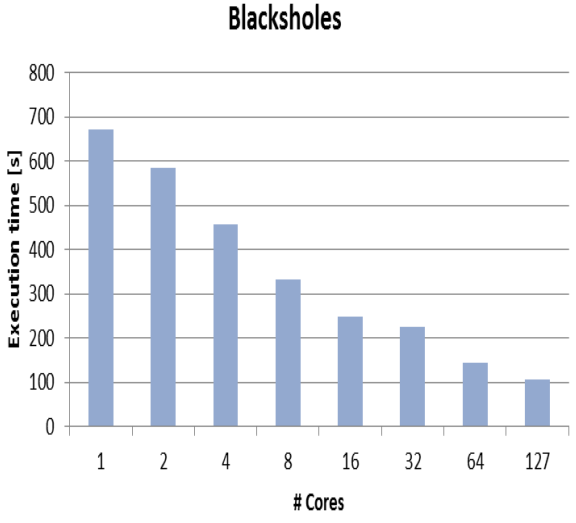


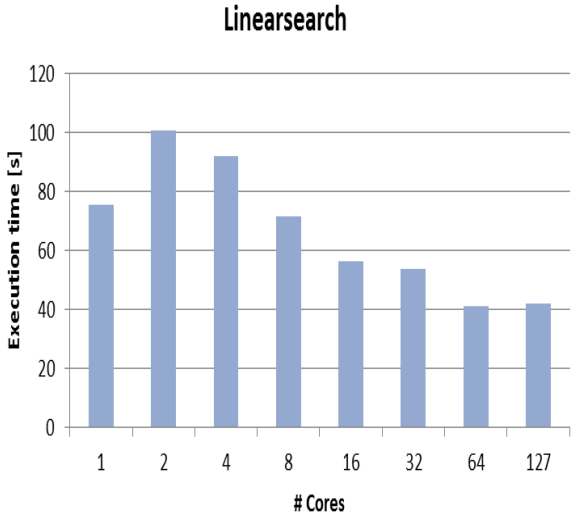
128コアSMYLEref/FPGA上での実行

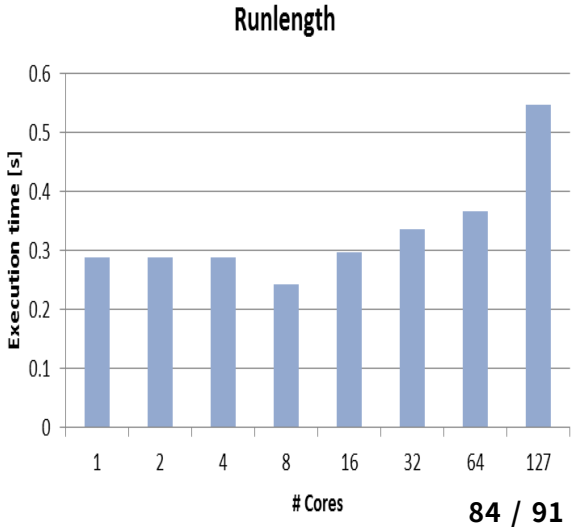






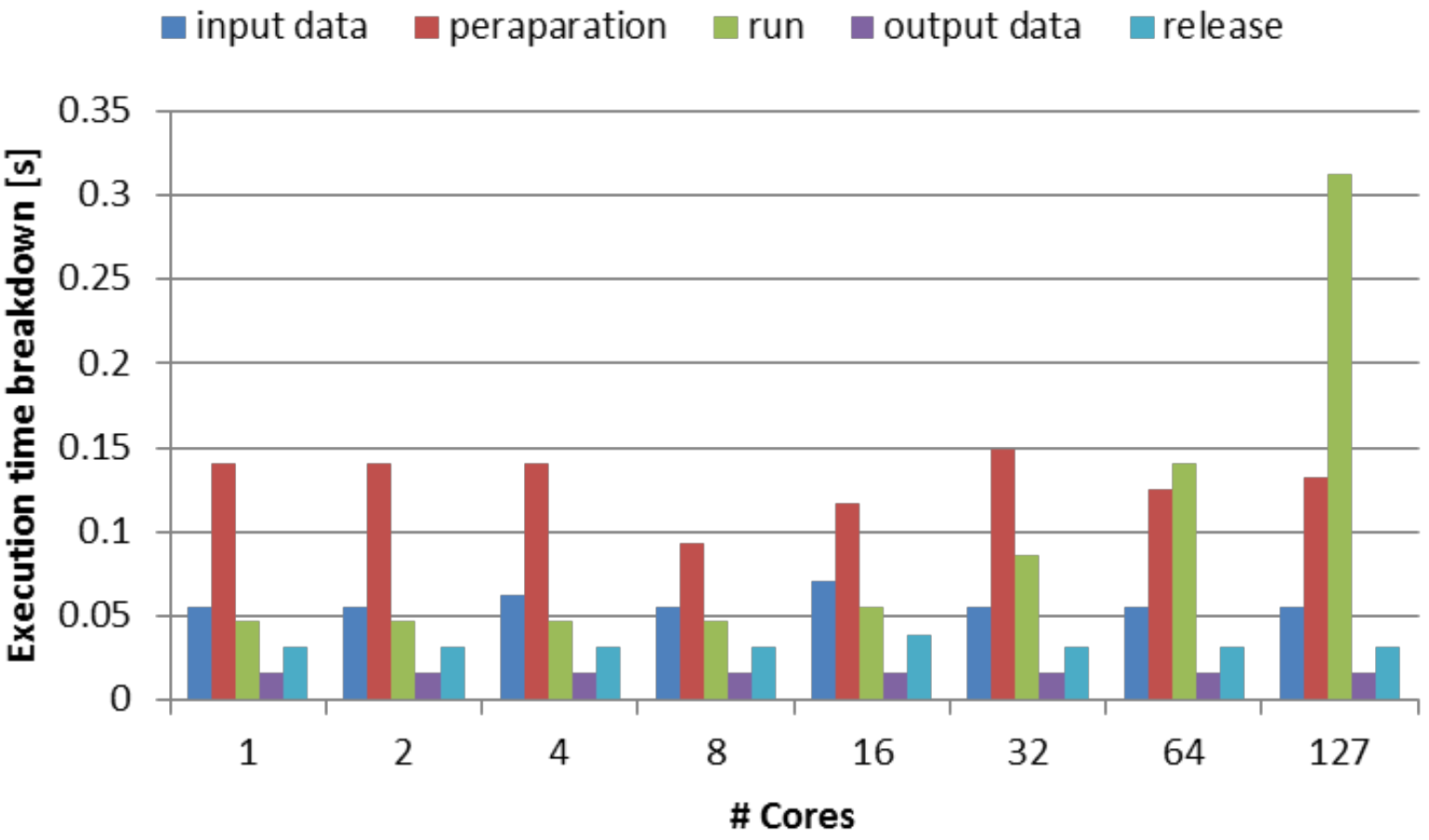






Runlengthの実行時間の内訳

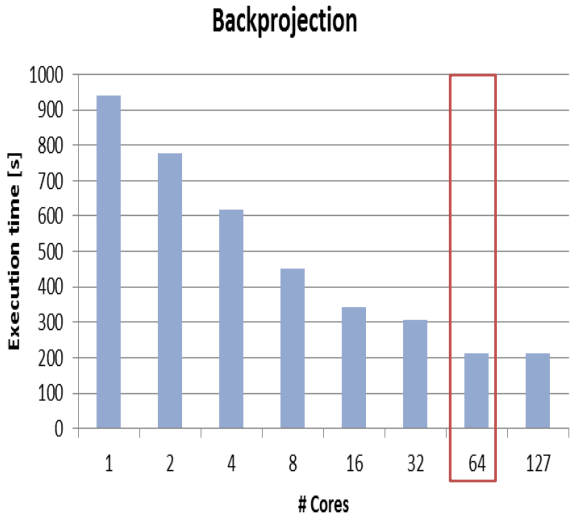
- 起動/終了時オーバーヘッドは(コア数に関わらず)ほぼ一定

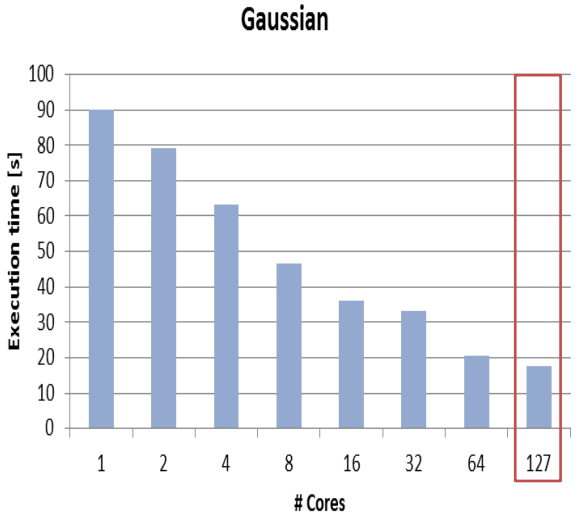


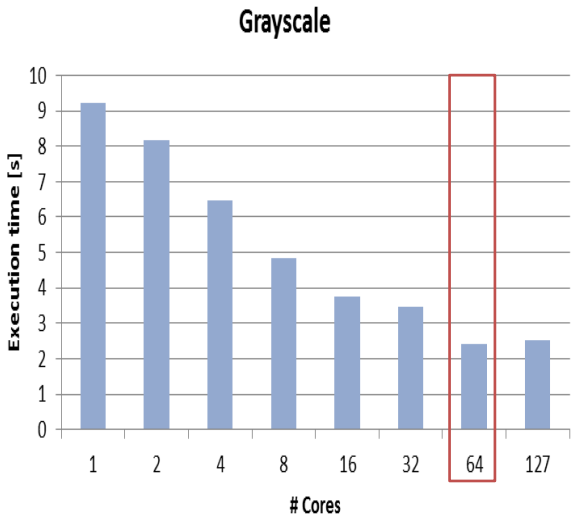
アプリレベル並列実行の評価

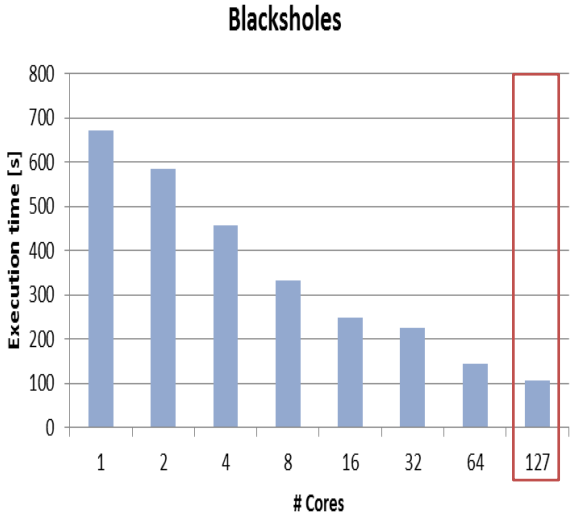
- 2つの実行方法のスループットを比較
  - A) 各アプリケーションのデータ並列性を最大限活用
    - 各アプリケーションに最大127コアを割り当て
    - アプリケーション間は逐次的に実行
  - B) 6つのアプリケーションを空間的に並列実行
    - アプリケーションが使用するコア数の合計が127以下
    - 各アプリケーションに割り当てるコア数を最適に決定
- 結果
  - アプリレベル並列実行（B）が4.03倍高速

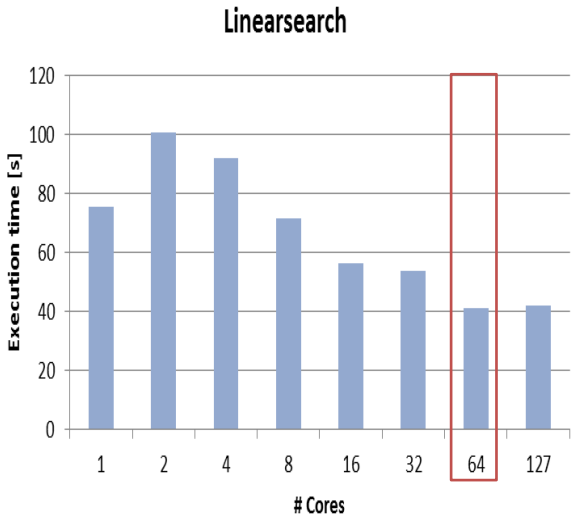
アプリ逐次実行の場合のコア数

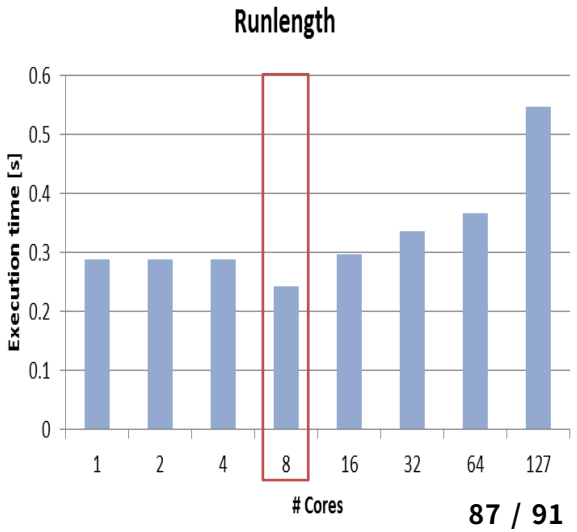




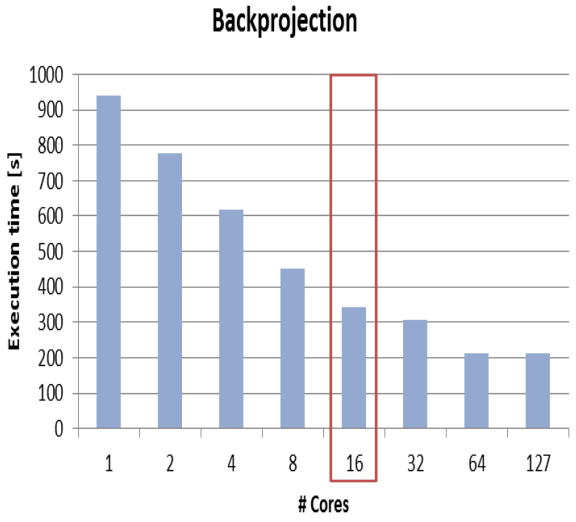


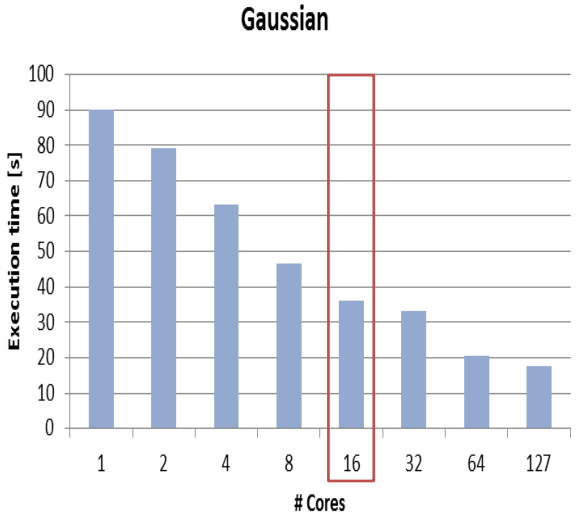


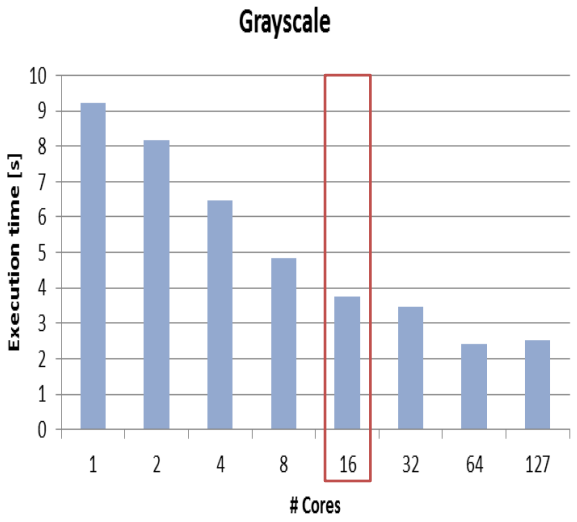


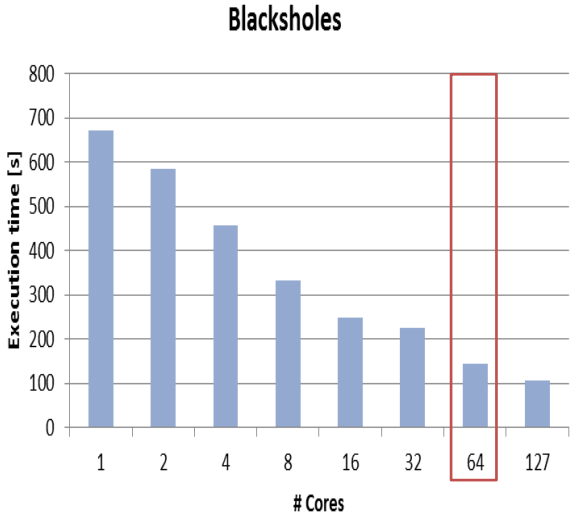


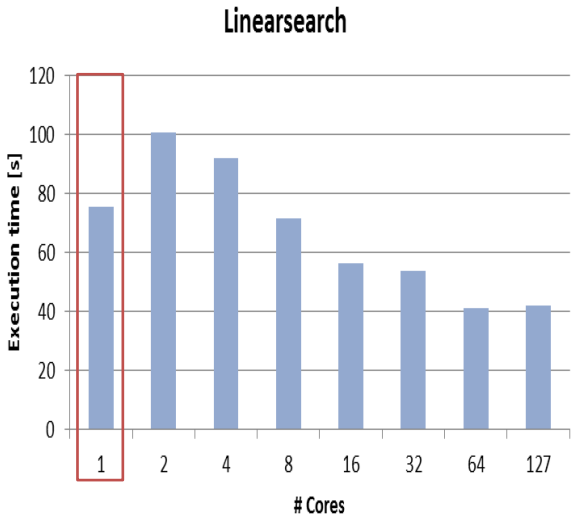
アプリ並列実行の場合のコア数

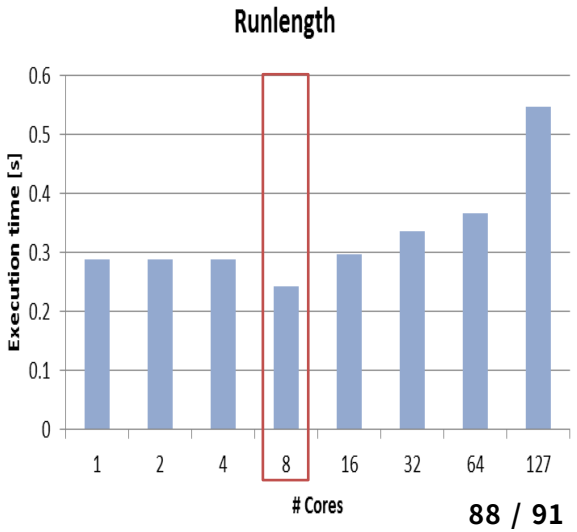












SMYLE OpenCLのまとめ

- SMYLE OpenCL環境を開発
  - ランタイムライブラリ
  - マッパ
  - 機能シミュレータ
- 特長
  - 小さな実行時オーバーヘッド
    - 静的なコンテキスト生成
    - 静的なマッピング
  - 様々なレベルでの並列実行
    - アプリケーション間の並列実行
    - アプリケーション内のデータ並列/タスク並列実行
- 128コアSMYLErefアーキテクチャ上で実証

まとめ

成果のまとめ

	九大＋立命館＋電通大＋農工大	トプスシステムズ	フィックスターズ	JEITA+eSOL+CATS
どのようなメニーコアを作れば良いのか?	SMYLEref SMYLE OpenCL	SMYLEvideo		市場調査
どのようにソフトウェアを開発すれば良いのか?		SMYLEvideo用ソフトウェア	CLtrump PEMAP	動作環境／開発環境調査
どのようにメニーコア研究開発を進めれば良いのか?	FPGAプロトタイプ 機能シミュレータ		BEMAP	プロジェクト提案