低消費電力マイクロプロセッサの研究動向

井上, 弘士
九州大学大学院システム情報科学研究院

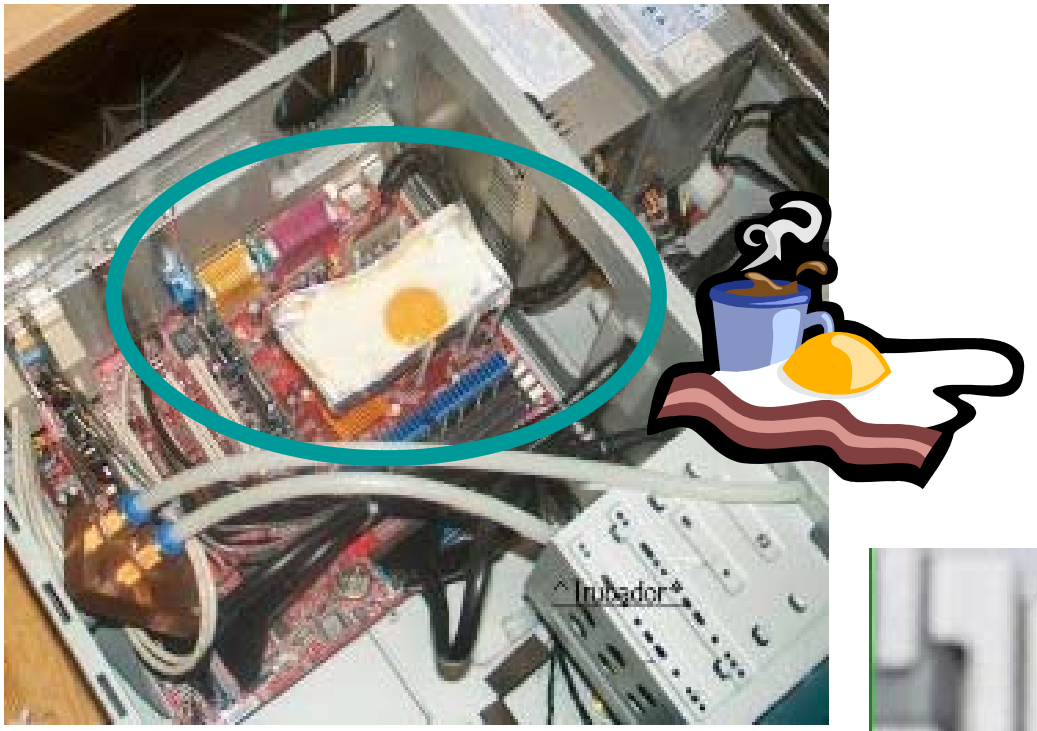
出版情報：SLRC プレゼンテーション, 2007-03-20. 九州大学システムLSI研究センター
バージョン：
権利関係：

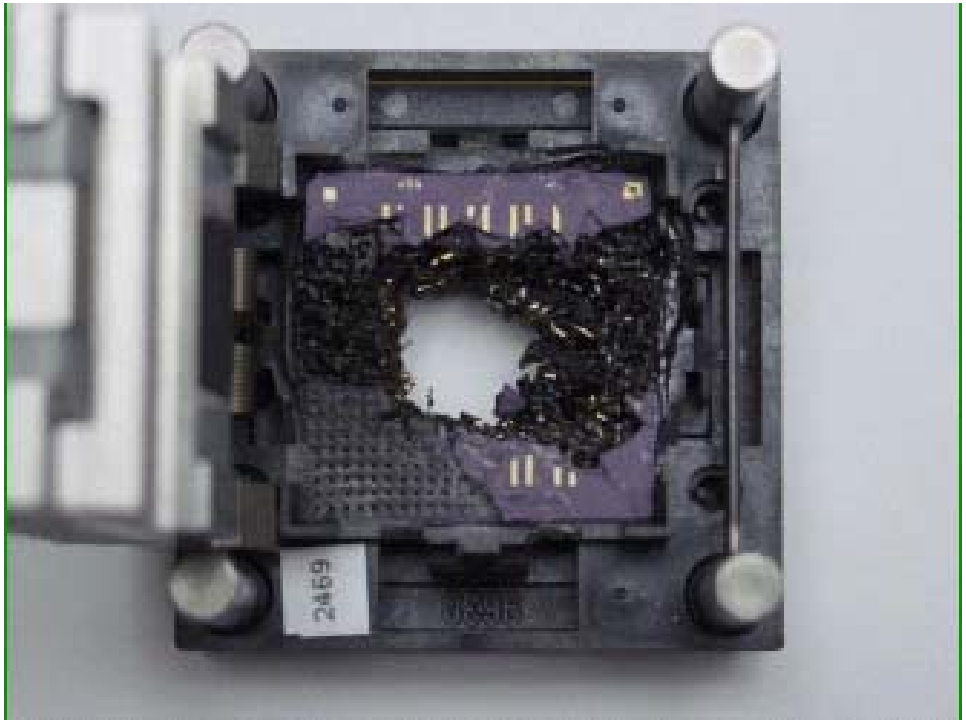
低消費電力マイクロプロセッサの研究動向

九州大学大学院システム情報科学研究院
井上弘士（いのうえこうじ）
inoue@i.kyushu-u.ac.jp

講演内容

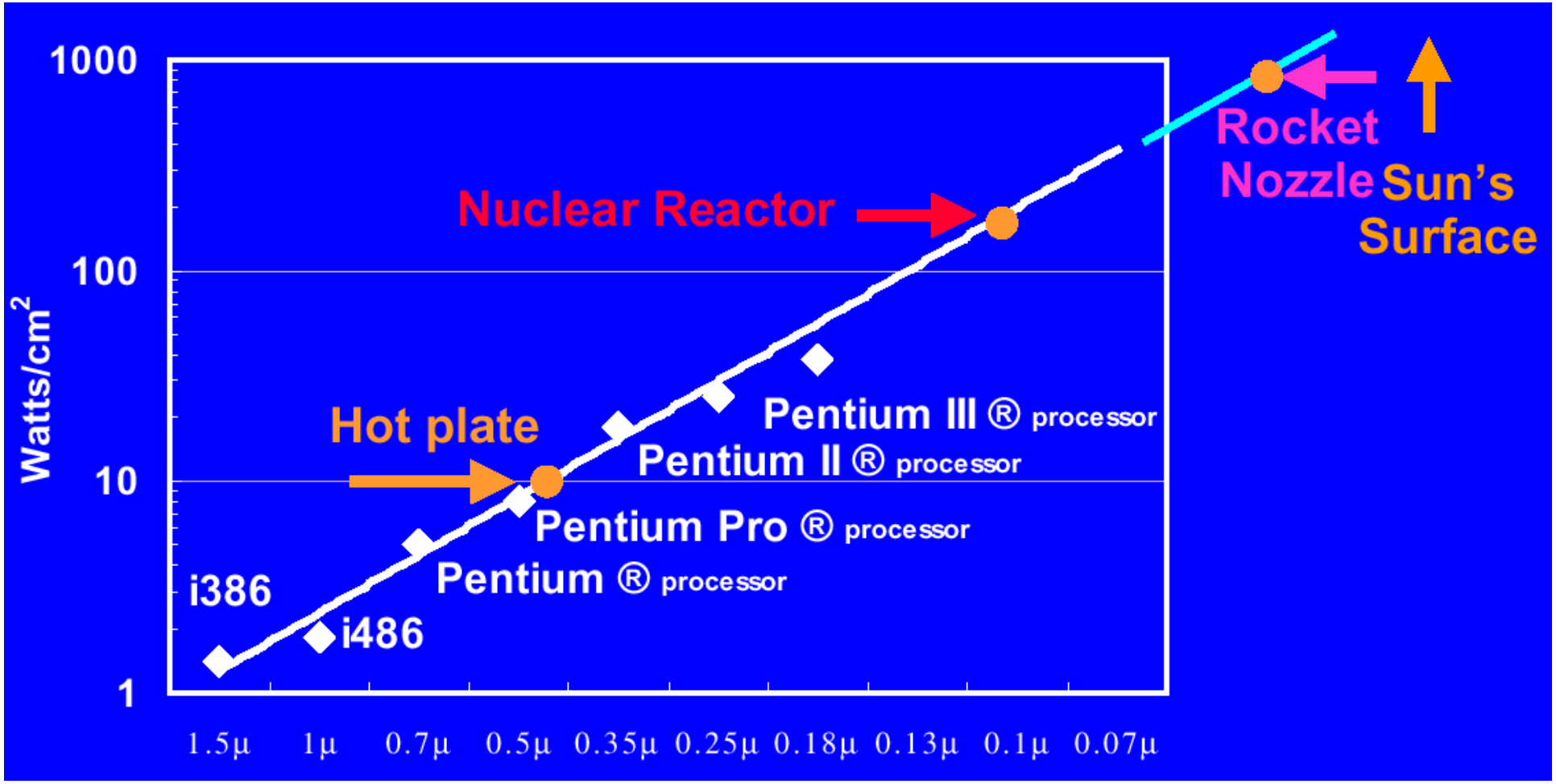
- マイクロプロセッサのトレンド
- 低消費電力化に向けた基本戦略
- TIPS：プログラムの実行において・・・
  - 「頻繁に実行される命令列」が存在する！
  - 「頻繁に実行される命令の種類」は少ない！
  - 「急いで実行すべき場合」と「そうでない場合」が存在する！
  - 「演算に必要なビット幅」は8～16ビットと小さい！
  - 「最近参照されたデータ」は再び参照される！
- 今後着目すべきは？





Next Generation Burn-in & Test System for Athlon Microprocessors : Hybrid Burn-in, Mark Miller, Burn-in & Test Socket Workshop, 2001

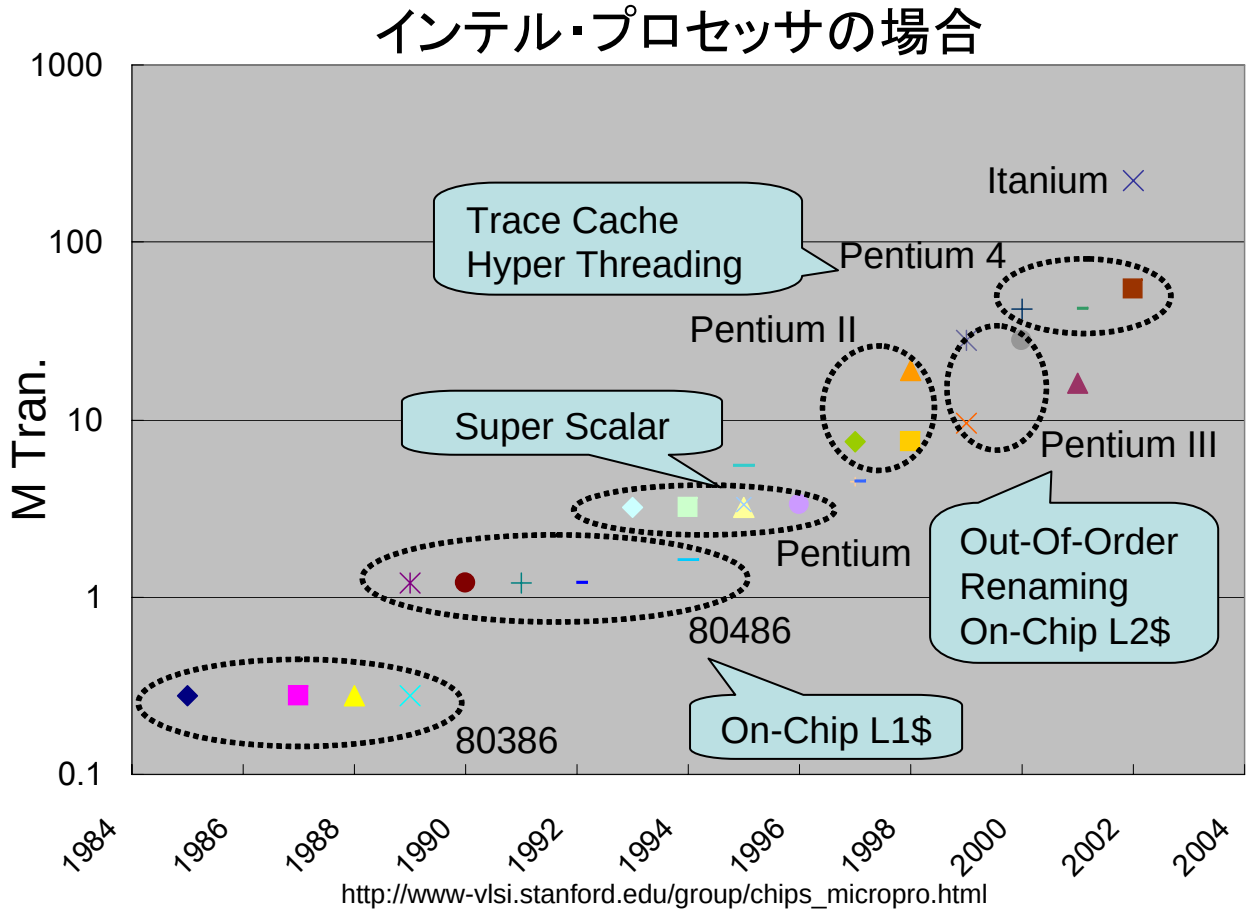
プロセッサの消費電力はどのくらい？



F. Pollack: New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies (MICRO99)

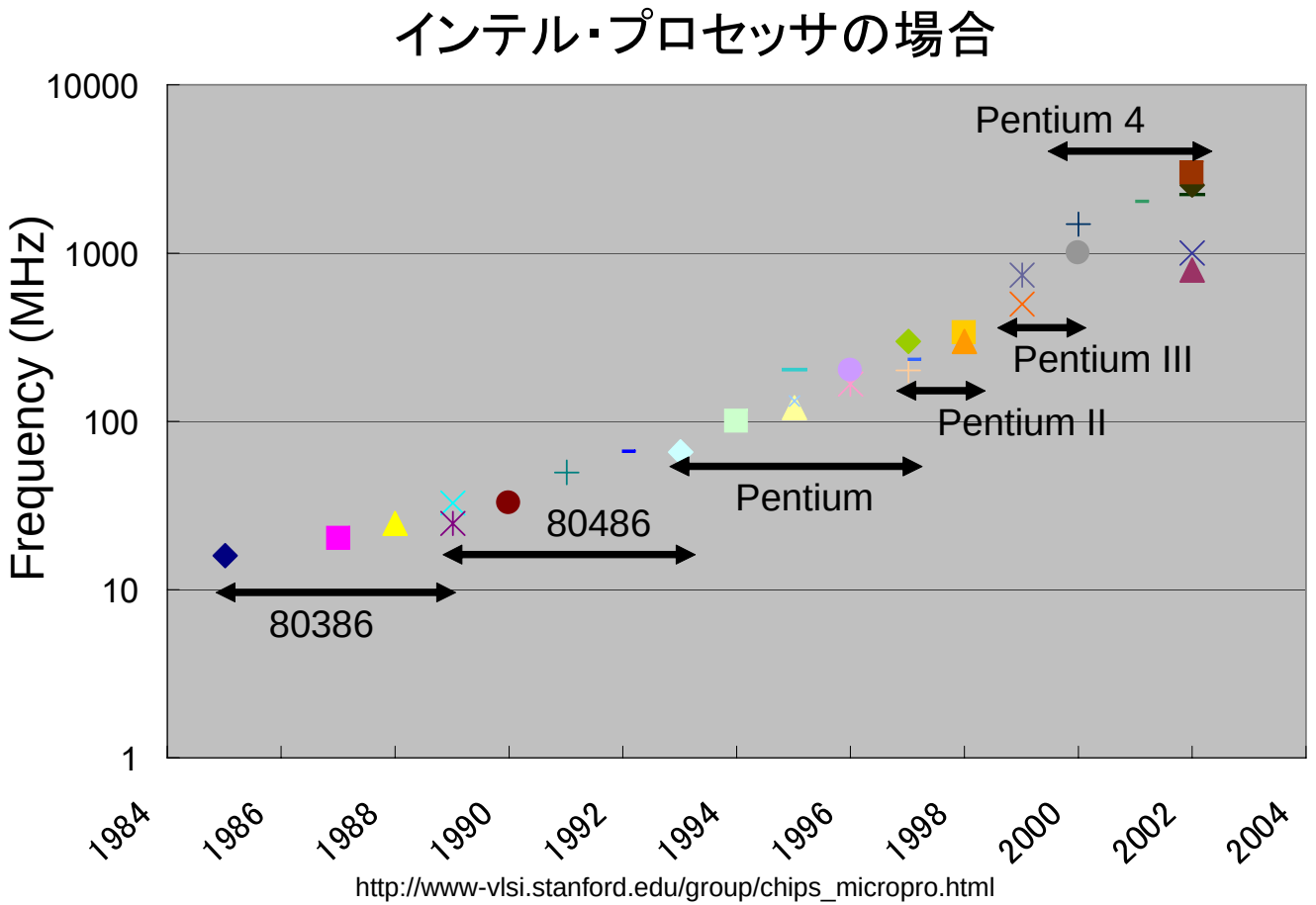
トランジスタ数の観点から（プロセッサの回路規模はどの程度？）

- 半導体集積度は3年で約4倍に！（ムーアの法則）



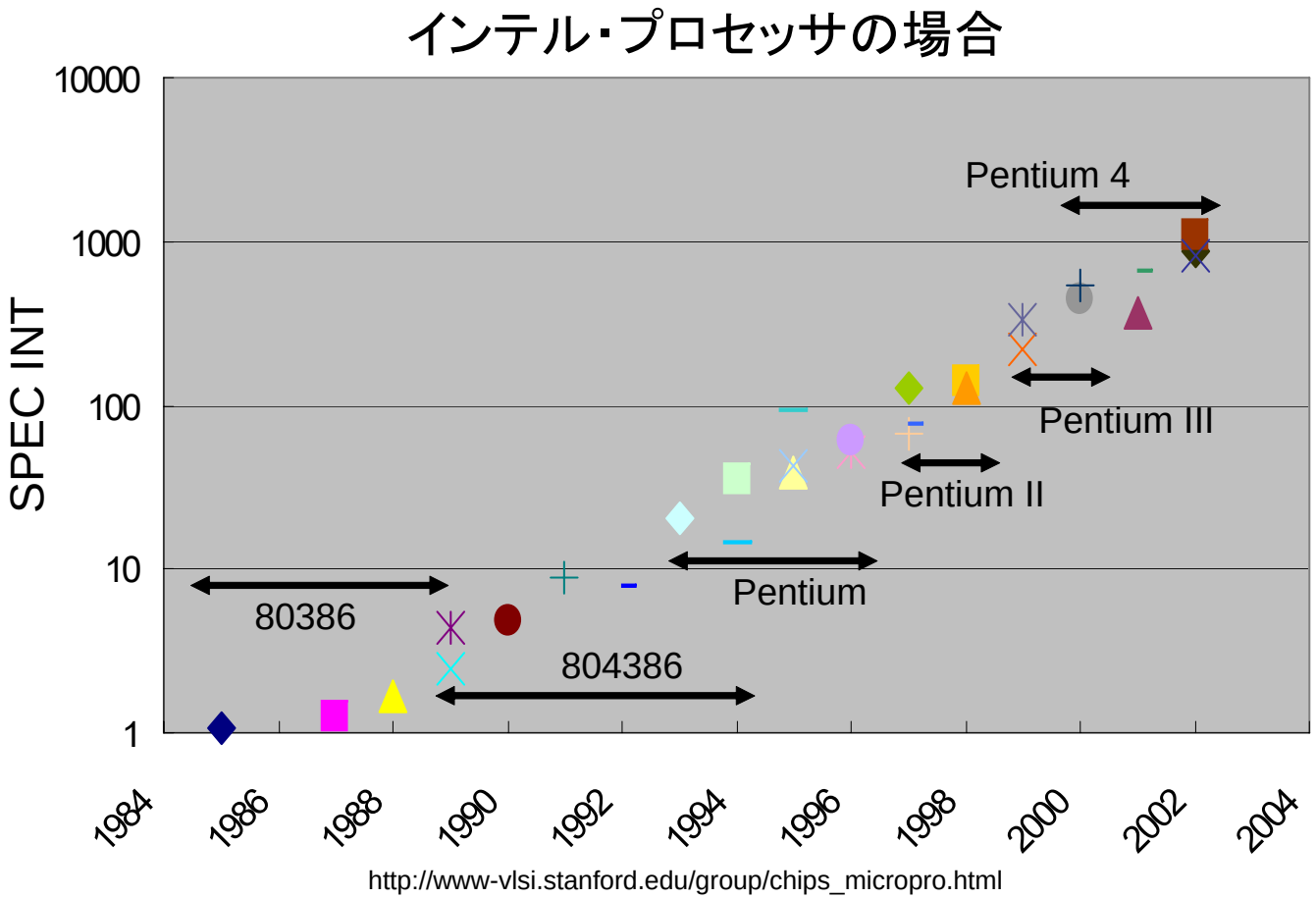
動作周波数の観点から（プロセッサはどの程度高速動作する？）

- プロセッサの動作周波数は3年で約2倍に！



性能の観点から
（プロセッサはどの程度高性能なのか？）

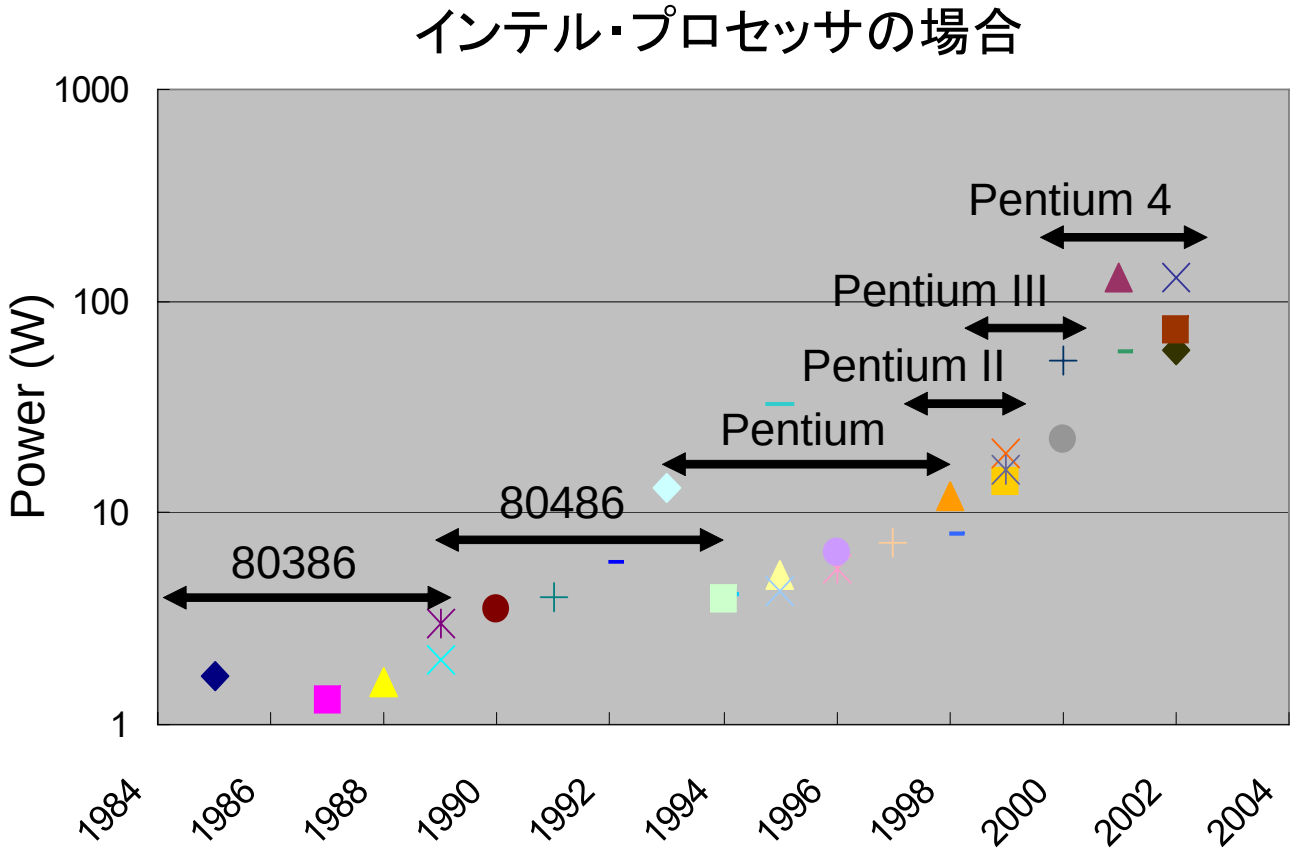
- プロセッサの性能は3年で約3～4倍に！



http://www-vlsi.stanford.edu/group/chips_micropro.html

消費電力の観点から
（プロセッサはどの程度電力消費するのか？）

- プロセッサの消費電力は3年で約3倍に！



http://www-vlsi.stanford.edu/group/chips_micropro.html

このまま性能向上を維持できるのか？

- 現状（３年毎に）
  - 性能：×３
    - 集積度：×４
    - 動作速度：×２
  - 消費電力：×３
- 高性能化への期待（３年毎に）
  - 性能：×３⇒維持したい！
    - 集積度：×４⇒もうしばらくは大丈夫！
    - 動作速度：×２⇒〇△□×・・・（なぜ？）
  - 発熱の問題が顕著化！
    - 「高速MPU開発を中止 インテル」 日経新聞2004/10/16より
      - 周波数上昇によってチップ内の漏電・発熱の問題が深刻になり、それを技術的に克服するためのコストが高すぎると判断した
- 低消費電力化なくして高性能化無し！
  - 「消費電力：×１」を維持しつつ性能を向上！

低消費電力化/エネルギー化の必要性

- 集積回路として，
  - 発熱による信頼性低下の回避
  - 放熱効率の高いパッケージの必要性の軽減
  - ディジタル雑音の低減
  - 電力供給系の設計
- 携帯機器として，
  - バッテリー駆動時間の長時間化とバッテリーの軽量化
  - 発熱に対する対策（ヒートシンク，ファン，等）の軽減
- 地球的なエネルギー問題への対策
  - ユビキタス社会の到来

講演内容

- マイクロプロセッサのトレンド
- 低消費電力化に向けた基本戦略
- TIPS:プログラムの実行において・・・
  - 「頻繁に実行される命令列」が存在する！
  - 「頻繁に実行される命令の種類」は少ない！
  - 「急いで実行すべき場合」と「そうでない場合」が存在する！
  - 「演算に必要なビット幅」は8～16ビットと小さい！
  - 「最近参照されたデータ」は再び参照される！
- 今後着目すべきは？

CMOS論理回路の消費電力

$$P \propto C_L \cdot V_{DD}^2 \cdot F$$

$$P = C_L \cdot V_{DD}^2 \cdot f_{0\to1} + t_{SC} \cdot V_{DD} \cdot I_{peak} \cdot f_{0\to1} + V_{DD} \cdot I_{leakage}$$

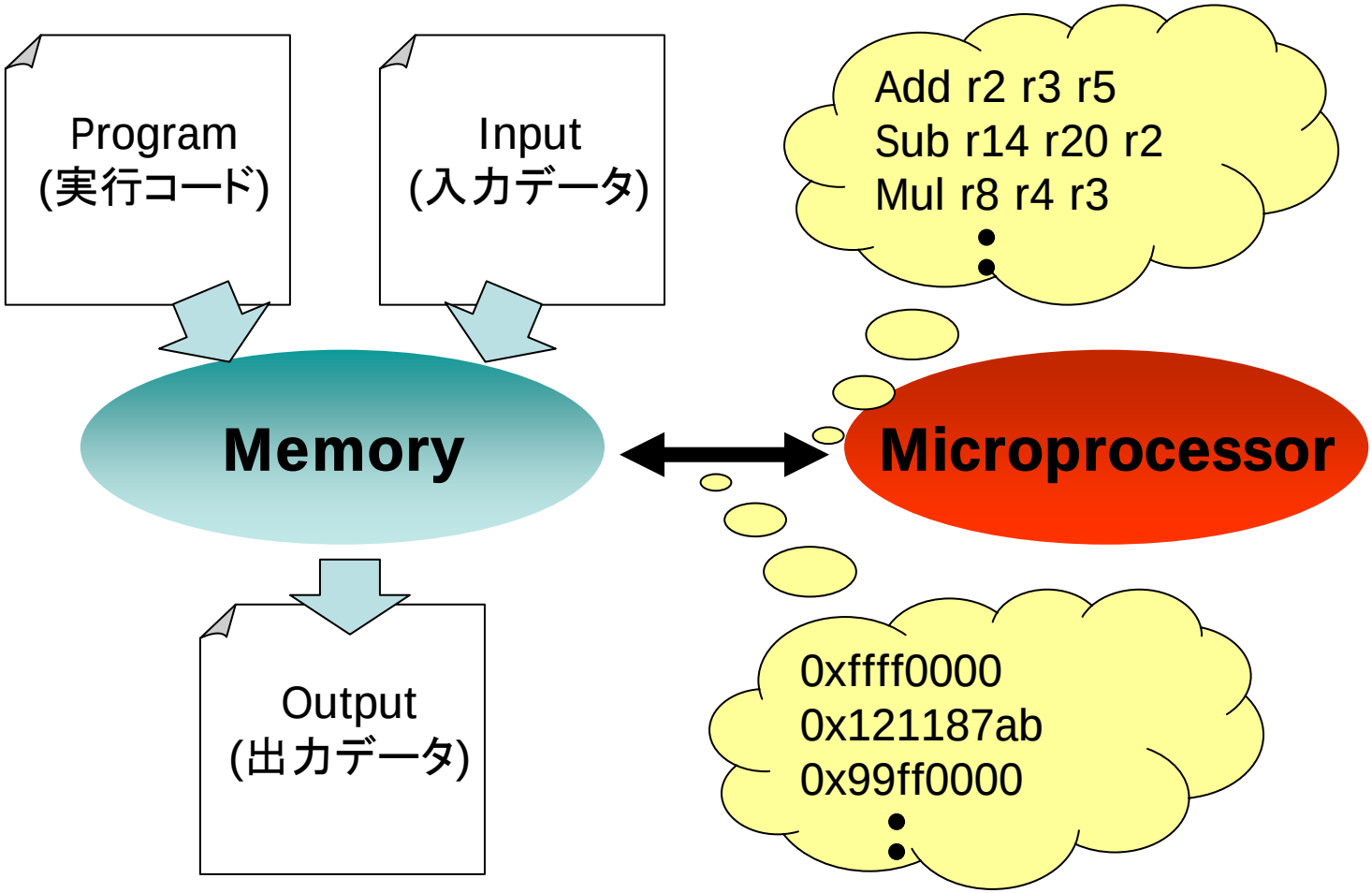
動的消費電力（スイッチング時の容量への充放電）

スイッチング時の貫通電流による消費電力

静的消費電力（ダイオードやトランジスでの漏れ電流）

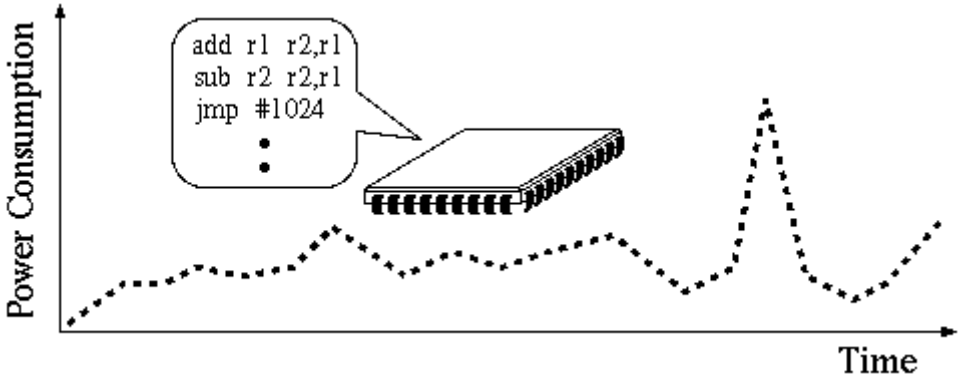
- C_L：負荷容量
- V_{DD}：電源電圧
- I_{peak}：貫通電流
- $I_{leakage}$：漏れ電流
- t_{SC}：スイッチング時間
- $f_{0\to1}$：単位時間の信号遷移頻度（$=S_{0\to1} \cdot F$）
- $S_{0\to1}$：信号遷移確率
- F：クロック周波数

マイクロプロセッサの処理



「消費電力」と「消費エネルギー」

- Quiz
  - Q:「消費電力」を削減すれば、かならず、「消費エネルギー」も削減できる。YesかNoか？
  - A:NO!
- 消費エネルギーは消費電力の時間積分
  - 消費電力を1/2にしても，プログラム実行時間が2倍になれば消費するエネルギーは同じ！



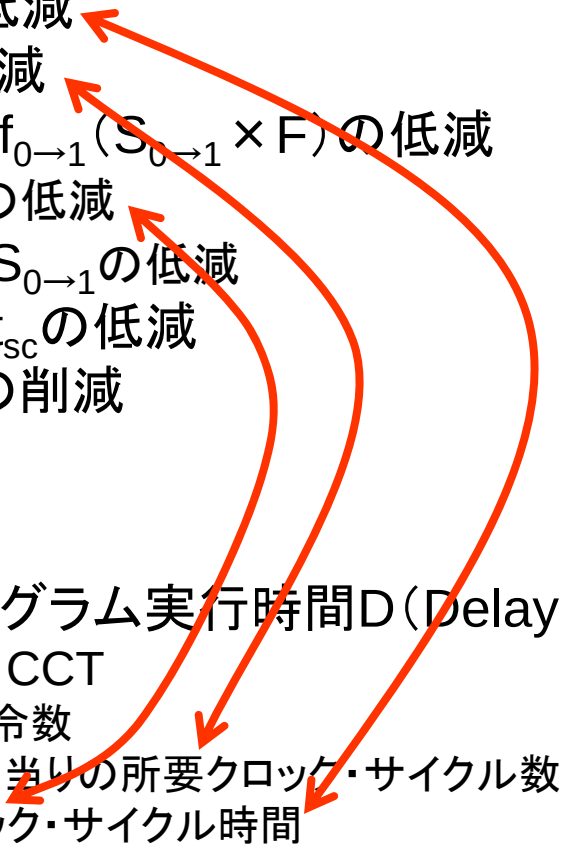
$$E_{Chip} = \int_0^t P_{Chip}$$

E_{Chip}：チップの消費エネルギー
P_{Chip}：チップの消費電力
t：プログラム実行時間

消費電力と性能のトレードオフ

$P = C_L \cdot V_{DD}^2 \cdot f_{0\to1} + t_{SC} \cdot V_{DD} \cdot I_{peak} \cdot f_{0\to1} + V_{DD} \cdot I_{leakage}$

- 電源電圧V_{DD}の低減
- 負荷容量C_Lの削減
- スイッチング頻度$f_{0\to1}$（$S_{0\to1} \times F$）の低減
  - 動作周波数Fの低減
  - 信号遷移確率$S_{0\to1}$の低減
- 波形の変化時間t_{sc}の低減
- 漏れ電流$I_{leakage}$の削減



相反する要求

$E = P \cdot D$

- 消費電力Pとプログラム実行時間D（Delay）との間のトレードオフ
  - D＝IC×CPI×CCT
    - IC：実行命令数
    - CPI：1命令当りの所要クロック・サイクル数
    - CCT：クロック・サイクル時間

低消費電力化へのアプローチ

- **戦略その１：できるだけソフトウェアに任せる！**
  - ハードウェア処理からソフトウェア処理へ
    - 命令スケジューリング：スーパスカラ vs. VLIW
    - 分岐予測：HW分岐予測 vs. SW分岐予測
  - ただし，「性能低下」と「コード互換性」が大きな問題

- **戦略その２：できるだけ無駄をなくす！**
  - 基本的な考え方⇒必要以上のことはやらない（できるだけ手を抜く）！

  ①Consider program-execution behavior

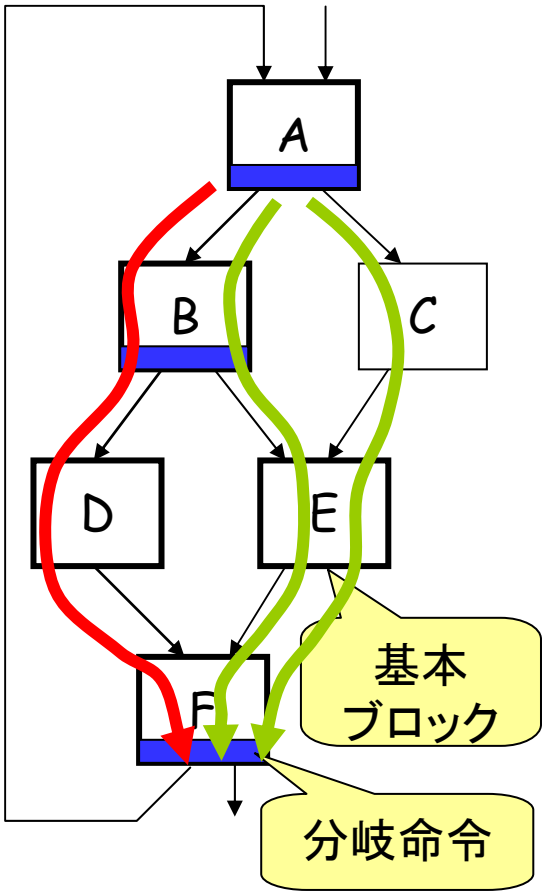
  ②Divide, Select/Allocate, and Reduce
  - Divide
    - 空間的分割：回路を少なくとも2つ以上に分割
    - 時間的分割：プログラム実行を少なくとも2つ以上の区間に分割
  - Select/Allocate
    - 活性化対象回路
    - 動作モード（電源電圧，閾値電圧，など）
    - など

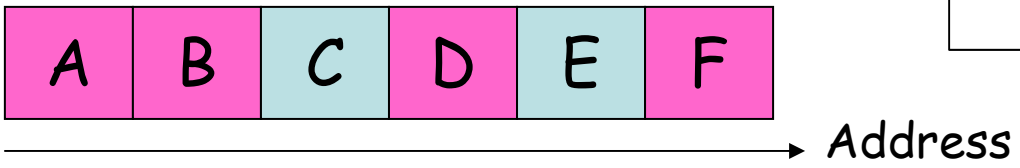
講演内容

- マイクロプロセッサのトレンド
- 低消費電力化に向けた基本戦略
- TIPS：プログラムの実行において・・・
  - 「頻繁に実行される命令列」が存在する！
  - 「頻繁に実行される命令の種類」は少ない！
  - 「急いで実行すべき場合」と「そうでない場合」が存在する！
  - 「演算に必要なビット幅」は8～16ビットと小さい！
  - 「最近参照されたデータ」は再び参照される！
- 今後着目すべきは？

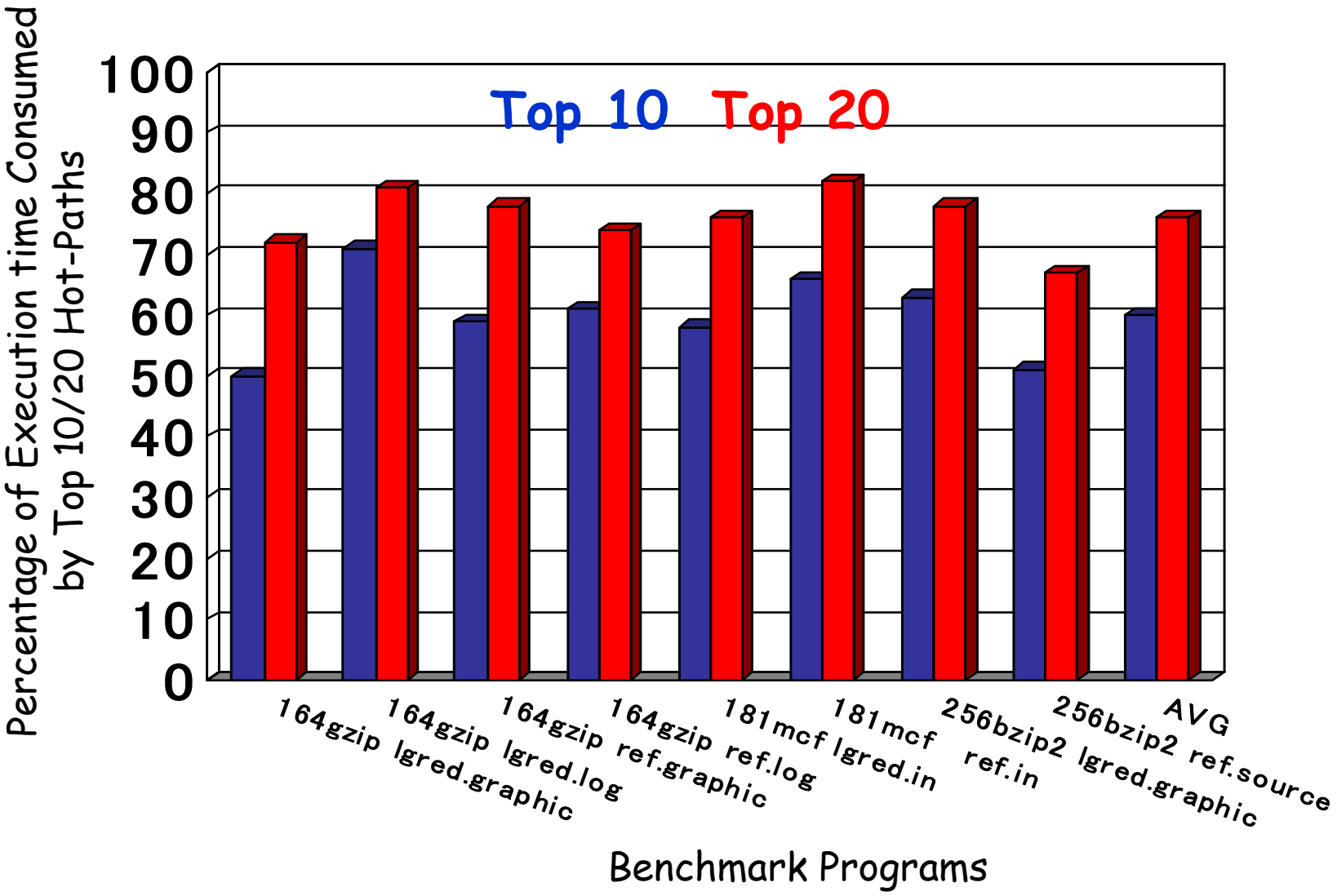
「頻繁に実行される命令列（ホットパス）」とは？

- 90/10の法則
  - プログラム実行時間の90%は，コード中10%の領域の実行によって費やされる
- ホットスポット
  - 頻繁に実行される連続したコード領域
- ホットパス
  - 頻繁に実行される命令列
  - 多くの場合はループボディに存在
  - 空間的に分散している場合もある



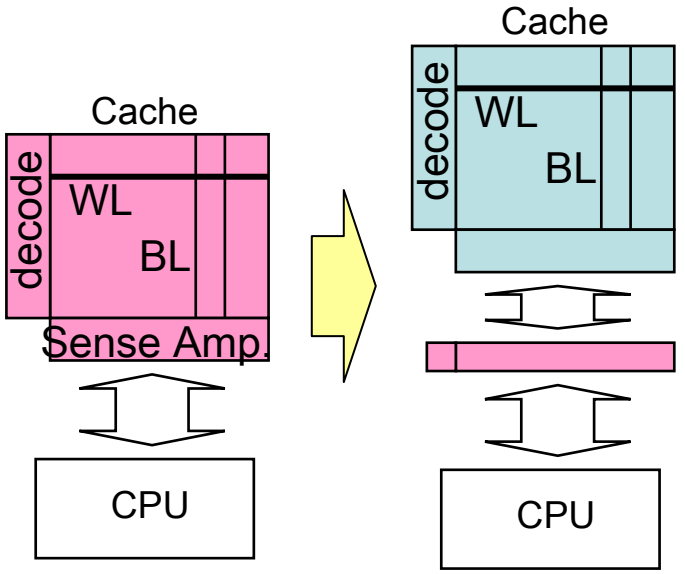


本当に「ホットパス」は存在するのか？



低消費電力化への応用
～命令キャッシュ～

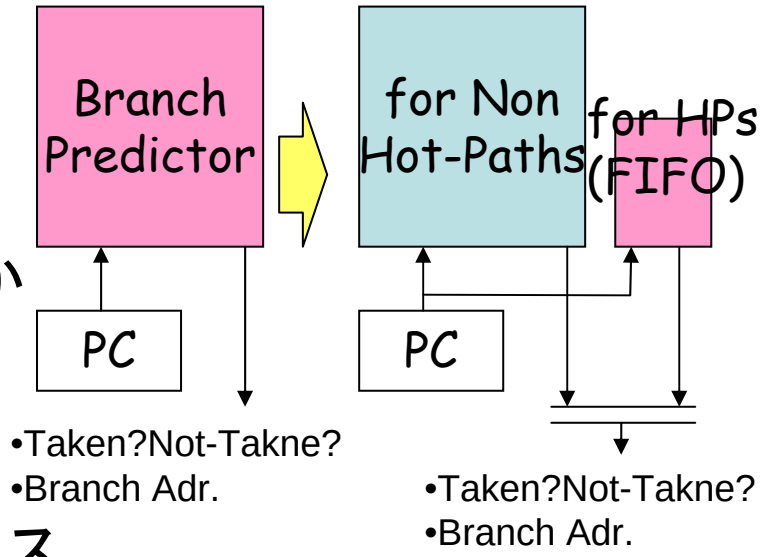
- Divide
  - 命令キャッシュを空間分割
    - 小容量かつ低消費電力なキャッシュ領域
    - 大容量かつ消費電力の大きいキャッシュ領域
- Select/Allocate
  - ホットパス上の命令を小容量キャッシュに割当て
- Reduce
  - 大容量キャッシュ（大きなC_L）でのスイッチング頻度（$f_{0\rightarrow1}$）を削減



[Kin97MICRO] The filter cache: an energy efficient memory structure
[Bellas98ISLPED] Architectural and Compiler Support for Energy Reduction in the Memory Hierarchy of High Performance Microprocessors
[Bellas99ISLPED] Using dynamic cache management techniques to reduce energy in a high-performance processor

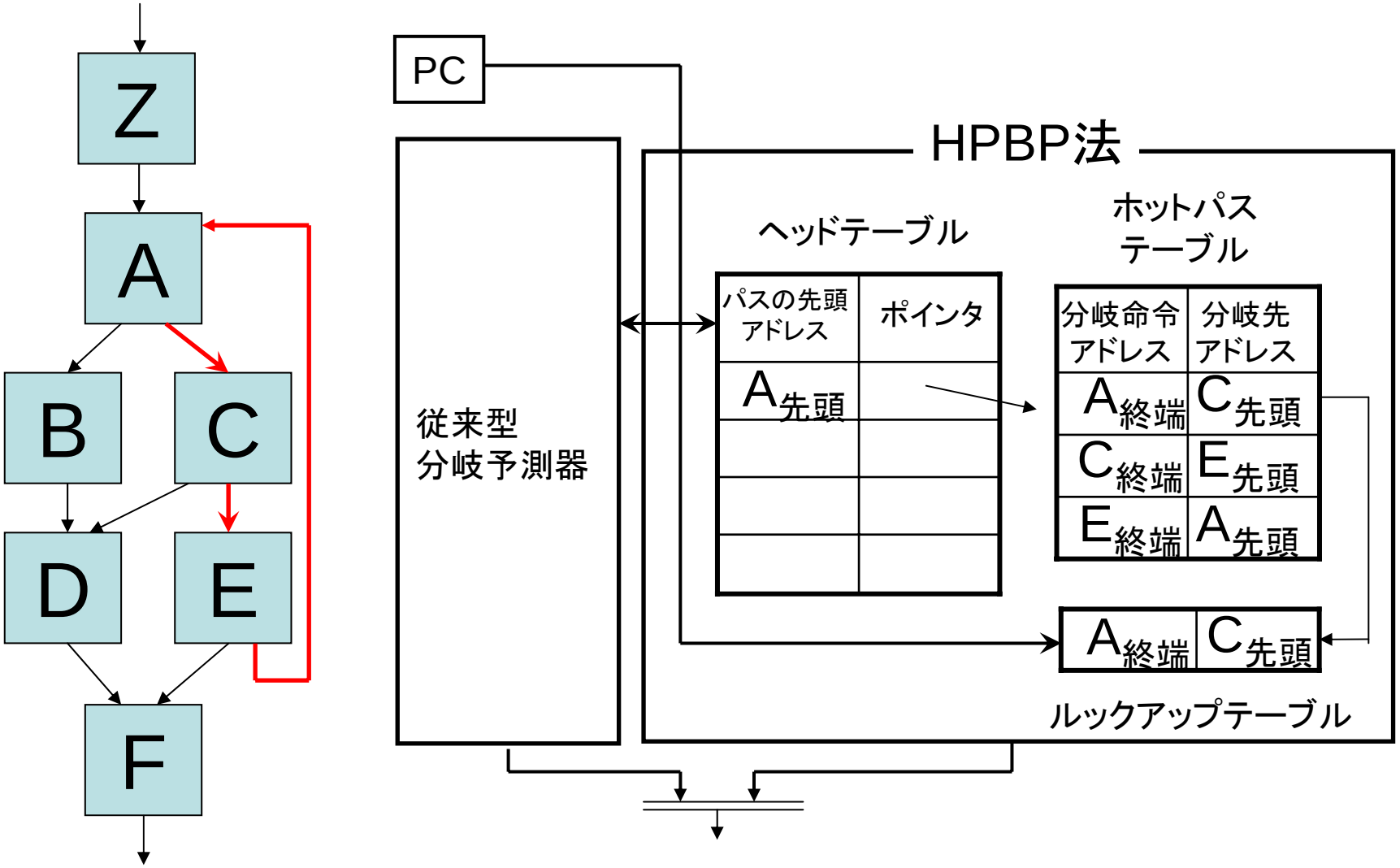
低消費電力化への応用
～分岐予測器～

- Divide
  - 分岐予測器を空間分割
    - 小容量かつ単純な分岐予測器
    - 大容量かつ複雑な分岐予測器
- Select/Allocate
  - ホットパス上の分岐命令を小容量かつ単純な分岐予測器に割当て
- Reduce
  - 大容量分岐予測器（大きなC_L）でのスイッチング頻度（$f_{0\rightarrow1}$）を削減

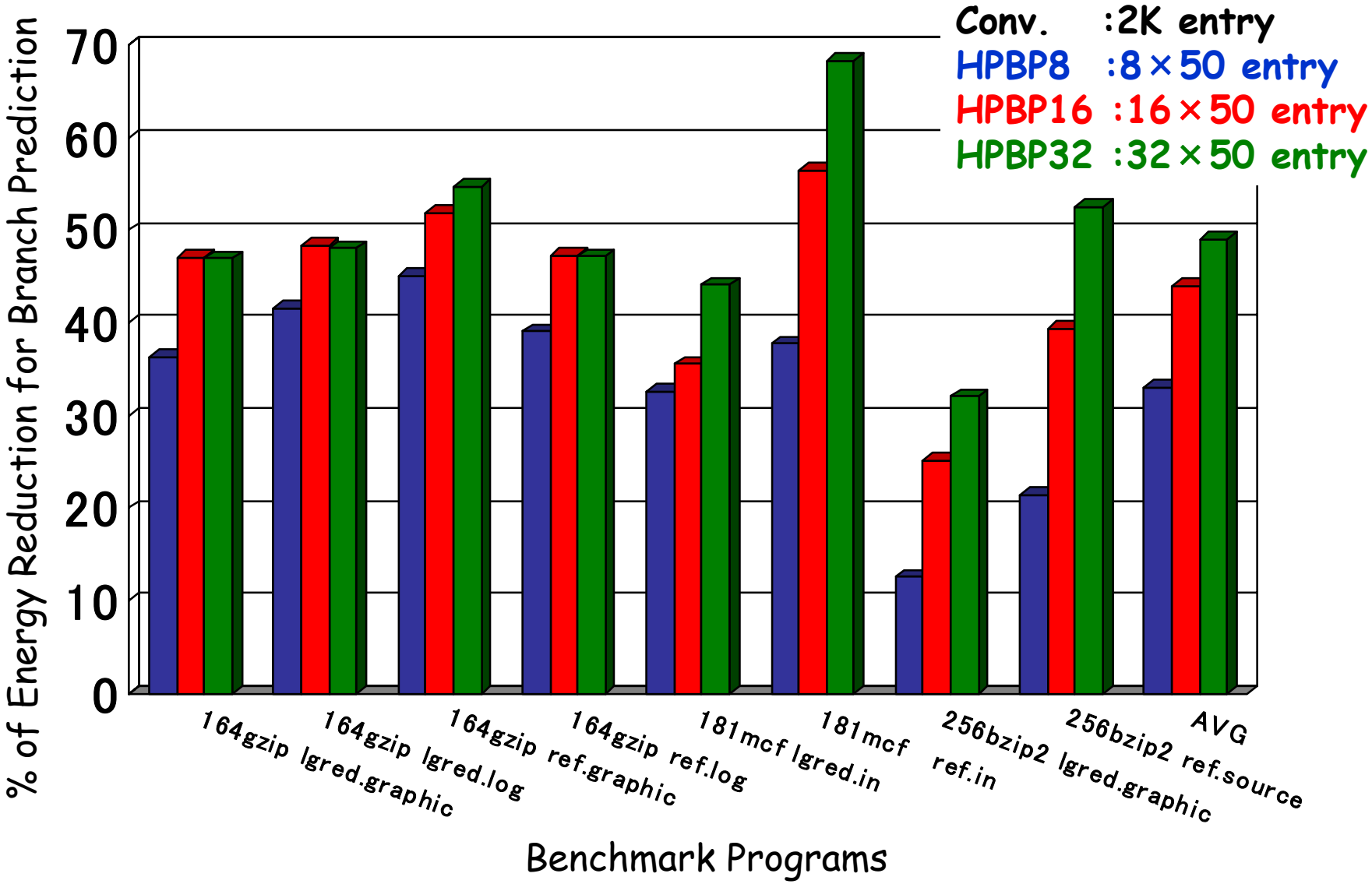


[築地06IPSJ-ARC]プログラムの実行経路の偏りに着目した分岐予測法

Hot-Path based Branch Prediction (HPBP)



HPBPの消費エネルギー削減能力は?

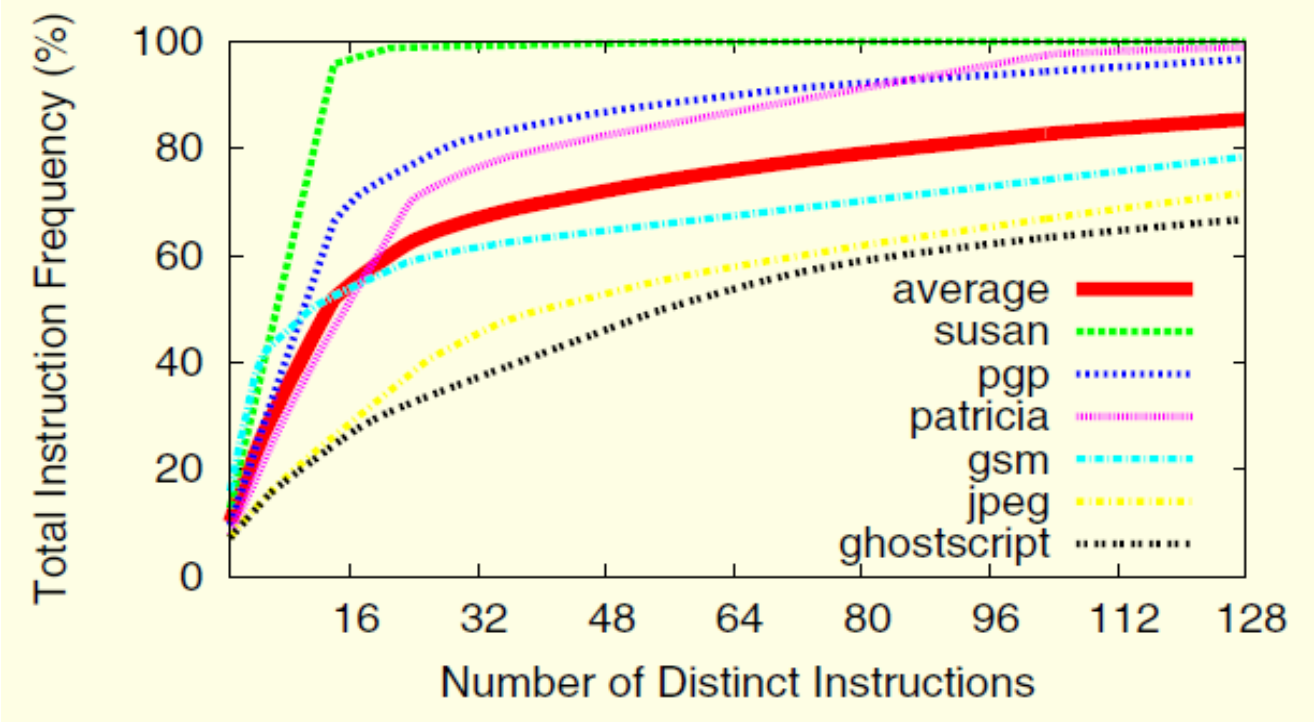


講演内容

- マイクロプロセッサのトレンド
- 低消費電力化に向けた基本戦略
- TIPS:プログラムの実行において・・・
  - 「頻繁に実行される命令列」が存在する！
  - 「頻繁に実行される命令の種類」は少ない！
  - 「急いで実行すべき場合」と「そうでない場合」が存在する！
  - 「演算に必要なビット幅」は8～16ビットと小さい！
  - 「最近参照されたデータ」は再び参照される！
- 今後着目すべきは？

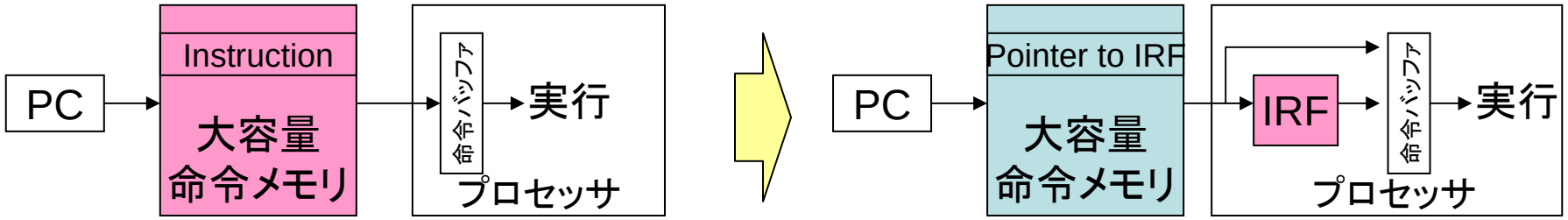
よく実行される命令の種類は少ない！

- 32種類の命令が全実行を60%を占める！
  - MIPS-base ISA, MiBench



低消費電力化への応用
～命令メモリ～

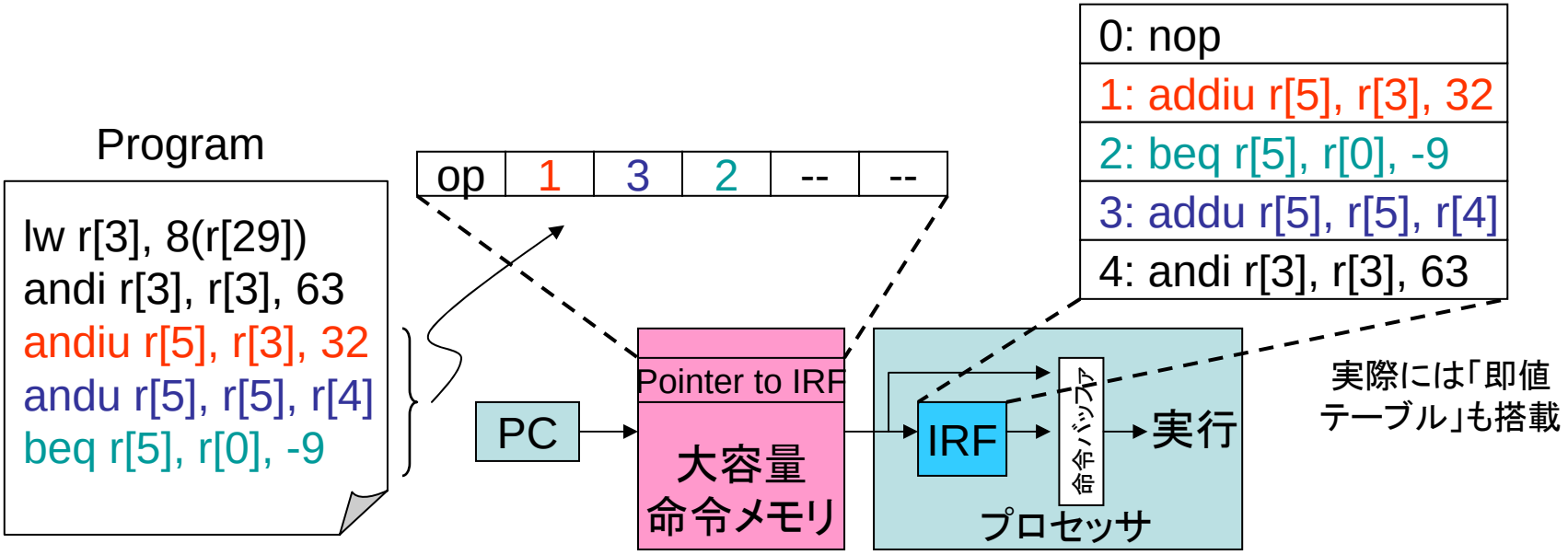
- Divide
  - 命令メモリを少なくとも以下の2つに空間分割
    - 頻繁にアクセスされる小容量メモリ領域（命令レジスタ・ファイル）
    - アクセス頻度の低い大容量メモリ領域（通常の命令メモリ）
- Allocate/Select
  - 高頻度実行命令を小容量メモリに割当て
- Reduce
  - 大容量キャッシュ（大きなC_L）でのスイッチング頻度（$f_{0\rightarrow1}$）を削減



[Hines05ISCA] Improving Program Efficiency by Packing Instructions into Registers

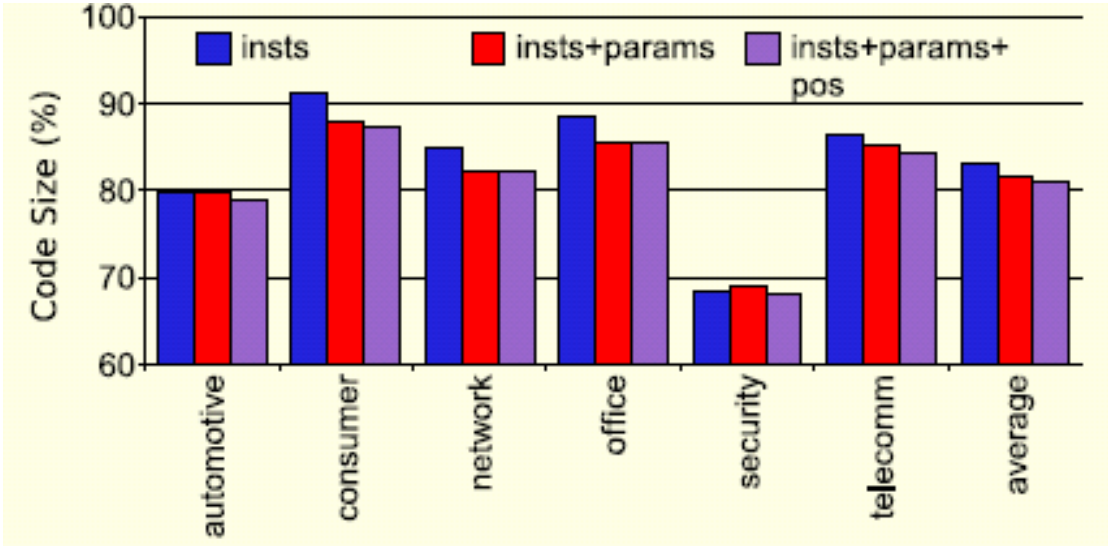
IRFを効率よく利用する！
～命令のパッキング～

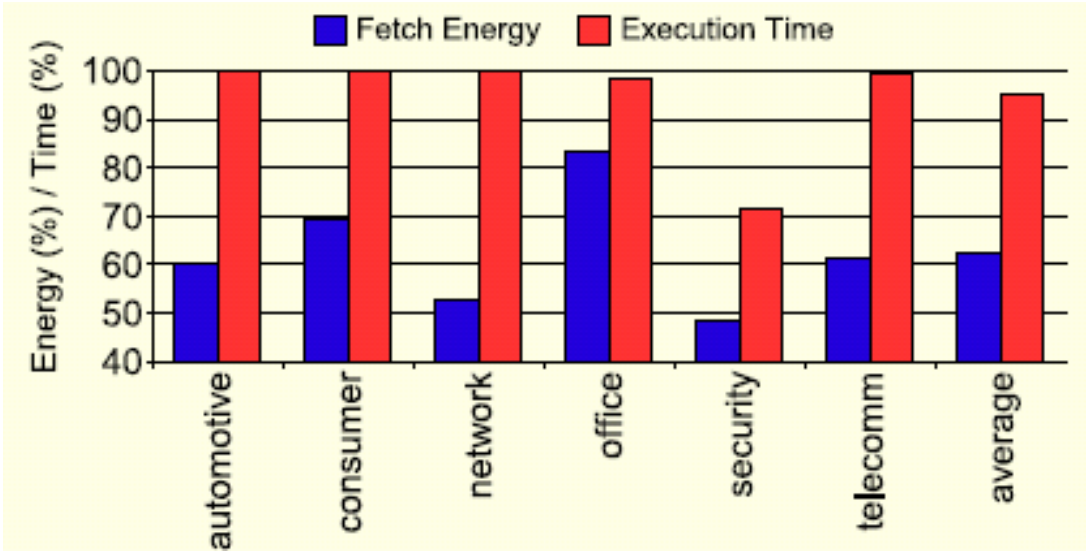
- 小容量メモリ（IRF）に格納すべき命令
  - よく実行される命令そのもの
- 大容量メモリにする命令
  - IRF内の命令へのポインタ
  - １個の命令で「複数個のIRF内命令」を指定



命令レジスタファイルの3つの削減効果

- コードサイズ
  - 20%程度
- 消費エネルギー
  - 40%程度
- 実行時間
  - 5%程度





講演内容

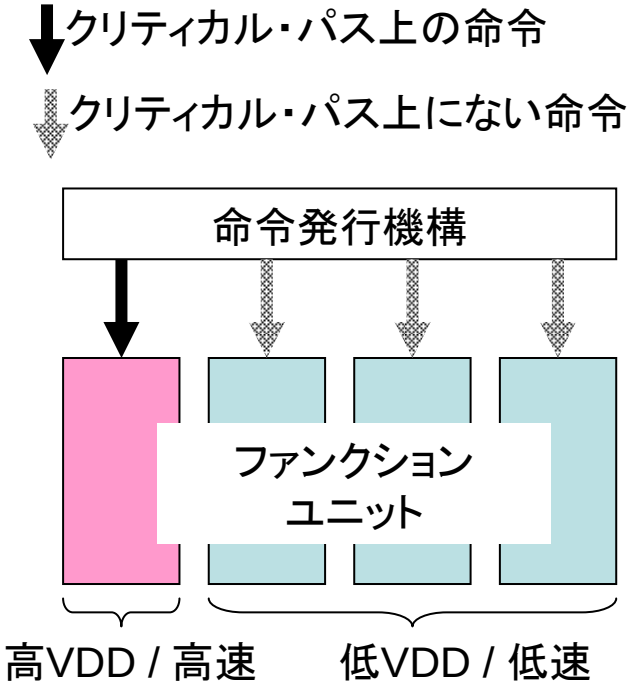
- マイクロプロセッサのトレンド
- 低消費電力化に向けた基本戦略
- TIPS:プログラムの実行において・・・
  - 「頻繁に実行される命令列」が存在する！
  - 「頻繁に実行される命令の種類」は少ない！
  - 「急いで実行すべき場合」と「そうでない場合」が存在する！
  - 「演算に必要なビット幅」は8～16ビットと小さい！
  - 「最近参照されたデータ」は再び参照される！
- 今後着目すべきは？

なぜ，急いで実行する必要がないのか？

- （特に組込みシステムにおいて）リアルタイム性を十分保障できる場合
- プログラム実行における「クリティカルパス」の上には乗っていない命令
- キャッシュミスに伴うオフチップ・アクセスが頻発する場合
- などなど

低消費電力化への応用
～ファンクション・ユニット（ALU）～

- Divide
  - 演算ユニットを少なくとも以下の2つに空間分割
    - 高速かつ高消費電力（高いV_{DD}）
    - 低速かつ低消費電力（低いV_{DD}）
- Allocate/Select
  - 命令実行の緊急度が低い場合は「低速演算ユニット」を使用
  - 緊急度は動的に検出
- Reduce
  - 「急ぐ必要のない」演算におけるV_{DD}とFを削減

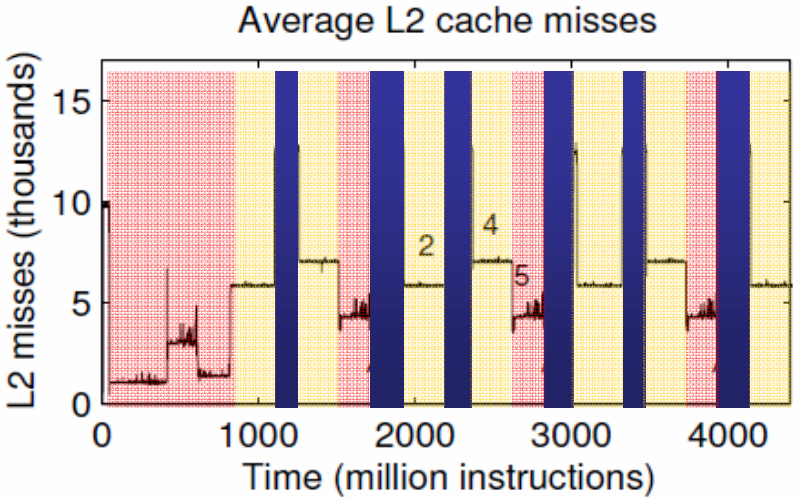


緊急度の低い命令の検出
- 消費者が少ない命令
- 高いスラックを有する命令
- などなど

[Chiyonobu03PACT] On Identifying Instruction Criticality for Energy-Aware Applications
[福山05SACSIS]スラック予測を用いた省電力アーキテクチャ向け命令スケジューリング
[小林05SACSIS]発見的手法に基づくローカルスラック予測機構

低消費電力化への応用
～キャッシュミスに基づくDVFS制御～

- Divide
  - プログラム実行を少なくとも以下の2つに時間分割
    - キャッシュミスが頻発する区間
    - キャッシュミスが頻発しない区間
- Allocate/Select
  - キャッシュミスが頻発する区間においては低い電源電圧と低い動作周波数を割当て
- Reduce
  - 「急ぐ必要のない」演算におけるV_{DD}とFを削減



L2ミス発生状況（SPEC2000 173.applu.）

- キャッシュミス発生状況をモニタリング
- V/F制御命令を実行時にコードへ挿入（コード変換）
- FPプログラムで70%の消費エネルギー削減，性能低下はわずか0.5%

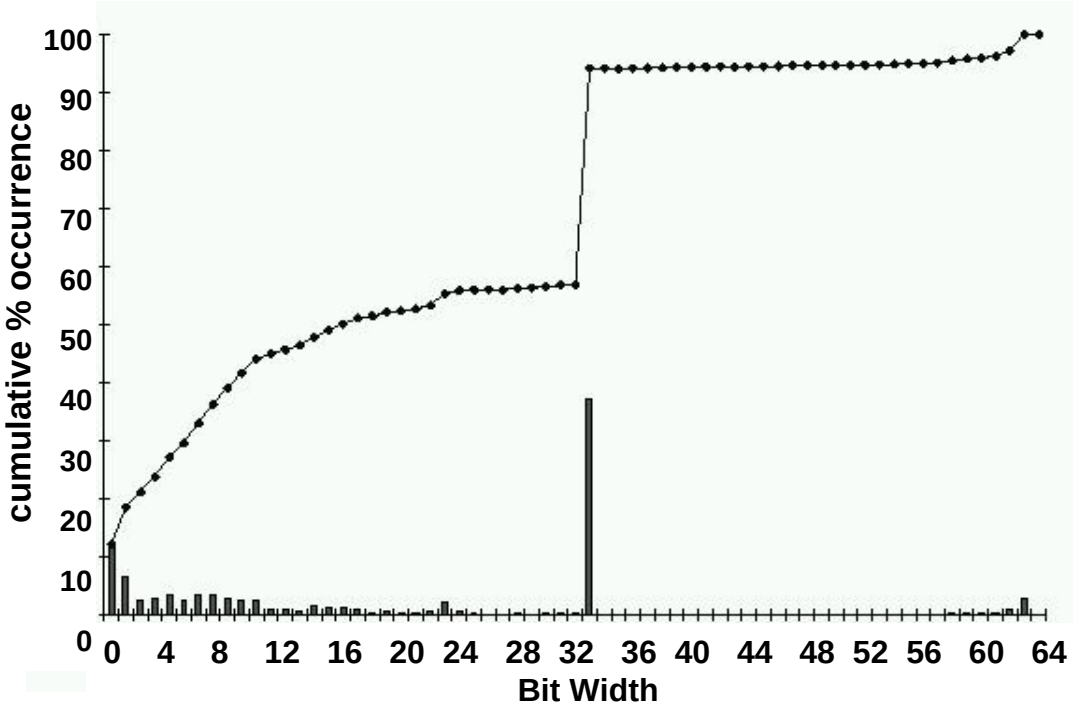
[Wu05MICRO] A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance

講演内容

- マイクロプロセッサのトレンド
- 低消費電力化に向けた基本戦略
- TIPS：プログラムの実行において・・・
  - 「頻繁に実行される命令列」が存在する！
  - 「頻繁に実行される命令の種類」は少ない！
  - 「急いで実行すべき場合」と「そうでない場合」が存在する！
  - 「演算に必要なビット幅」は8～16ビットと小さい！
  - 「最近参照されたデータ」は再び参照される！
- 今後着目すべきは？

演算に必要なビット幅は以外と小さい!

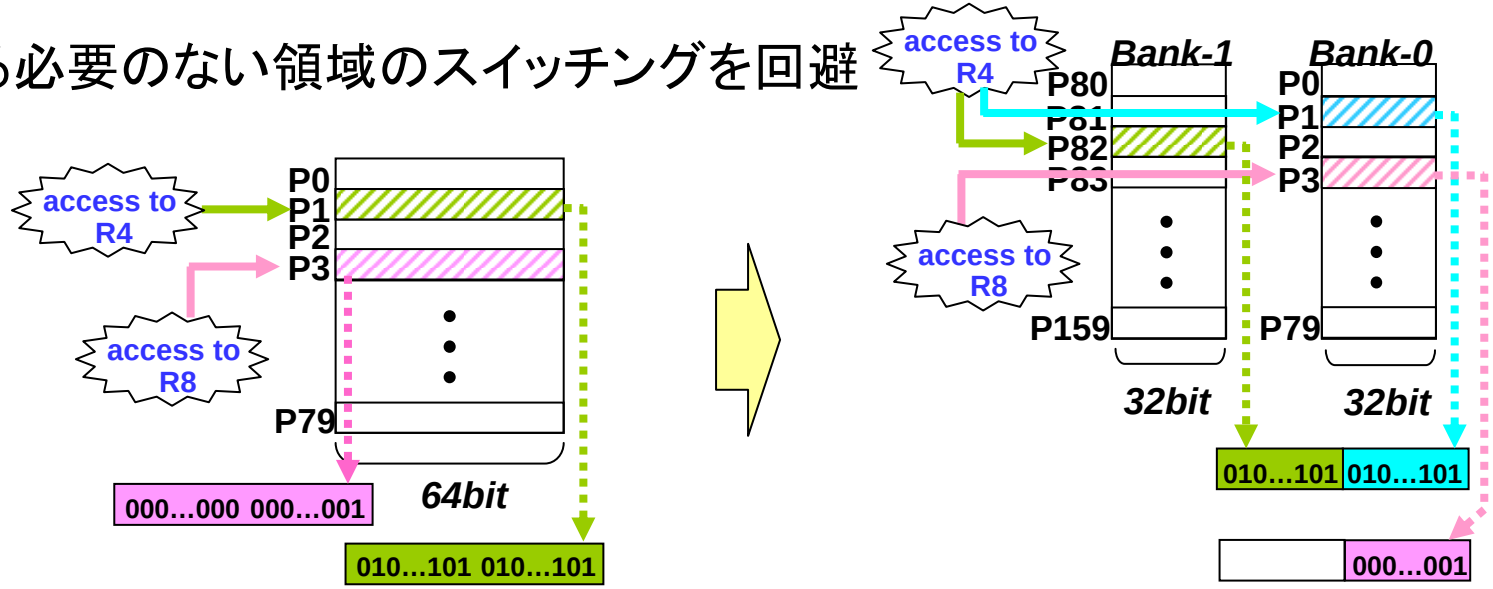
- Narrow Bit-Width（NBW）
  - プログラム実行において、約50%の命令は16ビット以下のオペランドに対する演算



64-bit Alpha ISA、SPECint95全プログラム平均

低消費電力化への応用
～レジスタ･ファイル～

- Divide
  - レジスタファイルを少なくとも以下の2つに空間分割
    - 上位ハーフワード用レジスタ･バンク
    - 下位ハーフワード用レジスタ･バンク
- Allocate/Select
  - 記憶すべきデータの必要ビット幅に基づき使用するレジスタバンクを選択
- Reduce
  - 動作する必要のない領域のスイッチングを回避

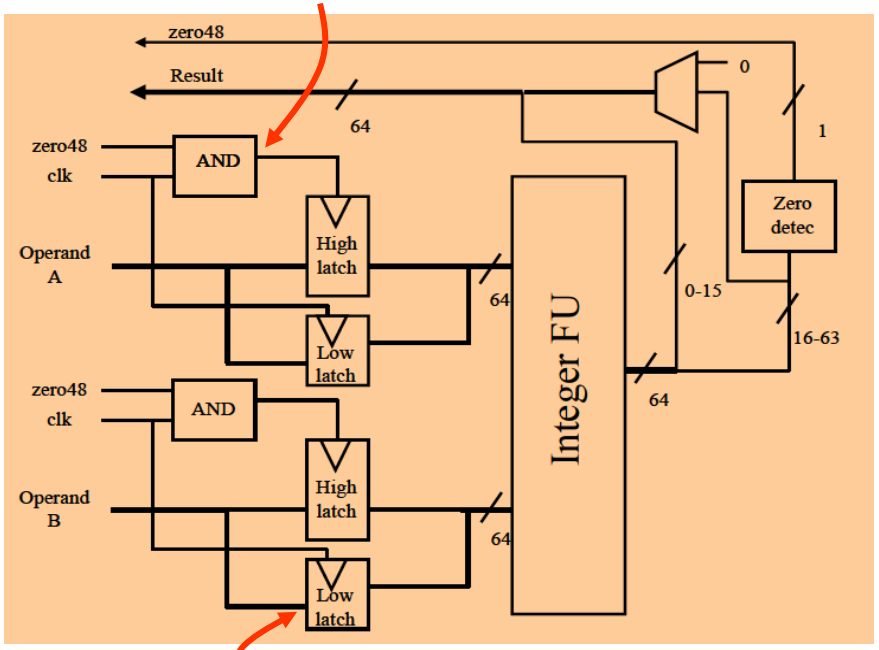


[Kondo05HPCA] A Small, Fast and Low-Power Register File by Bit-Partitioning

低消費電力化への応用
～ファンクション・ユニット～

- Divide
  - 演算ユニットを少なくとも以下の2つに空間分割
    - 上位ビット（例えばハーフワード）演算用の回路
    - 下位ビット演算用の回路
- Allocate/Select
  - 必要に応じて上位用演算回路のクロック供給を停止（クロックゲーティング）
  - オペランド値における有効なビット幅を動的に検出
- Reduce
  - 動作する必要のない領域のスイッチングを回避

上位48ビット用FFへの
クロック供給を停止

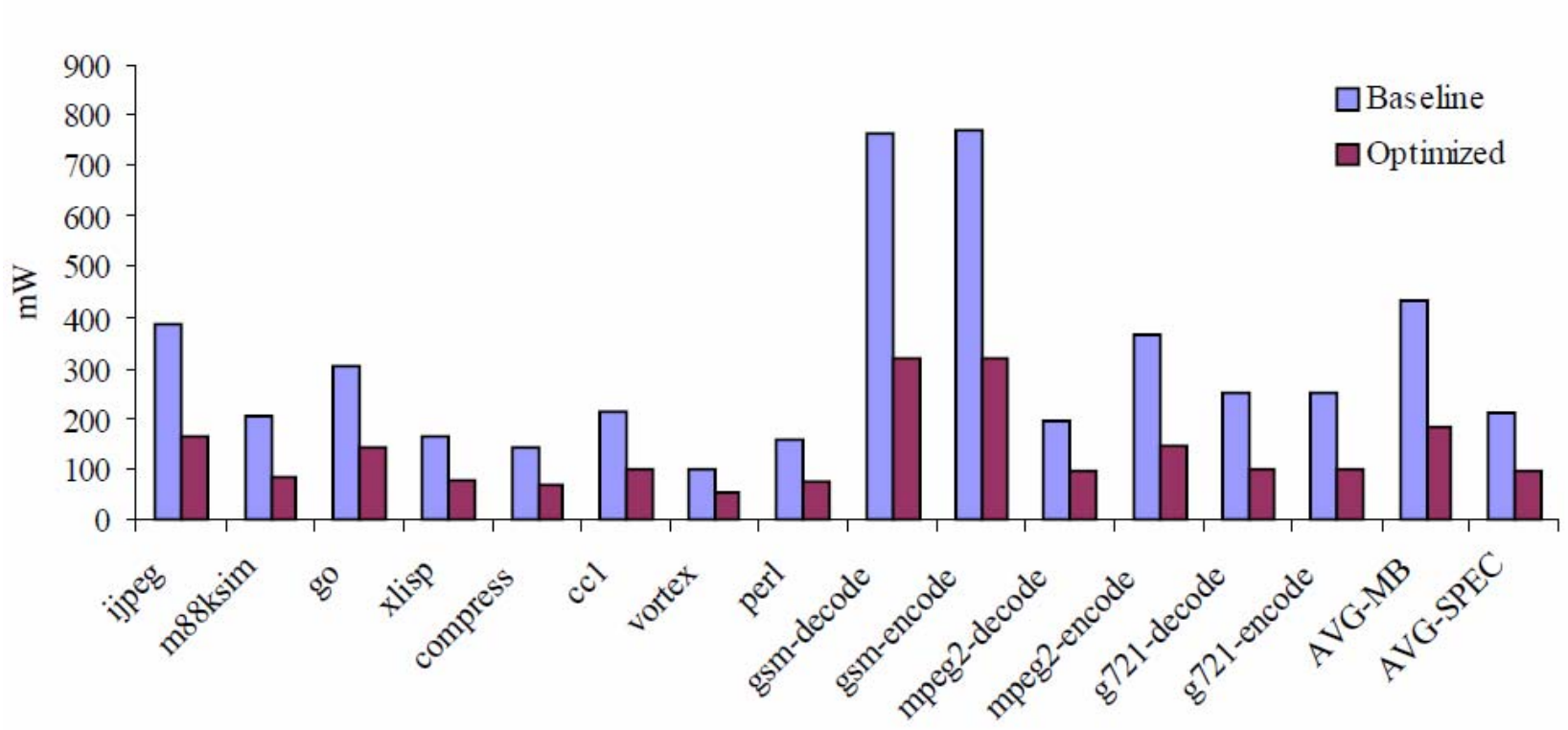


下位16ビット
は常に動作

[Brooks99HPCA] Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance

NBWの活用による消費電力削減効果

- 整数演算ユニットでの消費電力
- 約50%～60%の消費電力削減



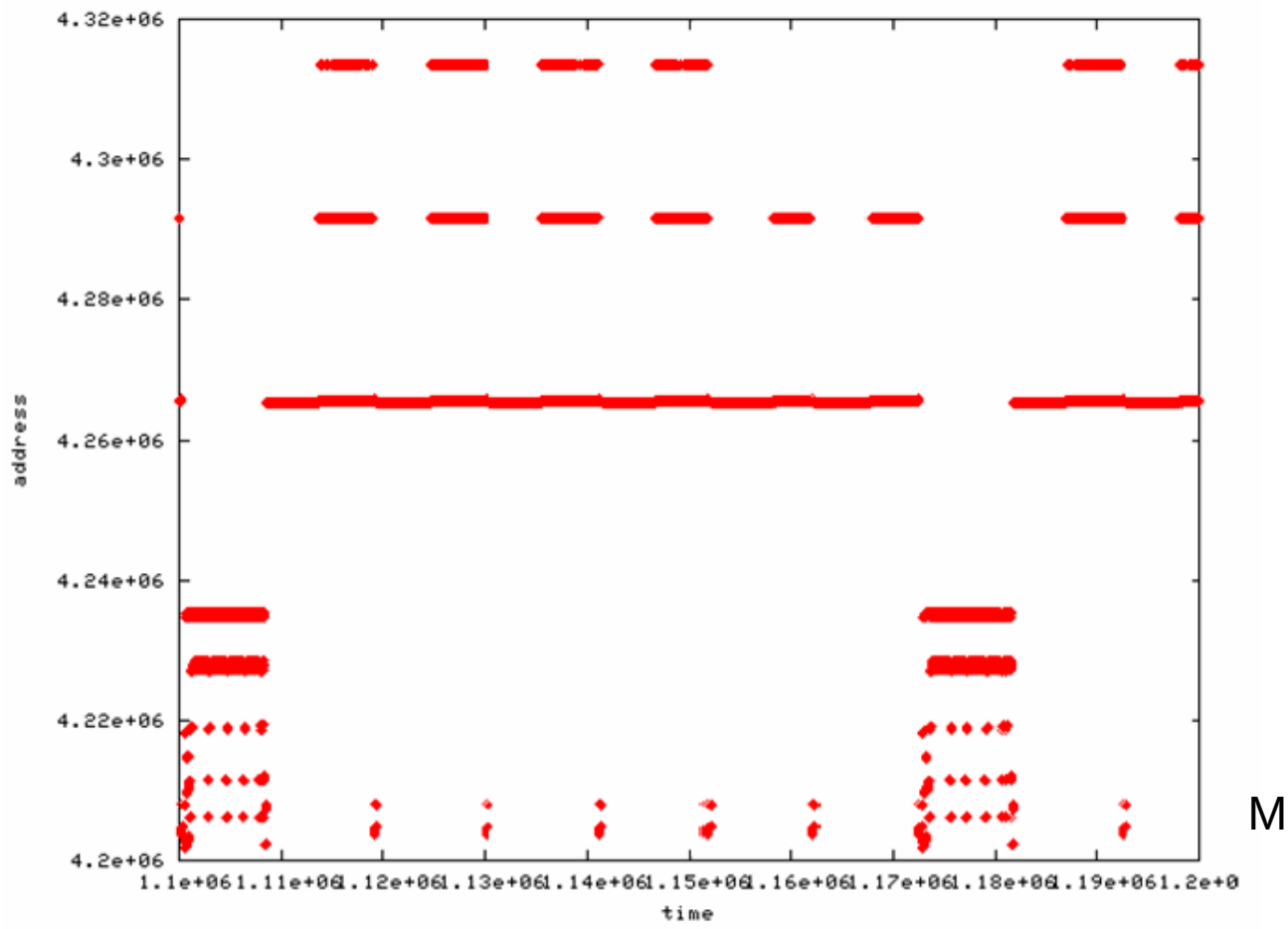
[Brooks99HPCA] Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance

講演内容

- マイクロプロセッサのトレンド
- 低消費電力化に向けた基本戦略
- TIPS:プログラムの実行において・・・
  - 「頻繁に実行される命令列」が存在する！
  - 「頻繁に実行される命令の種類」は少ない！
  - 「急いで実行すべき場合」と「そうでない場合」が存在する！
  - 「演算に必要なビット幅」は8～16ビットと小さい！
  - 「最近参照されたデータ」は再び参照される！
- 今後着目すべきは？

参照されたデータは近い将来参照される!
（参照されてないデータは当分参照されない）

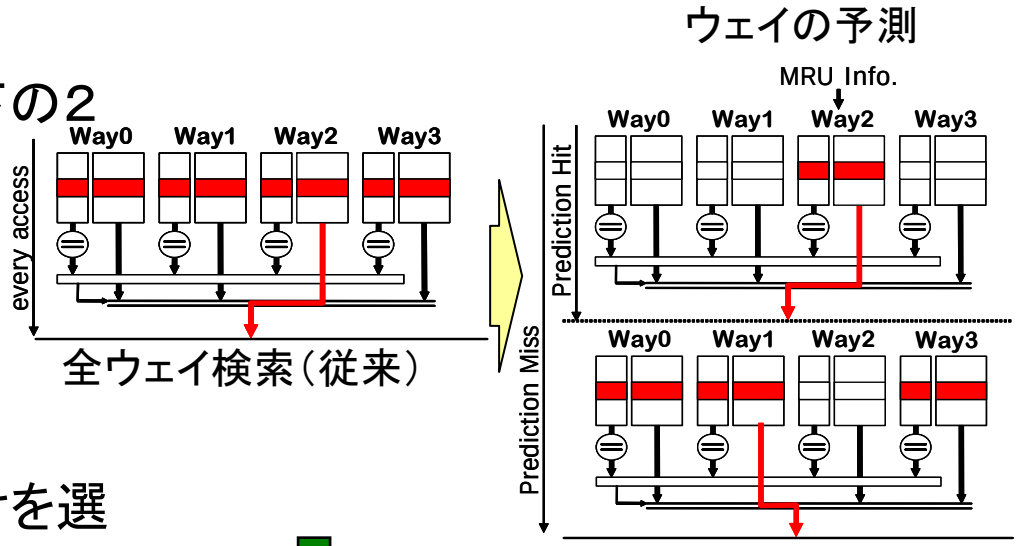
- いわゆる，メモリ参照の時間局所性



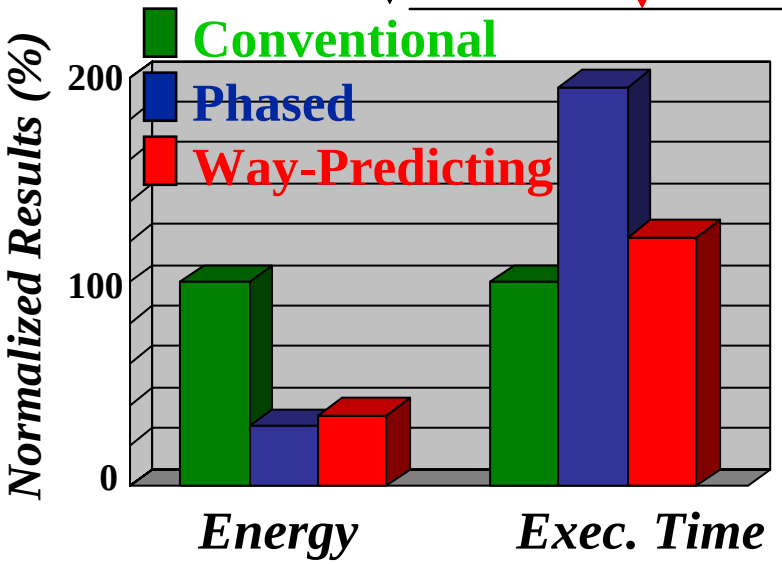
MPEG2

低消費電力化への応用
～キャッシュメモリ～

- Divide
  - キャッシュ・メモリを少なくとも以下の２つに空間分割
    - アクセスする必要のある領域
    - アクセスする必要のない領域

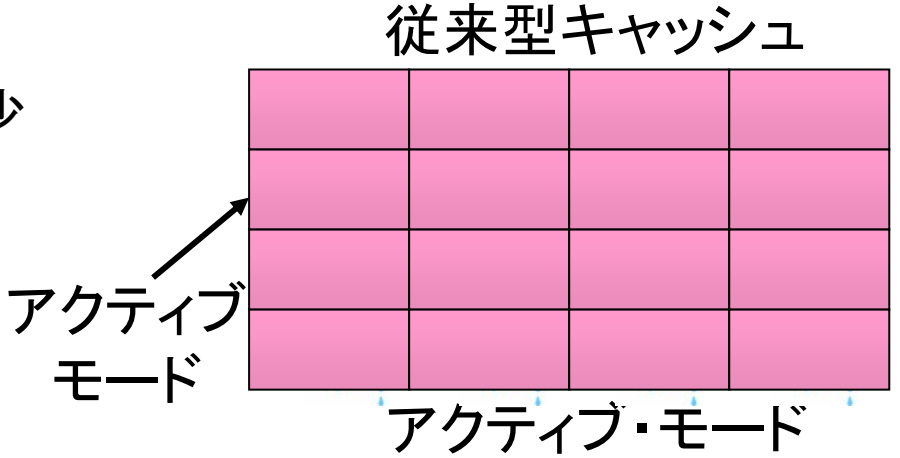


- Allocate/Select
  - アクセスする必要のある領域だけを選択して活性化
    - ウェイ予測（アクセスする必要のある領域を予測）
    - MRU（Most Recently Used）アルゴリズム：局所性の活用
- Reduce
  - アクセスする必要のない領域のスイッチングを回避

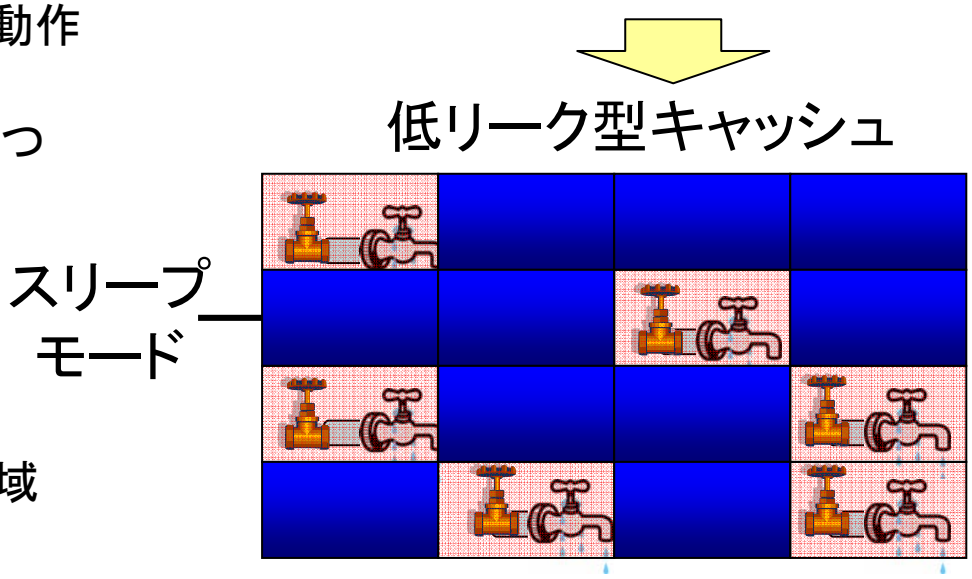


低消費電力化への応用
～キャッシュメモリ(リーク消費電力)～

- Divide
  - キャッシュ・メモリをある単位に空間分割
  - 各領域に対し，プログラム実行において少なくとも以下の2つに時間分割
    - プロセッサが必要とするデータを記憶している区間
    - プロセッサから必要とされない区間



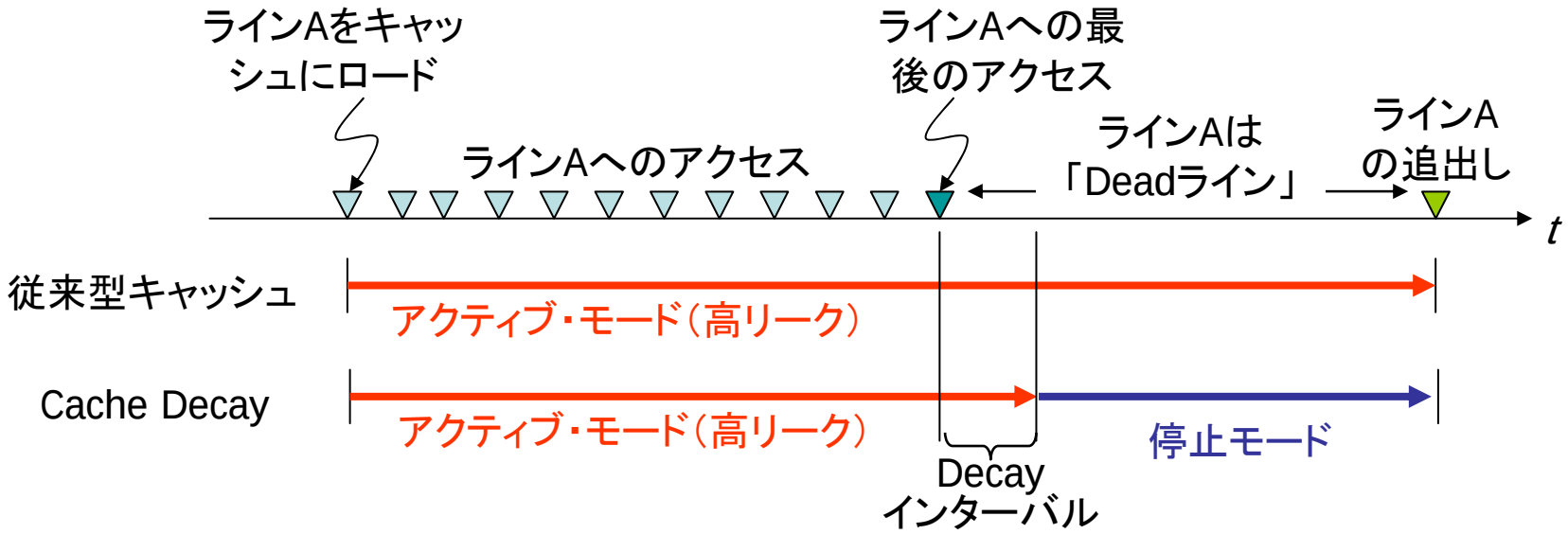
- Allocate/Select
  - プロセッサから参照される領域は通常動作
    - アクティブ・モード
  - 必要とされない区間において低性能かつ低リークモードを割当て
    - スリープ・モード(状態保持)
    - 停止モード・モード(状態破壊)



- Reduce
  - プロセッサから必要とされないメモリ領域でのリークを削減

Cache Decay
～使用済みになったら寝かせる～

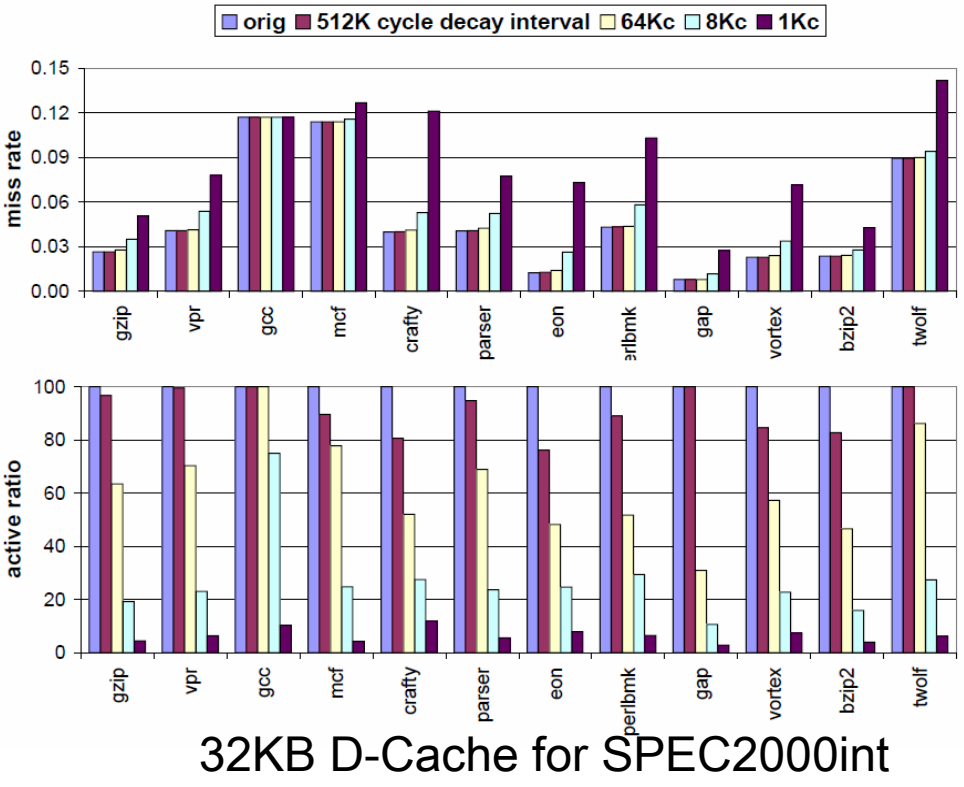
- 基本アプローチ
  - モード切替可能な領域：キャッシュ・ライン
  - 動作：アクティブ・モードと停止モード（状態破壊）
  - モード切替
    - →アクティブ・モード：キャッシュへのラインロード時
    - →停止モード：一定期間参照が無い場合（ラインの衰退）



[Kaxiras01ISCA] Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power

Cache Decay のリーク削減効果

- 効果はDecayインターバルに大きく依存
  - 「如何にてdeadラインを正しく検出するか」が重要
  - 1Kサイクルの場合は90%以上のリーク削減の見込み
- 最適なDecayインターバルはアプリケーションによって異なる（性能制約条件下）
  - Decayインターバルを動的に調整する研究もある



32KB D-Cache for SPEC2000int

講演内容

- マイクロプロセッサのトレンド
- 低消費電力化に向けた基本戦略
- TIPS:プログラムの実行において・・・
  - 「頻繁に実行される命令列」が存在する！
  - 「頻繁に実行される命令の種類」は少ない！
  - 「急いで実行すべき場合」と「そうでない場合」が存在する！
  - 「演算に必要なビット幅」は8～16ビットと小さい！
  - 「最近参照されたデータ」は再び参照される！
- 今後着目すべきは？

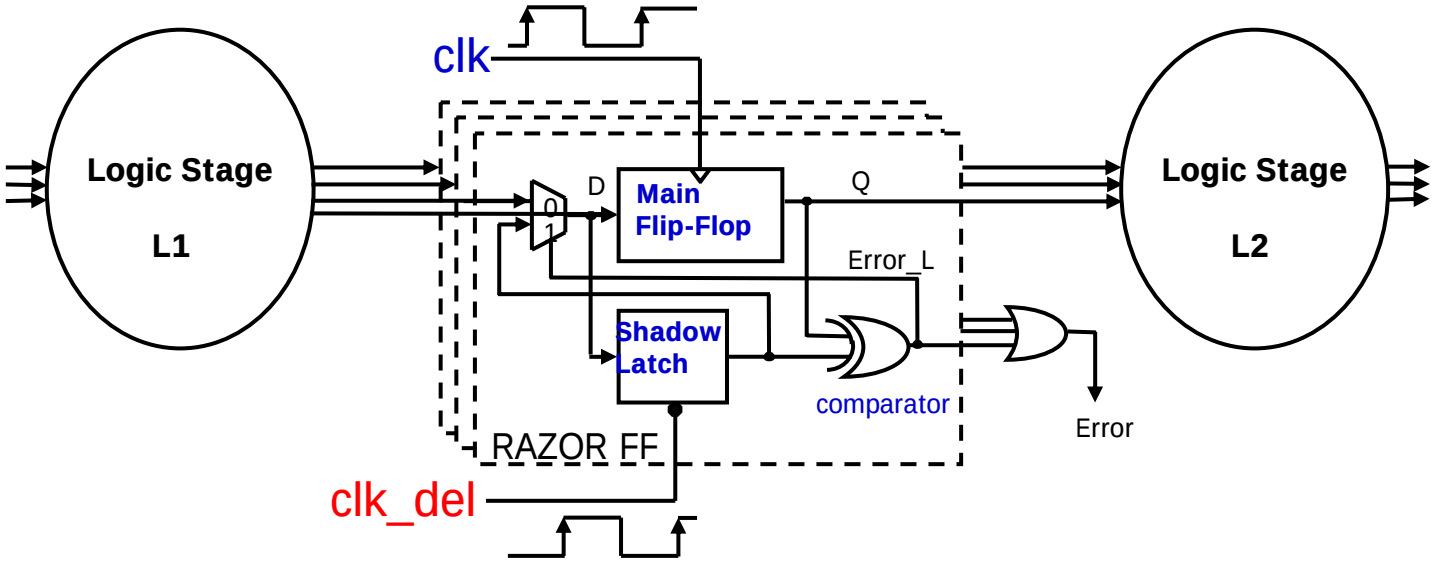
様々な「パラダイムシフト」

- 「シングルコア」から「マルチコア」の世界へ
  - 複数コアにおけるV/F制御など
- 「オフチップ主記憶」から「オンチップ主記憶」の世界へ
  - DRAM貼付け技術の実用化
  - 高オンチップ・メモリバンド幅の効率的な活用
- 「固定」から「可変」の世界へ
- 「汎用」から「専用（アクセラレータ）」の世界へ
  - 正確には，汎用＋専用アクセラレータ
- 「性能と消費電力」から「性能と消費電力と信頼性/安全性」の世界へ

信頼性と消費電力/消費エネルギー

「高信頼化技術」により消費電力を削減する！ ～Razor～

- 回路動作が不安定になっても「低電圧化」を進める
  - つまり，ある程度のエラー（誤り）発生を許す
- ただし，高信頼化技術によりこのエラーを回復！



[Ernst03MICRO] Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation

信頼性と消費電力のトレードオフを考える!

- Adaptive Error Protection
  - キャッシュ・メモリの高信頼化と低消費エネルギー化
  - メモリでのソフトエラー対策
    - SED（Single Error Detection）⇒低い消費電力
      - ただし，誤り訂正にはバックアップ・データが必要
    - SEC（Single Error Correction）⇒高い消費電力
  - これら2方式を保護対象データの状態に応じて使い分け

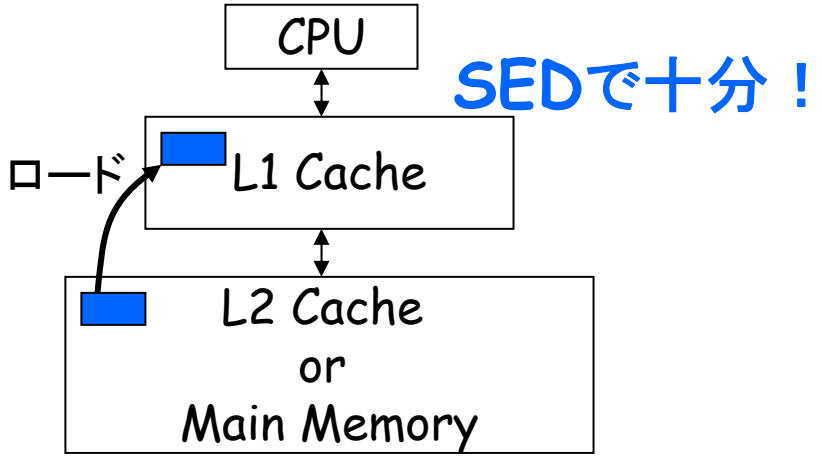
	Power (mW)		Delay (ns)	
	Coding	Decoding	Coding	Decoding
SED(Parity)	6.9232	7.2239	1.41	1.41
SEC(Hamming Code)	14.4871	26.2962	1.45	2.66

[Li04ISLPED] Soft Error and Energy Consumption Interactions: A Data Cache perspective

SEDとSECの使い分け

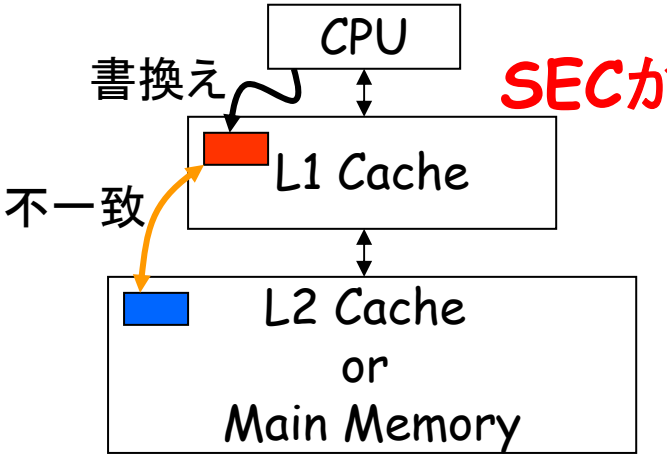
- L1キャッシュに記憶されたデータの状態
  - クリーン：下位記憶階層の対応するデータと一致する
  - ダーティ：下位記憶階層の対応するデータと一致しない
- SEDとSECの使い分け
  - バックアップ・データが下位階層に存在する場合（クリーン）⇒SED
  - 存在しない場合（ダーティ）⇒SEC

L1データがクリーンな状態（未更新）



SEDで十分！

L1データがダーティな状態（更新）

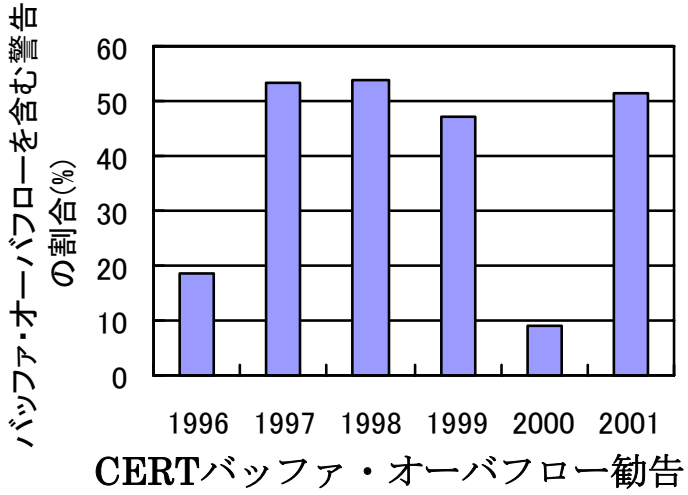


SECが必要！

安全性と消費電力/消費エネルギー

バッファ・オーバフロー攻撃

- バッファ・オーバフロー
  - 最も多く活用される脆弱性の1つ
  - Blaster@2003, CodeRed@2001



CERTバッファ・オーバフロー勧告

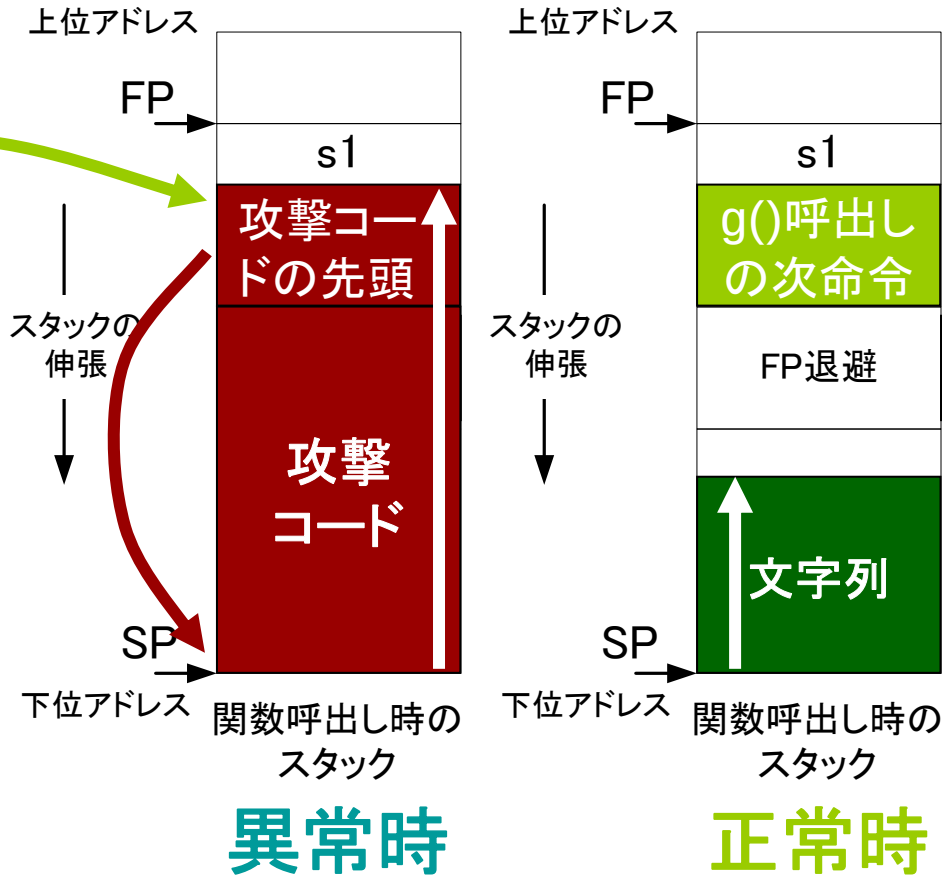
R.B.Lee, D.K.Karig, J.P.McGregor, and Z.Shi, “Enlisting Hardware Architecture to Thwart Malicious Code Injection,” Proc. of the Int. Conf. on Security in Pervasive Computing, Mar. 2003.

- メカニズム
  - データ境界を越えた書込み
    - C標準ライブラリ内に存在（strcpyなど）
  - スタックの破壊（スタック・スマッシング）
    - 悪質コードの挿入と戻りアドレスの改ざん
  - プログラム実行制御の乗っ取り
    - 改ざん後の戻りアドレスがPCへ設定

スタック・スマッシングによる実行制御の乗っ取り

実行コード

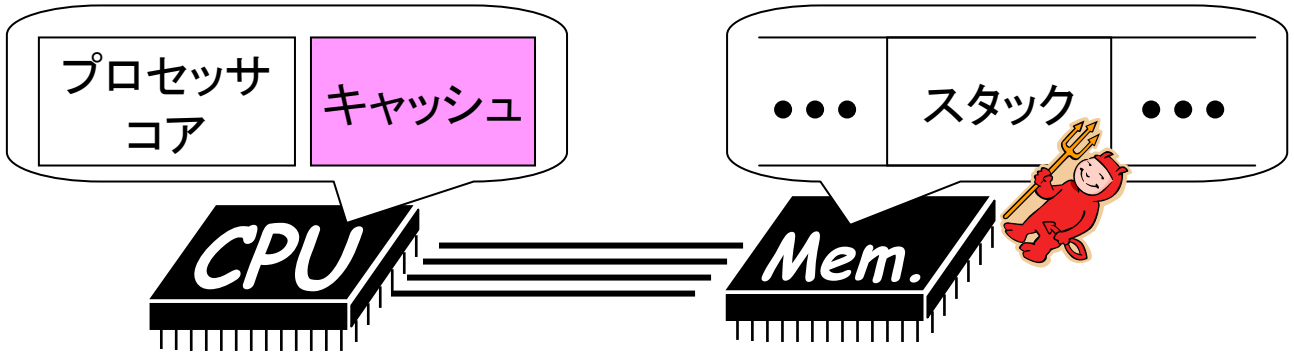
```
int f ( ) {
  …
  g (s1);
  …
}

int g ( char *s1) {
  char buf [10];
  …
  strcpy(buf, s1);
  …
}
```



処理手順

1. 関数f ( )の実行
2. 関数g( )の呼出し
3. 文字列コピー
4. 関数f( )へ復帰

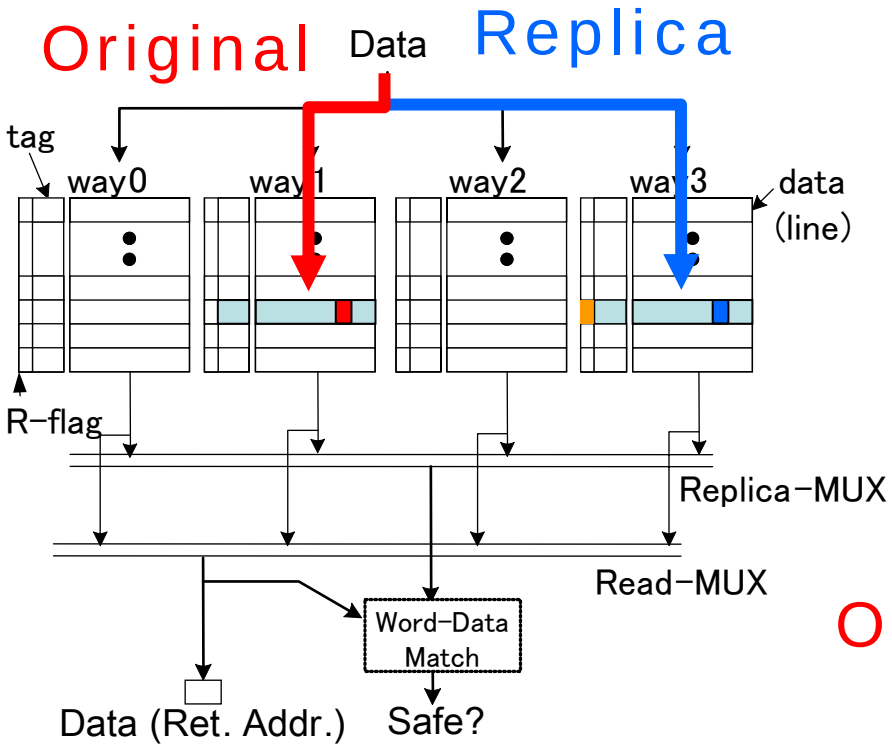
動的な戻りアドレス改ざん検出
～Secure Cache: SCache～



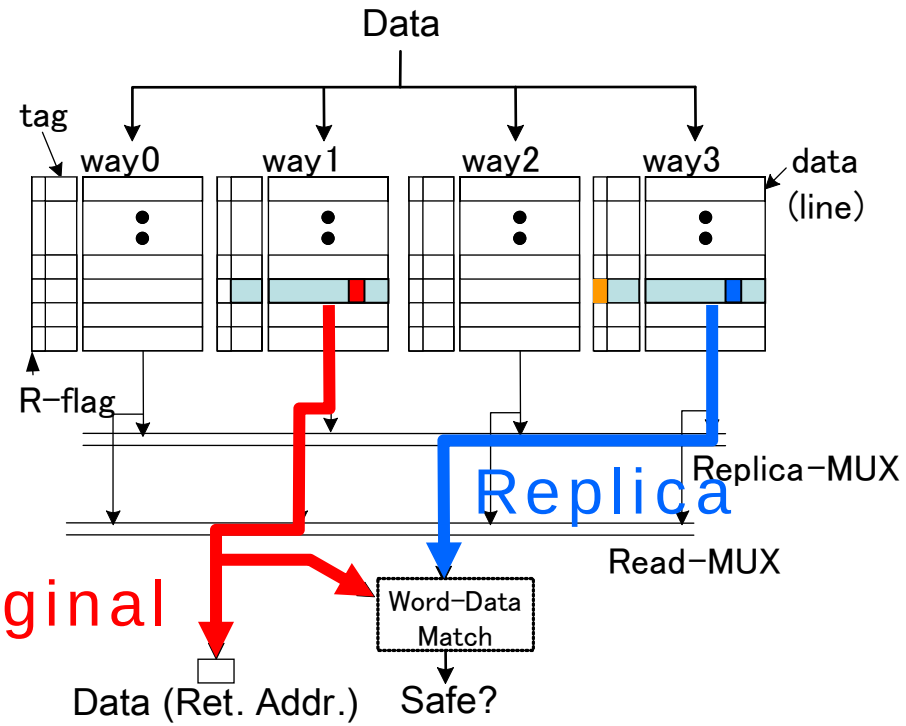
- 問題点:
  - スタックに書込んだ戻りアドレスが改ざんされる
- 解決策:
  - キャッシュで戻りアドレスを保護しよう！
- 手段:
  - 戻りアドレス書込み時に複製（レプリカ・ライン）を作成
  - 戻りアドレス読出し時に複製と比較
  - 不一致であれば戻りアドレス改ざん発生と判定

SCacheの動作

戻りアドレス書込み時



戻りアドレス読出し時



キャッシュ・ヒット時を想定

SCacheの特徴（利点と欠点）

Pros

- 戻りアドレス改ざんの動的検出が可能
  - ただし、レプリカ・ラインが存在する場合のみ
  - プロセッサの内部構成へ与える影響は極めて小さい
  - アクセス時間/面積オーバヘッドは極めて小さい
- レプリカ数を変更可能
  - 安全性と消費エネルギーのバランスを決定可能

Cons

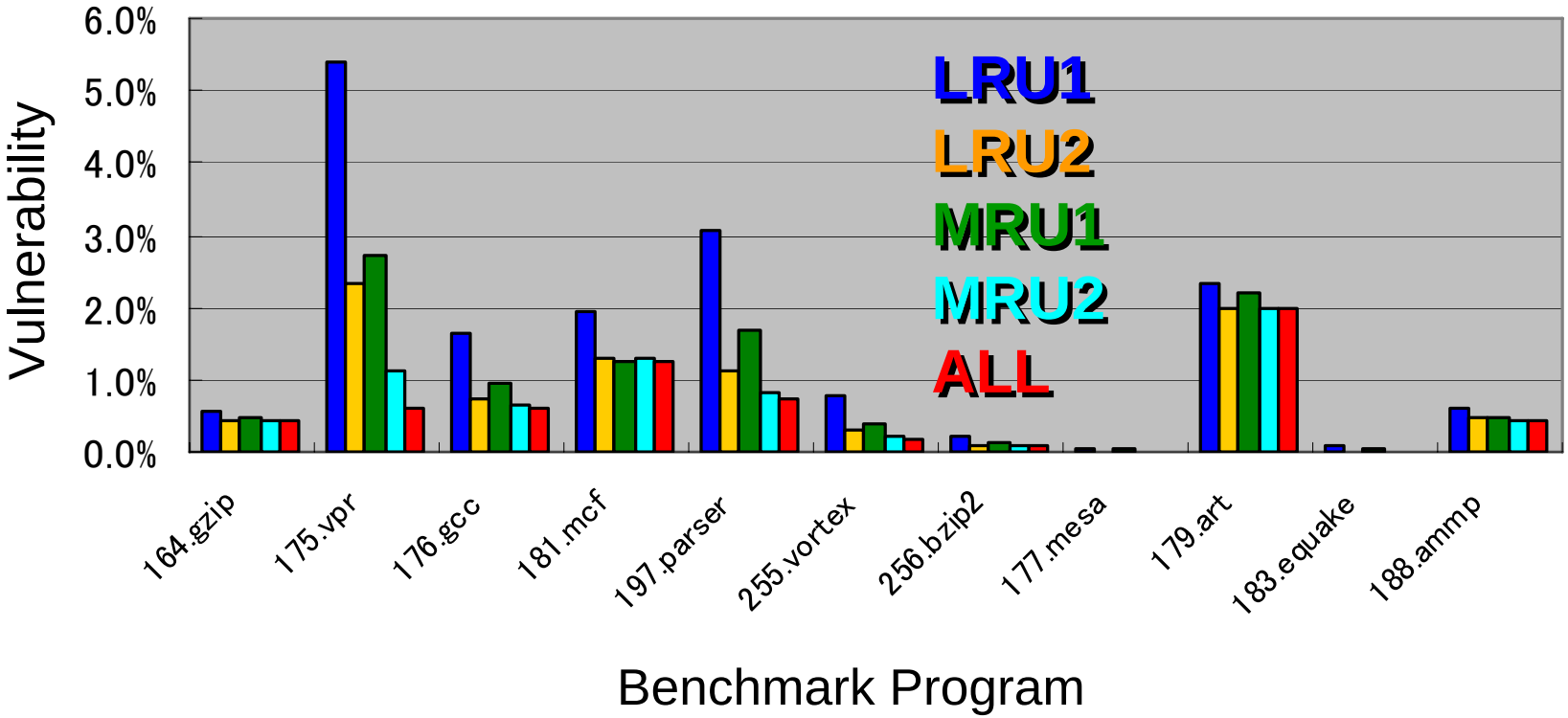
- レプリカ作成に伴うヒット率の低下
  - 平均メモリアクセス時間の増大
  - 下位階層メモリへのアクセス消費エネルギー増大
- レプリカ作成に伴う消費エネルギーの増加
  - 書込みエネルギーの増大
  - ライトバック・エネルギーの増大
  - 読出しエネルギーの増大（他低消費電力キャッシュと比較して）

安全性に関する評価

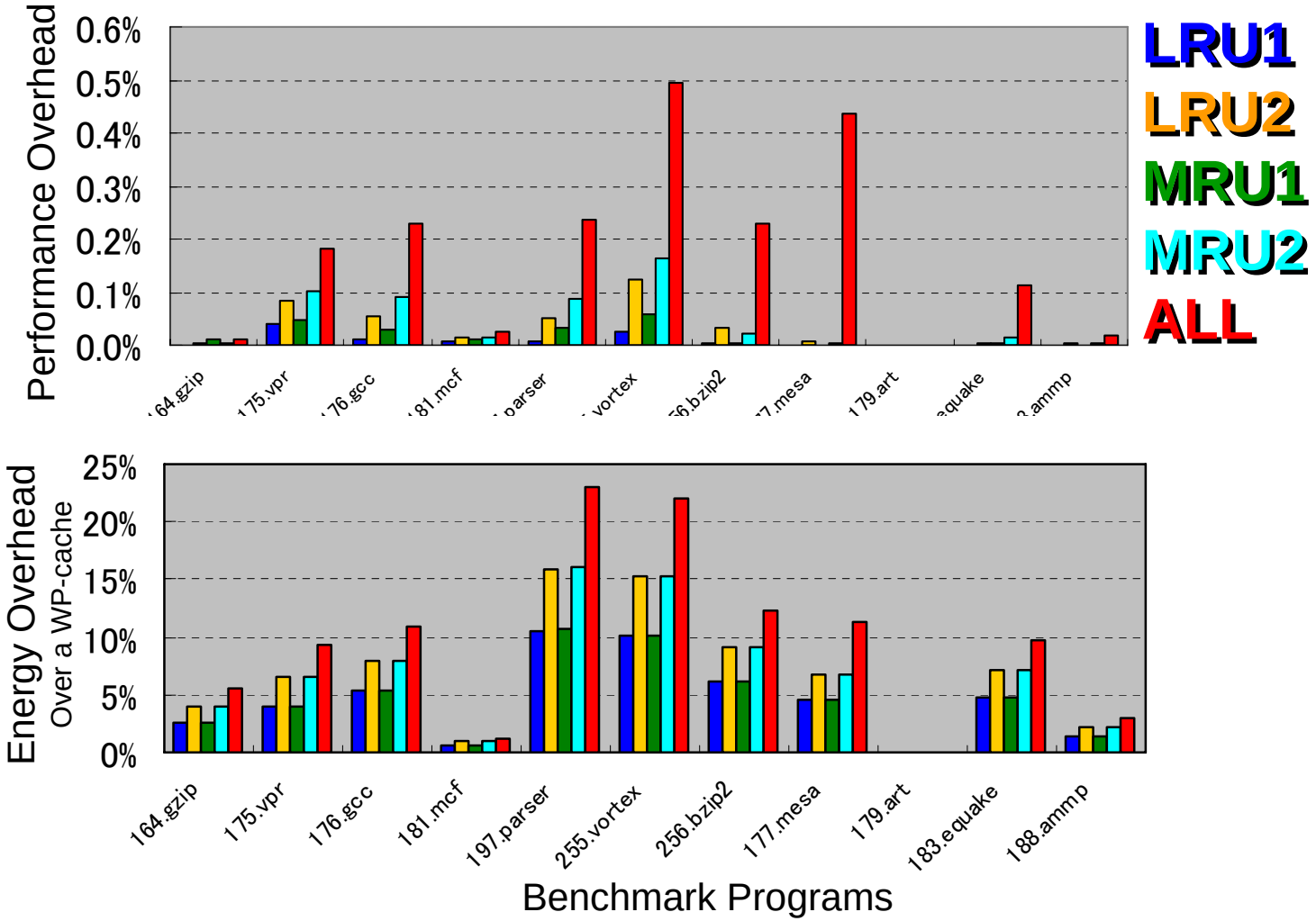
Vulnerability = (Nv-rald / Nrald) * 100

安全性を保障できない
戻りアドレス・ロード回数

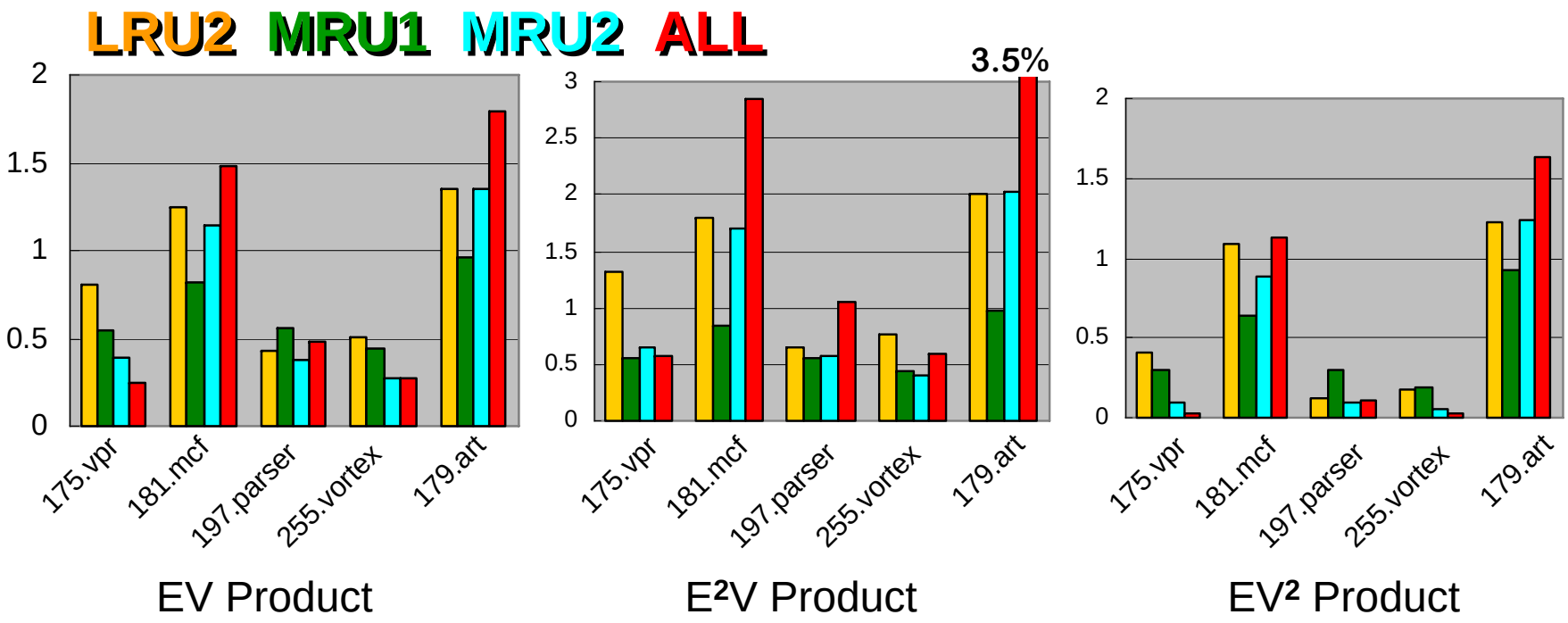
全戻りアドレス
ロード回数



性能/消費エネルギー・オーバヘッド



安全性と消費エネルギーのトレードオフ



EV Product　　E^2V Product　　EV^2 Product

低消費電力化への努力は続く・・・

- 今や，低消費電力化は（ほぼ）全てのコンピュータ・システムにおいて必須
  - スパコンから組込みシステムまで
  - 低消費電力プロセッサ==高性能/低コスト・プロセッサ
- 低消費電力化の基本アプローチ
  - ソフトウェアに任せる!
  - 無駄をなくす!
    - **Consider program-execution behavior**
    - **Divide, Select/Allocate, and Reduce**
- 今後の課題
  - 信頼性や安全性と消費電力
  - プロセスばらつきと消費電力
  - など

超高性能と低消費電力/エネルギー

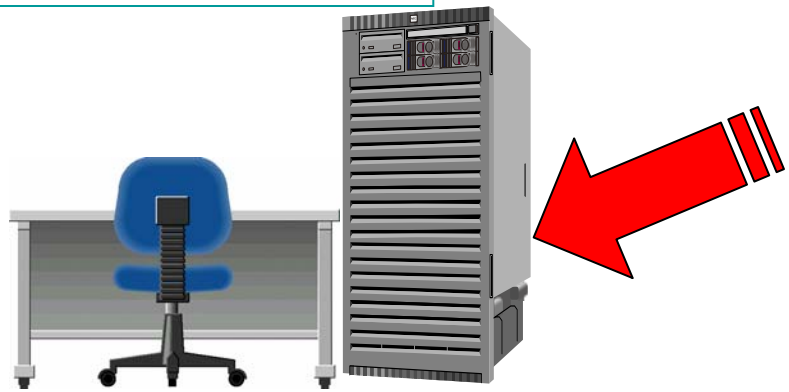
JST CRESTプロジェクト
名古屋大学、横浜国立大学、九州大学

10TFLOPS コンピュータ

90nm CMOS
並列プロセッサ方式

消費電力：約150kW
ラック30本

SFQ
新アーキテクチャ

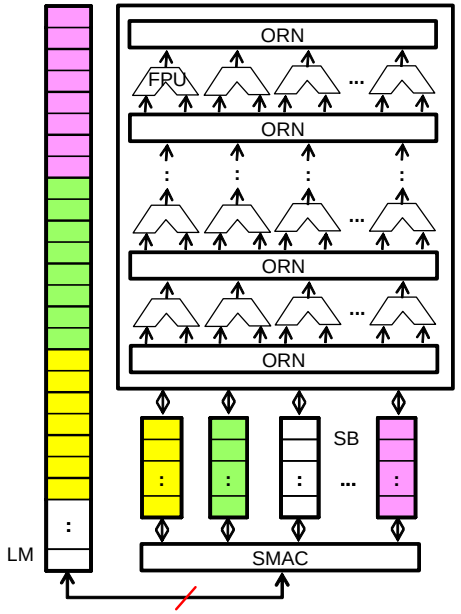


消費電力：約2kW
（冷凍機込み）

基盤技術の確立

デバイス/回路とアーキテクチャの協創
～単一磁束量子(SFQ)回路＋LSRDP～

- **「メモリアクセス」そのものを減らそう！**
  - 不必要なメモリアクセスがあるのでは？
    - スピルコード(中間結果の一時退避)
  - 中間結果の一時保存の必要性を無くす！
  - 依存関係のある大量の命令を「一度に実行」する！



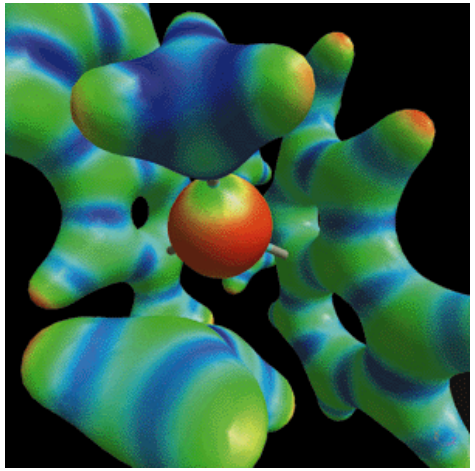
LSRDP(Large Scale Reconfigurable Data Path)

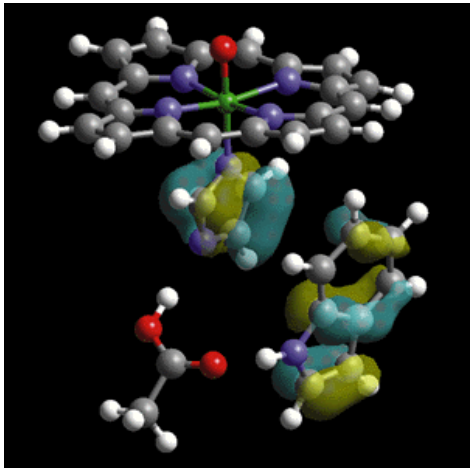
- 1個のチップに1,000個程度のALUを搭載
- ALU間をオンチップ・ネットワークで接続
- データフロー・グラフを直接マッピング

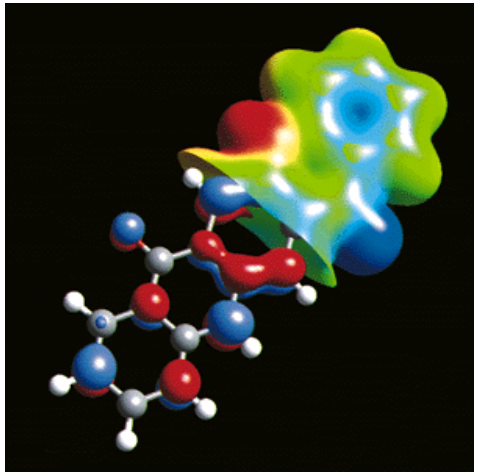
メモリアクセス回数を1/5に！

RDPの応用例：分子軌道法計算

- 電子が原子・分子内でどのような運動をしており、エネルギーを持っているかを計算により求める。
  - 分子物性の解析
  - 創薬、新素材の開発
  - ex) プリンタのカラーインク、液晶ディスプレイ







RDPの応用例

～分子軌道法における二電子積分計算（$\mu\nu\|\lambda\sigma$）の場合～

tei(4,4,4,4)=(((3+2*p*(4*PAx*PBx+PBx**2+PAx**2*(1+2*p*PBx**2)))*(3+2*q*(4*QCx*QDx+QDx**2+QCx**2*(1+2*q*QDx**2)))*f(0,t))/(p**2*q**2)+(4*(3+2*p*(4*PAx*PBx+PBx**2+PAx**2*(1+2*p*PBx**2)))*PQx*(QCx+QDx)*(3+2*q*QCx*QDx)*f(1,t))/(p*q*(p+q))(4*(PAx+PBx)*(3+2*p*PAx*PBx)*PQx*(3+2*q*(4*QCx*QDx+QDx**2+QCx**2*(1+2*q*QDx**2)))*f(1,t))/(p*q*(p+q))(8*(PAx+PBx)*(3+2*p*PAx*PBx)*(QCx+QDx)*(3+2*q*QCx*QDx)*(((p+q)*f(1,t))+2*p*PQx**2*q*f(2,t)))/(p*q*(p+q)**2)+(2*(3+2*p*(4*PAx*PBx+PBx**2+PAx**2*(1+2*p*PBx**2)))*(3+q*(QCx**2+4*QCx*QDx+QDx**2))*(((p+q)*f(1,t))+2*p*PQx**2*q*f(2,t)))/(p*q**2*(p+q)**2)+(2*(3+p*(PAx**2+4*PAx*PBx+PBx**2))*(3+2*q*(4*QCx*QDx+QDx**2+QCx**2*(1+2*q*QDx**2)))*(((p+q)*f(1,t))+2*p*PQx**2*q*f(2,t)))/(p**2*q*(p+q)**2)+(4*(3+2*p*(4*PAx*PBx+PBx**2+PAx**2*(1+2*p*PBx**2)))*PQx*(QCx+QDx)*(3*(p+q)*f(2,t)+2*p*PQx**2*q*f(3,t)))/(q*(p+q)**3)¥+(8*(3+p*(PAx**2+4*PAx*PBx+PBx**2))*PQx*(QCx+QDx)*(3+2*q*QCx*QDx)*(3*(p+q)*f(2,t)+2*p*PQx**2*q*f(3,t)))/(p*(p+q)**3)(8*(PAx+PBx)*(3+2*p*PAx*PBx)*PQx*(3+q*(QCx**2+4*QCx*QDx+QDx**2))*(3*(p+q)*f(2,t)+2*p*PQx**2*q*f(3,t)))/(q*(p+q)**3)(4*(PAx+PBx)*PQx*(3+2*q*(4*QCx*QDx+QDx**2+QCx**2*(1+2*q*QDx**2)))*(3*(p+q)*f(2,t)+2*p*PQx**2*q*f(3,t)))/(p*(p+q)**3)+((3+2*p*(4*PAx*PBx+PBx**2+PAx**2*(1+2*p*PBx**2)))*(3*(p+q)**2*f(2,t)+4*p*PQx**2*q*(3*(p+q)*f(3,t)+p*PQx**2*q*f(4,t)))/(q**2*(p+q)**4)(8*(PAx+PBx)*(3+2*p*PAx*PBx)*(QCx+QDx)*(3*(p+q)**2*f(2,t)+4*p*PQx**2*q*(3*(p+q)*f(3,t)+p*PQx**2*q*f(4,t)))/(q*(p+q)**4)(8*(PAx+PBx)*(QCx+QDx)*(3+2*q*QCx*QDx)*(3*(p+q)**2*f(2,t)+4*p*PQx**2*q*(3*(p+q)*f(3,t)+p*PQx**2*q*f(4,t)))/(p*(p+q)**4)+(4*(3+p*(PAx**2+4*PAx*PBx+PBx**2))*(3+q*(QCx**2+4*QCx*QDx+QDx**2))*(3*(p+q)**2*f(2,t)+4*p*PQx**2*q*(3*(p+q)*f(3,t)+p*PQx**2*q*f(4,t)))/(p*q*(p+q)**4)+((3+2*q*(4*QCx*QDx+QDx**2+QCx**2*(1+2*q*QDx**2)))*(3*(p+q)**2*f(2,t)+4*p*PQx**2*q*(3*(p+q)*f(3,t)+p*PQx**2*q*f(4,t)))/(p**2*(p+q)**4)(4*p*(PAx+PBx)*(3+2*p*PAx*PBx)*PQx*(15*(p+q)**2*f(3,t)+4*p*PQx**2*q*(5*(p+q)*f(4,t)+p*PQx**2*q*f(5,t)))/(q*(p+q)**5)+(8*(3+p*(PAx**2+4*PAx*PBx+PBx**2))*PQx*(QCx+QDx)*(15*(p+q)**2*f(3,t)+4*p*PQx**2*q*(5*(p+q)*f(4,t)+p*PQx**2*q*f(5,t)))/(p+q)**5+(4*PQx*q*(QCx+QDx)*(3+2*q*QCx*QDx)*(15*(p+q)**2*f(3,t)+4*p*PQx**2*q*(5*(p+q)*f(4,t)+p*PQx**2*q*f(5,t)))/(p*(p+q)**5)(8*(PAx+PBx)*PQx*(3+q*(QCx**2+4*QCx*QDx+QDx**2))*(15*(p+q)**2*f(3,t)+4*p*PQx**2*q*(5*(p+q)*f(4,t)+p*PQx**2*q*f(5,t)))/(p+q)**5+(8*(PAx+PBx)*(QCx+QDx)*(15*(p+q)**3*f(3,t)+30*p*PQx**2*q*(p+q)*(3*(p+q)*f(4,t)+2*p*PQx**2*q*f(5,t))8*p**3*PQx**6*q**3*f(6,t)))/(p+q)**6+(2*(3+p*(PAx**2+4*PAx*PBx+PBx**2))*(15*(p+q)**3*f(3,t)30*p*PQx**2*q*(p+q)*(3*(p+q)*f(4,t)+2*p*PQx**2*q*f(5,t))+8*p**3*PQx**6*q**3*f(6,t)))/(q*(p+q)**6)+(2*(3+q*(QCx**2+4*QCx*QDx+QDx**2))*(15*(p+q)**3*f(3,t)30*p*PQx**2*q*(p+q)*(3*(p+q)*f(4,t)+2*p*PQx**2*q*f(5,t))+8*p**3*PQx**6*q**3*f(6,t)))/(p*(p+q)**6)

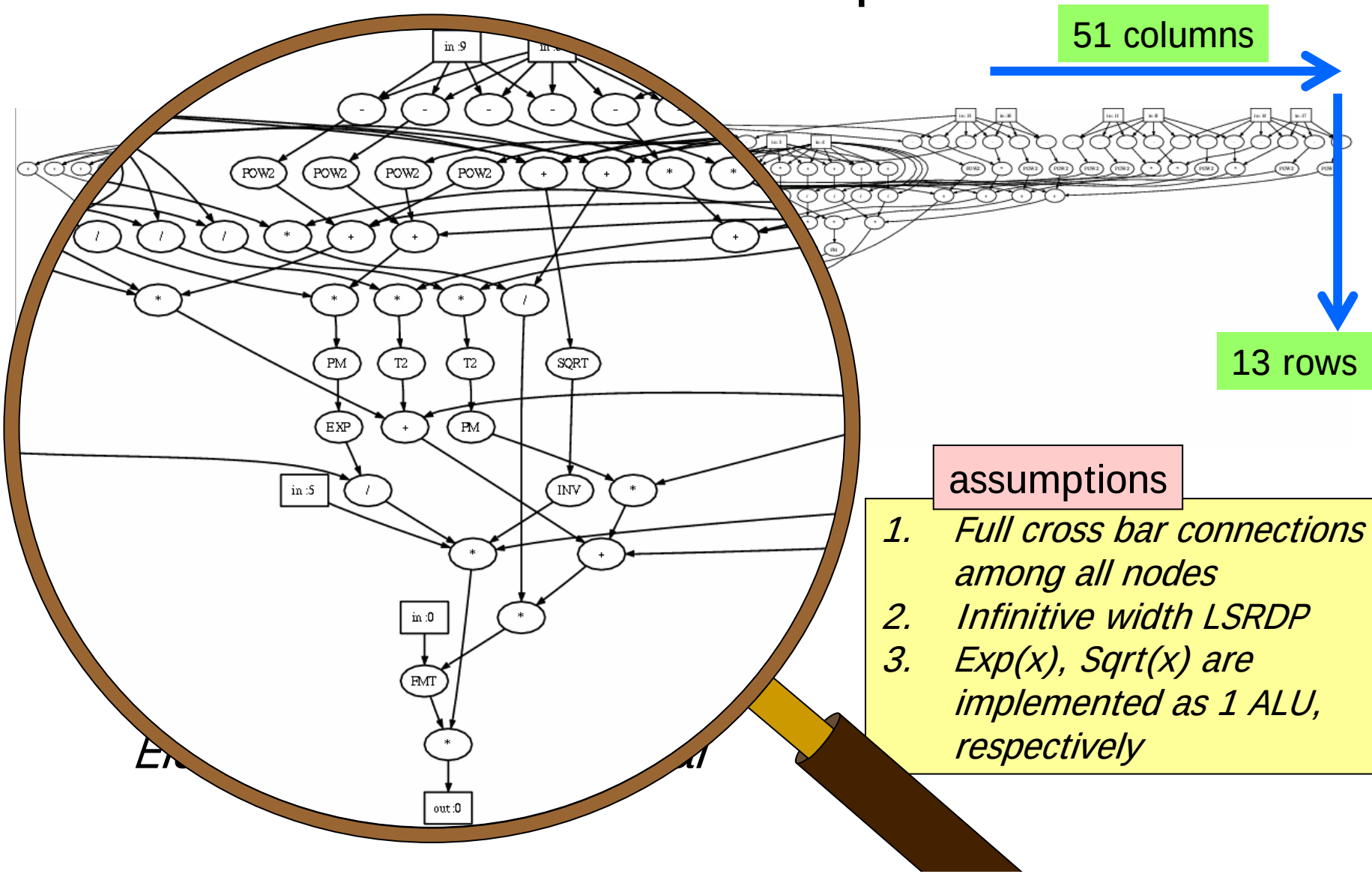
➔ **787 MUL, 261 ADD, 69 FUNC**

tei(3,1,1,1)=((PAy*(1+2*p*PAx*PBx)*(1+2*q*QCx*QDx)*f(0,t))/q+((((p+q)**4*((PAy+PQy)*q*(1+2*q*QCx*QDx)+p*(PAy+2*PAx*PAy*PQx*q+2*PAy*PBx*PQx*q+2*PAx*PBx*PQy*q2*PAx*PAy*q*QCx2*PAy*PBx*q*QCx2*PAy*PQx*q*QCx+2*q*((PAy*(PAx+PBx+PQx))+2*(PAy*(PAx+PBx)*PQx+PAx*PBx*PQy)*q*QCx)*QDx)2*p**2*PAx*PAy*PBx*(1+2*PQx*q*(QCx+QDx)))*f(1,t))/q)+(p+q)*((p+q)*((p+q)*(3*p*PAy+6*p**2*PAx*PAy*PQx+6*p**2*PAy*PBx*PQx+2*p**2*PAy*PQx**2+4*p**3*PAx*PAy*PBx*PQx**2+p*PQy+2*p**2*PAx*PBx*PQy+2*p*PAy*PQx**2*q+PQy*q+2*p*PAx*PQx*PQy*q

➔ **116 MUL, 31 ADD, 2 FUNC**

【出典：青柳 睦 教授（九大）】

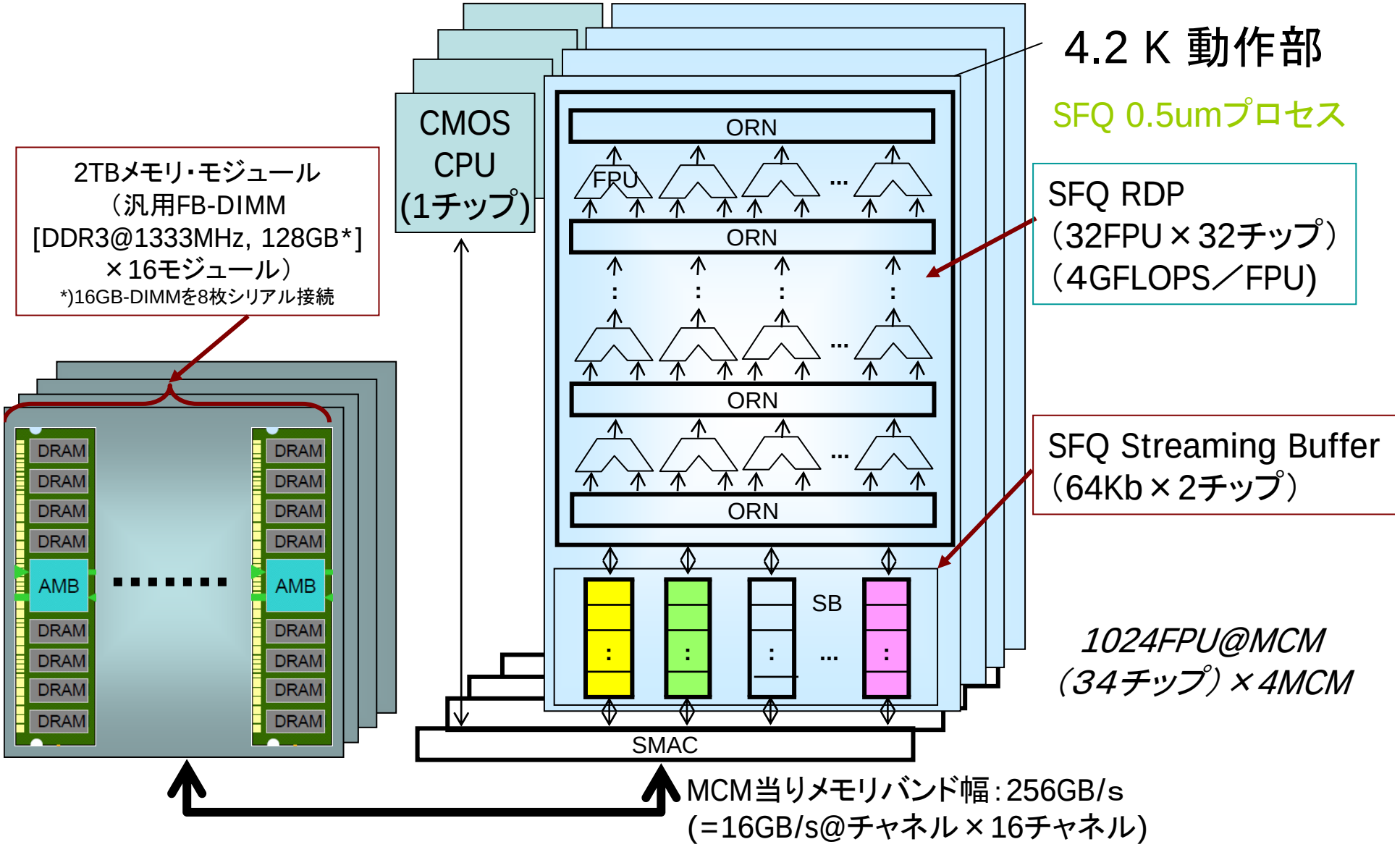
Example：Test Calculation(1/2)
Data Flow Graph



assumptions

1. *Full cross bar connections among all nodes*
2. *Infinitive width LSRDP*
3. *Exp(x), Sqrt(x) are implemented as 1 ALU, respectively*

10TFLOPS SFQ-RDP プロセッサ



まとめ
～低消費電力化技術は新しいステージへ～

- 「シングルコア」から「マルチコア」の世界へ
  - 複数コアにおけるV/F制御など
- 「オフチップ主記憶」から「オンチップ主記憶」の世界へ
  - DRAM貼付け技術の実用化
  - 高オンチップ・メモリバンド幅の効率的な活用
- 「固定」から「可変」の世界へ
- 「汎用」から「専用（アクセラレータ）」の世界へ
  - 正確には，汎用＋専用アクセラレータ
- 「性能と消費電力」から「性能と消費電力と信頼性/安全性」の世界へ