エネルギー予測とハードウェアの効率的な制御に基づく計算機システムの消費電力削減に関する研究

A Study on Reducing Power Consumption Based on Energy Prediction and Efficient Hardware Control for Computer System

2015年3月　博士学位論文

東京農工大学 大学院 工学府 電子情報工学専攻

坂本　龍一

Ryuichi Sakamoto

要　旨

近年，計算機の省電力化を目指し，様々な省電力化技術が研究開発されている．一方で，これらの多くはソフトウェアからの制御を想定しており，アプリケーションプログラマが省電力のための制御を行う必要があり，煩雑である．そこで，本論文ではこれらの煩雑な省電力制御を隠ぺいする，システムソフトウェアについて明らかとし，簡単に省電力化を実現できることを目指す．そのために，省電力な計算機アーキテクチャを提案し，この省電力なアーキテクチャを仮想化するシステムソフトウェアを提案する．また，システムソフトウェアによるストレージの電力モデルを用いた電源制御により，省電力制御の仮想化を行うと同時に，モデルにより省電力な計算機システムの方式を提案した．

第 1 章「諸言」では近年の計算機を取り巻く電力要求の問題について示し，計算機の省電力化の重要性を示す．計算機の省電力技術に関する研究は数多く行われているが，省電力化のためには，アプリケーションプログラマによる省電力制御が必要であり，省電力制御の簡単化が重要であることを述べる．そこで，省電力制御の簡単化を目標とし，システムソフトウェアによる電力制御の仮想化と資源管理手法について明らかとすることを示す．

第 2 章「関連研究と問題点」では計算機の省電力化に関するプロセッサ，ストレージ，メモリ，回路技術，ソフトウェアの関連研究を調査する．また，これらの問題点として，省電力化を実現するためにはアプリケーションプログラマが省電力制御を行う必要があり，煩雑であることを示す．具体的には，省電力プロセッサではプロセッサの特徴を利用するプログラミングが必要であり煩雑であることを示す．さらに，省電力ストレージにおいては，省電力機能の特性を考慮した電源制御が重要であることを述べる．

第 3 章「目標と設計方針」では問題分析を踏まえ，システムソフトウェアによる省電力制御の隠ぺいにより，煩雑な省電力制御を簡単化することを目標とすることを示す．具体的には，パイプライン並列プロセッサの制御の高速化による省電力アーキテクチャを提案し，煩雑なパイプライン並列を簡単化するシステムソフトウェアを提案する．また，SSD(Solid State Drive) と HDD(Hard Disk Drive) を組み合わせた環境において，HDD の省電力機能とエネルギー予測を用いたストレージの省電力化を行うシステムソフトウェアを提案する．

第 4 章「パイプライン並列プロセッサの省電力化と並列ライブラリ実行環境」では，電力性能に優れたパイプライン並列プロセッサの実行環境におけるアーキテクチャ改善による省電力化と, システムソフトウェアによるパイプライン並列制御の簡単化について示す．オンチップマルチコプロセッサにおいて，プロセッサ間でデータを受け流すパイプライン並列はメモリの利用効率が高く電力効率が高い特徴を持つ．一方で，パイプライン並列は個々のプロセッサ，メモリ，データ転送をそれぞれ制御する必要があり，これらの実行制御が時間的にオーバヘッ

ドとなる．また，パイプライン並列制御が複雑となる．そこで，制御オーバヘッドの高速化を実現するアーキテクチャと制御を簡単化する OpenCL ライブラリを提案する．これにより，パイプライン並列制御を抽象化し，高速化による省電力化と実行制御の簡単化を実現する．評価では，実行制御の高速化により 5 割程度エネルギー効率を改善できたことを示す．

第 5 章「省電力プロセッサ向け電力計測環境の設計と実装」では 4 章で述べたパイプライン並列プロセッサの電力評価環境について述べる．システムソフトウェアからの省電力制御を考慮した省電力化のための計算機アーキテクチャ向けの電力評価環境を示す．そのために，電力センサを介して直接消費電力を観測できる電力評価環境の設計と実装を示す．さらに，計測対象である省電力プロセッサからも自身の消費電力を計測するための機構について示す．合わせて，プロセッサに対する電源電圧制御を実現するための構成について示す．

第 6 章「SSD を HDD のディスクキャッシュとして用いる省電力ストレージ」では電力性能と速度性能に優れた SSD を HDD のキャッシュとして利用するシステムにおける省電力化手法について示す．プロファイリングと電力モデルにより HDD と SSD の電力性能を予測し，予測にて省電力化が可能な際に HDD をスピンダウンすることで HDD の消費電力を大きく低下させる．評価では，HDD 単体で利用した場合と比較し，平均で 21 倍のエネルギー効率の向上を実現したことを示す．

第 7 章「結言」では本論文の目的である，システムソフトウェアによる省電力制御の隠ぺいによる，省電力制御の簡単化を達成したことを述べる．また，今後の課題として，パイプライン並列向け OpenCL 環境と電力モデルを用いた省電力化の適用可能性について示す．

目 次

図 目 次

表 目 次

第1章　緒言

本章では研究の背景と目的について述べる．

1.1　研究の背景

近年，計算機の性能向上や，計算機に対する高性能化の要求の高まりにより，計算機システムの消費電力の増加が大きな課題となっている．そのため，計算機の省電力化が大きな課題となっている．

データセンタにおいては，電気代や冷房能力により，利用できる電力量は大きくスケールすることはできないが，ビッグデータ等へ対する分析の期待は大きく高まっており，データを保持するストレージや，分析を行うプロセッサのエネルギー効率を向上させることが大きな課題となっている．一方で，モバイル機器においても，限られたエネルギーの中で高いエネルギー効率を達成する事が求められている．

これらを背景として，計算機の省電力化に関する研究が進められている．また，数多くの省電力な製品が販売されている．計算機システムの省電力化では，プロセッサやストレージ，メモリ等の計算機の要素ごとに盛んに省電力化が行われている．さらに，ソフトウェアによる省電力化も行われている．

一方で，これらの計算機システムの省電力化技術には課題がある．それは，これらの多くの省電力化技術の制御はソフトウェア側に任されている点である．そのため，アプリケーションプログラマは省電力化を意識した省電力制御が必要であり，制御が煩雑である課題がある．

そこで，本研究では省電力制御の簡単化を目指す．これにより，アプリケーションプログラマに対して，省電力制御を意識することなく利用できる省電力な計算機を目指す．そのために，本研究では計算機の省電力化技術に関する関連研究を調査し，システムソフトウェアによる省電力制御を隠蔽するシステムソフトウェア構成を明らかとする．

1.2　計算機の省電力化技術の課題

計算機システムの省電力化では，プロセッサやストレージ，メモリ等の計算機の要素ごとに盛んに省電力化が行われている．さらに，ソフトウェアによる省電力化も行われている．

プロセッサの省電力化においてはコア数の増加による演算性能の向上によるエネルギー効率の改善，オンチップメモリの効率的な利用による省電力化，チップ内の通信の抑制による省電力化等が研究されている．また，メニーコアやマルチコアのそれぞれのコアに対する動的電源電圧周

波数制御 (DVFS) などが行われている．ストレージの省電力化では，Hard Disk Drive(HDD) の回転を止めることによる省電力化や，Solid State Drive(SSD) を用いることでスループットを改善し，エネルギー効率を向上させる研究や，製品が多く見られる．さらに，異なる特性を持つストレージを組み合わせた階層ストレージにより，容量とスループットの両立を目指した省電力ストレージがある．メモリにおいては，不揮発性の省電力なメモリを用いたノーマリーオフメモリが盛んに研究されている．ノーマリーオフメモリでは，使うときのみ不揮発メモリの電源を On にし，使わない時は常に Off とする事により省電力化を行う．また，メモリの LSI チップを 3 次元方向に積層した 3 次元積層メモリが普及している．3 次元に積層する事により，通信距離を短くする事が可能となり，省電力，高スループットを達成できる特徴がある．

この様に，計算機の要素ごとに盛んに省電力化研究が行われている．一方で，これの計算機システムの多くが適切な省電力制御をソフトウェア側から行う必要があるという事である．具体的には，アプリケーションプログラマが資源管理や電源制御を行う必要がある．プロセッサにおいては，多数のコアに対する資源管理が重要となる．また，HDD においては，適切なタイミングでディスクの回転を止める必要がある．省電力化のためには，これらの省電力機能をアプリケーションプログラマが考慮する必要がある．これは，極めて煩雑であり，アプリケーションブログラマが省電力機能を考慮して省電力化を実現する事は，現実的ではない．

1.3 研究の目的

そこで，本研究では，これらの省電力制御の隠蔽を目指す．これにより，アプリケーションプログラマは省電力制御を意識しなくとも省電力な計算機環境を利用できるようにする．具体的には，システムソフトウェアにより省電力制御を隠蔽する．アプリケーションより下位のレーヤで省電力制御を行うことにより，省電力制御をアプリケーションプログラマに対して隠蔽する．具体的には，省電力プロセッサの資源管理の簡単化とストレージの省電力制御の隠蔽をシステムソフトより行う．そこで，本研究では下記の 2 点の省電力化制御を隠蔽するシステムソフトウェアを目指す．

- 省電力プロセッサの資源管理の隠蔽
  演算性能と電力性能に優れたマルチコアアクセラレータの省電力制御の隠蔽を目指す．マルチコアアクセラレータでは，各コア等の資源管理が重要である．省電力化のためには，適切なコア制御，データ転送制御が必要であり，アプリケーションプログラマがこれらを制御する必要があり，煩雑である．そこで，省電力プロセッサの資源管理の隠蔽を目指す．

- ストレージの省電力機能制御の隠蔽
  HDD はディスクの回転を止めるスピンダウンを行うことにより省電力化が可能である．一方で，これらの省電力制御にはエネルギーやスループットのオーバヘッドがある．HDD を利用する際にはディスクを回転させる必要があり，エネルギーと遅延のオーバヘッドが生じる．アプリケーションプログラマがこれらを考慮し省電力制御を行う事は煩雑である．そこで，スピンダウンによるエネルギーやスループットのオーバヘッドを

考慮した省電力制御をシステムソフトから行う事で，アプリケーションプログラマに対して省電力制御を隠蔽する．また，本環境を SSD を HDD のディスクキャッシュとして用いる階層キャッシュストレージに適応し，省電力，高スループット，大容量を実現するストレージの実現を目指す.

これらにより，本研究では省電力制御を隠ぺいするシステムソフト環境を明らかとする事を目指す.

1.4 本論文の構成

本論文では，2 章で計算機の省電力化に関する省電力化技術について調査し，計算機の省電力化の問題点を明らかとする．さらに，それらの問題点について考察し，本研究の位置づけを明らかとする．続いて 3 章では，2 章での省電力化技術の問題分析を踏まえ，研究の目標，および，設計方針を示す.

4 章から 6 章では，3 章での目標を達成するための研究について示す．4 章では，ハードウェアによる高速化を行うことによる省電力化と，システムソフトウェアによる資源管理の簡単化について示す．具体的には，マルチコアパイプラインアクセラレータの省電力化手法について示す．マルチコアパイプラインアクセラレータにおける同期処理や実行制御を専用のハードウェアによって高速化することにより，電力効率を改善する．さらに，システムソフトウェアにより，専用ハードウェアを隠蔽し，アプリケーションプログラマに対して汎用性を担保するシステムを提案する．さらに，提案手法に基づく実際のシステムの設計と実装，評価について示す.

5 章においては，省電力ヘテロジーニアスプロセッサ向けの電力評価環境について示す．省電力化を目指し，省電力ヘテロジーニアスプロセッサの研究を進めている．一方で，これらのプロセッサを評価するための環境が必要である．そこで，5 章では省電力ヘテロジーニアスプロセッサ向けの電力評価環境の設計と実装について示す.

6 章ではストレージの省電力機能を隠蔽するシステムソフトウェアについて示す．具体的には，性能や電力特性の異なる SSD と HDD を用いた省電力ストレージシステムの研究について示す．SSD を HDD のディスクキャッシュとして用いる階層キャッシュストレージにおいて，HDD の省電力制御をシステムソフトから行う事により，省電力制御の隠蔽を行う．最後に 7 章にて本研究の成果，知見を示し，今後の課題についてまとめる.

第2章　関連研究と問題点

本研究の位置づけを明確にするため，計算機の省電力化に関する研究について調査を行い，その問題点について考察した．本章ではこれらの詳細について述べる．

2.1　計算機の構成要素と省電力化研究

近年，計算機の省電力化の研究が盛んに行われている．これらの省電力化研究の多くは，計算機の構成要素であるプロセッサ，メモリ，I/O の 3 つに着目し，研究が進められている．本研究では，これらの 3 つの要素のうちの，プロセッサと I/O に着目する．本研究で調査する分野を図 2.1.1 に示す．プロセッサの中でもアクセラレータに注目する．I/O では主にストレージに着目する．ストレージでは，HDD の省電力化や階層キャッシュストレージの省電力化について調査を行う．メモリは今後の課題である．

2.2　アクセラレータの省電力化研究

本節では計算機の構成要素であるプロセッサの省電力化に関する研究について調査し，問題点をまとめる．プロセッサの省電力化では，主にアーキテクチャを改善することにより実行効率を上げることによる省電力化，パワーゲーティングや DVFS などの回路が提供する省電力化機能を制御することによる省電力化などがあげられる．ここでは，主にアーキテクチャの改善により，実行効率を改善することによる省電力化について注目する．

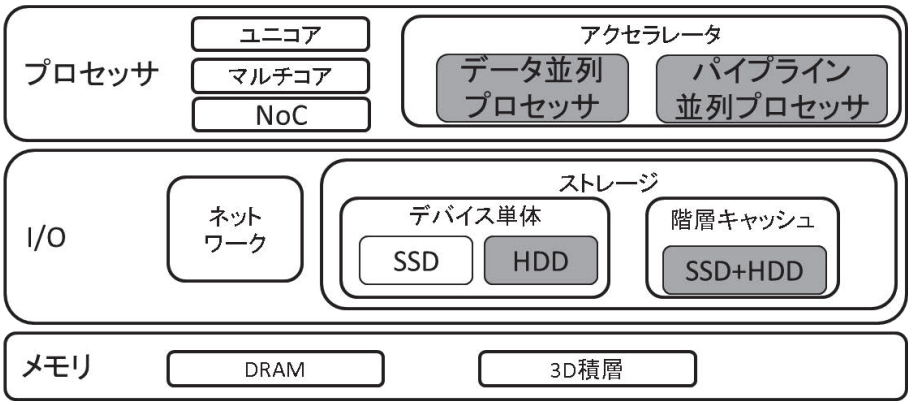

図 2.1.1 計算機の構成要素と省電力化研究

プロセッサのエネルギー効率の改善のアプローチとして1つのチップの中に，多数のコアを搭載することにより，演算効率を大幅に向上させ，エネルギー効率の向上により，省電力化を行う製品や研究が多く見られる．これらは，メニーコアプロセッサやアクセラレータと呼ばれている．他にも，プロセッサ内のデータ通信を効率化することによる，実行効率の改善による省電力化の研究が盛んに行われている．

2.2.1 メニーコアプロセッサの研究

はじめに，メニーコアプロセッサの研究について述べる．メニーコアプロセッサは，主にプロセッサの数を増やすことにより，演算の性能を向上させるアプローチである．演算性能を向上させることで演算の高速化を実現し，エネルギー効率を改善する．次に，メニーコアプロセッサの製品や研究の例を示す.

- Epiphany Chip[1][2]
  Epiphany Chip はアメリカのファブレスベンチャー Adapteva 社が開発した組込み向け省電力メニーコアプロセッサである．Adapteva 社では，将来の組込み向け省電力メニーコアプロセッサのコンセプトモデルとして，省電力性，プログラミング手法，スケーリングについて着目している．これらの要件は，組込み向けマルチコアプロセッサの革新を目指しているものである．現在，Epiphany Chip は，16 コア版と，64 コア版が製造販売されている．また，将来的には，4096 コア〜64K コアまでのスケールアウト目指し，プロジェクトがすすめられている．最大 1Ghz で動作する RISC プロセッサを 64 個格子状に並べた構成をとる．各プロセッサには，32KBytes のローカルメモリを持つ．そのため，チップ全体では，2MBytes のメモリを持つ．また，プロセッサ間は 800GB/sec を越える高速な NoC によって接続されている．64 コアで 100GFLOPS の性能を持ち，さらに，チップの消費電力が 2W 程度と非常に省電力であることが特徴である．プログラミング環境としては，C 言語を利用したプログラミング環境と OpenCL によるプログラミング環境が提供されている．

- Intel MIC(Knights Ferry)
  Epiphany Chip は組込み向けの省電力なプロセッサである．一方で，高性能数値演算，データセンタ向けのメニーコアプロセッサとして，Intel MIC[3][4] があげられる．Intel MIC は Intel の Larrabee[5] の後継プロセッサである．Intel MIC は PCI Express スロットに搭載される形で提供される．

  Intel MIC は 32 コアのインオーダ x86 プロセッサから構成される．また，それぞれのコアは 1.2Ghz で動作し，32Kbytes の L1，256Kbytes の L2 キャッシュを搭載する．L2 キャッシュは 32 コアで共有となり，合計で，Epiphany の 4 倍の 8Mbytes のキャッシュを有する．また，Epiphany と大きく異なる部分として，NoC の構成が異なる．Epiphany は格子状の NoC を採用するが，Intel MIC はリングバスを利用している．性能は，Epiphamy の 7 倍以上の 750GFLOPS の演算能力を持つが，消費電力はペリフェラルの

DDR5 等のメモリを含めた場合，300W 程度と，組込み機器を対象とした Epiphamy の 150 倍となり，非常に消費電力が大きい．

Intel MIC は PCI Express スロットにさし，ホストプロセッサから演算をオフロードするアクセラレータの形をとる．また，プログラミング環境についても，Intel が提供する環境において，C 言語によるプログラミング，OpenCL によるプログラミング環境がサポートされている．

- MPPA-MANYCORE[6]
  256 コアのプロセッサを搭載した組込み向けメニーコアプロセッサ MPPA-MANYCORE[6] も存在する．省電力かつ，高性能を目指し，256 コアでの消費電力が 10W 程度と省電力なプロセッサである．STMicroelectronics の Transputer/Occam の研究を行っていた，Monnier 氏が社長であり，Transputer の特徴を引き継いでいる．

  256 コアを搭載した MPPA-256 は，16 個のコンピュートクラスタから構成される．さらに，1 つのコンピュートクラスタは，16 個の VLIW Core から構成される．1 つのコンピュートユニットには，2MBytes のシェアードメモリが搭載され，コア全体で，32MBytes のメモリが搭載されている．また，400Mhz の周波数で動作し，230GFLOPS の演算性能を持つ．最大消費電力は 15W と Intel MIC よりも 20 分の 1 の消費電力である．電力あたりの演算性能は Epiphany Chipy より低いが，PCI Express Gen3 や 10G Ethernet 等の入出力を持ち，インタフェースが豊富である．また，MPPA MANYCORE は MMU のサポートを行っており，ボード単体で動作し，OS の動作もサポートしている点で，Epiphany Chip や Intel MIC と異なる．

  また，プロセッサの他に，開発環境が充実していることが，MPPA-MANYCOER の特徴である．リファレンス開発ボード，SDK，デバッグ環境が用意されている．また，プログラミング方法も他のメニーコア環境と比較して多数の環境を提供している．POSIX を用いたプログラミング環境や，OpenCL によるプログラミング環境，独自の Dataflow プログラミング環境をサポートしている．

- Cavium Octeon Fusion CNF71xx アーキテクチャ[7]
  Epiphany Chip，Intel MIC，MPPA-MANYCORE は同一のコアを多数並べた，ホモジニアスメニーコアの構成をとっていた．一方で，汎用のプロセッサと数値演算に特化したアクセラレータを複数組み合わせたヘテロジーニアスメニーコアも存在する．Cavium 社の Octeron Fusion CNF71xx アーキテクチャは，LTE やルータ等のネットワーク処理向けプロセッサである．Octeon Fusion アーキテクチャは，4 コアの MIPS64 コアと，6 コアのハードウェアアクセラレーションブロックの，合計 10 コアから構成される．

  Octeon Fusion アーキテクチャが他のメニーコアプロセッサと大きく異なる点は，アクセラレータがプロセッサと対等な位置づけになっている点である．MIC や Epiphany Chip では，ホストとなるプロセッサから，タスクをオフロードするモデルを構成している．すなわち，ホストメモリのデータを，メニーコア側のメモリへコピーし，アクセラレー

タをキックするモデルである．一方で，Octeon Fusion アーキテクチャでは，リアルタイムなトラフィック処理を想定し，トラフィック処理の各ステージを，MIPS プロセッサ，ハードウェアアクセラレーションブロックの得意なほうが行うというモデルをとっている点である．

- 345mW Heterogeneous Many-Core Processor[8]
  Octeron Fusion アーキテクチャは基地局等の常に電力が供給されることを想定したプロセッサである．一方で Lee らは，電力制限が厳しいウェアラブル組込み向けの省電力メニーコアプロセッサの開発を行っている．カメラの画像から，人や物等のオブジェクトの抽出を低電力で行うための研究を行っている．汎用の制御 RISC プロセッサ部（Cognitive Control Layer）と 4 つのアクセラレータ (Feature Extraction Cluster:FEC) から構成される．さらに，FEC は 1 つの Vector Procesing Element と 8 つの Scalar Processing Element から構成される．また，画像検出時の消費電力は 345mW と非常に省電力である．

上記のメニーコアアーキテクチャににについて比較を示す．初めに，コア数や今後のスケールの計画などについて示し，次に，電力性能について示す．最後に，省電力化機能の有無を示す．コア数等を表 2.2.1 に，電力性能を表 2.2.2 に，省電力化機構について表 2.2.3 に示す．

表 2.2.1 より，最大 64K コアのものが計画されていることが分かる．64K ものコアを搭載するプロセッサの場合共有資源へ対するロックが大きな課題となる．具体的には，64K コアで spin lock を行った場合，8ms もの時間がかかり，大きなオーバヘッドとなる．また，表 2.2.2 より，それぞれの，チップによって省電力化機能は様々であり，適切な制御が求められる．

2.2.2 メニーコアプロセッサの課題

メニーコア化による省電力化のアプローチではコア数を大きく増加させることにより，演算性能を向上させることによってエネルギー効率の改善を達成している．また，多くの場合すべてのコアで，同一のカーネルを実行しデータを分割する，データ並列モデルが多く用いられて

表 2.2.1 基本性能による比較

プロセッサ名	コア数	将来的なコア数	周波数 (Mhz)	消費電力 (W)	GFLOPS	GOPS	プロセス
Adapteva	64	4k〜64k	800	2	100		GF 28nm
Intel MIC	32		1200	300	750		45nm
Kalray	256	1024	400	10	230	700	TSMC 28nm
Cavium	14		2000	10			28nm
	(6+8)		1000				
Lee's chip	37		200	0.345		228	130nm

表 2.2.2 電力性能

プロセッサ名	単位コアあたりの消費電力	単位電力あたりの性能		単位コアあたりの性能	
	W/core	FLOPS/W	GOPS/W	GFLOPS/core	GOPS/core
Adapteva	31mW	50	データ無	1.56	データ無
Intel MIC	9.3W	2.5	データ無	23.44	データ無
Kalray	39mW	23	70	0.9	2.73
Lee's chip	9.3mW	データ無	661	データ無	6.16

表 2.2.3 省電力化機能

プロセッサ名	プロセッサ全体で DVFS は可能か	コアごとに周波数変更は可能か	コアごとに電圧を制御可能か	コアごとに電源を切れるか
Adapteva	不可	不可	不可	不可
Intel MIC	不可	不可	不可	可能
Kalray	可能	可能	可能	可能
Lee's chip	可能	不可	不可	不可

いる．一方で，メニーコアを用いたデータ並列演算にはOff-Chipへのデータアクセス時に課題がある．プロセス技術の微細化によりコア数は大きく増加しているが，チップ自体のサイズは大きく変わらない．そのためOff-Chipと接続するためのボンディングワイヤ数はスケールしない．この結果，Off-Chipへのアクセスがボトルネックとなり，性能が向上しなくなり，エネルギー効率も悪化する．

2.2.3 パイプライン並列プロセッサの研究

メニーコアプロセッサを用いたデータ並列演算時は，Off-Chipへのメモリアクセスがボトルネックとなる．これらの問題を解決するために，プロセッサ間でデータを受け渡すパイプライン並列処理が注目されている．パイプライン並列では，複数のコアに対しそれぞれ異なるタスクを割り当てる．さらに，これらのコア間でデータを受け渡すことにより，パイプライン処理を実現する．また，これらのデータの受け渡しには，On-Chipの小規模なFIFOを用いてコア間を接続する．

データ並列処理の場合，多くのコアが頻繁にOff-chipのメモリへアクセスするのに対し，パイプライン並列の場合，隣接するコア間で通信が発生するため，Off-Chipへのメモリアクセスが削減される特徴がある[9]．メモリアクセスが抑制されることで演算性能の向上や電力性能が向上する．また，キャッシュの代わりにFIFOを用いるため，ハードウェアの構成が簡単になり，さらなる省電力化が可能である．次にこれらのコア間パイプラインをサポートするプロセッサについて示す．

- RAW [10]
  RAWプロセッサはホモジーニアスなコアを16個搭載したマルチコアプロセッサである．MIPSベースのプロセッサを4 x 4個格子状に並べた構成をとっている．また，コア間でのパイプラインを行うために，プロセッサのレジスタとしてFIFOを有している．レジスタに対する書込み，読出しがFIFO操作に対応する．そのため，算術演算レベルでのパイプライン並列を主に想定している．

- Tilera [11]
  Teraは上記のRAWプロセッサをベースとした64コアのプロセッサである．8 x 8個のプロセッサを格子状に配置した設計となっている．また，コア間の通信にはFIFOではなく，フレキシブルなNetwork-On-Chip(NoC)を用いることが特徴である．通常NoCを用いることで任意のコア間で通信を行うことが可能であるが，TileraのNoCはあらかじめ，通信を行うプロセッサを限定することができる．これにより，通信ホップ数の減少により遅延の削減，省電力化を実現している特徴を持つ．

- C-5 [12]
  C-5プロセッサはFreescale社のネットワーク向けプロセッサである．パケットに対する演算処理をパイプライン化することで高いスループットを達成している．

- Cube [13]
  Cube プロセッサは筆者らが研究開発を進めるメディアプロセッシング向けのパイプラインプロセッサである．1 つの MIPS ベースの汎用コアと，3 つの CMA アクセラレータから構成される．CMA には 4KB のダブルバッファが搭載されており，ダブルバッファを用いることで CMA 間のパイプライン処理をサポートする．

2.2.4 パイプライン並列プロセッサの課題

パイプラインプロセッサは On-Chip の FIFO を利用し，コア間でデータを受け渡すことにより，高い演算性能と電力性能を達成している．一方で，これらのパイプラインの制御が課題となっている．具体的には，それぞれのコアや FIFO に対する実行制御オーバヘッドの課題と，実行制御を行うためのプログラミングが複雑である課題があげられる．

パイプライン処理のデータの受け渡しには小規模な On-Chip の FIFO(数 KB〜十数 KB) を用いる．そのため，各コアで実行するタスクの実行時間が短くなる（数百クロックサイクル〜千クロックサイクル程度）．これは，データ並列を得意とするメニーコアプロセッサと比較し，細かい粒度である．そのため，ソフトウェアによるパイプラインの実行制御が大きな課題となる．ソフトウェアによる制御では数百クロックサイクルから 2 千クロックサイクル程度要する．そのため，大きく演算性能が悪化する．これにより電力遅延積が増加し，エネルギー効率も悪化する．

また，パイプラインを扱うプログラミング環境にも課題があげられる．パイプライン制御を行う場合は，それぞれのコアの実行制御，データを受け渡す FIFO に対する get/put 操作，Off-Chip 間でのデータの入出力操作が必要である．MIT のパイプライン向けのコンパイラ StreamIt[14] では，FIFO に対する get/put オペレーションを API として提供することで，プログラマがパイプラインを記述できるようにしている．一方で，一般にこれらの FIFO と get/put を用いたパイプラインのプログラミングは非常に煩雑である．そのため，パイプライン制御の簡単化が大きな課題である．

2.2.5 実行制御の高速化の研究

パイプライン並列プロセッサでは実行制御オーバヘッドが課題としてあげられる．そこで，実行制御をハードウェアにて高速化する研究が多数行われている．Plural Architecture[15][16] では，メニーコアプロセッサを対象として，コア制御をハードウェアにて高速化している．一方で独自のパイプライン言語を用いたプログラミング環境を提供しているものの，独自仕様のため汎用性が低い．ほかにも，ハードウェアによる高速化のアプローチとして，コア上のタスク制御をハードウェアにて行うものが研究されている．コア制御の高速化には，Carbon[17] や ADM[18] などがある．一方でこれらは，タスクキューにあるタスクに対するタスクスティーリングを対象としている．よって，FIFO をもちいたパイプライン並列には対応していない課題がある．

2.2.6 アクセラレータの省電力化における問題点

プロセッサの省電力化の研究では，コア数増加による演算性能の改善やデータ転送に着目した省電力化が盛んに行われている．これらのアプローチでは，多数のコアにてデータ並列を行うことで演算性能を向上させるが，メモリアクセスがボトルネックとなり，演算の性能，電力性能が向上しない．

そこで，メモリアクセスのボトルネックを改善するために，パイプライン並列が研究されている．一方で，パイプライン並列プロセッサでは，プロセッサ間での同期処理やデータ転送，演算の実行指示などが重要になる．また，これらの処理をソフトウェアにて行った場合，オーバヘッドが大きい問題がある．また，今後，コア数が増加することにより，同期や DMAC などの制御オーバヘッドはさらに，大きくなると考えられる．これに対して，ハードウェアによる実行制御の高速化のアプローチは行われいているが，パイプライン並列には対応できない課題がある．

2.3 ストレージの省電力化研究

ストレージの省電力化に関する研究では，大きく分けて 2 つのアプローチがある．1 つめは，スループットなどの I/O 性能を改善することにより，実行性能を改善し省電力化を行うアプローチである．これらは，主にメモリの記憶階層において，データの配置を工夫することで，キャッシュを有効に利用するアプローチや，新しい記憶素子を用いることで，性能を向上させるアプローチである．

また，2 つめのアプローチは，ストレージ自身の省電力化機能を利用するアプローチである．具体的には，データの配置を工夫することでストレージへの I/O を抑制し，抑制された合間に省電力化機能を利用するというものである．具体的には，HDD に対する省電力化機能を用いている．HDD の省電力化機能では HDD のディスクの回転を停止させることにより，省電力化を実現する．

2.3.1 スループットを改善することによる省電力化

ストレージの性能向上を目指した研究では，近年大きく普及している SSD を用いた研究が多く行われている．また，SSD を用いた省電力化の研究として，多くが SSD と HDD の双方を利用することを想定している．SSD と HDD を利用することにより，それぞれの利点をいかすことができる．SSD は HDD と比較して高速，省電力という特徴を持つ．一方，容量においては，HDD の 10 分の 1 以下である欠点がある．そこで，SSD の性能を生かしつつ，HDD 並の容量をもつ，SSD を HDD のディスクキャッシュとして用いる階層キャッシュの研究が盛んに行われている．階層キャッシュを用いることで，高い実行性能を保ちつつ，HDD 並の容量を扱うことが可能となる．次に，これら SSD を用いて HDD のスループットを改善する研究について示す．

- Solaris ZFS ファイルシステム [19]：Solaris の ZFS ファイルシステムにおいて，L2ARC

（Level 2 Adaptive Replacement Cache）と呼ばれるメモリキャッシュの二次キャッシュ領域や，ZIL（ZFS Intent Log）と呼ばれる書き込みログ領域へ設定するボリュームにSSDを用いることで，ファイル入出力の高速化を図る方式である．ZFSファイルシステム以外では利用できない．

- Flashcache[20]：SSDをHDDのディスクキャッシュとして利用するもので，Linuxのブロック I/O レイヤを扱うデバイスドライバにより実装されている．そのため，任意のファイルシステムや，ブロックデバイスとして仮想化される任意のストレージデバイスで利用可能であるが，SSD上のキャッシュブロックの管理に，4KB〜16KBの固定サイズのブロック一つ当たり24バイト（64bit環境時）の主記憶領域が必要であり，メモリオーバヘッドが大きい．100GBのSSDを，4KBのブロックを用いて管理した場合，600MBの主記憶が必要となる．

- Flaz[21]：Linuxのブロック I/O レイヤでHDDのデータブロックを圧縮してSSDにキャッシュすることで，SSDへの格納容量の効率化を実現する．ファイルシステムやストレージデバイスには依存しないが，オンラインで行うデータブロックの圧縮・解凍にかかるCPUオーバヘッドとのトレードオフが発生する．

- SieveStore[22]：過去の一定期間のアクセス履歴を分析して，再利用される可能性の高いデータブロックのみをSSDに割り当てる方式である．SSDへのデータブロック割り当て量を大幅に削減し，同時にSSDのヒット率の向上を実現しているが，アクセス履歴を保持する記憶領域の確保と分析作業が必要となる．履歴の保存にはHDDのブロック単位でのアクセス数をすべて記録しているため，大きなコストが必要である．

- Turbo Memory[23]:
  SSDディスクキャッシュシステムにおいて，省電力化のためにHDDのスピンダウンを行うものとして，Intelの Turbo Memoryがあげられる．これらは，PCI-Express接続の専用のSSDデバイスとチップセットを用いて，ディスクアクセスの高速化とHDDへのアクセス低減とスピンダウンを組み合わせて省電力化を図る技術である．しかし，専用ハードを用いるために，使用できるハードウェアが限定されてしまう．このように，専用ハードウェアを用いずに，任意のファイルシステムやストレージデバイスから利用できるSSDディスクキャッシュの省電力化については提案されていない．

このように，SSDをHDDのディスクキャッシュとして用いることで，スループットを大きく改善することができる．また，これらのSSDをHDDのディスクキャッシュとして用いるストレージ方式について，それぞれの方式の比較を示す．1つめの比較ではスループットの良し悪しについて示す．スループットの向上率が高いほど，エネルギー効率の改善率が高くなるためである．2つめの比較では，ライトキャッシュを利用しているか，していないかである．ライトライトキャッシュを利用すると，スループットが向上するが，不慮のエラー等によって，キャッシュデータが破壊され，データの一貫性が失われる恐れがある．3つめの比較では，

表 2.3.1 SSD を HDD のキャッシュとして用いる研究の比較

	スループット	ライトキャの利用	キャッシュのオーバヘッド	プラットフォームへの依存性	スピンダウンによる省電力化
ZFS	中程度	無し	小	依存有り	無し
Flashcache	高速	有り	中	依存無し	無し
Flaz	高速	有り	大	依存無し	無し
SiveStore	高速	有り	大	依存無し	無し
Turbo Memeory	高速	有り	小	強い依存有り	有り

キャッシュの利用効率について示す．4 つめの比較では，キャッシュにかかるオーバヘッドを示す．5 つめに特定のプラットフォームへの依存性について比較する．6 つめに，HDD のスピンダウンによる省電力化が行われているかを示す．これらの比較を，表 2.3.1 に示す．

Flashcache は最もスループットが高く，高いエネルギー効率を達成することができる．一方で，キャッシュの対応管理に多くのメモリを消費する欠点がある．また，Flaz と SieveStore は SSD の利用効率は高いが，アクセスパタンの分析やデータの圧縮解凍を行うため CPU コストが高い．ZFS はライトキャッシュを用いないため，トラブルの際にデータを失わない特徴を持つが，ややスループットが落ちる欠点がある．さらに，Solaris のみサポートしているため，他の OS では利用できない欠点がある．Turbo Memory はスループットも高く，HDD をスピンダウンすることによる省電力化を行うことが特徴であるが，専用の PCI Express のデバイスを利用するため，極めて汎用性が低いと言える．

2.3.2 ストレージの省電力化機能を用いた省電力化

次に，HDD の省電力化機能を利用する省電力化研究について詳細を示す．これらの研究では，複数の HDD に対してデータの配置を工夫することにより，一部の HDD へのアクセスを抑制し，アクセスが抑制された HDD をスピンダウンすることで省電力化を実現している．下記に，HDD のスピンダウンを行うことによる省電力化研究について示す．

- RIMAC[24]:
  RIMAC は RAID5 と 2 段階の階層キャッシュを用いて，HDD の I/O を抑制することにより省電力化を行なっている．1 段目のキャッシュでは，I/O 要求レベルでキャッシュを行う．2 段目では，RAID5 に対するパリティデータのキャッシュを行うことで，HDD への I/O 要求を大きく削減する．

- EERAID[25]:
  EERAID は RAID コントローラにて，キャッシュ管理と I/O 要求管理を行うことで，HDD の省電力化を行う．RAID1 と RAID5 に 2 つについて省電力化手法を提案している．RAID1 ではミラーリングを用いるが，この際，1 つのディスクに，連続してして I/O を

表 2.3.2 HDD の省電力化機能を利用する省電力化研究の比較

	一般的な HDD に適応可能か	対象とするストレージ	オーバヘッド
RIMAC	可能	RAID	計算，キャッシュメモリ
EERAID	不可	RAID	キャッシュメモル
Prefeching and Caching	可能	1 台の HDD	計算，キャッシュメモリ
RAPoSDA	可能	RAID	キャッシュメモリ

発行するようにすることで，もう 1 つのディスクへのアクセスを抑制する．RAID5 はブロック単位でのパリティ分散記録方式であるため，複数ある HDD のうち特定の HDD はアクセスが抑制される．さらに，パリティの書き込みをキャッシュすることで，HDD への I/O をさらに抑制する．この抑制されたタイミングで HDD の省電力機能を用いる．

- Prefetching and Caching[26]:
  Athanasions らは，HDD を搭載するモバイル PC 環境を想定し，HDD への I/O 要求を学習するることにより，データのプリフェッチをおこない，HDD へのアクセスを抑制することで，省電力機能を用いて，省電力化をおこなっている．

- RAPoSDA[27]:
  RAPoSDA は複数の HDD を用いた環境にて，頻繁に利用するデータを特定の HDD にキャッシュすることで，他の HDD をスピンダウンさせることにより，少電力化を実現している．また，主記憶によるキャッシュメモリと HDD キャッシュディスクを用いた 2 段キャッシュストレージとなっている．

次に，これらの比較について示す．これらの研究では，HDD のディスクの回転速度を低下させることができるマルチスピードディスクを用いているものも多くある．一方で，マルチスピードに対応したディスクは一般的には普及していない課題がある．そのため実環境に対して省電力化を行う場合は，一般的に市販されている HDD に対応できるかが重要となる．そのため，比較では，一般的な HDD に適応可能かについて確認する．また，1 つのディスクを対象とするのか，RAID のように複数のディスクに対して省電力化を適応するのかについて比較する．あわせて，どのようなオーバヘッドがあるかについて示す．これらの比較を表 2.3.2 に示す．RIMAC や Prefeching and Cacheing, RAPoSDA は一般的な HDD を用いており，実用的と言える．また，RAID を対象とする省電力化が大半であり，1 つの HDD を対象とした省電力化は Prefeching and Cacheing しかない．さらに，すべての方式において，主記憶上にキャッシュメモリを用いるため，主記憶のメモリ消費が大きくなる欠点がある．

一方で，これらの HDD をスピンダウンを利用した際には問題点がある．HDD はスピンダウンを行うことにより省電力化が可能だが，HDD に対するデータの読み書きを行う場合は，止まっているディスクを回転させる必要がある．この回転させるスピンアップには大きなエネルギーが必要である．具体的には，頻繁に HDD のスピンアップやスピンダウンを繰り返した場合，電力性能が悪化する．さらに，スループットの低下による性能低下が発生する．

先行研究の場合，これらの電力の損得が問題であることは述べているが，きちんと考慮した研究はなく，多くはHDDのI/O要求がなくなり次第スピンダウンを行なっている．このような場合，電力的に損となるスピンダウンを起こす場合があり，これらの考慮が課題となっている．

2.3.3 スループットの向上とHDDの省電力化機能による省電力階層ストレージ

ストレージの省電力化のアプローチには，スループットの向上によりエネルギー効率を改善する方法と，HDDの省電力化を用いる2通りの方式があることを示した．スループットの向上には，SSDをHDDのディスクキャッシュとして用いる．また，ストレージの省電力化では，複数のHDD環境において，データの配置を工夫することで，スピンダウンを行うことにより省電力化が可能である．これらを組み合わせ，SSDをHDDのディスクキャッシュとして用いることで，HDDへのI/Oが抑制されるため，このタイミングでHDDの省電力化機能を用いることにより，さらなる，省電力化が期待できる．

2.3.4 ストレージの省電力化に関する課題

上記に示したように，SSDによるスループットの改善と，HDDにスピンダウンを行うことで，高いエネルギー効率を期待できる．一方で，HDDの省電力化機能であるスピンダウンとスピンアップには大きなエネルギーオーバヘッドがある．関連研究ではこれらがオーバヘッドである問題であることは述べられているが，これらのオーバヘッドを考慮した具体的な研究は行われていない．

2.4 関連研究と問題点のまとめ

アクセラレータの省電力化研究とストレージの省電力化研究の問題点についてまとめる．課題として，実行制御がオーバッドとなること，実行制御の簡単化が必要であること，単純な省電力化機能では電力的に損となる場合があることがあげられる．下記に，これらの詳細について示す．

- パイプラインプロセッサの実行制御オーバヘッド (2.2.4, 2.2.5)
  パイプライン並列プロセッサは，On-ChipのFIFOを利用し，コア間でデータを受け渡すことにより，高い電力性能を得ている．一方で，コア数の増加によりソフトウェアによる実行制御，同期制御，データ転送制御，FIFO制御がオーバヘッドとなっている．パイプライン処理ではデータの受け渡しに小規模なOn-ChipのFIFO(数KB〜十数KB)を用いる．そのため，各コアで実行するタスクの実行時間が短くなる（数百クロックサイクル〜千クロックサイクル程度）．これは，データ並列を得意とするメニーコアプロセッサと比較し，細かい粒度である．そのため，ソフトウェアによるパイプラインの実行制御が大きな課題となる．ソフトウェアによる制御では数百クロックサイクルから2千クロックサイクル程度要する．そのため，大きく演算性能が悪化する．これにより電力遅延積が増加し，エネルギー効率も悪化する．これらの問題を解決するためにハードウェ

アによる実行制御の高速化を目指した研究はあるが，パイプライン並列には対応していない課題がある．

- 複雑なパイプライン制御 (2.2.4)
  コア間でのパイプラインを行うことにより，演算効率，エネルギー効率を向上させることが可能である．一方で，一般的にパイプライン制御は複雑である課題がある．アプリケーションプログラマは FIFO に対する get/put と各コアの演算制御，Off-Chip とのデータ転送制御を行う必要があり制御が複雑である．

- HDD と SSD キャッシュ環境における HDD のスピンダウン制御による効率的な省電力化 (2.3.4)
  HDD がもつ省電力化機能の制御だけでは，エネルギー効率が悪化する可能性がある．HDD はスピンダウンを行うことによる省電力化機能を有するが，スピンアップには大きなエネルギーを要するため，頻繁にスピンダウンとスピンアップを繰り替えした場合，電力的に損となる可能性がある．

さらに，これらの制御をアプリケーションプログラマが考慮することは煩雑である．パイプラインプロセッサにおいては，省電力化のために煩雑なパイプラインアーキテクチャの実行制御が必要となる．さらに，ストレージ環境においては，HDD のスピンアップオーバヘッドを考慮した省電力制御が必要となる．3 章ではこれらの問題に対して，具体的な目標と問題解決のためのアプローチについて示す．

第3章　目標と設計方針

本研究では，計算機の省電力化をめざし，アプリケーションプログラマが省電力制御を意識せずとも，簡潔に省電力化が可能である計算機環境を目指している．そのために，システムソフトウェアによる省電力制御の隠蔽を目指している．2章では省電力化研究について，調査を行い，問題点を示した．本章では，2章の問題点を受け，省電力制御の隠蔽を行うシステムソフトウェアの目標を示す．これらの実現により，省電力制御の隠蔽を達成する．

計算機の問題点として，パイプライン並列制御に要するオーバーヘッド，パイプライン並列制御の複雑化，HDDの省電力機能の問題について示した．これらの課題を下記にまとめる．

- パイプラインプロセッサの実行制御オーバヘッド
  プロセッサの省電力化を目指し，コア間でデータを受け流すパイプライン並列が注目されている．一方で，パイプライン並列では個々のコアや，FIFO，データを行うDMACに対する実行制御が課題となっている．具体的にはソフトウェアによるコアの実行制御，コア間での同期制御，データ転送制御やFIFO制御がオーバヘッドとなっている．パイプラインプロセッサに利用するFIFOは数KB〜十数KBの容量であるため，各コアで実行するタスクの実行時間が短くなる（数百クロックサイクル〜千クロックサイクル程度）．この結果，ソフトウェアによるパイプラインの実行制御オーバヘッドが大きな課題となる．ソフトウェアによる制御では数百クロックサイクルから2千クロックサイクル程度要する．そのため，大きく演算性能が悪化する．これにより電力遅延積が増加し，エネルギー効率も悪化する課題がある．

- 複雑なパイプライン制御
  コア間でのパイプラインを行うことにより，演算効率，エネルギー効率を向上させることが可能である．一方で，パイプライン制御は複雑である課題がある．アプリケーションプログラマはFIFOに対するget/putと演算制御，Off-Chipとのデータ転送制御を行う必要があり制御が複雑である．

- HDDとSSDキャッシュ環境におけるHDDのスピンダウン制御による効率的な省電力化
  HDDがもつ省電力化機能の制御だけでは，エネルギー効率が悪化する可能性がある．HDDはスピンダウンを行うことによる省電力化機能を有するが，スピンアップには大きなエネルギーを要するため，頻繁にスピンダウンとスピンアップを繰り替えした場合，電力的に損となる可能性がある．

3.1 本研究の目標

本研究では，これらの問題を解決し，計算機の省電力化を目指す．さらに，本研究ではこれらの課題をシステムソフトウェアとアーキテクチャの改善にて解決することにより，省電力化制御を隠蔽する．下記に，課題に対応する目標を示す．

- パイプラインプロセッサにおける実行制御の高速化
  パイプライン制御においては，ソフトウェアによるコアの実行制御，同期制御，FIFO 制御，データ転送制御がオーバヘッドとなる．さらに，これにより，エネルギー効率も悪化する．そでこ，これらの実行制御を高速化することを目指す．これにより，演算時間を短縮し，エネルギー効率を向上させる．

- パイプライン制御の簡単化
  パイプラインの制御は複雑である．そのため，複雑なパイプライン制御の簡単化を目指す．これにより，簡単にアプリケーションプログラマがパイプラインプロセッサを利用できるようにする．

- HDD と SSD の電力と性能を考慮した電源制御
  HDD のスピンダウンを用いた省電力化の場合，スピンダウン直後にスピンアップを起こすような場合が頻繁すると，エネルギー効率が悪化する．そこで，これらのエネルギー的に損となるスピンダウンを回避することによる省電力化を目指す．

これらをシステムソフトウェアより下位のレーヤにて達成することで，省電力化制御を隠蔽し，アプリケーションプログラマに対して省電力制御を意識させることなく利用できる，計算機環境を実現する．

3.2 本研究の設計方針

次に，目標を実現するための具体的な設計方針を示す．実行制御の高速化の実現のために，専用ハードウェアによる実行制御の高速化を行う．さらに，これらの専用ハードウェアの隠蔽とパイプライン制御の簡単化のために，汎用のプログラミング環境を利用した汎用性の高い開発環境を実現する．また，HDD と SSD の電力特性を考慮した省電力化を行うために，電力モデルを用いた電源制御を行う．下記に，詳細を示す．これらの 3 つの設計と実装を行うことにより，計算機のエネルギー効率の改善とわせて省電力制御の隠蔽を目指す．これらの概要を図 3.2.1 にしめす．

- ハードウェアによる実行制御の高速化
  パイプラインプロセッサでは，ソフトウェアによる，実行制御がオーバヘッドとなっている．そこで，本研究ではこれらの実行制御を専用のハードウェアにて高速化する．コア制御，同期制御，データ転送制御をハードウェアにて高速化することにより，実行時間を短縮しエネルギー効率を改善する．

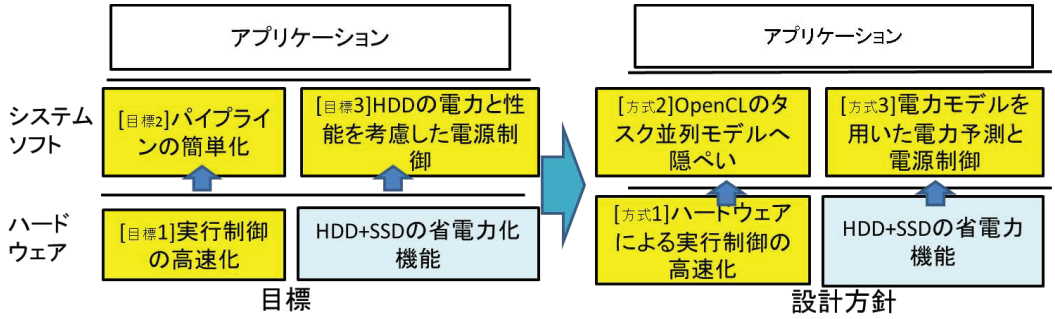

図 3.2.1 本研究の設計方針

- パイプライン制御の隠蔽
  煩雑なパイプライン制御を隠蔽し，パイプライン制御を簡単化する．これにより，アプリケーションプログラマは簡単にパイプラインを利用できるようにする．具体的には，汎用の並列プログラミング環境の OpenCL からパイプラインを利用できるようにする．そこで，OpenCL のタスク並列モデルにパイプライン制御を完全に隠蔽する．これにより，アプリケーションプログラマは OpenCL のタスク並列プログラムを記述するだけで，パイプラインプロセッサの利用を可能とする．OpenCL ライブラリ内でパイプラインへの展開とパイプライン制御を行う．これにより，コア間パイプラインの省電力化と制御の簡単化を達成する．

- 電力モデルを用いた電力予測と電源制御
  HDD のスピンダウン，スピンアップを頻繁に繰り返した場合，電力的に損となる場合がある．そのため，損となるスピンダウンを極力避けるために，電力モデルとアドミッションテストを用いた省電力手法を提案する．アドミッションテストにて HDD に発行される I/O パタンを分析し，エネルギーの予測を行う．この予測をもとにスピンダウンを行うタイミングを制御する．

3.3 本研究の全体構成

本研究ではシステムソフトウェアによる省電力化を実現する．これにより，煩雑な省電力制御をアプリケーションプログラマに対して隠蔽する．研究におけるこれらの関係を図 3.3.1 に示す．

本研究では，パイプラインプロセッサの実行制御にかかるオーバヘッド，複雑なパイプライン制御，HDD の省電力化にかかるオーバヘッドの 3 点を問題としてあげている．これらの問題に対し，実行制御オーバーヘッドの改善による省電力化，パイプライン並列制御の隠ぺい，HDD のオーバヘッドを考慮した制御を実現することを目指す．

そのために，実行制御を専用ハードウェアにて高速化する．これにより，演算時間を短縮し省電力化を実現する．煩雑なパイプライン制御を OpenCL のタスク並列モデルに対し抽象化を行う．これによって，アプリケーションプログラマが煩雑なパイプライン制御を行わずとも

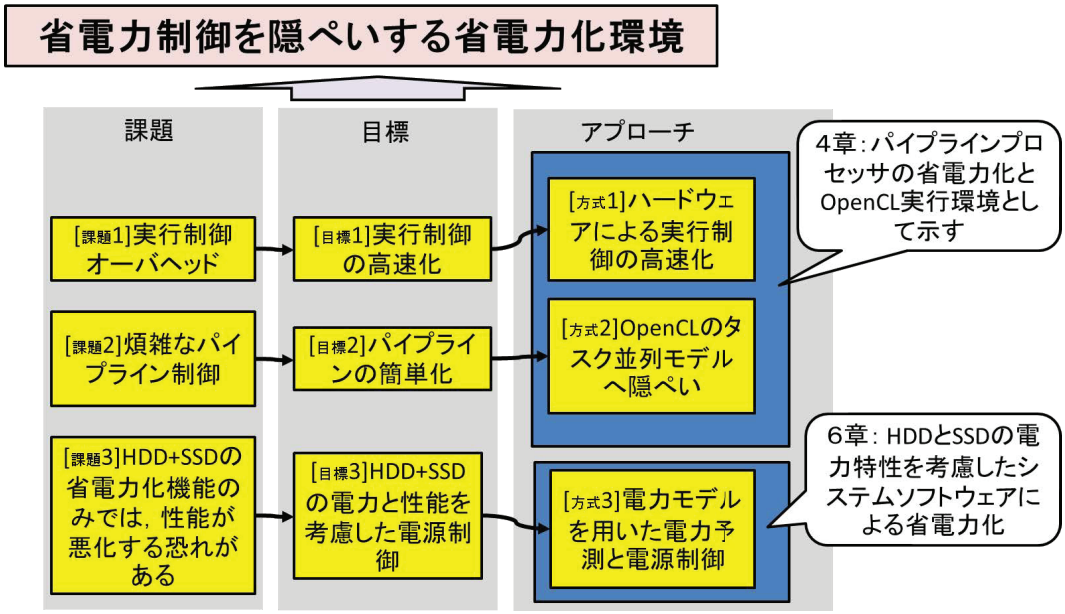

図 3.3.1 それぞれの目標と研究の関係

OpenCL のタスク並列モデルのみで簡潔に省電力効果を得られるようにする．また，電力モデルを用いた HDD の電力予測と電源制御を実現する．このように，3 つの課題に対し，システムソフトウェアによる省電力制御の隠ぺいを行うことにより，煩雑な省電力制御をアプリケーションプログラマに対して隠蔽する．

3.4 関連研究と本研究の関連

本研究では，計算機の省電力化を目指し，アクセラレータの省電力化とストレージの省電力化を目指している．パイプラインプロセッサ環境においては，ハードウェアによる実行制御の高速化による省電力化と OpenCL ライブラリによる制御の簡単化を実現している．これらの関係を図 3.4.1 に示す．パイプライン制御は FIFO に対する get/put 制御が必要であり煩雑である課題を，OpenCL のタスク並列モデルに仮想化することにより，並列計算の難易度を改善する．

また，ストレージ環境においてはスピンダウンを行うことにより，エネルギー効率を改善する．さらに，SSD と HDD を用いる階層キャッシュによって，容量を増加させる．これらを図 3.4.2 に示す．

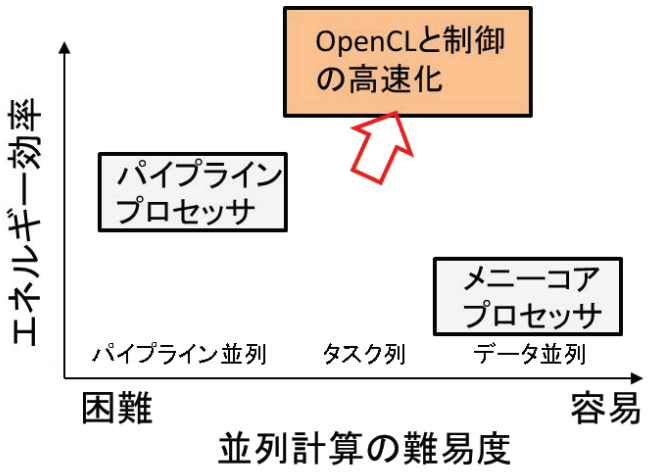

図 3.4.1 **パイプラインプロセッサによる省電力化**

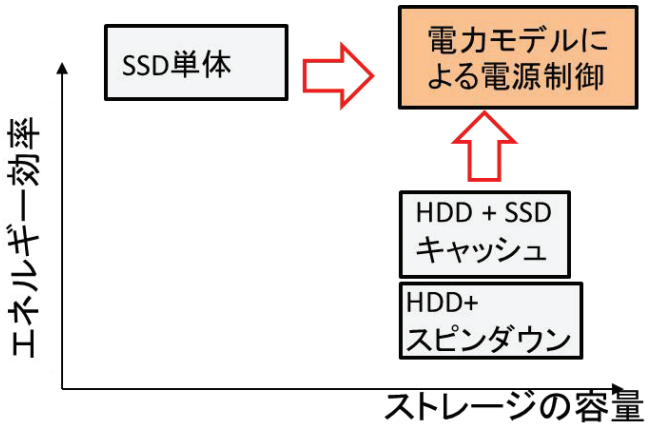

図 3.4.2 **ストレージ環境における省電力化**

第4章　パイプライン並列プロセッサの省電力化と並列ライブラリの実行環境

本章では，マルチコアアクセラレータ環境の省電力化に関する研究について示す.

4.1　はじめに

近年，マルチコアアクセラレータやメニーコアプロセッサの消費電力の増加が大きな課題となっている．そこで，電力の省電力化を目指して，On-Chip 中のアクセラレータ間でデータを受け渡すパイプライン並列処理が注目されている．パイプライン並列処理では，複数のアクセラレータに対して，それぞれ異なるタスクを割り当てる．さらに，アクセラレータ間で演算データを受け渡す．また，パイプライン並列処理は On-Chip の数 BK から十数 KB 程度の小規模なメモリを FIFO として利用するため，シェアードメモリを用いるデータ並列処理と比較しデータの移動にかかるコストが小さい．また，Off-Chip メモリへのアクセスが抑制される．そのため，高い電力効率とスループットを得ることができる [9].

近年は，多くのコア間パイプラインプロセッサ [10] が研究されており，コア間パイプラインをサポートする商用のプロセッサ [11][12] なども普及している．我々もプロセッサの省電力化をめざし，アクセラレータ間でパイプラインをサポートするアクセラレータ Cube[13] の研究開発を進めている．

一方で，これらの様々なプロセッサを利用するためには，それぞれのプロセッサ向けの開発環境を利用し，プロセッサ固有の API 等を理解する必要がある．そのため，非常に手間が大きい問題がある．そこで，これらの問題に対して，汎用の並列プログラミング環境として OpenCL[29] が提案されている．OpenCL では抽象化された OpenCL デバイスを提供しており，様々なアクセラレータやメニーコアプロセッサ，マルチコアプロセッサを共通のアプリケーションコードから利用できる特徴を持っている．そのため，コア間パイプラインプロセッサ向けの OpenCL 環境が重要となる．これにより，簡潔にコア間のパイプラインプが利用可能となる．

一方で，OpenCL ではデータ並列モデルとタスク並列モデルのみサポートしており，パイプライン並列はサポートしていない問題がある．パイプライン処理ではアクセラレータ間を FIFO で接続し，FIFO に対するデータの出し入れを get/put などの API を用いて制御する必要があり，OpenCL はこれらの制御をサポートしていない．そのため，OpenCL から直接パイプライン制御を記述する能力がない課題がある．

また，実行時のパイプライン制御においても実行制御オーバヘッドの課題がある．パイプラインには On-Chip のメモリを FIFO として利用するため，アクセラレータで動作するカーネル

の実行時間が短く（数百クロックから千クロック程度）なり，細粒度なアクセラレータやFIFOに対する実行制御が必要となる．ソフトウェアにてこれらの実行制御を行った場合，数百から2千クロック程度のオーバヘッドが生じる．その結果，実行制御に要するオーバヘッドが増加し，実効性能が大きく悪化する課題がある．

そこで，本研究ではOpenCLによるパイプライン制御環境の実現と実行制御オーバヘッドの高速化により，パイプライン制御の簡単化と高速化の実現を目指す．高速化により，エネルギー遅延積を改善し，エネルギー効率を改善する．そのために，パイプライン並列処理を隠蔽するOpenCLライブラリと，ハードウェアによる制御の高速手法を提案する．

4.2 研究の背景と目標

近年プロセッサの演算性能の向上のために，1つのチップの中に多数のコアを搭載することにより，演算性能を大幅に向上させるマルチコアアクセラレータやメニーコアアクセラレータが多く普及している．これらのアクセラレータの多くは，データ並列を得意としている．データ並列では，それぞれのコアで同一のカーネルを実行し，大きなデータをそれぞれのアクセラレータに分割する．

これらの，データ並列を得意とするアクセラレータは制御が簡単である特徴を持つが，多くのアプリケーションにおいて，Off-Chipへのメモリアクセスがボトルネックとなり，演算性能がスケールしない問題がある．プロセス技術の向上により，1つのチップに搭載できるアクセラレータ数は増加するが，チップの物理的な大きさは大きく変わらない．よって，チップとOff-Chipメモリへの転送帯域は大きく変化しない．そのため，メニーコア環境ではメモリへのアクセスにより演算性能が低下する．そこで，メモリボトルネックを解決するためにパイプライン並列を用いたマルチコアパイプラインプロセッサが注目されている．

4.2.1 マルチコアパイプライン並列プロセッサ

データ並列の場合，チップ内のそれぞれのアクセラレータは同一のカーネルを実行し，演算に用いるデータを分割する．一方で，パイプライン並列の場合，チップ内のそれぞれのアクセラレータに異なるカーネルを割り当て，さらに，これらのアクセラレータ間で演算データを受け渡すことにより，パイプライン処理を実現する．

パイプライン処理はメモリの利用効率が高く，高スループット，省電力である特徴を持つ．パイプラインによるデータの受け渡しにはOn-Chip上の小規模 (数KB〜十KB程度) なFIFOを用いる．このFIFOを用いて，アクセラレータ間で直接データの受け渡しを行うためメモリの競合がなく高いスループットを達成できる．また，FIFOはキャッシュのように複雑なコヒーレンシ制御が不必要となる．この結果電力効率も高くなる．さらに，データ並列と比較し，パイプラインではOff-Chipへのアクセスが抑制される特徴を持つ．

4.2.2 パイプライン並列制御と OpenCL との対応の課題

パイプラインを行う際は，それぞれのアクセラレータに対し，カーネルを割り当てて，FIFO を介して他のアクセラレータと通信する．具体的には，各アクセラレータの実行制御，FIFO に対する get/put 制御，Off-Chip とのデータ転送制御が必要となる．

MIT のパイプライン向けのコンパイラ StreamIt [14] では，FIFO に対する get/put オペレーションを API として提供することで，アプリケーションプログラマがパイプラインを記述できるようにしている．一方で，一般にこれらの FIFO と get/put を用いたパイプラインのプログラミングは非常に煩雑である．

また，近年，広く普及している並列プログラミング環境の OpenCL では，データ並列とタスク並列のみをサポートし，パイプラン並列はサポートしていない．具体的には，OpenCL に FIFO に対する get/put 制御を行う能力がないため，パイプライン制御を記述できないことが問題である．

筆者らの先行研究 [28] では，OpenCL に FIFO を扱うためのインタフェースを拡張し，アプリケーションプログラマが FIFO を利用してパイプラインを記載できるようにした．具体的には，FIFO 機能を持つ特殊なメモリオブジェクトと FIFO 機能を持つメモリオブジェクトに対する get/put 制御用の API の 2 つの拡張を行った．しかし，アプリケーションプログラマは依然，FIFO に対する制御が必要であり，制御は複雑である．さら OpenCL に対して API を追加する拡張を行っているため，アプリケーションコードの汎用性が担保できなくなる．OpenCL では共通のアプリケーションコードが，異なるプラットフォーム上でも動作することを特徴としているため，アプリケーションコードの汎用性が低下することは望ましくない．よって，簡単に OpenCL からパイプラインが制御できることが望ましい．

4.2.3 パイプライン並列の制御オーバヘッドの課題

パイプライン処理でのデータの受け渡しには小規模な On-Chip の FIFO(数 KB〜十数 KB) を用いる．そのため，各コアで実行するタスクの実行時間が短くなる（数百クロックサイクル〜千クロックサイクル程度）．これは，データ並列を得意とするメニーコアプロセッサと比較し，細かい粒度である．そのため，ソフトウェアによるパイプラインの実行制御が大きな課題となる．ソフトウェアによる制御では数百クロックサイクルから 2 千クロックサイクル程度要する．そのため，大きく演算性能が悪化する．これにより電力遅延積が増加し，エネルギー効率も悪化する．

4.2.4 パイプライン並列プロセッサのための OpenCL 実行環境

そこで，煩雑なパイプライン制御と，制御オーバヘッドの課題を解決するために，パイプライン制御の簡単化とパイプライン制御の高速化を目指す．具体的には，FIFO に対するパイプライン制御を隠蔽することで，簡単化を実現する．さらに，ソフトウェアによる制御オーバヘッドを改善する．これにより，アクセラレータ間パイプラインの簡単化と実行性能向上のための実行環境の実現を目指す．

本稿ではこれらの目標を達成するために2つのアプローチをとる．

- OpenCL ライブラリによるパイプライン並列制御の隠蔽
  OpenCL のタスクモデルに対してパイプライン並列制御を隠蔽する．アプリケーションプログラマは OpenCL のタスク並列モデルによるプログラミングを行うことで，パイプラインの制御を行えるようにする．これにより，パイプライン制御の簡単化を達成する．
- パイプライン制御を行うハードウェアタスクディスパッチャによる制御の高速化
  実行制御を高速に行うためのハードウェアディスパッチャRCH(Reconfigurable Control Hardware) を提案する．ソフトウェアによるアクセラレータ制御，FIFO 制御，Off-Chip とのデータ転送制御をハードウェアにて行うことにより，ソフトウェアの制御オーバヘッドを削減し，制御の高速化を実現する．

これらの実現により，汎用のアクセラレータ間パイプランの実行環境を構築する．

4.3 システムの全体構成

アクセラレータ間パイプラインの簡単化と実行性能の向上をめざし，OpenCL ライブラリと RCH にてパイプライン制御の高速化と簡単化を実現する．本システムの全体構成図を図 4.3.1 に示す.

4.3.1 システムの概要

OpenCL ライブラリはパイプライン制御の隠蔽により，アプリケーションプログラマに対して簡単なパイプライン実行環境を提供する．また，RCH はソフトウェアにて行っていた実行制御をハードウェアにて高速に実現し，アクセラレータの実行効率を改善する役割を担う．また，OpenCL ライブラリと RCH のインタフェースには，RCH の機能を抽象化した RCH コマンドを用いる．具体的には，RCH コマンドはアクセラレータ制御や FIFO 制御，DMAC に対する制御を抽象化を行ったものである．

4.3.2 OpenCL ライブラリの役割

OpenCL ライブラリは主に3つの役割を担う．1つ目はアプリケーションプログラマに対して抽象化された OpenCL デバイスを提供することである．アプリケーションプログラマはこれらの抽象化された OpenCL デバイスを用いてタスク並列制御を記載する．これらの情報を OpenCL オブジェクト情報部としてタスク並列の情報を管理する．2 目は FIFO 制御の隠蔽である．タスク並列により記載された DAG とメモリオブジェクトの情報よりパイプライン情報を抽出する．このようにパイプラインを OpenCL の内部で隠蔽する．具体的には，OpenCL のコマンドより RCH コマンドを生成する．これは，RCH コマンド対応生成部にて行う．3 つ目

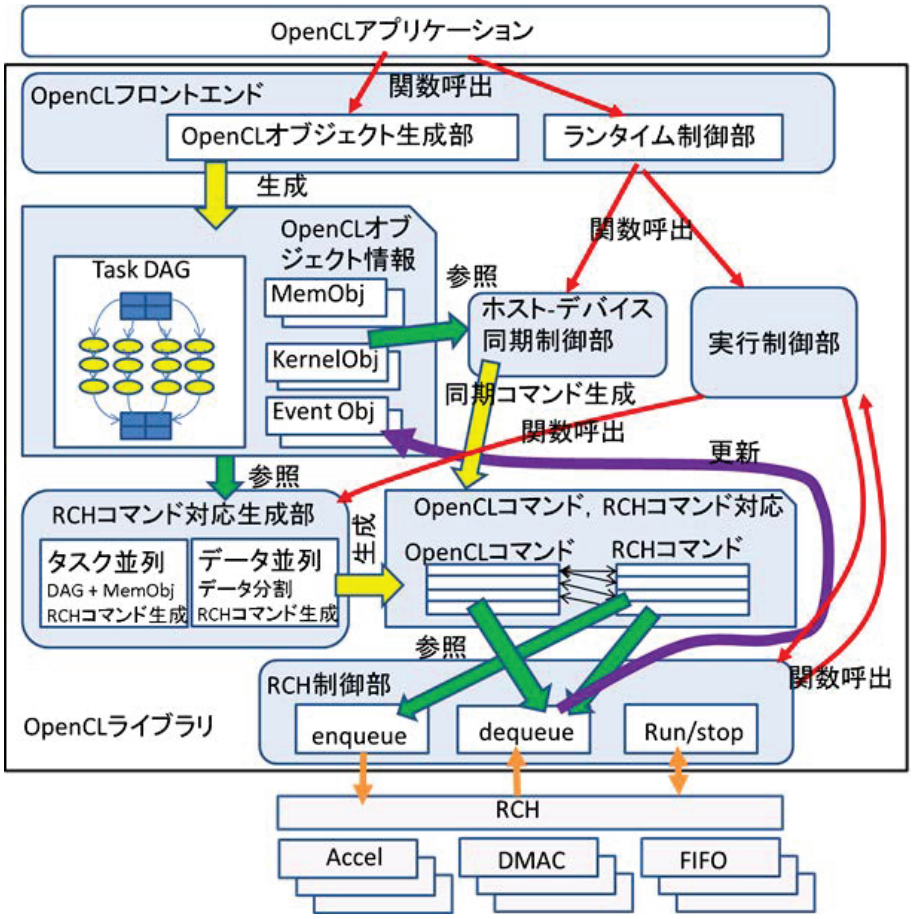

図 4.3.1 提案システムの全体構成

は，RCH の実行制御を行う役割である．RCH の起動や停止などの制御と生成された RCH コマンドを RCH に対しエンキューし RCH の実行制御を行う．

4.3.3 RCH の役割

RCH はソフトウェアによる実行制御を高速化する役割を持っている．具体的には個々のアクセラレータの実行制御や個々の DMAC へ対するデータ転送制御をハードウェアから高速に行う．また，次節で述べる RCH コマンドをベースに実行制御を行う．RCH ではアーキテクチャ依存の実装を隠蔽する役割をもつ．さらに，OpenCL ライブラリとの同期を行うために，RCH コマンドの終了を RCH に知らせる仕組みを持つ．

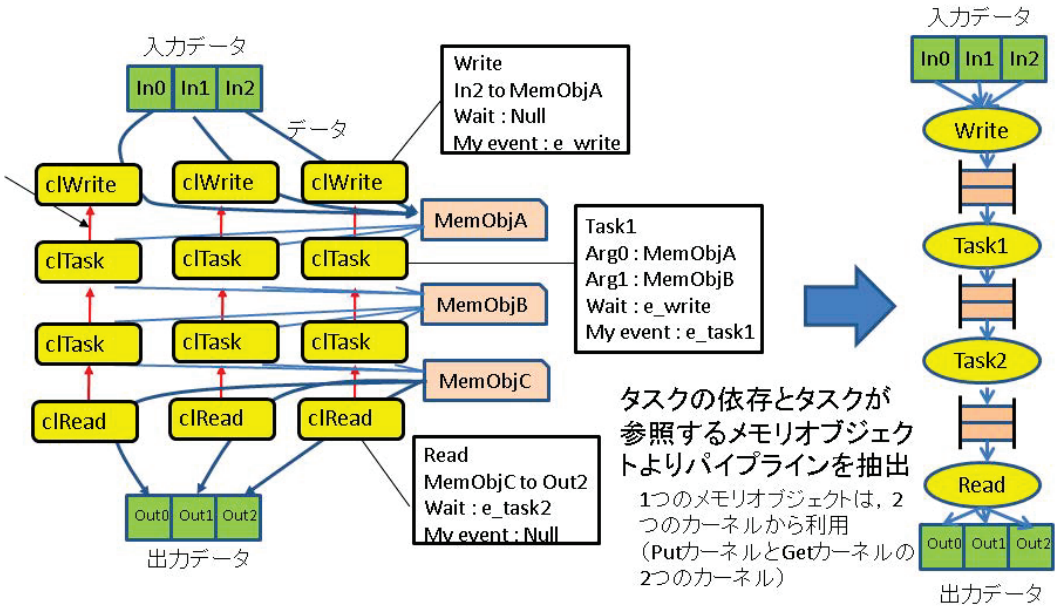

図 4.4.1 OpenCL **によるタスク並列からのパイプライン展開**

4.3.4 OpenCL ライブラリと RCH 間のインタフェース

OpenCL ライブラリと RCH 間はパイプライン制御を抽象化した RCH コマンドにてインタフェースをとる．この RCH コマンドによって，ホストプロセッサと非同期にアクセラレータ制御や FIFO 制御が可能となり，ホストプロセッサと RCH 間で並列を向上させることが可能となる．

4.4 OpenCL ライブラリによるパイプラインの隠蔽

本 OpenCL 環境はアプリケーションプログラマに対してパイプライン制御を隠蔽する役割を担う．これにより煩雑なパイプライン制御を隠蔽する．そのために，パイプラインを隠蔽する OpenCL を用いたタスク並列モデルについて示す．これらの概要を図 4.4.1 に示す．

パイプラインを行う際は，アクセラレータ間で FIFO を利用する．また，FIFO には必ず 1 つの入力と 1 つの出力がある．そこで，OpenCL のメモリオブジェクトを 1 つの FIFO として抽象化を行う．アプリケーションプログラマはこのメモリオブジェクトに対するデータの書き込みと読出しを行う 2 つのタスクを指定する．この際，OpenCL ではメモリオブジェクトをタスクの引数として指定する．一方で，OpenCL ではカーネルの引数に対して入力と出力の区別がない．そのため，FIFO の方向を決定することができない．そこで，タスク間のデータの依存関係を用いることにより FIFO の入力と出力を決定する．

パイプラインへの展開時は，タスクモデルにて依存関係のある 2 つのカーネル間でともに利用されているメモリオブジェクトを FIFO として対応付ける．さらに，データの生産者側と消費者側のカーネルをそれぞれアクセラレータへ割り当てる．これらは OpenCL ライブラリ中

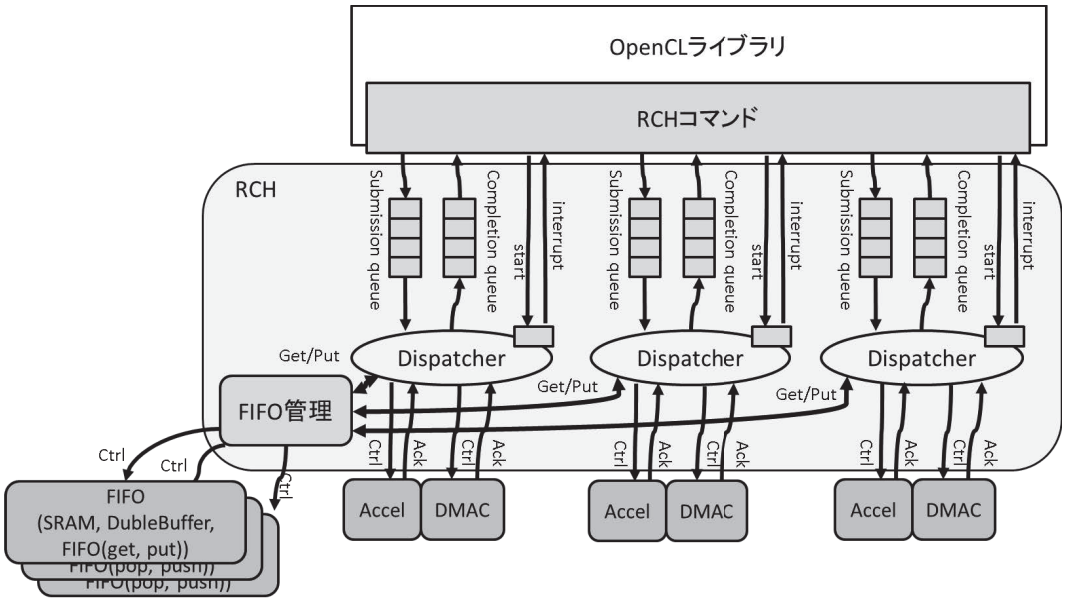

図 4.5.1 RCH **の概要**

の RCH コマンド対応生成部のタスク並列部にて行う.

4.5 RCH による実行制御

RCH はアクセラレータ，DMAC，FIFO の実行制御を，ソフトウェアに変わって行う．RCH の概要を図 4.5.1 に示す．パイプラインを行うためにはそれぞれのアクセラレータが異なるカーネルを実行する必要がある．そのため，それぞれのアクセラレータを独立して動作する必要がある．そこで，RCH は 1 つ 1 つのアクセラレータを独立して動作させるために，アクセラレータに 1 対 1 に対応するディスパッチャから構成される．ディスパッチャは他のディスパッチャと独立してアクセラレータを制御する．この際，DMAC も合わせてディスパッチャから制御を行う.

また，パイプラインを行う際は，2 つのアクセラレータ間で FIFO を用いてデータのやり取りを行っている．FIFO を用いることで 2 つのアクセラレータ間での非同期なカーネル実行をサポートする．一方で，FIFO full や FIFO empty により 2 つのアクセラレータ間の同期を保証する役割を持つ．そこで，任意のアクセラレータ間での FIFO による通信を実現する必要がある.

そのため，本研究ではこれらの FIFO の管理をハードウェアとして一括して行う．FIFO の状態をハードウェアにて一括管理することで任意のアクセラレータ間での FIFO を簡単に利用できるようにする.

さらに，これらの FIFO に対する put/get 制御もハードウェアから行うことにより高速化を実現する．具体的には，RCH に FIFO を制御する GET/PUT RCH コマンドにより，FIFO の制御を行う．ディスパッチャは GET コマンドを受け取った場合，FIFO 管理モジュールに対して GET 制御を要求する．FIFO 管理部では FIFO のデータの有無を確認し，有効なデータが無

ければ，ディスパッチャはロックする．有効なデータがある場合は，ロックが解除される．同様に，ディスパッチャがPUTコマンドを受け取った場合は，FIFO管理部にPUT制御を要求する．FIFOに空きが無ければロックされ，空きがあればロックが解除される．このように，ハードウェアにてアクセラレータ制御，データ転送制御，FIFO制御を行うことで制御オーバヘッドを大きく改善する．

4.5.1 コマンドセット

RCHではパイプライン制御に用いる基本的な実行制御を抽象化したRCHコマンドを持つ．これらをのコマンドを表4.5.1に示す．

表 4.5.1 RCH **コマンドの一覧**

コマンド名	コマンドの機能
EXE	アクセラレータを起動する
READ	DMACを用いてOn-Chipのデータ Off-Chipへコピーする
WRITE	DMACを用いてOff-Chipのデータを をOn-Chipへコピーする
GET	FIFOからデータを取り出す
PUT	FIFOへデータを詰める
SYNC_HOST	ホストプロセッサと同期を行う

4.5.2 汎用化のためのアーキテクチャの隠蔽

本OpenCL環境とRCH環境は汎用のアクセラレータ間パイプラインの実行制御環境を目指している．一方で対象となるアクセラレータは多岐にわたるため個々のアクセラレータとRCH間でのインタフェースが重要となる．

- アクセラレータやDMACに対するインタフェース
  アクセラレータの制御に関しては，プロセッサのアドレス空間にマップされたRun/Done Clearレジスタを有するアクセラレータに対応する．また，DMACについても同様である．
- FIFOに対するインタフェース
  ハードウェアによるFIFOの実装は多岐に及ぶ．本RCH環境においては，
  - アドレスマップされたRAMを用いたFIFO
  - 複数のページ（2BK程度のメモリ）を切り替えるFIFO
  - ダブルバッファを用いた深さ2のFIFO

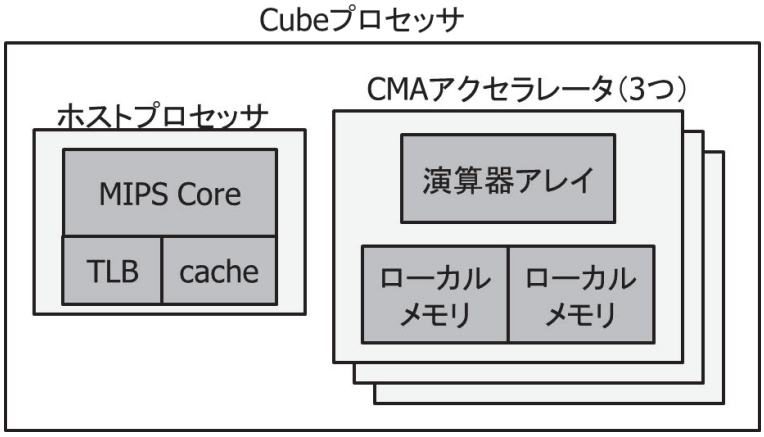

図 4.6.1 Cube **アーキテクチャの概要**

などに対応する．具体的には，FIFO 管理部の FIFO 側のインタフェースの制御部分をアーキテクチャに応じて設計することにより，適応が可能である．

4.6 評価環境とバス構成

評価では，我々が研究開発をすすめる省電力なパイプラインアクセラレータ Cube[13] を対象として，RCH と OpenCL を実装し評価を行った．図 4.6.1 に Cube の概要を示す．Cube は一つの MIPS ホストプロセッサと三つの省電力な CMA アクセラレータから構成される．評価では RCH やバス，DMAC を含むハードウェアを Xilinx 社の FPGA を用いて実装を行った．また，ホストプロセッサと CMA は実際の LSI チップを利用している．実際のチップを用いることで，動作周波数の向上を図った．また，CMA に対して高速なデータ転送を行うために，クロスバー方式のバスにて，RCH, ホストプロセッサ，CMA, DMAC，主記憶等を接続した．さらに，ソフトウェアの評価には，2 次記憶や Uart などの I/O が必須となるため，これらは，外部の FPGA ボード上に実装を行なっている．本ボードには，Xilinx 社の ML605[37] を利用した．評価環境のブロック図を図 4.6.2 に示す．また，評価環境の写真を図 4.6.3 に示す．これらの評価環境の詳細については，5 章で詳しく述べる．また，本評価では，利用する FPGA のピン数の制約により，CMA の数を 1 とした．CMA 数が 2，3 での評価は今後の課題である．

4.7 RCH を用いることによる実行制御オーバーヘッドの削減

上記の環境上にて RCH を用いることでの実行速度の評価を行う．本節では，実機上でアプリケーションの速度向上について評価する．エネルギー効率の向上については，本速度向上をもとに次節にて試算する．

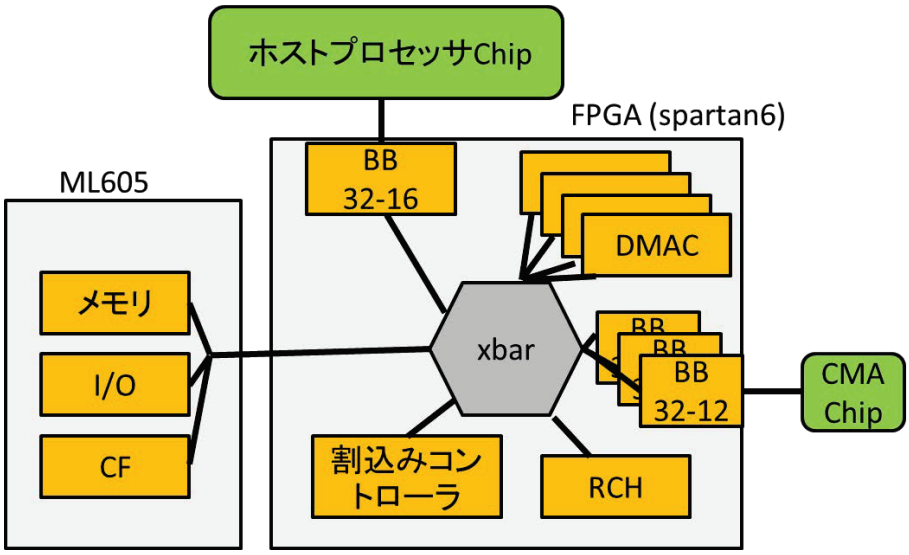

図 4.6.2 RCH を含むバス構成のブロック図

4.7.1 評価内容

本評価では，RCH 実機上で CMA アプリケーションを OpenCL から制御した際の全実行時間について計測を行った．また，本評価ではアプリケーションとしてセピア変換を用いた．下記に実験の詳細を記す．

- シーケンシャルな実行
  評価では，OpenCL フロントエンド，ライブラリによる資源割り当て，CMA コマンドの生成，RCH へコマンドをエンキュー，RCH による演算実行，コンプリーションコマンドのデキューを順に行う．本来はそれぞれがオーバラップ可能な処理であるが，基礎評価のためこれらをシーケンシャルに行った．

- ベンチマークアプリケーション
  CMA アプリケーションにはセピア変換を用いた．入力データを 2K バイトに分割し，CMA の LM に転送する．その後，CMA にてセピア変換を行い，演算結果を主記憶へ書き戻す．

- ソフト実行と RCH 実行
  評価では，RCH を用いて CMA・DMAC を制御した場合と，ホストプロセッサ上のソフトウェアから直接 CMA・DMAC を制御した場合の実行時間について比較する．また，本評価では OS は用いすに，ホストプロセッサ上で動作するアプリケーションとして評価プログラムを作成した．

- 動作環境
  表 4.7.1 に動作環境を示す．

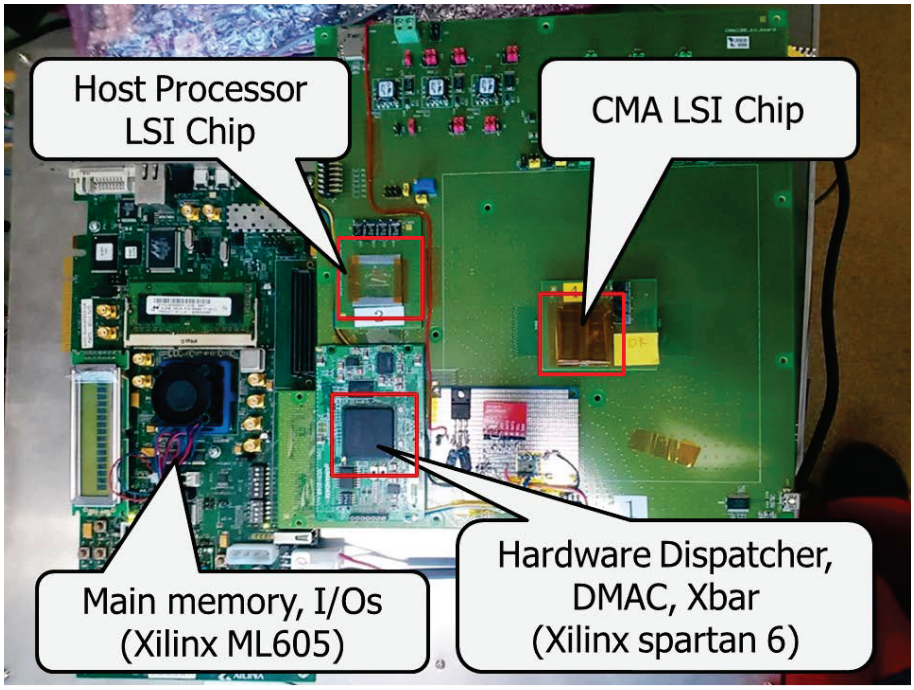

図 4.6.3 評価環境の写真

表 4.7.1 動作環境のパラメタ

CMA の動作周波数	25Mhz
ホストプロセッサの動作周波数	50Mhz
クロスバー，DMAC,RCH の動作周波数	25Mhz
CMA の個数	1 個

4.7.2 Sepia 変換における実環境での総実行時間

これらの環境にて，アプリケーションの総実行時間の計測を行った．これらの結果を図 4.7.1 に示す．データサイズは 2K バイトに分割したデータに対する演算を行う回数である．図 4.7.1 では OpenCL ライブラリにて要する時間，CMA・DMAC の制御に要する時間について示している．OpenCL ライブラリにて要する時間は，OpenCL のフロントエンド，資源割り当て，CMA 制御コマンドの生成からなる．また，CMA・DMAC の制御に要する時間に関しては，ソフトウェア制御の場合，制御の時間と CMA・DMAC での演算時間の和となっている．また，RCH の場合は，コマンドのエンキューとデキューの時間，RCH による制御と CMA・DMAC での演算時間の和を示している．これらより，ソフトウェアによって制御した場合と，RCH を用いた場合の総実行時間は大きく変わらない結果となった．N=64 について注目した場合，ソフトウェア制御と，RCH 制御では全体の 3 分の 1 が OpenCL ライブラリによる処理であることがわかる．(OpenCL Front-End, Resource Map, Create CMA Cmd）ソフトウェア実行につ

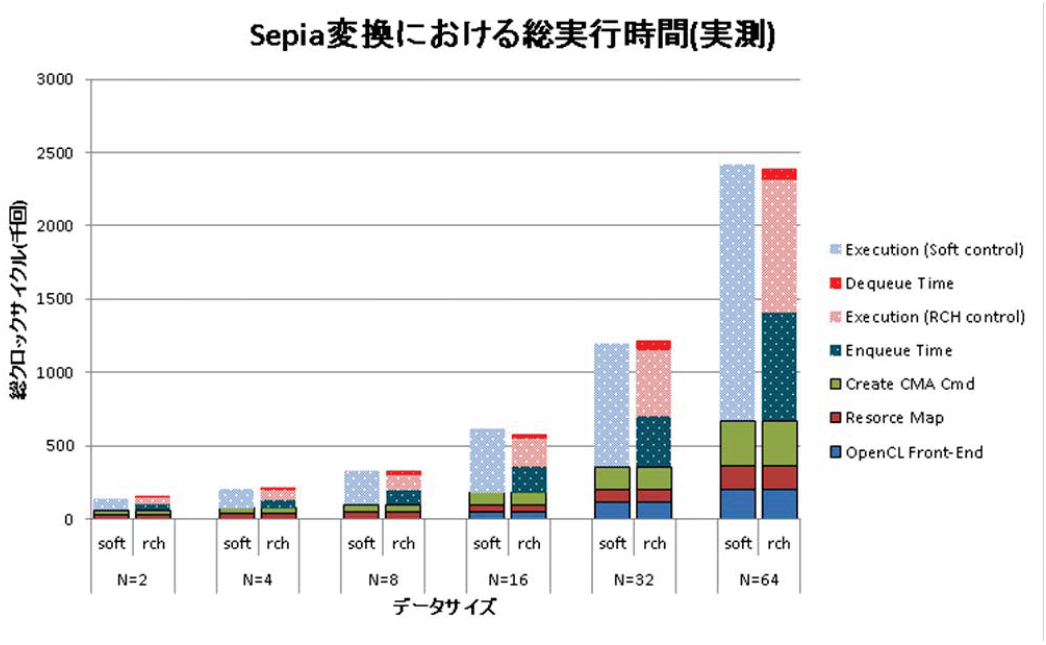

図 4.7.1 Sepia **変換での実行時間**

いては，残りの 3 分の 2 が演算と制御にかかる時間である．RCH では，RCH が制御を行っている時間は，ソフトウェアの半分以下であるが，RCH へのコマンドのエンキューとデキューが大きな時間を占めており，全体として速度向上が見込めない結果となっている．しかし，エンキューとデキューに関しては RCH の演算とオーバラップが可能であり，隠ぺい可能であるといえる．また，本 RCH 実装ではコマンドの入出力がシングルアクセスを用いており，バースト転送によるコマンドのエンキュー・デキューにより高速化が可能である．

4.7.3 RCH による実行制御時間の高速化

次に，これらの OpenCL ベンチマークの結果より CMA による演算時間と CMA の実行制御，同期制御，DMAC へのデータ転送制御に要する時間の変化について示す．これらの結果を図 4.7.2 に示す． これらの結果より，RCH を用いた場合 N=64 の際には全体で 48%の実行時間の改善を達成できることが確認できる．CMA に対する実行制御，DMAC へのデータ転送制御を RCH によって高速化できたためである．このため，本提案手法でアクセラレータ制御を高速化することができたと言える．

4.7.4 アクセラレータと DMAC 制御の詳細

次に，ソフトウェアによる実行制御と，RCH による制御での制御オーバヘッドについて示す．本評価では，データサイズが N=4 の場合の CMA・DMAC の動作を示す．これには，

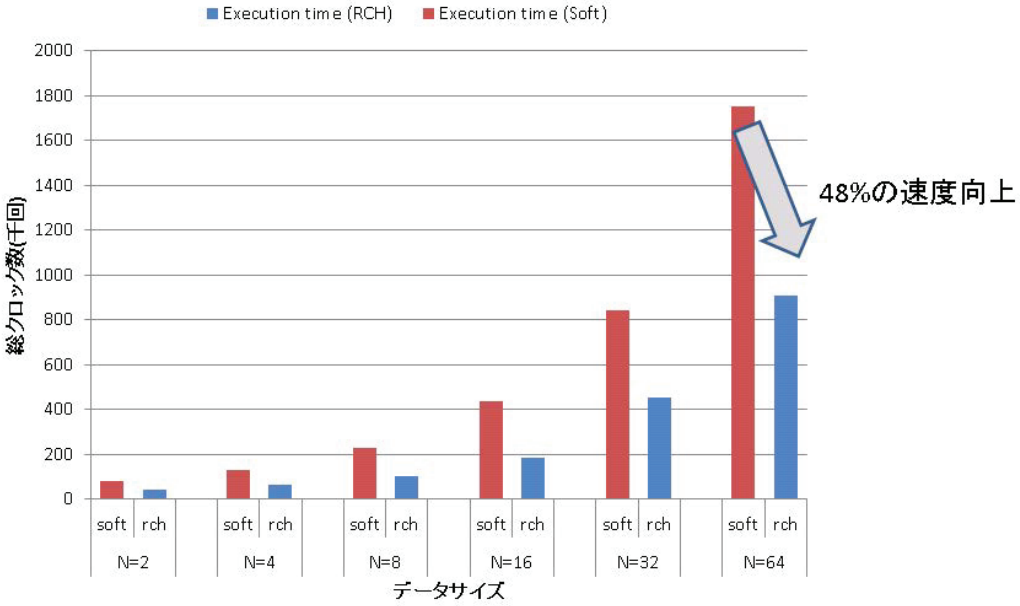

図 4.7.2 演算時間と実行制御時間の変化

Xilinx の ChipScope を用いた．ChipScope を用いることで，リアルタイムに FPGA 内部の回路の動作を確認することができる．これらの結果を図 4.7.3 に示す．図中の赤い箱の部分は DMAC の稼働時間を示し，青い部分は CMA による演算が行われている時間を示す．これらより，RCH を用いた場合には DMAC によるデータ転送と，CMA による演算がオーバラップしていることがわかる．さらに，DMA 転送が絶え間なく行われていることから，効率よく制御が行われていることが確認できる．一方で，ソフトウェア実行では，DMA 転送と次の DMA 転送の間隔が大きいことがわかる．さらに，CMA による演算と DMAC によるデータ転送のオーバラップもできていないことが確認できる．ソフトウェア制御では CMA・DMAC の利用率が低いことが確認できる．よって，RCH にて実行制御を行うことにより，実行速度を大きく改善することができたと言える．

4.8 エネルギー効率の算出

本節では，前節にて得られた速度向上の値より，エネルギーを試算する．現行の環境では，FPGA 中に RCH, DMAC, XBAR を実装し，CMA とホストプロセッサは実際の LSI を利用している．そのため，プロセスや構成の違いにより，単純に消費電力の比較を行うことができない．そこで，提案手法と従来手法の回路規模を基準にして消費電力の算出を行う．さらに，この消費電力に，実測の演算時間をかけることによりエネルギーを求める．これらをもとに，既存手法と提案手法のエネルギーの比較を行う．

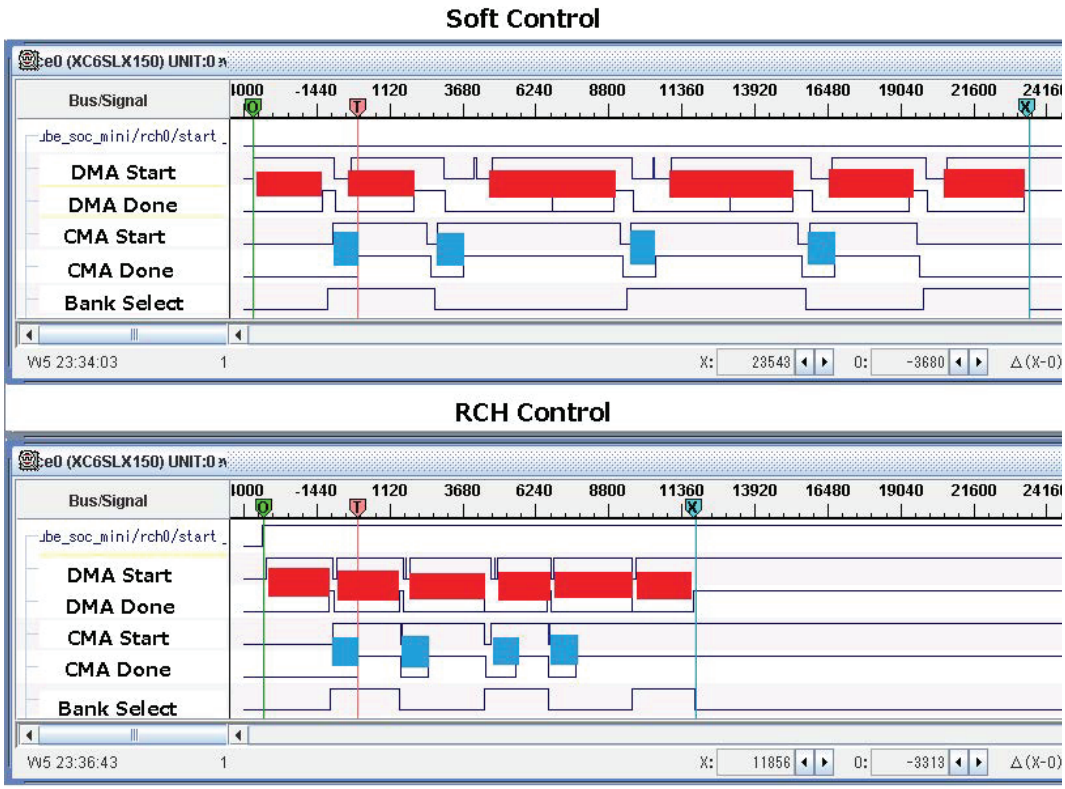

図 4.7.3 ソフトウェアによる CMA・DMAC 制御と RCH による制御の詳細

4.8.1 エネルギーモデル

一般的に，プロセッサの電力 (P) は動作周波数 (f), 回路の静電容量 (C), 回路の電圧 (V) より，

$$P = fCV^2 \tag{4.1}$$

となる.（静的なリークはないものと仮定する）また，現在のホストプロセッサと CMA と提案手法の回路部分の合計電力 $P_{host+cma+rch}$ は，

$$P_{host+cma+rch} = P_{host} + P_{cma} + P_{rch} \tag{4.2}$$

となる．式 (4.2) の右辺の $P_{host} + P_{cma}$ の部分は実機の実測値より算出が可能である．よってここでは，P_{rch} に着目する．P_{rch} は式 (4.1) より，

$$P_{rch} = f_{rch}C_{rch}V^2 \tag{4.3}$$

となる．ここで，静電容量は回路規模に比例すると仮定することとすることにより，C_{rch} を求める．具体的には，ホストプロセッサの回路規模と提案手法による RCH の回路規模を比較することにより求める．ホストプロセッサの静電容量を C_{host} ，ホストプロセッサの回路規模を R_{host} ，提案手法部の回路規模を R_{rch} とすると，求めたい C_{rch} は

$$C_{rch} = C_{host}\frac{R_{rch}}{R_{host}} \tag{4.4}$$

とすることで算出する． R_{rch} は FPGA の合成ツールが出力するリソース数を用い，R_{host} はホストプロセッサの HDL を FPGA に実装した際のリソース数から算出する．また，C_{host} については，式 (1) より，実チップの電力から算出する．よって，合計電力は式 (4.2)(4.3)(4.4) より，

$$P_{host+cma+rch} = P_{host} + P_{cma} + f_{rch}C_{host}\frac{R_{rch}}{R_{host}}V^2 \tag{4.5}$$

となる．また，エネルギーは，演算に要する実行時間を式 (4.5) にかけることにより算出する．最終的に求めたいエネルギーを E，アプリケーションの実行時間 $T_{application}$ とすると，
$E = P_{host+cma+rch}T_{application}$
と求めることができる．また，実行時間は前節で示した実機環境より計測した実測値を用いる．

4.8.2 実機における各種パラメタ

ここでは，電力モデルに必要なパラメタについて示す．また，アプリケーションの実行時間については，セピア変換，blender, IDCT を行った場合について示す．また，128KB の入力データに対して演算を行った際の時間を示している．表 4.8.1 に動作周波数，表 4.8.2 に回路規模，表 4.8.3 にチップ単体の電力，表 4.8.4 にアプリケーションの実行時間を示す.

表 4.8.1 動作周波数

CMA の動作周波数	25Mhz
ホストプロセッサの動作周波数	50Mhz
クロスバー，DMAC,RCH の動作周波数	25Mhz

表 4.8.2 回路規模（FPGA の LUT を指標として利用）

ホストプロセッサ (FPGA に実装した場合のリソース)	21,309
XBAR + DMAC + RCH	4,947 (XBAR:2,530, DMAC;1,125, RCH:1,292)

4.8.3 エネルギー効率と考察

エネルギーモデルと各種パラメタから算出したエネルギー値を図 4.8.1 に示す．図 4.8.1 より，3 つのすべてのベンチマークにおいて，RCH を用いることで，演算に要するエネルギーを大きく削減できることが確認できる．これらの主なな要因として表 4 からわかるように，大きくアプリケーションの実行時間を高速化できたためである．3 つのベンチマークともに 2 倍程度高速化を達成している．回路規模としてみた場合は，25%程度回路規模が増加しているが，動作周波数もプロセッサの半分のため，回路増加による電力増加は全体の 10%程度である．

4.9 アクセラレータ向け省電力化環境のまとめ

本研究では，マルチコアパイプラインアクセラレータの省電力化を目指し，実行制御に優れた専用のハードウェアを提案し，演算を高速化することでのエネルギー効率の向上の手法について明らかとした．旧来はソフトウェアで行なっていたマルチコアパイプラインアクセラレータにおける同期処理や，アクセラレータの実行制御，データ転送を専用のハードウェアにて行うことで，演算の高速化を達成した．また，実環境上に本提案手法を実装し，演算速度の高速化を確認した．これにより，大きなエネルギー効果が得られることが明らかとなった．具体的には，5 割程度エネルギー効率を改善することができた．このように，ボトルネックとなる実行制御をハードウェアにて高速化することで，大きくエネルギー効率を改善することができることを明らかとした．

表 4.8.3 チップ単体の電力

ホストプロセッサチップの電力	12.1mW
CMA チップの電力	13.3mW

表 4.8.4 アプリケーションの実行時間 (実測値)

	Sepia	blender	IDCT
ホストプロセッサからの制御	35.0ms	28.1ms	116.7ms
RCH からの制御	18.2ms	13.5ms	69.3ms

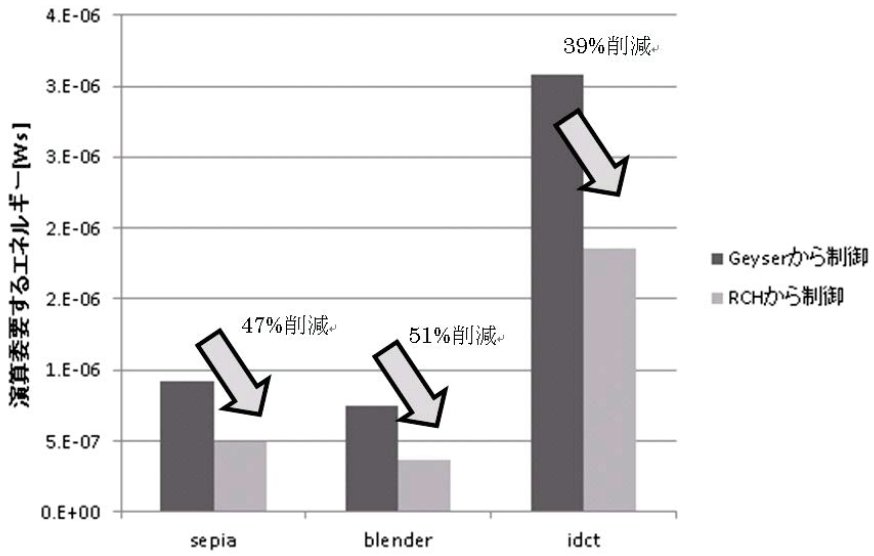

図 4.8.1 演算に要したエネルギー

さらに，パイプライン制御を汎用の並列プログラミング言語である OpenCL 環境から利用する環境を提案した．これにより，アプリケーションプログラマは，煩雑な FIFO を利用したパイプライン制御を行わずとも，高速かつ省エネルギーにパイプライン並列を利用できるようになった．

本研究では，ハードウェアを用いることで省電力化を実現し，さらに，ソフトウェアにて汎用のプログラミング環境を提供することで専用ハードウェアの隠蔽を実現した．このようにアプリケーションプログラマがハードウェアを意識することなく省電力化を実現する環境を明らかとした．

第5章　省電力プロセッサ向け電力評価環境の設計と実装

本章では，省電力プロセッサのための，電力評価環境の設計と実装について示す.

5.1　はじめに

近年，昨今の電力事情により計算機の省電力化が重要となっている．そのため，計算機システムの省電力化を目指した研究が盛んに行われている．その中で，我々は共同研究として，プロセッサの省電力化を目指した研究を進めている．本研究では，省電力ヘテロジーニアスマルチコアアクセラレータ (Cube) の研究と開発を進めている．4 章で示したアクセラレータも，これらの一環である．本研究では，回路レベル，アーキテクチャ，システムソフトウェアが協調することによる省電力化を目指している．また，シミュレーションレベルでの電力評価のみならず，実際にチップを作製し，実環境における省電力化研究を行なっている．

そのため，本電力評価環境は上記のすべての分野の評価を支える非常に重要な役割を担っている．実際に，上記のすべての分野で本評価環境は利用され，数多くの評価や成果物に大きく貢献している．一方で，それぞれの分野において，評価環境に対する要求は異なっており，それぞれの，要求を満たすことが重要となる．

本章では，これらの評価環境の要求や特徴，機能を述べる．はじめに，Cube プロセッサの概要について示す．さらに，電力評価のための評価環境における問題点と課題を示す．さらに，問題を解決するための電源評価環境の概要を示す．続いて実装について示す．さらに，電源評価環境を利用するためのソフトウェア環境について示し，最後にまとめを述べる．

5.1.1　省電力ヘテロジーニアスプロセッサ Cube

はじめに，我々が研究開発を行なっている省電力ヘテロジーニアスプロセッサ Cube[13] の概要を示す．Cube プロセッサは 4 枚のプロセッサを 3 次元方向に積層した 3 次元積層技術を用いたプロセッサである．さらに，Cube プロセッサは 1 つのホストプロセッサと，3 つの CMA アクセラレータから構成される．これらの概要を図 5.1.1 に示す．また，下記に，Cube プロセッサの特徴であるホストプロセッサの Geyser，アクセラレータの CMA について示す．さらに，Cube の大きな特徴である 3 次元ワイヤレス積層についても示す.

- Geyser

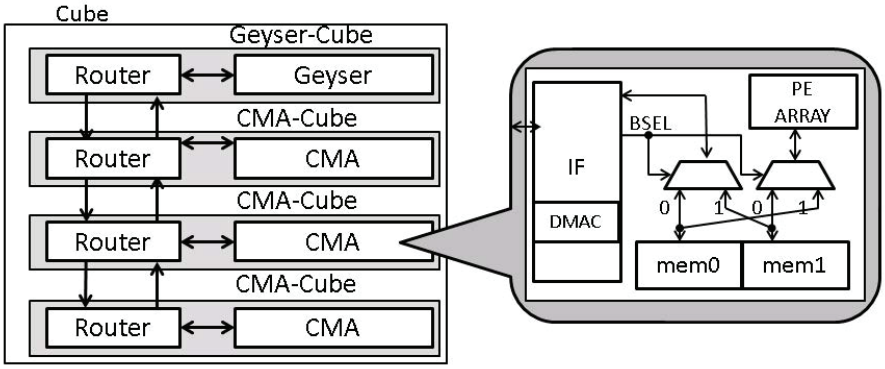

図 5.1.1 3 次元積層された Cube アーキテクチャの概要

Geyser は MIPS R3000 をベースとした汎用プロセッサであり，Cube の資源管理を行う．キャッシュや TLB を搭載し Linux や組込み OS が動作する．また，Geyser は細粒度パワーゲーティング技術を用いた省電力なプロセッサである．4 つの演算器である ALU, Shift, Mult, Div に対してそれぞれ，クロックレベルでパワーゲーティングが可能である．また，4 章でホストプロセッサと述べていたプロセッサは本 Geyser である．

- CMA
  CMA は電力性能に優れたアクセラレータである．演算器から構成されるプロセッシングエレメント (PE) を 80 個搭載している．PE は格子状に配置され，PE アレイを構成している．これらによって同時に数多くの演算を並行して行うことが可能である．さらに，データストア用のローカルメモリと PE へのデータ転送を制御するマイクロコントローラから構成されている．また，PE アレイは電源電圧を下げることにより省電力化を行う機能を備えている．

- 3 次元ワイヤレス積層
  近年プロセスの微細化の限界に伴い，チップを 3 次元方向に積層する 3 次元積層技術が普及している．3 次元方向に積層することで，より多くの回路を搭載することができる．Cube プロセッサは Geyser チップと CMA チップを 3 次元方向に積層したプロセッサである．さらに，Cube の大きな特徴として，3 次元積層されたチップ間を TCI(ThruChip Interface) を用いている点である．一般的には，3 次元積層には積層されたチップ間に TSV と呼ばれる物理的なピンをうつが，Cube は物理的なピンを持ちいず，チップ間近距離無線にてデータ転送を行なっている．TCI は TSV と比較して，コストが安く，省電力，高信頼，高速である特徴があげられる．

- 有線結合
  Cube は無線にて 3 次元積層されたチップ間を接続することを特徴としている．一方で，3 次元ワイヤレス積層技術を用いた場合，バス構成などのペリフェラルの構成を変更す

ることは難しい．そのため，Cube は Geyser チップと CMA チップを 3 次元積層せずに，有線にて配線する有線結合もサポートしている．有線結合を用いる場合は，3 次元積層はせずにそれぞれのチップを FPGA を介して接続する．4 章で述べた RCH を用いたアクセラレータの省電力化には，有線結合を用いている．

また，Cube は省電力化のための様々な機能を有する．先に述べたように，細粒度パワーゲーティングや電源電圧を降下することによる省電力化技術が搭載されている．次にこれらの詳細を示す．

- 細粒度パワーゲーティング
  細粒度パーワーゲーティングは，チップ内の回路レベルにて電源の On/Off を行う省電力機能である．一般的なパワーゲーティングと異なり，非常に細い面積，短期間で電源の On/Off できることが特徴である．具体的には，Geyser プロセッサの ALU, Shift, Mult, Div モジュールに対して，パワーゲーティング制御が可能である．さらに，クロックレベルでの制御が可能である．これにより，きめ細やかな電源制御を可能とし，高い省電力効果を期待できる．一方で，アクティブ状態と省電力状態の遷移にエネルギーを要する．そのため，短期間での省電力化機能の利用はオーバヘッドとなる．さらに，これらの課題を解決するのが，コンパイラとシステムソフトウェアの役割である．

- 動的電圧制御 (DVS)
  CMA の PE アレイは電源電圧を変化させることで電力を大きく改善することができる特徴を持つ．プロセッサや回路で消費される電力は，電圧の 2 乗に比例するため，電圧を下げることによる省電力効果は大きい．CMA の PE は組み合わせ回路から構成されており，PE の利用率が低い場合はクリティカルパスが短くなる．このタイミングで電圧を下げることにより，大きく消費電力を削減することが可能となる特徴を持つ．

さらに，細粒度パワーゲーティングや DVS による省電力効果が確認できるように，Geyser と CMA は複数の電源バンクを持つ．Geyser では，細粒度パワーゲーティング機構を備える 4 つのモジュールに対する電源バンクとその他のロジックや I/O などのバンクに電源が独立している．そのため，I/O や他のロジックの消費電流の影響を受けることなく，細粒度パワーゲーティングを行う部分のみの電流を観測ができる．また，CMA においても，PE 部分とその他のロジック，I/O は電源がそれぞれバンクに独立しており，PE のみで消費される電流のみを確認することができる特徴を持つ．

5.2 電力評価における目標

Cube プロセッサは省電力化のためのプロトタイププロセッサである．そのため，数多くの評価が必要である．また，回路技術，3 次元ワイヤレス積層技術，省電力アーキテクチャ，コンパイラ，システムソフトウェアレベルで，電力評価を行うが，それぞれ視点が異なるため，

それぞれのレベルの評価に対する要求を満たす必要がある．そこで，それぞれの階層での評価に対する要件をまとめる．さらに，要件を受け達成すべき目標を示す.

5.2.1 評価内容

はじめに，Cube プロセッサの評価内容について示す.

- Geyser チップ単体での簡易プログラム動作の確認
  Geyser チップは試作チップであり，基本的なバスアクセスやそれぞれの命令が正しく動作することから確認する必要がある．また，キャッシュや TLB を搭載しておりこれらの評価も必須である．そのために，簡単なプログラムが動作することを確認することが必要となる.

- Geyser チップでの Linux 動作の確認
  Geyser チップは MIPS R3000 をベースとしたプロセッサである．また，Linux を用いたパワーゲーティングの省電力化手法を提案している．Linux の動作には，大容量の主記憶や，ストレージや Uart などの I/O が必要となる.

- CMA チップ単体での動作の確認
  Geyser チップと同じく CMA チップも試作チップであり，基本的な動作の確認から行う必要がある．また，CMA による電力評価の場合も，より単純な環境での電力評価が重要となる．外付けの DDR メモリなどを用いると，データ読み書きの遅延により，CMA 本来の電力評価が難しくなる.

- 3 次元積層とワイヤレス通信を用いた Cube の動作確認
  Cube はアプリケーションプロセッサとして，世界で初めて 3 次元ワイヤレス積層を採用した実用的なプロセッサである．そのため，3 次元ワイヤレス積層技術の様々な評価が必要である.

- 有線結合を用いた Geyser と CMA の接続確認
  Cube プロセッサはリングバスを用いて，Geyser と CMA を接続している．また，CMA は Geyser のスレーブデバイスとして構成されている．マルチコアプロセッサでは，これらのプロセッサを接続するプロセッサ間のネットワークやバス構成が重要となる．そのため，他のバス構成やデバイスの構成も柔軟に対応できる環境である必要がある．そこで，Geyser と CMA を FPGA を介して有線で接続ができる評価環境構成をとる．また，4 章の RCH を用いた高速化の際は，これら有線結合を用いている.

- Geyser における細粒度パワーゲーティングにおける電力評価
  Geyser は細粒度パワーゲーティング技術を用いたプロセッサである．そのため，これらの電力評価を行うための環境が必要である.

- CMA における PE アレイの電力評価
  CMA は DVS をサポートするアクセラレータである．CMA の PE アレイに対して電圧を

変えることにより，電力を削減する機能を持つ．そのため，CMA の PE アレイに対する電流計測が必要である．

- CMA における DVS を行った場合の電力評価
  上でも示したように，CMA は DVS をサポートするアクセラレータである．そのため，CMA の PE アレイに 0.6V〜1.2V の電圧を制御しながら供給する必要がある．また，これらはソフトウェアから制御する必要がある．

- Geyser や CMA の温度が変化した場合の電力評価
  一般的に，チップの温度が変化した場合，消費電流が大きく変化する．そのため，チップの温度を変化させた場合の評価が必要である．

このように，電源評価環境では，これらの要件をすべて満たす必要がある．

5.2.2 電力評価環境の目標

上記のすべての評価を実現するため，下記に電力評価環境の目標を示す．

- Geyser での簡単なプログラムの実行
  単純な MIPS のプログラムを実行するための環境を目指す．Geyser は評価チップであり，命令レベルできちんと動作することを確認することが重要となる．また，低遅延なメモリや ROM を実現する．細粒度パワーゲーティングの場合，クロックレベルで電力効果が異なる．そのため，DDR メモリなどを用いた場合，読み出し遅延などにより，正確に省電力効果を評価できないためである．

- CMA での簡単なプログラムの実行
  CMA も Geyser 同様に，命令レベルで動作を確認することが必要である．また，Geyser と同じく低遅延な RAM をサポートする．

- 3 次元ワイヤレス積層と有線結合の双方をサポート
  Cube は 3 次元ワイヤレス積層を行うことを特徴としているため，3 次元ワイヤレス積層向けの評価を実現する必要がある．また，計算機アーキテクチャレベルで見た場合，マルチコアプロセッサなどのバス構成は非常に重要となる．そのため，これらのバス構成などを自由に変えることができる，有線結合のサポートも行う．

- Geyser での Linux の動作
  Geyser のパワーゲーティング制御はコンパイラと OS により，行われる．そのため，軽量 OS や Linux からの制御をサポートする．

- チップ毎の消費電力の計測
  ホストプロセッサ，CMA ともにエネルギー性能が高いプロセッサである．また，ホストプロセッサは細粒度パワーゲーティングによりエネルギー削減を行う特徴を持つ．CMA

もPEアレイ部分の電圧を制御することによる省電力化機構を有している．そのために，これら4枚のチップ（ホストプロセッサ1枚，CMA 3枚）のすべての消費電力の計測を実現する．

- CMAに対する電源電圧制御
  CMAのPEアレイ部分は電源電圧を変化させることによりエネルギー効率を向上させる機能を持つ．そのため，これらのCMAのための動的に電源電圧を変更するための電源を搭載する．また，3つのCMAのそれぞれに対して異なる電圧が設定可能なため，3つとも独立して制御できるようにする．

- 電流計測，電圧設定のためのインタフェース
  ホストプロセッサから電流値の参照，電圧の設定が可能であるインタフェースをプロセッサに提供する．これにより，Geyserからの電源制御を可能とする．

- 電力計測によりプロセッサの電力が消費されないこと
  電力計測を行う際に，その電力を計測する行為によって電力が増加する恐れがある．そのため，これらの影響を最小限に抑えることを目指す.

- チップの交換が容易にできること
  試作したチップはプロセスのばらつきにより，個体差がある．この個体差に対する評価も重要な評価項目である．また，試作チップのため動作不良が頻繁に発生する．そのため，簡単にチップを取り替え，再実験が容易な環境を目指す.

- 恒温槽のサポート
  細粒度パワーゲーティングでは温度が大きな要素となる．そのため，チップの温度を変えた場合の特性評価が重要である．この，温度を変化させる評価には恒温槽と呼ばれる装置を用いる．しかし，恒温槽には温めることができる対象の大きさに制約があるため，評価環境も小型にする．

5.3 電力評価環境の設計

本節では電力評価環境の設計について示す.

5.3.1 システムの全体設計

電力評価環境では主に，3つの部分から構成される．まずは，Cubeプロセッサを搭載する小型のボードである(以降LSIボード)．本LSIボード上にGeyserとCMAを3次元積層したCubeを搭載する．次に，ホストボードである．ホストボードには電源やFPGAチップ，簡単なI/Oを搭載する．ホストボード上にLSIボードを搭載する事で，簡単な評価を可能とする．3つ目が大容量のメモリやストレージを搭載するペリフェラル用のボードである．ペリフェラ

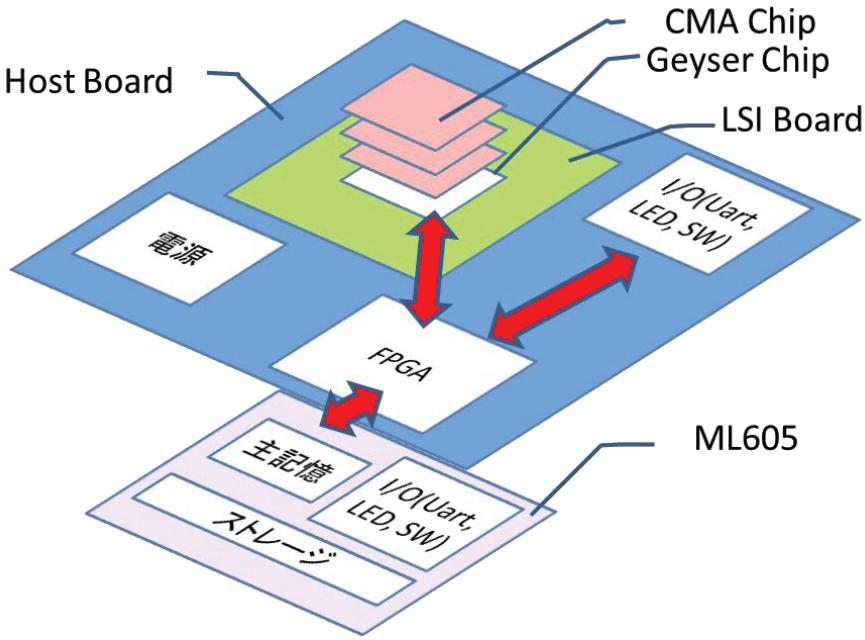

図 5.3.1 評価環境の構成

ルのボードは，Xilinx の ML605[37] を利用している．LSI ボード，ホストボードに関しては，新たに設計を行い，基板を作製した．これらの 3 つのボード構成の概容を図 5.3.1 に示す．

3 つのボードに分類することで，評価をより効率よく行うことを可能としている．具体的には，LSI ボードとホストボードを用いることで，Geyser と CMA の簡単なテストから，3 次元積層や有線結合，各種電力評価，温度を変えた場合の評価を可能とする．さらに，主記憶や大容量のストレージを搭載した ML605 を用いることで，Linux の動作確認と，Linux を用いた電力評価を可能とする．以下に，LSI ボード，ホストボード，ML605 の特徴を示す．

- LSI ボード
  LSI ボードには，試作を行った LSI を搭載する．LSI は個体差の問題や歩留まりの問題があるため，チップのみを LSI ボードに搭載することで，簡単にチップの交換を可能とする．また，本 LSI ボードには，4 種のバリエーションがある．1 つ目は，3 次元積層を行った Cube プロセッサである．1 つの Geyser チップと 1 枚〜3 枚の CMA チップを積層したものである．2 つ目は Geyser チップ単体を搭載した物である．また，3 つ目が CMA 単体を搭載した物である．この様にすることで，Geyser 単体でのテスト，CMA 単体でのテスト，Cube のテストを可能とする．

- ホストボード
  ホストボードは LSI ボードとセットで用いることで，チップの簡単な評価を可能とする．そのため，各種電源から簡単なプロセッサ向けのペリフェラルを搭載する．具体的には，アナログ回路の各種電源，電流計測回路，DVS 回路とデジタル回路の，UART, SD,

FPGA を搭載する．FPGA を用いてチップと Uart 等を接続する簡易的なペリフェラルを実装する．具体的には，小規模なブートローダ，512K バイトのメモリ，Uart を FPGA に搭載する．さらに，これに，電流計や DVS 回路を制御する電源制御回路を含める．この様にすることで，LSI ボードとホストボードのみで多くの評価を可能とする．また，ML605 と接続するための FMC コネクタを搭載する．

- ML605
  Geyser チップはソフトウェアによる電源制御により，高い省電力化効果を実現するチップである．具体的には，細粒度パワーゲーティング機構を Linux 等のシステムソフトウェアから制御を行うことで，省電力化を達成する．そのため，Linux を動作させる必要がある．このためには，大容量な主記憶，ストレージ，Uart や Ethernet 等のペリフェラルが必要となる．そのため，これらを，Xilinx 社の評価ボードである ML605 を用いてこれらペリフェラルを構築した．具体的には，EDK を用いたプロセッサ環境を作製し，これを改良した．また，ML605 上にある FMC コネクタを介して，ホストボードを接続する．

この様に，3 つのボードに分類することで，評価を効率よく進めることを可能とした．

5.3.2 ホストボードの設計

次に，ホストボードの設計について示す．ホストボードは，LSI ボードとともに用いることで Cube の評価を行う事を目的とし，電力評価環境にて最も重要な役割を担う．そのため，ホストボードと LSI ボードで計算機システムとしての簡単な評価を行う事ができる必要がある．具体的には，LSI ボード向けの電源回路，計算機としての簡単なペリフェラルを搭載する．図 5.3.2 にこれらの概容を示す．また，ホストボードは簡単なプログラムの実行と電力評価の 2 つの目的を持つ．そのため，それぞれについて下記に設計の方針を示す．

- 簡単なプログラムの動作環境
  Geyser や CMA の評価を行うために，ホストボード上に，ROM, RAM，Uart を搭載する．ROM 上のプログラムがロードされ，簡単なプログラムの評価が可能である．ROM や RAM，Uart は FPGA 上に実装した．また，簡単に異なるプログラムの評価をロードして評価ができるような構成とした．ROM の中に評価を行いたいプログラムを置くことも可能であるが，評価のたびに ROM を書き換える必要があり，面倒である．そこで，Uart 経由で任意のプログラムをロードし，評価プログラムを簡単に変更できるようにした．具体的には，ROM に小規模なブートローダを搭載する．その後，Uart 経由でプログラムをロードするようにした．このようにすることで，ROM の書換をしなくとも簡単に評価プログラムを変更できるようにした．

- 電力評価用マイクロコントローラ
  電力評価ではホストボード上に半導体電流センサを搭載することにより計測を行う．正確な電力評価を行う場合には，センサから読み込まれたデータに対して平均化等のフィ

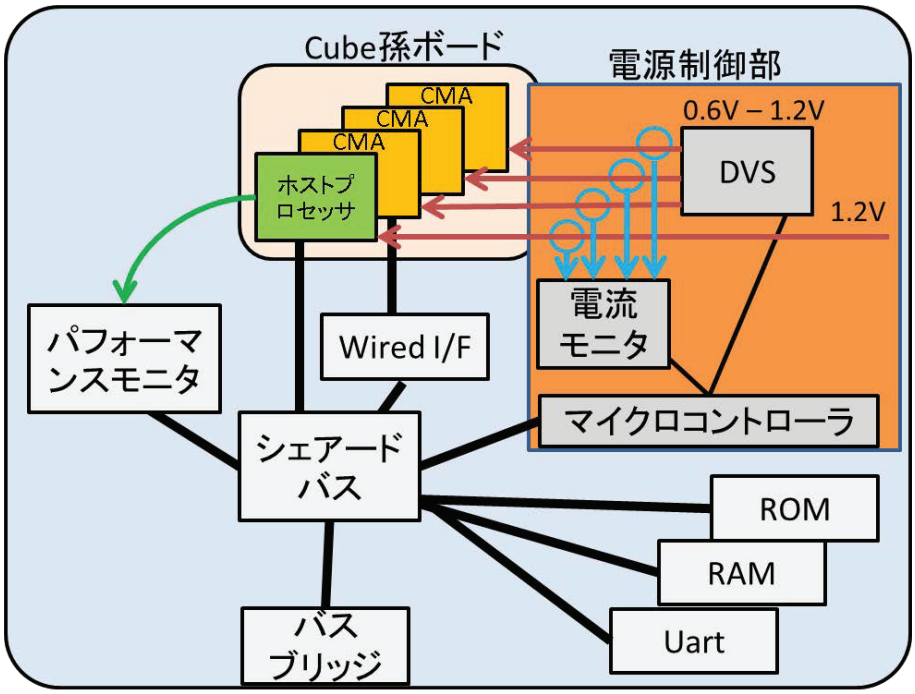

図 5.3.2 ホストボードの概要

ルタを適応することが望ましい．一方で，これらのフィルタ処理を Geyser で行う場合，電源計測を行うことで，消費電力が増加する恐れがあり，望ましくない．そこで，本ホストボードには電流センサや DVS などの電源部を管理するマイクロコントローラを別途用意する．このマイクロコントローラを用いることで，フィルタ処理や電流センサ，DVS 回路を制御する．このようにすることで，電流計測を行う行為による Geyser の電力増加を抑制する．また，これらのマイクロコントローラは Geyser のペリフェラルとして Geyser から制御できるようにする．このようにすることで，Geyser のベンチマーク実行と電流計測を同期することができるため正確な電力計測が可能となる．

次に，ペリフェラルやバス構成，電源等について詳細に示す．

- 入出力インタフェース
  ホストボードには最低限のプログラム評価や動作の確認用の入出力を設けている．これらによって，他の部品等を付け足すことなく評価を可能とする．まず，簡単な入力出力の確認ができるように，4 つの LED, 4 つの DIP スイッチを設けている．さらに，リセットスイッチなどに使う押しボタンスイッチを 2 つ搭載した．また，プログラムの入力やデータの確認用に利用するための Uart のポートを 1 つ搭載している．さらに，大規模なプログラムのロードやストアを想定し，Micro SD ソケットを搭載した．これらにより，ホストボード 1 つで，大半の評価ができるように設計を行った．

- ペリフェラルとバス構成

簡単なプログラムの評価を可能とするため，ROM，RAM，Uart を搭載することを述べた．これらは，FPGA 内部でシェアードバスを経由し Geyser から読み書き，及び制御できるようになっている．電力評価用のマイクロコントローラもアドレスマップの一部にマップされ，Geyser から制御できるようになっている．また，Linux 動作時には，ML605 に搭載された主記憶やストレージを利用する必要がある．そのため，Geyser からのバスアクセスアドレスがホストボードのいずれのモジュールのアドレスに該当しない場合に，ML605 側へバスアクセスが転送されるようした．

- 電源
  電流の計測については省電力効果の評価のために，Geyser の細粒度パワーゲーティング部分の電源と CMA の PE アレイ部の電源について電流の計測を行うようにした．さらに，Geyser や CMA のその他のロジック部の電流も確認できるように電流センサを搭載した．これにより，デジタル部全体で消費される電力を計測できるようにした．また，CMA の PE の電源の動的電圧制御を可能とするため，デジタル制御が可能な可変電源を採用した．これにより，ホストボード上のプログラムから電圧を制御可能とする．また，電流センサ，DVS 電源は電力制御用のマイクロコントローラから制御できるようにした，

5.3.3 ML605 の設計

Linux の動作のためには，大容量の主記憶とストレージが必要である．また，デバッグや実用的な評価のために Ethernet, USB, ディスプレイ出力を搭載する．このようにすることで，デバッグの効率を高める．また，ディスプレイ出力を用いることで，CMA を用いた信号処理等の結果を可視化することが可能となる．

5.4 電力評価環境の実装

次に，電力評価環境の実装について示す．実装では電源回路の詳細，バス構成などの詳細について示す．電力評価環境は LSI ボード，ホストボード，ML605 から構成される．また，LSI ボードとホストボード間は，ヒロセの小型コネクタを用いて接続している．このようにすることで，LSI ボードを簡単に交換可能にした．ホストボードと ML605 の間には，FMC コネクタを用いている．また，これらのコネクタ間の接続には Xilinx 社の FPGA である Spartan6 を用いた．本 FPGA 内部に，簡易 ROM や 512KB の小容量の RAM やバスを搭載する．図 5.4.1 にこれらの実装の際のブロック図を示す．また，図 5.4.2 に評価環境の写真を示す． 次に，ホストボードと ML605 の実装の詳細を次に示す.

5.4.1 ホストボードの実装

ホストボードは各種電源や LSI ボードを搭載するコネクタ，ML605 との接続用のコネクタ，FPGA から構成される．FPGA は市販の FPGA ボードをホストボードの上に搭載する形とし

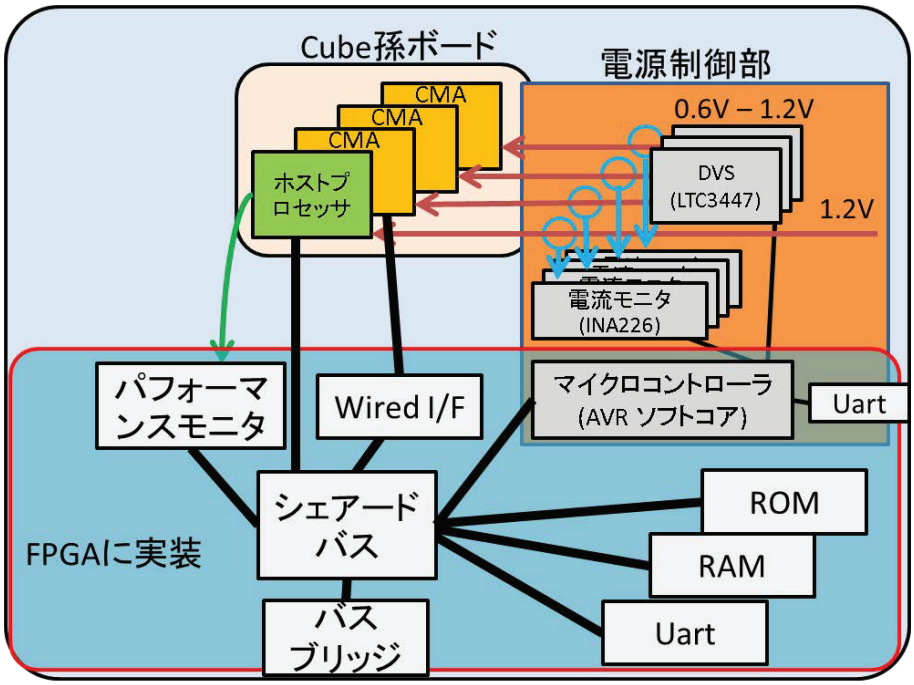

図 5.4.1 評価環境の実装におけるブロック図

た．FPGA チップを用いた基板作成は非常にコストが大きいため，市販の FPGA ボードを用いることで，電源と信号線をつなぐでけでよくなり，非常に開発が簡単になる特徴がある．

また，LSI ボードを接続するコネクタを 2 つ用意した．このようにすることで，3 次元ワイヤレス積層の Cube と Geyser と CMA を有線で接続した有線結合をサポートする．3 次元ワイヤレス積層を用いた場合は，1 つの LSI ボードに Cube を搭載し，1 つのコネクタに搭載する．一方で，有線結合を利用する場合は，Geyser チップを搭載した LSI ボードと，CMA を搭載した LSI ボードの 2 つを用意し，これらを，2 つのそれぞれのコネクタに搭載する．このようにすることで，有線結合を可能とし，4 章で述べたような RCH 環境の構築などに利用することができる．次に，FPGA 内部のバス構成，Uart 等のペリフェラル，LSI 向けコネクタ，電源回路について示す．

FPGA とバス構成

ホストボードでは簡単なプログラムの評価を行うために，ROM, RAM, Uart の簡単なペリフェラルを持つ．ROM を書き換えることなく評価プログラムをロードできるようになっている．具体的には，ROM にブートローダが書かれており，ブートローダから XMODEM を用いて Uart 経由で，ホストボードの RAM にプログラムをロードすることができる．このように，PC 側から Uart 経由でプログラムを実行できるため，ROM を書き換える必要がなく簡便に評価をおこなうことができる．これらは，すべて，ホストボード上の FPGA に実装している．

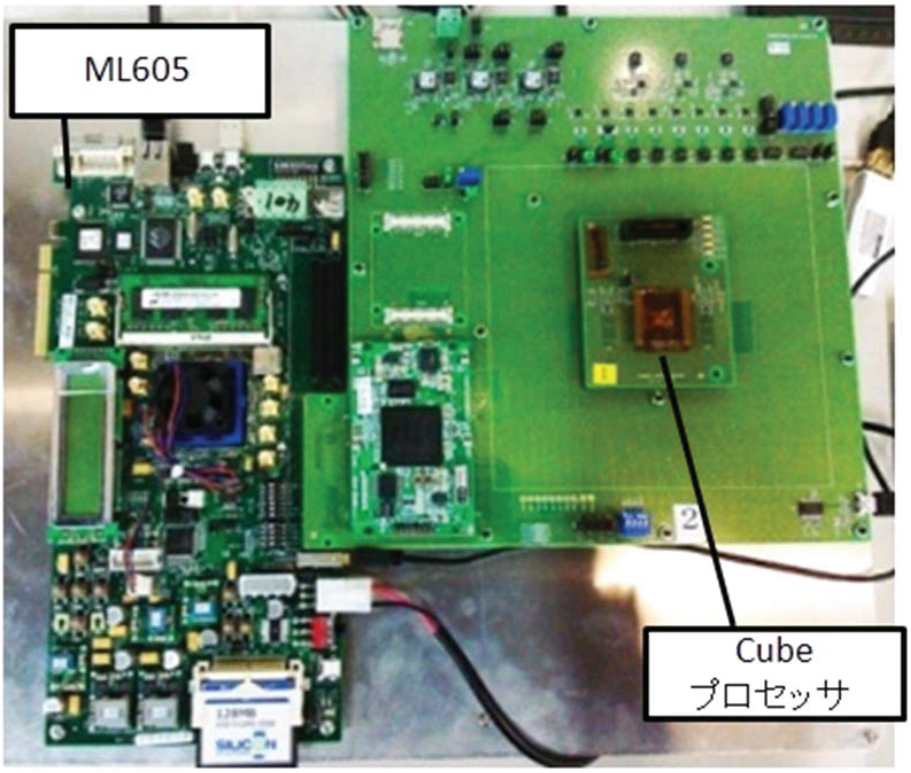

図 5.4.2 評価環境の写真

この他にも，ML605 へアクセスするためのバスブリッジや CMA を有線で接続するための CMA バスブリッジが含まれる．また，電源制御のためのマイクロコントローラはソフトコアとして FPGA 中に実装されている．マイクロコントローラと Geyser はシェアードメモリを介して，互いに制御が可能である．これらを利用することで，Geyser からの電流計測や CMA の PE の電圧制御が可能となる．また，これらのモジュールは簡単なシェアードバスによって接続されている．

ペリフェラル

ホストボードには，1 つの Uart，4 つの LED, 4 つの DIP スイッチ，2 つの押しボタンスイッチが搭載されている．Uart は FTDI の FT232RL[32] にて USB に変換され，PC に接続される．また，FPGA 内部には，Geyser のスレーブとなる Uart と電源制御のためのマイクロコントローラに接続された Uart の 2 つが搭載されている．そのため，DIP スイッチを用いて，これらの接続を切り替えることができるようになっている．また，押しボタンスイッチはリセットボタンになっている．

LSI ボード向けコネクタ

Geyser や CMA, Cube はテストチップのため個体差による電力の違いや，歩留まりによる不良等がある．よって，チップを頻繁に交換し，動作の確認を行うことができる必要がある．このために，LSI ボードを簡単に取り外しができるよに実装した．このコネクタにはヒロセ社のコネクタを用いている．さらに，ヒューマンデータ社の FPGA ボード [33] と互換性があるようになっている．このようにすることで，ピンコンパチブルな FPGA ボードを持ちいてデバッグが可能となる．

FMC コネクタ

ホストボードと ML605 の接続には，ML605 に搭載された FMC コネクタを介して接続する．FMC は 70 本のピンを利用可能であり，これらを介してホストボード上の FPGA と ML605 を接続した．また，ホストボードから ML605 へクロックを供給するようになっている．このようにすることで，ホストボード側の動作周波数を変更しても，ML605 側は FPGA の再プログラミングを行わずとも，動作させることが可能となる．

電源計測回路

電流の計測には，シャント抵抗を用いた電流計測方法を用いた．本方式では，電源ラインに直列に抵抗を入れ，シャント抵抗の両端にかかる電圧値をもとに電流値を算出する．また，抵抗は，流れる電流に対して無視できる程度の小さな値の抵抗値を用いている．一方で，これらの電圧値をデジタル変換する必要がある．そこで，本実装では，Texas Instruments 社の INA226[34] を用いることとした．INA226 はシャンと抵抗を用いた電力監視用の IC であり，簡単に電流値と電圧値を計測することが可能となる．さらに，これらの電流値と電圧値は I2C 経由で取得が可能である．本実装ではこれらを電源制御用のマイクロコントローラに接続している．

電源電圧制御回路

CMA の PE 部分は電圧を下げることにより，消費電流を大きく低下させることが可能である特徴を持つ．具体的には，PE アレイ中の PE の利用数が少ない場合は，クリティカルパスが短くなり，動作電圧を下げることが可能となる．PE の電圧は PE の利用率にあわせて，0.6V〜1.2V の間で動作させることが可能である．また，これらの電源電圧をソフトウェアから制御できる環境が必要である．そこで，PE 向けの電源のために Linear Technology の LTC3447[35] を用いた．LTC3447 は I2C 経由の制御にて，0.69V から 2.05V の間で電圧を出力することが可能である．CMA に 2.05V をかけると CMA が故障する恐れがあったため，電源制御用のマイクロコントローラ側で，最大の電圧を制限するようにしている．

電源制御用マイクロコントローラ

電流の計測には，平均化などのフィルタ処理が必要である．フィルタ処理を行うことで，電源のノイズを低減し電流計測が可能となる．一方で，これらの処理を Geyser で行うと，電流を計測する行為が消費電力を増やす要因となる．そこで，これらの問題を解決するために電源制御用のマイクロコントローラを用いている．さらに，基板上に新しくマイクロコントローラを搭載すると設計コストなどが増加するため，本実装では，FPGA 内部に AVR のソフトコアを用いることとした．このようにすることで，余計な部品が増えずに開発が容易になる．

AVR からは，上記で示した電流計測 IC の INA226 と電源電圧可変用の LTC3447 を I2C 経由で接続した．また，AVR には Uart を接続し，電流の計測結果等を直接ホストボード外へ出力できるようにした．このようにすることで，容易に電力の計測を可能とした．また，Geyser とは共有メモリを介して AVR を接続する方法を用いた．これにより，AVR の制御を Geyser からできるようにした．このようにすることで，AVR のプログラムを供給メモリ経由で簡単に再プログラミングできるようになっている．さらに，評価プログラムの実行と電流計測を同期して計測が可能となる．これらの，結果は Uart を経由して PC に送信される．

5.4.2 ML605 の実装

ML605 には主記憶やコンパクトフラッシュを始めに，Ethernet や USB, デスプレイ出力を搭載する．これらのペリフェラル部分の実装は，研究ではないが必ず必要であるため，可能であれば手間がかからないほうが望ましい．そこで，本環境では，これらに Xilinx の EDK[36] を利用した．EDK はソフトウェアコアを用いた組込みプロセッサ評価環境であり，ソフトコアプロセッサや，Ethernet, USB，ディスプレイ等の各種ペリフェラルが標準で利用可能である．本 EDK 環境から，ソフトコア部分を取り除き，ML605 の FMC 部分にバスを引き出すことでホストボードと接続できるようにした．このようにすることで，ペリフェラル開発のコストを大きく削減することができる．

5.4.3 可視化プログラム

電流の計測や分析等は上記の評価環境を利用することで，簡単に評価が可能となる．AVR からの Uart 経由で電流値や電圧値が出力される．電力評価や実験時はこれらの値を用いて評価を行なっている．一方で，これらの成果を視覚的に示すことも重要である．具体的には，研究によって得られて省電力効果を視覚的に示す必要がある．これにより，より直感的に研究の効果を示すことが可能となる．

そこで，Geyser や CMA の省電力効果をリアルタイムに可視化し，直感的に省電力効果が得られるようにした．具体的には，プロセッサの電流を計測し，リアルタイムに PC 上のディスプレイに表示する評価環境の構築を行った．これらの可視化した様子を図 5.4.3 に示す．図では，リアルタイムに消費される CMA や Geyser の消費電流を見ることができる．また，省電力化研究の成果を適応した場合と，適応しない場合の比較をおこなっており，研究を適応した結

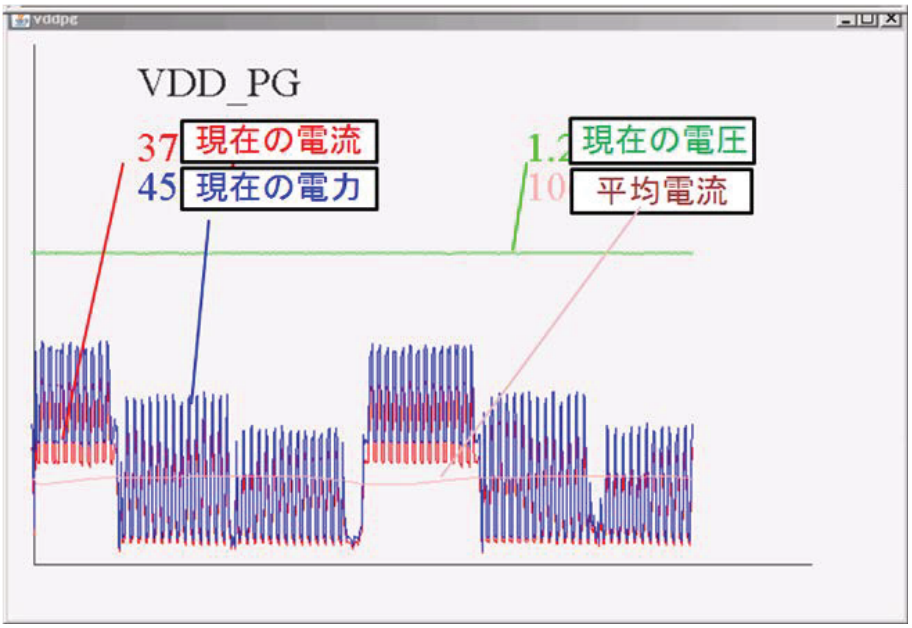

図 5.4.3 消費電力の可視化の様子

果，電流を削減できていることが直感的に見てわかるようになっている．これらは，実際に研究成果を適応した際の消費電力を示しており，省電力化研究の効果を直感的に示すことができたといえる．

5.5 まとめ

本章では，Cube プロセッサのための電力評価環境について示した．Cube プロセッサは回路レベルからソフトウェアレベルに至るまで，複数の階層の研究から構成されている．そのため，それぞれの階層ごとの評価を実現するための電力評価環境について示した．はじめに，各階層ごとの評価内容について整理し，電力評価環境の要件を示した．また，要件を満たすべく目標を明らかとした．さらに，設計と実装について示した．これにより，本電力評価環境は回路技術，3 次元ワイヤレス積層技術，省電力アーキテクチャ，ソフトウェアのすべての階層における，それぞれの評価を可能とした．実際に，それぞれの階層の評価や分析に用いられており，本プロジェクトに対して大きく貢献している．

第6章　SSDをHDDのディスクキャッシュとして用いる省電力ストレージ

本章では，HDDの省電力化機能のオーバヘッドを考慮した，省電力キャッシュストレージについて示す．

6.1　研究の概要

近年，データセンタ等で扱うデータが爆発的に増大している．データセンタでは増大するデータに対応するために，多くの場合，容量単価の安価なHDDが多数用いられる．データセンタでは，消費電力削減が課題であり，データセンタにおける消費電力の30%がディスクによって消費されている[30]．そのために，HDDの省電力化手法が注目されている．

HDDの省電力化では，ディスクを止めることによるスピンダウンが利用される．ディスクの回転を止めることで，電力を削減することができる．一方で，ディスクが止まった状態から，ディスクを回転させるためのスピンアップには，大きなエネルギーが必要である．そのため，頻繁にスピンダウン，スピンアップを繰り返した場合，電力的に損となる．そのため，このような頻繁に発生するスピンダウンを抑制し，エネルギー的に損となるスピンアップを抑制することが重要である．

また，NAND型フラッシュメモリを記憶素子に用いたストレージデバイスとしてSSDが普及している．SSDはHDDと比較して優れたアクセス性能を発揮し，消費電力も比較的小さいという特徴を持っている．しかし，容量単価がHDDの十倍以上と高価な装置である．そのため，ビット単価で有利なHDDと省電力性に優れたSSDとを併用することによって，より省電力効果が期待できる．図6.1.1に示すように，主記憶と二次記憶装置との間の新たな記憶階層としてSSDを活用する方式が提案されている．[19–23]. SSDをHDDのディスクキャッシュとして用いることでHDDの遅延時間を隠蔽し処理速度の向上を行うことができ，結果的にエネルギー効率が改善する．また，SSDをディスクキャッシュとして用いることで，HDDへのI/O要求数を削減することが可能である．

本論文では，SSDをHDDのディスクキャッシュとして用い，そのディスクキャッシュにおいて，HDDのスピンダウンを行うことで，ディスクの省電力化を実現する．筆者らが既に実現しているSSDディスクキャッシュシステム[31]をベースとして，スピンダウン機構を追加することで，省電力化を実現する．

SSD上のディスクキャッシュがヒットしている間は，HDDをスピンダウンすることで電力を削減できる可能性がある．しかし，キャッシュミスしたときには，スピンアップのための電力

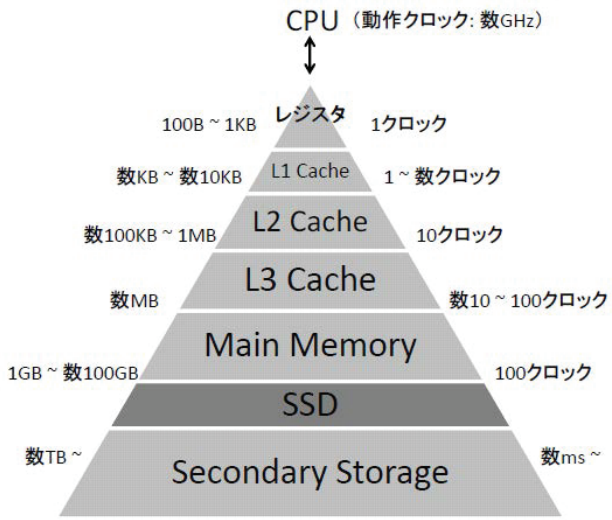

図 6.1.1 SSD **を含む記憶階層の例**

とアクセス可能になるまでのアクセス時間のペナルティが大きい．スピンアップのオーバヘッドを避けるにはSSDとHDDの両方を通電状態にすべきであるが電力的には損をする．これらのことから，適切なタイミングでのスピンダウンが極めて重要な課題となる．

6.2 本研究の目標

HDDの省電力化機能であるスピンダウンを利用した場合，スピンアップまでに，電力と遅延のオーバヘッドが生じる．そこで，本研究では，HDDとSSDの電力と性能を考慮した電源制御を行うことにより，オーバヘッドを抑えることを目指す．

そのために，電力モデルを用いた電力予測と電源制御を行う．本方式では，SSDに対してキャッシュミス時のHDDのread，writeの要求数を計測し，I/O要求の頻度を監視し，一定期間HDDへのアクセスがない場合に，スピンダウンする．この間隔については，HDDが持つスピンダウンに伴う電力オーバヘッドが損をしない時間を用いた．

評価ではHDDへ発行されるI/O要求パタンを参考にし，スピンダウンを行うまでの待機時間を設定した．本ディスクキャッシュシステムは，Linuxのデバイスドライバとして実現した．ブロックデバイスのデバイスドライバとして実現することで，特定のファイルシステムやHDDに依存しないで，本方式を適用できる．

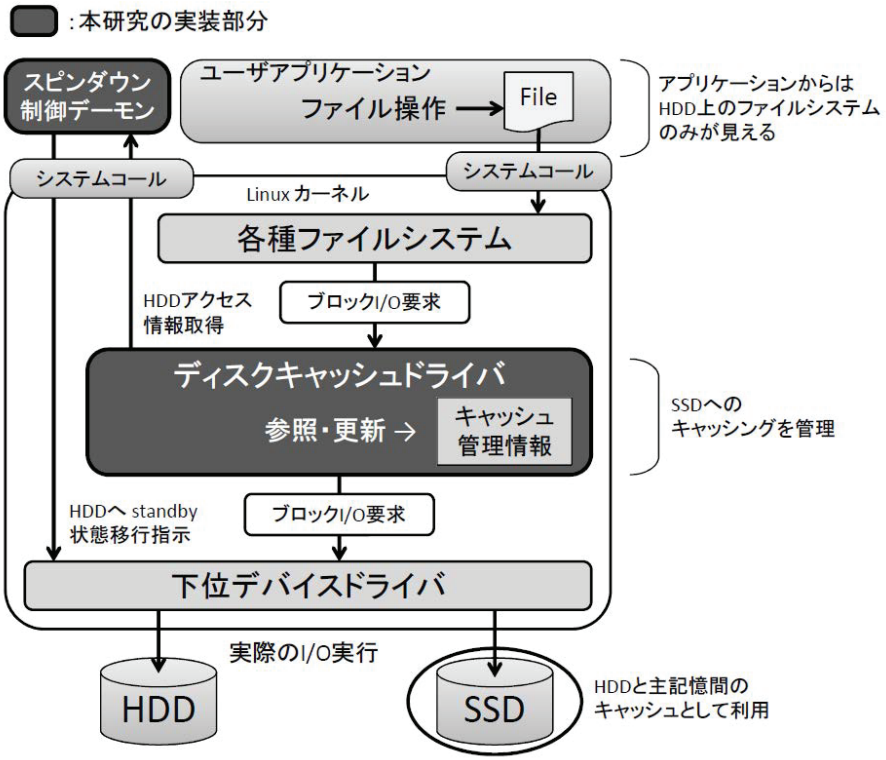

図 6.3.1 省電力 SSD キャッシュシステムの全体構成

6.3 省電力SSDキャッシュシステムの設計

本手法では，SATA などの SSD と HDD を組み合わせ，SSD を OS のデバイスドライバレベルでキャッシュとして用いる．デバイスドライバとして仮想化しながら HDD のキャッシュとして用いることで OS に修正を加えることなく，任意のファイルシステムで省電力化を行える．

図 6.3.1 に本省電力キャッシュシステムの全体構成を示す．Linux では，各種ファイルシステムやバッファキャッシュから HDD 等のブロックデバイスに対してブロック I/O 要求が発行される．本システムでは，ファイルシステムと HDD や SSD 等のブロックデバイスの間にディスクキャッシュドライバを導入しキャッシュ管理を行う．Linux からはブロックデバイスとして仮想化される．あわせて，新たに新設するスピンダウン制御デーモンがスピンダウンを実行する．次にディスクキャッシュと HDD の電源制御について述べる．

6.3.1 SSD による HDD の仮想化とディスクキャッシュドライバ

SSD を HDD のキャッシュとすることから，ファイルシステムから見た場合，HDD と SSD は一つのブロックデバイスとして仮想化されている必要がある．そのため，ディスクキャッシュドライバは，SSD をキャッシュとして利用しながら，HDD と SSD の組を単一のブロックデバイスとして仮想化し，ファイルシステムやユーザアプリケーションへ提供する．

本ディスクキャッシュドライバはSSDに対するブロック単位でのキャッシングの管理を行う．あわせて，ファイルシステムから発行されるブロックI/O要求の処理を行う．具体的には，ディスクキャッシュドライバはファイルシステムからのI/O要求を受けSSDにキャッシュされたブロックがあるかを確認し，キャッシュされていれば，キャッシュされたブロックをファイルシステムに返す．キャッシュミスした場合は，HDDからブロックを読み出し，ファイルシステムに返す．あわせて，SSDにブロックの新規割り当てを行う．

6.3.2 キャッシュ管理方式

SSDのキャッシュについては筆者らの文献[31]に詳細があるが，ここで，基本的な構成の概要を示す．ディスクキャッシュドライバでは，HDDからSSDへの領域の対応付けを管理している．これらは，ブロック単位での対応付けの管理を行い，キャッシュの置き換えなどを容易に行えるように固定サイズで行う．HDDとSSDのブロック対応表は主記憶上で管理する．

図6.3.2に本ディスクキャッシュドライバにおけるキャッシュ管理の概念図を示す.「セット」と「ブロック」という単位を用意し，領域の対応付けをセット単位で行い，セット内のデータの実際の割り当てをブロック単位で行うことで，データ割り当てにかかるオーバヘッドを最小に維持しつつ，対応表の管理コストを調節可能にする．各単位は次の通りに定義される．

- ブロック：SSDへの実データ割り当ての最小単位．2^N セクタで指定（Nは自然数）．
- セット：HDD領域とSSDキャッシュ領域の対応付けの単位．2^M ブロックで指定（Mは自然数）．

ディスクキャッシュドライバでは，HDDの領域をセット単位に区切って管理する．このHDD上のセットのことを「HDDセット」と呼ぶ．SSD上にはHDDセットに対応付けるセットを管理し，これを「キャッシュセット」と呼ぶ．キャッシュセットは，一つのHDDセットに対応付けられ，HDDセットとキャッシュセットの対応付けを対応表によって管理することで，任意のセットの対応付けを行える．また，HDDのセットとSSDのセット対応はLRUリストを用いて管理を行っている．そのため，キャッシュの置換が必要となった場合は，利用率の低いキャッシュセットを追い出すことが可能であり，キャッシュの利用効率が良い．

また，ディスクキャッシュドライバはライトバック方式をとっている．具体的には，キャッシュされたブロックへの書き込みが行われた場合，SSDにあるキャッシュブロックへのデータの書き込みを行い，セット対応表が管理するダーティフラグを有効にする．この際，HDDの対応するブロックへの書き込みはすぐには行わない遅延書き込みとなっている．以下では，SSDキャッシュにおいてダーティフラグが有効となったブロックのことを，ダーティブロックと呼ぶ．これらのダーティブロックのHDDへの追出しは，SSDのキャッシュ利用率がシステムで設定した閾値を越えた場合と，SSDをアンマウントする際に発生する．本論文では，この閾値として，基本的なシーケンシャルライトを行うマイクロベンチマークで閾値を変えながら実測して得た最良値を評価時のパラメータとして用いた．

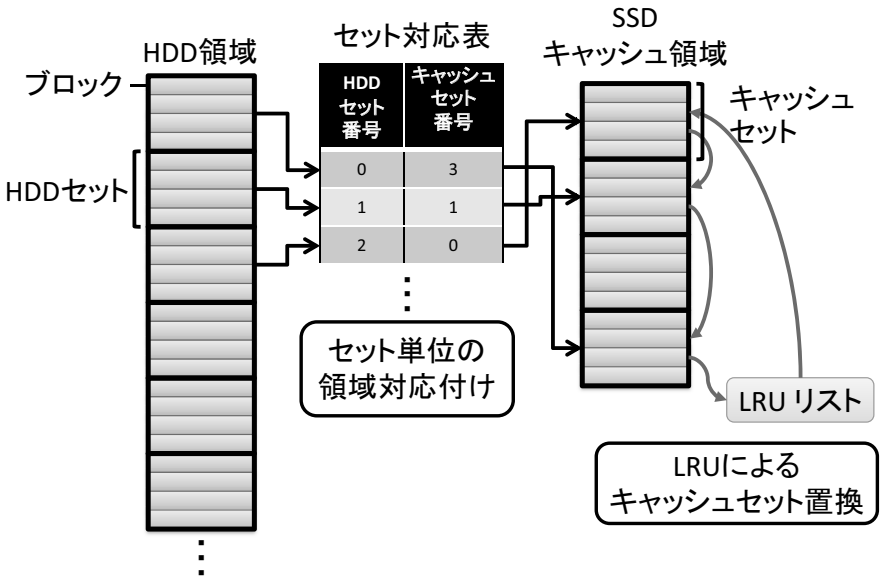

図 6.3.2 SSD **キャッシュ管理の概要**

本方式では，キャッシュブロックの対応やダーティフラグを主記憶上で管理している．そのため，ディスクキャッシュドライバは永続化を実現するために，キャッシュ内のダーティブロックの追出しを行う．これによって，SSD キャッシュと HDD の一貫性を保ち，永続化を行う．

6.3.3 ダーティブロック追出しによる I/O 制御

6.3.2 節で述べたように，SSD のキャッシュ利用率が閾値以上となった場合，ダーティブロックの追出しが発生する．しかし，閾値を越えた際にダーティブロックの追出しを行う場合，閾値近辺で，小規模な追出しが頻発し，HDD のアイドル時間を減らしてしまう．そこで，ダーティブロックの追出しを，一定量まとめて行うことによって，アイドル時間を増やす．これによって，HDD がスピンダウンできる時間を確保することができる．

6.4 HDD 電力削減手法

本方式の基本的着想は，SSD キャッシュで HDD への I/O を削減し，この HDD のアイドル時間にスピンダウンを行うことである．一方で，HDD をスピンアップする際は，ディスクの回転に伴いエネルギー・I/O 遅延のペナルティが大きい．そのため，得となるスピンダウンを行うには，ある一定以上スピンダウンの状態を維持する必要がある．この時間のことを損益分岐点 (BET : Break-Even Time) とする．BET 以上スピンダウンを行うことで，エネルギーを削減できる．一方で，HDD へのアクセスがなくなり次第スピンダウンを行ったにも関わらず，BET 以内に SSD キャッシュミスに伴うスピンアップが発生した場合には，電力的に損となる．

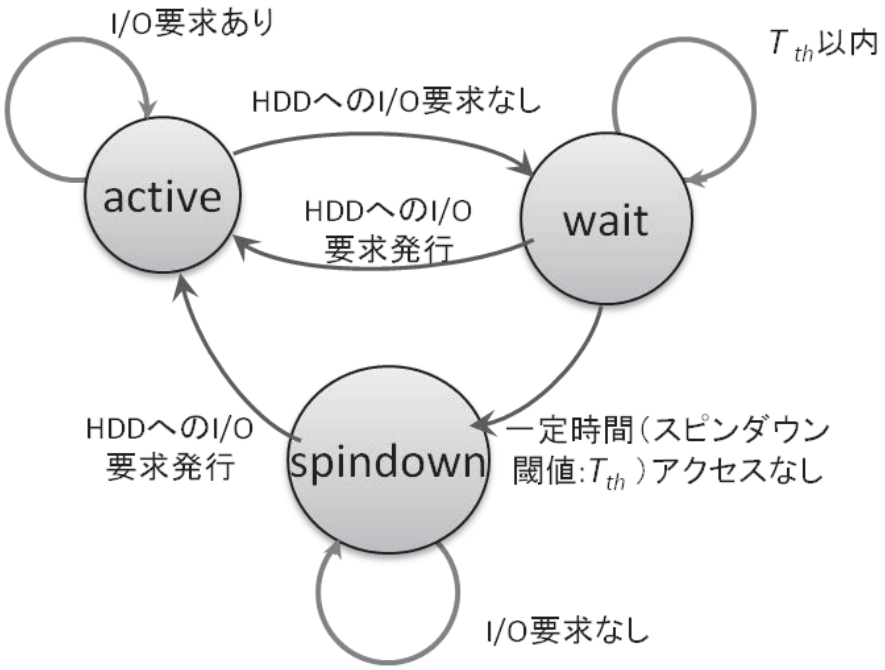

図 6.3.3 スピンダウン制御の状態遷移図

このような電力的に損となる状況に対応するために，本手法ではSSDディスクキャッシュからのHDDへのI/Oがなくなっても，すぐにはスピンダウンを行わずに，一定期間待つようにする．本スピンダウン制御の状態遷移を図6.3.3に示す．本スピンダウン制御手法では，通常状態のactive状態からスタートし，HDDのアクセスがない場合は，wait状態へ遷移する．wait状態において1度でもHDDへのアクセスが発生したら，active状態へ遷移する．一方でwait状態において，ある一定期間HDDへのアクセスがない場合，spindown状態へ遷移する．この一定期間のことをスピンダウン閾値（T_{th}）とする．spindown状態へ遷移した場合は，HDDのスピンダウンを実行する．また，spindownの状態でHDDのアクセスが発生した場合はactive状態へ遷移する．

本スピンダウン制御を行う疑似コードを図6.4.1に示す．まず，HDDへのI/O要求の有無の確認を行う(i)．本確認では，前回の確認以降にHDDへのI/O要求が発行されたか否かの結果を得る．確認した結果より，HDDへのI/O要求があった場合にはidle_timeを初期化し(ii)，HDDへのI/O要求がなかった場合にはidle_timeをインクリメントする(iv)．この時，idle_timeがスピンダウン閾値tthを越えていた場合にはスピンダウンを実行する(iii)．本処理手順を一定間隔で実施し，BETを考慮したスピンダウン制御を行う．

以上の制御により，頻発するBET以下のスピンダウン・スピンアップを回避する．一方で，アイドル時間がBET以上の場合でも，T_{th}の期間はアクティブとなるために，電力削減効果が小さくなる可能性がある．このため，適切にT_{th}を設定することが重要となる．

```
static idle_time = 0;   //HDDのアイドルサイクル数
tth;                    //スピンダウン閾値

I/O要求の有無の確認;          //( i )

if(I/O要求があった場合){      //( ii )
  idle_time = 0;
}else{ // (I/O要求がなかった場合)
  if(idle_time > tth){       //( iii )
    スピンダウンの実行;
  }
  idle_time++;               //( iv )
}
```

図 6.4.1 スピンダウン制御の疑似コード

6.4.1 エネルギーモデルとスピンダウン閾値の決定

本方式の特徴は，SSD キャッシュヒット中，HDD へアクセスしていない期間に HDD をスピンダウンすることにより，できる限りシステムが消費するエネルギーを削減することである．一方で，SSD ディスクキャッシュへのミスがいつ発生して HDD へアクセスするかを事前に把握することはできないため，HDD をスピンダウンするタイミングを的確に決定できないことが問題となる．そのため本方式では，6.4 節で述べたスピンダウン閾値 T_{th} を定め，T_{th} 経過後まで HDD へのアクセスがない場合に HDD をスピンダウンして省電力化を図る．

本提案方式では，スピンダウンを行わない状態の SSD キャッシュを適用した環境において，想定するワークロードを事前に実行させて得られたプロファイリングデータに基づいて T_{th} を定める．本システム上ではワークロードごとに異なる計算環境を用いることを想定しているため，実行環境とワークロードごとにプロファイリングを行う．実行環境に依存する主たるパラメータは，ファイルシステムの種別，SSD のサイズ，HDD と SSD の電力である．特に HDD のアイドル時平均消費電力 W_{active}^{hdd}，SSD のアイドル時平均消費電力 W_{active}^{ssd}，HDD のスピンダウン時の平均消費電力 W_{sleep}^{hdd}，HDD のスピンアップに要するエネルギー E_{spinup}^{hdd}，HDD のスピンアップにかかる時間 T_{spinup} については実行環境に共通の値として事前に計測しておく．

さらに上記の値を用いて，HDD がアクティブである時の電力 W_{active}，HDD がスピンダウンされた時の電力 W_{sleep} の値も事前に算出しておく．本システムでは HDD と SSD とを併用するため，HDD と SSD の両方の電力を用いてこれらの値を求める．算出式は次のように表される．

$$W_{active} = W_{active}^{hdd} + W_{active}^{ssd} \tag{6.1}$$

$$W_{sleep} = W_{sleep}^{hdd} + W_{active}^{ssd} \tag{6.2}$$

事前実行では，SSD ディスクキャッシュシステムから HDD への I/O 要求が発生してから次

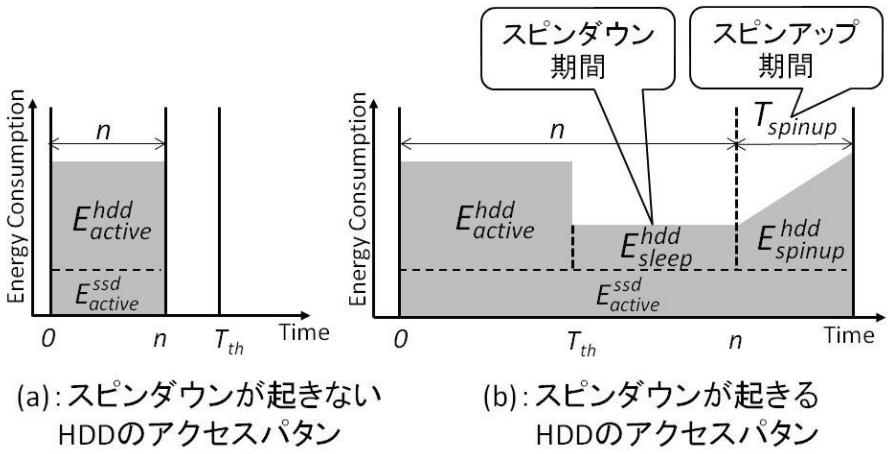

図 6.4.2 HDD **と** SSD **の消費エネルギーモデル**

の HDD への I/O 要求が発生するまでの時間間隔 n の発生回数 $C(n)$ をプロファイリングする．例えば，次の HDD へのアクセスまでが 10 秒間であった回数が 3 回あった場合は，$C(10) = 3$ となる．このように，特定のワークロードを実行したときの $C(n)$ をワークロードごとにプロファイルする．次に，ワークロードごとに計った本プロファイリングデータを用いてワークロード向けのスピンダウン閾値 T_{th} を決定する．この T_{th} を求めるために，T_{th} を変化させてエネルギーの推定値を机上算出し，その中からエネルギーが最小となる T_{th} を求める．以下，本論文におけるエネルギー算出方法について述べる．

エネルギーは T_{th} を基準として 2 種類のモデルで考える．図 6.4.2 に本提案方式における消費エネルギーモデルを示す．HDD に対する I/O 要求間隔 n がスピンダウン閾値 T_{th} より短い場合 (a) はスピンダウンが起きないため，HDD と SSD のアクティブ時エネルギー（E^{hdd}_{active}，E^{ssd}_{active}) を消費する．一方，n が T_{th} より長い場合 (b) はスピンダウンが起きるため，HDD のアクティブ状態とスリープ状態のエネルギー (E^{hdd}_{active}，E^{hdd}_{sleep})，SSD のアクティブ時エネルギー (E^{ssd}_{active})，HDD のスピンアップに要するエネルギー (E^{hdd}_{spinup}) を消費する．本モデルに対して，プロファイリングにより求めた HDD への I/O 要求間隔のデータをあてはめてエネルギー遅延積を算出することで，ワークロード実行時の総エネルギーを求める．以下，エネルギーはエネルギー遅延積とする．

スピンダウンが起きない場合のエネルギーを E_{th}，スピンダウンが起きる場合のエネルギーを $E_{active,sleep,spinup}$ とすると，本提案方式における総エネルギー E_{all} は次の式で表される．

$$E_{all} = E_{th} + E_{active,sleep,spinup} \tag{6.3}$$

本 E_{all} は，T_{th} により変動する値であるため，T_{th} の関数となる．また，$E_{active,sleep,spinup}$ は，スピンダウンするまでに要するエネルギー E_{active}，スピンダウンしている間のエネルギー E_{sleep}，スピンアップする際に要するエネルギー $E_{spinupall}$ の和により求めることができる．すなわち，

$$E_{active,sleep,spinup} = E_{active} + E_{sleep} + E_{spinupall} \tag{6.4}$$

となる．

式 (6.3) における E_{th} は，式 (6.1) の W_{active} に HDD がアクティブとなっている時間をかけたエネルギー遅延積として，

$$E_{th} = W_{active} \sum_{n=1}^{T_{th}} nC(n) \tag{6.5}$$

により求めることができる．HDD がアクティブとなっている総時間については，プロファイリングにより求めた $C(n)$ と HDD への I/O 要求間隔 n を用いて，$\sum_{n=1}^{T_{th}} nC(n)$ として求めることができる．

式 (6.4) の E_{active} については，式 (6.1) の W_{active} とその総時間数をかけたエネルギー遅延積として，

$$E_{active} = W_{active} \times T_{th} \sum_{n=T_{th}}^{T_{exe}} C(n) \tag{6.6}$$

で求めることができる．$\sum_{n=T_{th}}^{T_{exe}} C(n)$ は，スピンダウン閾値を越えてワークロードの実行時間 T_{exe} までに次の I/O 要求が発生した回数の総和である．

式 (6.4) の E_{sleep} については，式 (6.2) の HDD のスリープ時電力 W_{sleep} とスピンダウンしていた時間 $n - T_{th}$ をかけたエネルギー遅延積として，次の式で求めることができる．

$$E_{sleep} = W_{sleep} \sum_{n=T_{th}}^{T_{exe}} (n - T_{th})C(n) \tag{6.7}$$

上記 $n - T_{th}$ は T_{th} を起点として HDD をスピンダウンしていた時間であり，それに $C(n)$ をかけて加えたものが HDD をスピンダウンした総時間数である．

式 (6.4) の $E_{spinupall}$ については，一回のスピンアップのエネルギー E_{spinup} とスピンアップした回数の総和により，

$$E_{spinupall} = E_{spinup} \sum_{n=T_{th}}^{T_{exe}} C(n) \tag{6.8}$$

で求めることができる．なお，HDD のスピンアップ時のエネルギー E_{spinup} は，SSD の電力が加算されることを考慮して，

$$E_{spinup} = E_{spinup}^{hdd} + T_{spinup} \times W_{active}^{ssd} \tag{6.9}$$

という算出式で事前に求めておくものとする．

本方式では，ワークロードごとにプロファイリングした $C(n)$ に基づいて，上記の算出式 (6.1)〜(6.9) を用いて E_{all} を算出する．そして，E_{all} を最小にする，すなわち，エネルギーあたりの総オペレーション数であるエネルギー効率を最大にする T_{th} を，事前に簡単なプログラムで求める．この T_{th} はシステムの実行中ではなく，プロファイリング後に求めて，本システムへ運用時のパラメータとして与える．すなわち，そのシステムのユースケースを想定したワークロードから T_{th} を求めて設定することで，HDD の消費電力削減効果のより高い SSD ディスクキャッシュを実現する．

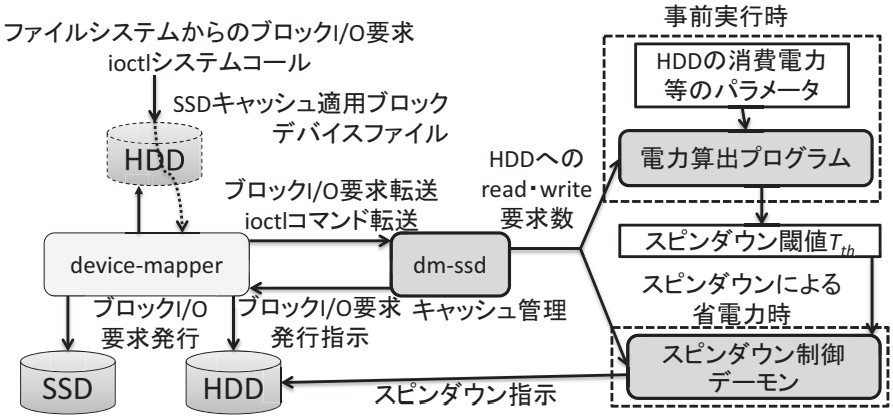

図 6.5.1 **ディスクキャッシュドライバの全体構成**

6.5 ディスクキャッシュドライバと HDD 自動スピンダウン機構の実装

ディスクキャッシュドライバは，ブロックデバイスの仮想化を device-mapper[38] のフレームワークを活用して実現する．device-mapper は，仮想ブロックデバイスの作成，管理に利用できる，ブロックデバイスドライバおよびそれをサポートするライブラリ群であり，Linux 2.6 系以降で利用できる．

ディスクキャッシュドライバでは，SSD キャッシュ適用ブロックデバイスの作成・削除，I/O 要求の受け付け，I/O の実デバイスへの発行などの機能を，この device-mapper により実現し，キャッシュ管理や，device-mapper へ I/O 発行を指示する部分をカーネルモジュール ‘dm-ssd’ として新しく作成した．ディスクキャッシュドライバの実装の全体構成の概略を図 6.5.1 に示す．device-mapper は，生成した仮想デバイスへの I/O 要求をファイルシステムから受け取ると，それを dm-ssd に転送し，dm-ssd の指示で，実デバイスへの I/O 発行を行う．また，I/O 完了後，dm-ssd から指定されたコールバック関数に処理を戻す.

dm-ssd は，キャッシュ管理情報を持ち，device-mapper から受け取った I/O 要求の内容とキャッシュ管理情報から，必要な実デバイスへの I/O 要求を構築し，device-mapper に指示する．また，I/O 完了後に呼び出されるコールバック関数により，ファイルシステムへ I/O 完了の応答なども行う.

6.5.1 device-mapper によるブロック I/O 要求制御

device-mapper の基本的な機能は，仮想的なブロックデバイスを作成し，その任意のセクタ範囲を，カーネル内の「ターゲット」と呼ばれるモジュールにマッピングすることである．マッピングされた範囲へのブロック I/O 要求は，マッピング先のターゲットに転送され，ターゲットが I/O 要求に様々な変換を施したり I/O の振り分けを行ったりすることが可能である．また，device-mapper を介することで，仮想ブロックデバイスにターゲット独自の ioctl コマン

ドを実装できる．本研究のディスクキャッシュドライバは，device-mapper に dm-ssd モジュールをターゲットとして登録し，仮想ブロックデバイスを作成してその全体を dm-ssd にマッピングすることにより実現する．

device-mapper の制御用インタフェースとして，/dev/mapper/配下の control キャラクタデバイスファイルによるシステムコールインタフェースを提供する．これを用いて，device-mapper に仮想ブロックデバイス作成，ターゲットへのマッピングの設定，仮想ブロックデバイスの削除などを指示する．

6.5.2 HDD 自動スピンダウン機構の実装

HDD 自動スピンダウン機構は，device-mapper を介した ioctl コマンドインタフェースを用いて，ディスクキャッシュドライバから HDD への I/O の要求数の情報を取得し，プロファイリングで決定した Tth より長く HDD へ I/O 要求が発生しない場合にスピンダウンを指示する．具体的には，6.4 節の図 6.4.1 の疑似コードで示した処理を行う．この HDD 自動スピンダウン機構はユーザ空間で動作するスピンダウン制御デーモンとして実装する．

6.5.3 プロファイリング

省電力 SSD ディスクキャッシュドライバを導入する環境において，初めにプロファイリングを行い，6.4.1 節で示したように SSD ディスクキャッシュドライバから HDD へ発行される I/O 要求を監視する．

dm-ssd モジュールは，HDD へ発行した I/O 数を ioctl を経由し取得できる．そのため，これらの I/O 数を監視し，HDD のアイドル時間と回数を記録する．これらの情報と 6.4.1 節のエネルギーモデルを用いて T_{th} を決定する．

6.6 評価

6.6.1 評価方法

提案手法のエネルギー効率を評価するために，従来方式の HDD 単体・SSD 単体と提案方式を含む 5 種類のブロック格納手法において，ベンチマーク実行時のエネルギー効率 (ops/J) を比較する．以下にこれら 5 種類のブロック格納手法を示す．

1. **HDD**：HDD を単体で使用する．
2. **SSD**：SSD を単体で使用する．
3. **SSD キャッシュのみ**：SSD キャッシュドライバにおいて，100GB の SSD の全容量をキャッシュとして利用する．パラメータを表 6.6.1 に示す．本手法は提案手法との比較のため

表 6.6.1 SSD キャッシュの設定

	SSD キャッシュパラメータ
キャッシュ容量	約 100GB
ブロックサイズ	4KB
セットサイズ	256 ブロック

表 6.6.2 各ワークロードのパラメータ設定

ワークロード	平均ファイルサイズ	ファイル数	I/O サイズ		スレッド数	R/W 比率（回数）
			Read	Write		
webserver	32KB	20,000	1MB	16KB	100	10:1
fileserver	256KB	50,000	1MB	16KB	100	1:2
varmail	16KB	50,000	1MB	16KB	100	1:1
oltp	0.8GB	10	2KB	2KB	200 + 10	20:1

に，スピンダウンを行わない．6.3.2 節で述べたダーティブロックの追出しの閾値は，マイクロベンチマークにより求めた最良値である 1GB を設定した．

4. **提案手法**：上記の SSD キャッシュのみの方式とあわせて，HDD のアイドル時間にスピンダウンを行い省電力化を行う．

5. **Flashcache**：キャッシュ方式の違い以外の公平性を確保するため，提案手法と同一のキャッシュ容量と，ブロックの大きさを設定した．キャッシュセット内のブロックの置換ポリシーは FIFO と LRU が選択できる．本評価では本研究の提案方式と似た方式である LRU を設定した．

   Flashcache を比較対象とした理由として，オープンな実装で，かつ，データセンタ等において多くの利用実績があるためである．Flashcache はスピンダウンを行わない手法ではあるが，Flashcache 以外の方式に比べて本 SSD ディスクキャッシュシステムとの実現方式が近いことも比較対象の要因となっている．

また，提案手法はファイルシステムより下位の，ブロックデバイスレーヤでの実装を行っているために，ファイルシステムが提案手法に与える影響が大きい．そこで，ファイルシステムが提案手法に与える影響を確認する．本評価では，ext3，xfs，nilfs を提案手法上に構築し，ファイルシステムによるエネルギー効率の違いを明らかにする．

これらの環境上で，ファイルシステムやストレージの性能評価を行うベンチマークを実行する．本評価では，様々なサーバのワークロードを再現する Filebench[39] における次の四つのワークロードを使用する．

- fileserver：複数のユーザのホームディレクトリを格納するファイルサーバを想定している．複数のユーザからのファイル操作を模倣するため，各スレッドは ID に基づきファイル操作を行う．
- varmail：電子メールサーバのワークロードを模倣する．マルチスレッド化された，各スレッドがメールの読み込み，メールの作成，メールの削除を行う．
- webserver：ウェブサーバを模倣する．多数のスレッドがそれぞれファイルをシーケンシャルに読み込む．すべてのスレッドは単一のログファイルに 16KB のログを追記するため，このログファイルに対するデータ・メタデータの更新が集中する．
- oltp：オンライントランザクション処理のデータベースサーバを模倣する．数百 MB のファイル対して，200 のスレッドがランダムに同期読み込み，10 の書き込みスレッドが非同期なランダム書き出しを行い，単一スレッドが 256KB 単位のログを書き出す．

本評価で設定したパラメータを表 6.6.2 に示す．各ワークロードにおいて，各サーバで扱う平均ファイルサイズ，ファイル数，I/O サイズを設定した．また，Filebench は実際のサーバを模倣するためにマルチスレッド化されたベンチマークとなっており，スレッド数，Read・Write を行うスレッドの割合もパラメータとして指定した．これらの設定は，サーバ用計算機におけるファイルシステムの性能とエネルギー効率について評価した文献 [40] を参考にした．これらのパラメータを用いることで，様々なサーバの I/O 要求のパタンを再現できると考える．

また，10 分間のワークロードを 5 回行い，平均を求める．これらのベンチマークを実行することによって，単位時間当たりのオペレーション数 (ops/s) が得られる．この値を，ベンチマーク実行中の平均消費電力で割ることで，エネルギー辺りのオペレーション数 (ops/J) を算出する．

評価環境として表 6.6.3 で示す計算機環境を利用する．また，1TB・3.5 インチの SATA 接続の HDD と，100GB・2.5 インチの SATA 接続の SSD を用いる．表 6.6.4 に HDD の仕様を示す．SSD の平均消費電力 (W^{ssd}_{active}) は 0.62W である．これらの電力の測定には，株式会社シナジェテックの ST-30400 を利用する．本測定器はクランプ式の電流センサを使用しており，対象となる電源の導線をクランプ型センサにて挟むことで電流を計測する．計測した電流に電圧をかけ合わせることで電力を算出する．

6.6.2 評価に用いたスピンダウン閾値の決定

6.4.1 節で述べたプロファイリングによって得られた I/O パタンと HDD の電力パラメータから適切なスピンダウン閾値 T_{th} を求める．初めに，6.6.1 節で示した 3 種のファイルシステム・4 種のワークロードの合計 12 種の組み合わせにおいて，それぞれ事前実行を行い，I/O パタンの取得を行った．実評価と同様に，10 分間のワークロードを 5 回行った．これらの結果を図 6.6.1

表 6.6.3 評価環境の諸元

項目	内容
CPU	Xeon X5550 2.66GHz
主記憶	4GB
SATA インタフェース	3Gbps
オペレーティングシステム	Linux 2.6.35 (x86_64)
評価用 HDD	SATA 3.5 インチ 7200rpm 1TB
	Segate Barracuda
	ST31000528AS
評価用 SSD	SATA 2.5 インチ MLC 100GB
	OCZ Technology
	Vertex 2 EX

表 6.6.4 HDD の仕様

項目	内容
アイドル時平均消費電力 (W^{hdd}_{active})	5.15W
スピンダウン時平均消費電力 (W^{hdd}_{sleep})	1.14W
スピンアップに必要なエネルギー (E^{hdd}_{spinup})	74.7J
スピンアップに要する時間 (T_{spinup})	8 秒
BET	19.8 秒

に示す．横軸は，HDD へのアクセス要求間隔を示し，縦軸にそれらがどれだけ発生したかの数を示している．横軸は，BET 近辺のアクセス要求間隔を明確にするために指数表示としている．

次に，評価に用いた HDD の表 6.6.4 のパラメータと図 6.6.1 の結果から，6.4.1 節で示したエネルギーモデルを用いて，エネルギーを求めた．図 6.6.2 にこれらのエネルギーを示す．図 6.6.2 では縦軸にモデルより算出されるエネルギーを示し，横軸に決定すべき変数の Tth を示している．また，BET 近辺のエネルギーの変化を明確にするために横軸を指数表示としている．

その結果，fileserver の ext3，webserver の ext3，webserver の xfs，oltp の xfs 以外の組み合わせでは，$T_{th} = 1$ のときにエネルギーが最小となり，T_{th} が増加するに従いエネルギーも増加する．fileserver の ext3 では，$T_{th} = 17$ まではエネルギーが減少するが，それ以降は増加する傾向にある．したがって，これらの組み合わせでは，エネルギーが最小となる T_{th} をスピンダウン閾値として選択した．

webserver の ext3，webserver の xfs では，T_{th} が増加すると，一部急にエネルギーが低下する部分がある．この原因は，図 6.6.1 における BET 近辺のスピンダウンが多いことによる．このようにエネルギーの増減はあるものの，これら二つの場合も，エネルギーが最小となる点が $T_{th} = 1$ であったため，これをスピンダウン閾値として選択した．

一方で，oltp の xfs では，T_{th} を変化させても，エネルギーがあまり変化しないことがわか

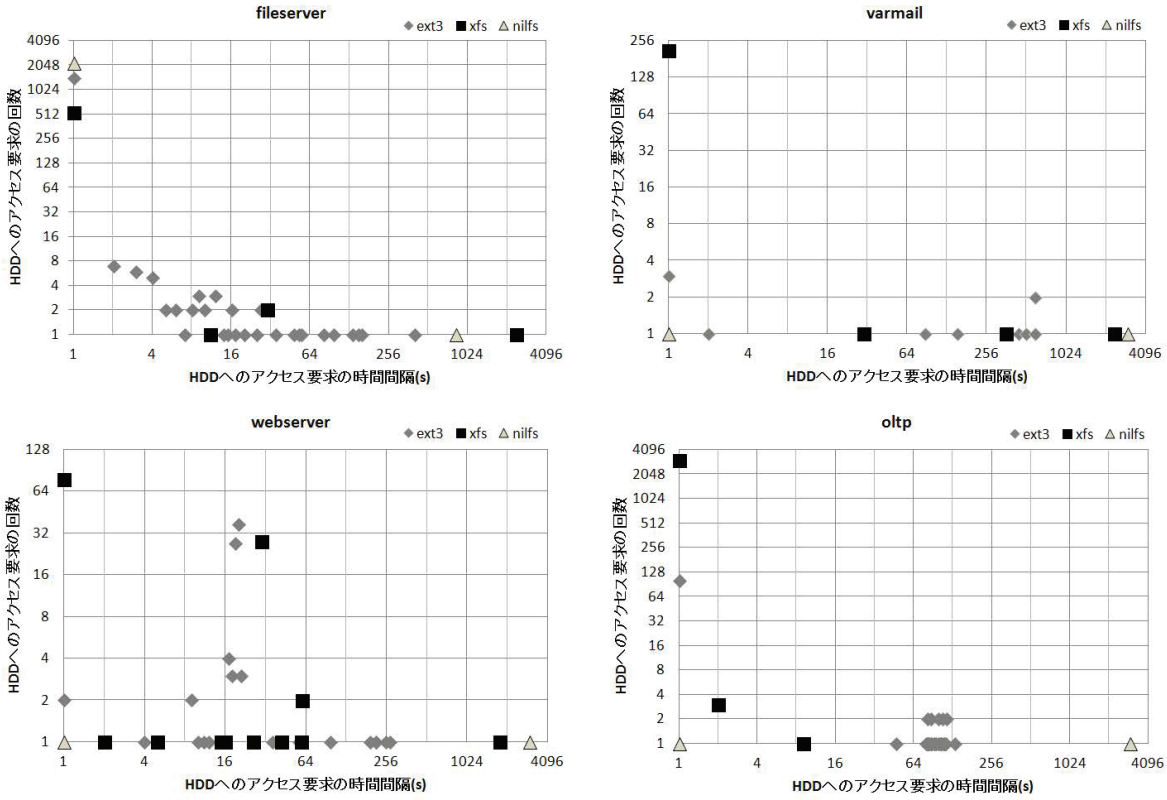

図 6.6.1 各ベンチマークごとの HDD の I/O 要求の時間間隔

る．図 6.6.1 における oltp の xfs では，常に HDD へのアクセスがありスピンダウンができないことを示しており，スピンダウン閾値を変化させてもエネルギー値に大きな変化は表れないが，机上計算の結果からは $T_{th} = 9$ でエネルギーが最小となる．したがって，この値をスピンダウン閾値として選択した．

6.6.3 評価結果と考察

評価では，初めに提案手法のエネルギー効率 (ops/J) を示す．その後，実際にどの程度 HDD をスピンダウンできたかを示す．あわせて，SSD キャッシュミス率について考察を行う．最後に，スピンアップにかかる I/O 遅延のオーバヘッドについて述べる．

エネルギーあたりの処理性能

提案手法のエネルギー効率の結果を示す．fileserver，webserver，varmail，oltp の消費したエネルギー 1J 当たりに処理できたファイル操作の数（ops/J）を示す．

ブロック格納手法の違いによるエネルギー効率

ブロック格納手法にてエネルギー効率を比較した結果を図 6.6.3 に示す．初めに，従来手法

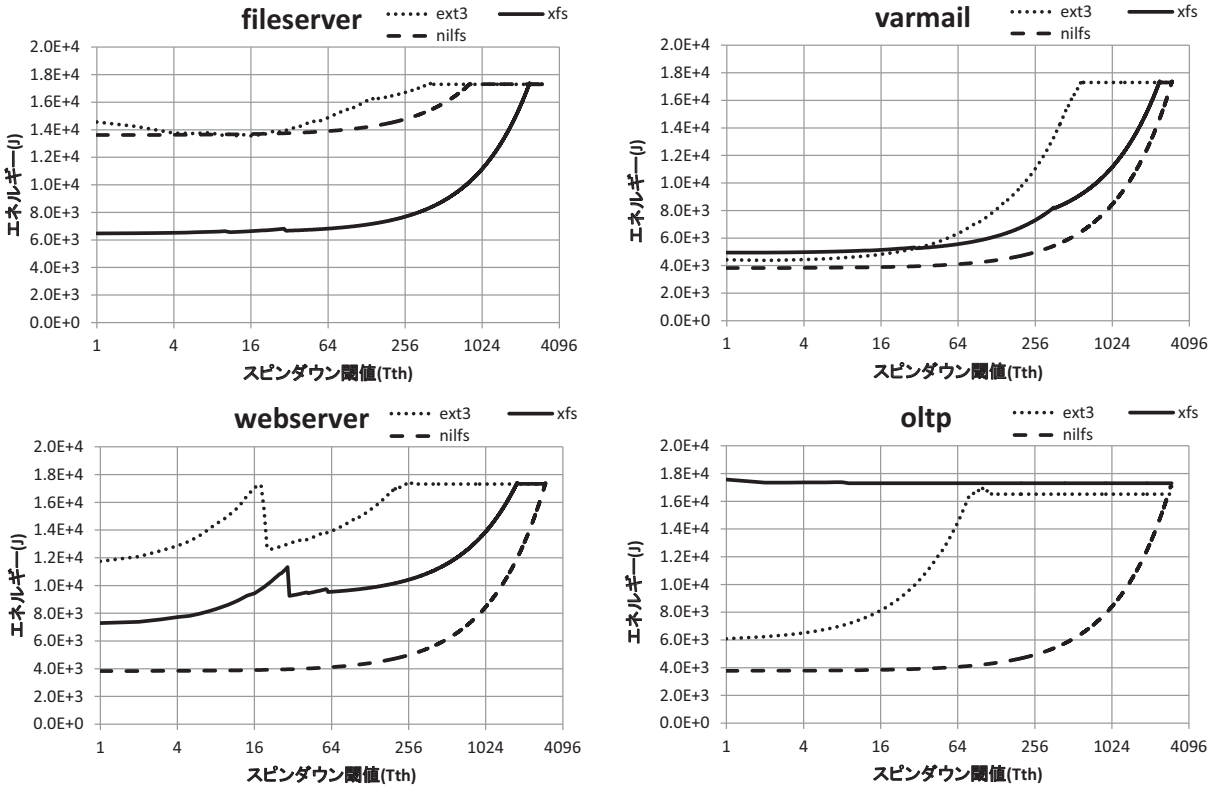

図 6.6.2 スピンダウン閾値ごとに算出したエネルギー

の SSD キャッシュのみと提案手法を比較した場合，varmail については，約 3.2 倍のエネルギー効率を実現し，oltp については 1.7 倍のエネルギー効率を達成できている．fileserver 上でのスピンダウンでは，平均消費電力は削減できたが，同等に処理速度が低下したために，エネルギー効率に変化が見られなかった．したがって，SSD キャッシュのみの手法と比較した場合，提案手法は，平均 1.88 倍のエネルギー効率を実現している．また，Flashcache と比較した場合もすべてのベンチマークで，エネルギー効率を改善することができた．varmail に関しては 5 倍以上のエネルギー効率の改善を達成した．提案手法と HDD での比較では fileserver で 11 倍，varmail で 59 倍，oltp で 17 倍のエネルギー改善を実現した．

ファイルシステムの違いによるエネルギー効率

ベンチマーク実行時の平均消費電力を図 6.6.4 に示し，ファイルシステムによるエネルギー効率の変化を図 6.6.5 に示す．HDD と提案手法を比較した場合，すべてのワークロード，すべてのファイルシステムにおいて，スピンダウンを行うことで，高いエネルギー効率を示した．

また，SSD キャッシュのみの手法と比較した場合，fileserver，varmail，webserver の組み合わせにおいて，エネルギー効率を改善できた．特に，fileserver の xfs で 2.5 倍，varmail の ext3 で 3.2 倍，varmail の xfs で 3.7 倍，varmail の nilfs で 3.8 倍，webservser の xfs で 3.2 倍，

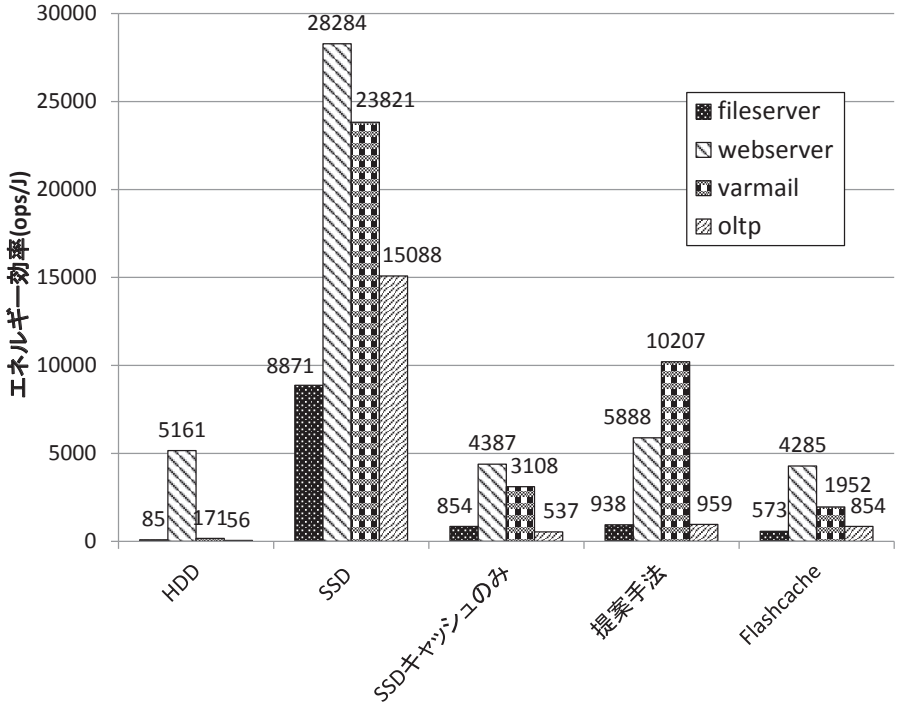

図 6.6.3 各ベンチマーク実行時のエネルギー効率

webserver の nilfs で 3.4 倍，oltp の nilfs で 3.9 倍の性能向上を達成した．

三つのファイルシステムと四つのワークロードの 12 種の組合せにおいて，平均で 2.5 倍のエネルギー効率向上を実現できた．よって，SSD ディスクキャッシュと HDD のスピンダウンの併用は，ストレージデバイス全体の省電力化に一定の効果があると言える．

スピンダウン頻度とエネルギー効率

電力の削減効果の詳細を明らかとするために，ワークロード実行中のスピンダウン時間・スピンダウン頻度の内訳の詳細を示す．まず，図 6.6.6 に全実行時間 3000 秒に占めるスピンダウ

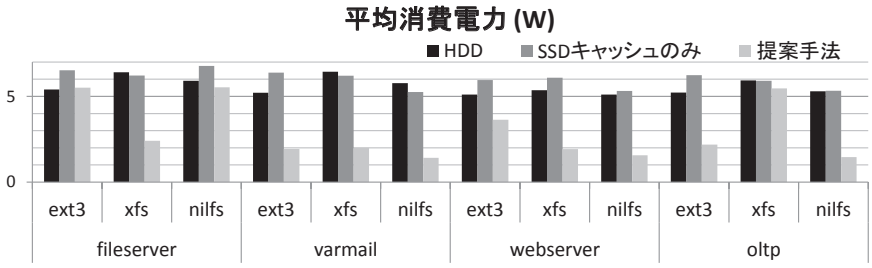

図 6.6.4 各ファイルシステムにおける各ベンチマーク実行時の平均消費電力

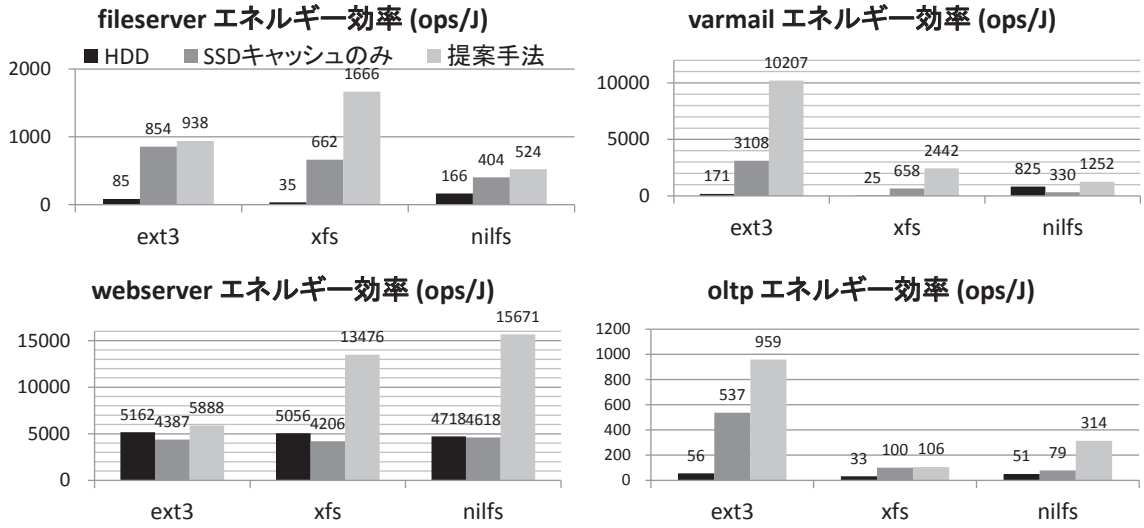

図 6.6.5 各ファイルシステムにおける各ベンチマーク実行時のエネルギー効率

ンを行っていた時間の割合を，図 6.6.7 にスピンダウンを行った回数を示す．本実験に用いた HDD のエネルギーの BET を基準として，エネルギー的に得をしたスピンダウン・損をしたスピンダウンに分けて示している．図 6.6.6 より，fileserver の xfs，varmail のすべてのファイルシステム，webserver のすべてのファイルシステム，oltp の ext3，nilfs において全実行時間の 8 割以上がスピンダウンしていることが確認できる．これらは，SSD ディスクキャッシュによって，HDD へ発行される I/O 要求が削減され，HDD のアイドル時間を生成できたためである．

一方で，oltp の xfs では，総スピンダウン時間が 2 秒で大半の間スピンダウンができていないことがわかった．xfs の oltp は図 6.6.9 で示すキャッシュミス率が 1.2%と小さいが，図 6.6.1 に示すように断続的な HDD アクセスが続く．oltp では扱うファイルのサイズが比較的大きいのと合わせ，xfs では環状バッファを用いてジャーナルを保存しているために，ジャーナルの更新に要する，断続的なにリードミスが発生する．このため，スピンダウンできない結果となった．

図 6.6.7 より，nilfs ファイルシステムの varmail，webserver，oltp では BET 以上のスピンダウンのみが発生し，スピンダウン回数も 10 回以下と少ないために，平均スピンダウン時間も長い．そのため，スピンダウンを行わない SSD キャッシュの環境で nilfs を選択した場合，スピンダウンを行うことで，エネルギー効率を大きく改善できる．また，キャッシュミス率も 0.1%以下と低い．nilfs はログ構造化ファイルシステムであり追記書き込みにてデータの更新を行うために，書き出しが SSD でキャッシュされるために，スピンアップが発生しない．

ext3 ファイルシステムでは，定期的なリードミスが頻発するため，他のファイルシステムに比べてスピンダウン回数が多くなる傾向がある．一方で，xfs ファイルシステムでは，webserver を除くワークロードで nilfs 同様，総スピンダウン数が 10 回以下と少ない回数であることがわかる．xfs の webserver においては，スピンダウン回数が 40 回と多く，スピンアップにかかる I/O 遅延によって，エネルギー効率が低下すると考えられるが，ワークロードの結

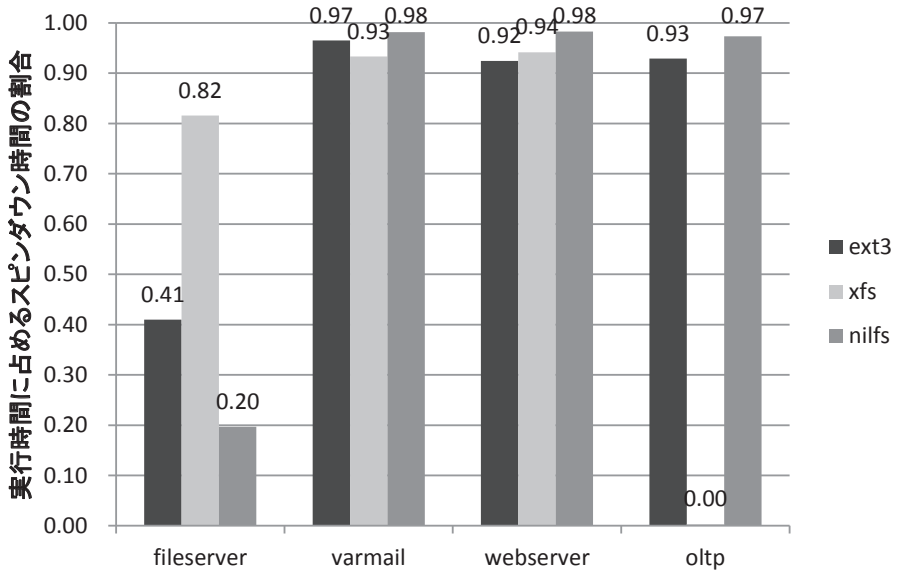

図 6.6.6 全実行時間に占めるスピンダウン時間の割合

果では，スピンダウンを行わない SSD キャッシュのみの場合と比較して 3 倍以上エネルギー効率が向上している．これは，xfs の webserver でのスピンアップは，データ更新とは非同期に行われる，メタデータ更新のジャーナルの記録のために発生し，処理速度が落ちないためである．そのため，ワークロードの結果に影響が出ない．

いずれの組み合わせにおいても，図 6.6.5，図 6.6.6 で示すとおり本提案方式によりスピンダウンが発生し，エネルギー効率が向上した．ワークロードとファイルシステムの組み合わせによってその効果に違いがあるものの，実行時間の 8 割以上をスピンダウンさせることができた組み合わせにおいては，スピンダウンを行わない SSD キャッシュのエネルギー効率に比べて平均 2.8 倍の改善が可能であった．

ワークロードの I/O 要求数・キャッシュヒット率とのエネルギー効率

各ワークロードのブロック I/O の read/write の要求数の割合を図 6.6.8 に示す．これらは，ファイルシステムから，SSD ディスクキャッシュブロックへ発行された I/O 要求数を示す．また，これらの I/O 要求が SSD キャッシュにてどの程度キャッシュされるかを示すために，SSD ディスクキャッシュドライバでのキャッシュミス率を図 6.6.9 に示す．

fileserver はファイルシステムから SSD ディスクキャッシュに発行される I/O 要求数が四つのワークロードで最も多い．また，ext3 と nilfs では，キャッシュミス率が 5%以上と高く，SSD キャッシュへ割り当てるキャッシュブロックが不足し，キャッシュブロックがあふれる状態が発生した．ext3 と nilfs では，図 6.6.9 に示すとおり write ミスが多く発生していることから，HDD へのダーティブロックの追出しと新規に割り当てるキャッシュブロックの読込みに伴う HDD への I/O 要求が増え，HDD のアイドルサイクルを作ることができず，他の組み合わせに

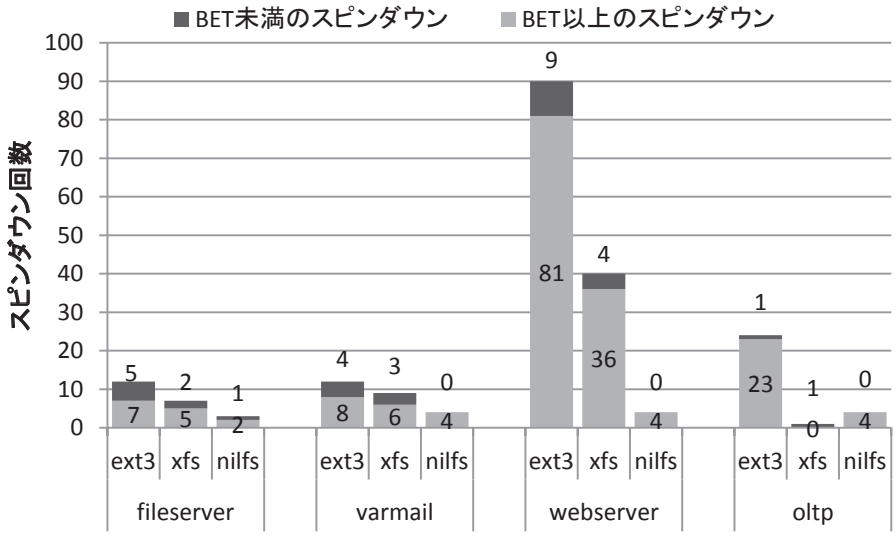

図 6.6.7 **スピンダウン回数**

比べてエネルギー効率をあまり改善できなかった．

varmail では webserver 同様にファイルシステムからの I/O 要求数が多い．しかし，HDD へ発行される write ミス率は 0%のため，write は SSD キャッシュで吸収されている．したがって，HDD のアイドル時間が増大し，エネルギー効率の改善率が高い．

webserver では，ファイルシステムからの I/O 要求数は四つのワークロードの中で最も少ない．これは，ファイルシステム内のファイルキャッシュの効果が大きいためである．また，ファイルシステムへ発行される write 要求もすべて，SSD でキャッシュしている．そのため，xfs・nilfs では HDD のアイドル時間が BET 以上の長さとなり電力効率が改善した．一方で，ext3 においては，BET 近辺のアイドル間隔の定期的な read ミスが発生し，スピンアップが発生するためにエネルギー効率が，xfs・nilfs と比較して小さい．

oltp ではファイルキャッシュから発行される I/O 要求数は write，read が同程度となった．ext3・nilfs のキャッシュミス率も 0.1%以下のために，エネルギーの改善率も高い．しかし，oltp では，ジャーナルデータの新規割り当てのための，少容量のリードミスが断続的に発生し，スピンダウンを行うことができずエネルギー効率を改善させることができなかった．このように，エネルギー効率が改善された xfs の webserver のキャッシュミス率と同程度で有っても，図 6.6.1 に示したように，小容量の断続的な read ミスが起こる場合があり，スピンダウンできない．

スピンアップオーバヘッドによる I/O 遅延

キャッシュミス時に生じるスピンアップには，大きな時間を要するため I/O 遅延が生じる．本実験に用いた HDD はスピンアップに 8 秒の時間が必要である．そのため，スピンアップを行うことでの処理性能低下の確認を行った．

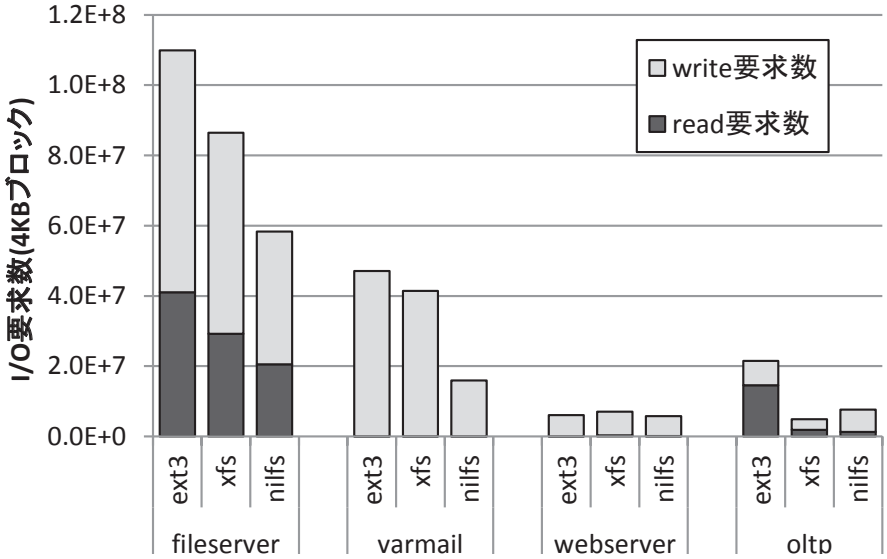

図 6.6.8 ファイルシステムから発行される I/O 要求数

スピンダウンを行わない SSD ディスクキャッシュを適用した場合に対する本提案手法の処理性能の比率を算出したところ，ext3 の fileserver，webserver，oltp においてそれぞれ 0.93，0.81，0.62 となり，処理性能の低下がみられた．その他の組み合わせでは 0.98〜1.17 の比率が得られ，ほぼ同等の処理性能で動作するという結果を得た．これらの処理性能低下の原因は，定期的に発生するジャーナルのリードミスである．ext3 では，ジャーナルの更新をデータの更新と同期して行うため，ファイルシステムがロックされる．一方，xfs のジャーナルの更新はデータの更新と非同期に行われるため処理性能の低下はなかった．nilfs では，ログ構造化ファイルシステムを採用し，追記書込みを行うため，リードミスは発生しない．さらに，追記書込みされたデータは，SSD キャッシュによって集約されているために，HDD へのアイドル時間も増加する．

よって，ジャーナルデータの新規割り当ての際に read ミスを起こし，ジャーナルの更新とデータの更新が同期して行われるようなファイルシステムの場合，処理性能の低下を起こす場合がある．ext3 では処理性能が低下するが，xfs，nilfs では起きない．

SSD キャッシュミス時のブロックあふれの評価

fileserver の ext3，nilfs では SSD キャッシュ容量不足によるブロックのあふれが発生したが，本提案方式によりエネルギー効率が向上した．さらに，SSD キャッシュの容量を 50GB，10GB，5GB と小さくした環境において SSD キャッシュブロックがあふれる状況を増やし，本提案方式の適用効果があるかを検証した．100GB と同様に 12 種類の環境をそれぞれの容量で評価した結果を表 6.6.5 に示す．表項目として，実測に用いたスピンダウン閾値（Tth），実測値から得られたエネルギー効率 (提案方式)，SSD をスピンダウンなしで適用した場合のエネルギー効率に対する提案方式の比率 (HDD+SSD との比率)，HDD 単体のエネルギー効率に対す

表 6.6.5 小容量の SSD ディスクキャッシュにおけるエネルギー効率

ブロックあふれ	SSD 容量 (GB)	WL[1]	FS[2]	Tth (sec)	提案方式 (ops/J)	HDD+SSD との比率	HDD単体との比率
(A) 無し	50	F	xfs	1	1459	2.09	41.7
		V	ext3	1	8655	2.6	50.5
			xfs	1	2192	2.71	86.61
		W	ext3	21	6466	1.43	1.25
			xfs	1	13596	3	2.69
			nilfs	1	15723	3.09	3.33
		O	ext3	1	1642	2.87	29.16
			xfs	11	91	1.02	2.73
			nilfs	1	319	2.77	6.24
	10	F	xfs	1	2189	2.61	62.55
		V	ext3	1	13771	3.23	80.34
			xfs	1	3268	3.52	129.12
		O	ext3	1	3521	4.7	62.52
			xfs	1	100	1.03	3.02
	5	V	xfs	1	2966	3.23	117.19
(B) 有り（エネルギー効率向上あり）	50	F	ext3	23	368	1.08	4.35
			nilfs	1	300	1.03	1.81
		V	nilfs	1	838	2	1.02
	10	W	xfs	5	8687	1.84	1.72
			nilfs	4	5995	1.33	1.27
		O	nilfs	1	103	1.26	2.01
	5	F	xfs	3	285	1.08	8.14
		V	ext3	1	13614	4.24	79.43
		W	nilfs	6	4908	1.18	1.04
		O	ext3	1	105	1.11	1.86
			xfs	1	61	1.03	1.85
			nilfs	1	60	1.02	1.17
(C) 有り（エネルギー効率向上なし）	10	F	ext3	2	96	0.94	1.14
			nilfs	1	178	0.97	1.07
		V	nilfs	1	708	1.07	0.86
		W	ext3	20	4676	0.98	0.91
	5	F	ext3	1	79	0.99	0.93
			nilfs	1	165	1.01	0.99
		V	nilfs	1	712	1.06	0.86
		W	ext3	20	4523	1.04	0.88
			xfs	31	4980	1.13	0.98

1 ワークロード(F:fileserver, V:vermail, W:webserver, O:oltp)
2 ファイルシステム

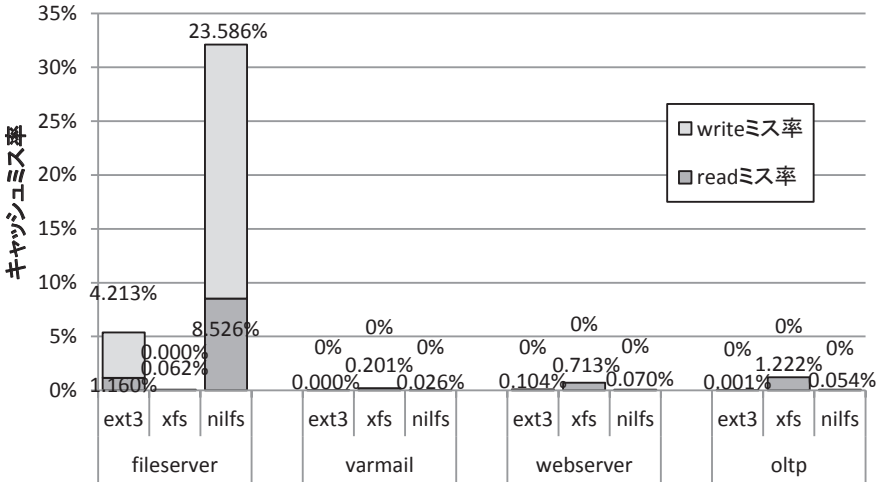

図 6.6.9 SSD **ディスクキャッシュのキャッシュミス率**

表 6.6.6 SSD **キャッシュブロックがあふれてエネルギー効率が向上しないケースの評価**

予測値がHDD単体を	SSD容量(GB)	WL[1]	FS[2]	予測値(ops/J)	提案方式(ops/J)	HDD+SSD(ops/J)	HDD単体(ops/J)	miss率(%)
(1)下回る	10	V	nilfs	803	708	661	825	91.1%
		W	ext3	4452	4676	4794	5161	30.4%
	5	V	nilfs	796	712	672	825	94.5%
		W	ext3	4556	4523	4340	5161	35.9%
			xfs	4471	4980	4400	5059	29.7%
(2)上回る	10	F	ext3	104	96	102	85	72.6%
			nilfs	206	178	183	166	83.4%
	5	F	ext3	88	79	80	85	82.3%
			nilfs	189	165	163	166	89.1%

1 ワークロード(F:fileserver, V:vermail, W:webserver)
2 ファイルシステム

る提案方式の比率 (HDD 単体との比率) を示し，評価結果を次の三つに分類した．

(A) SSD キャッシュからブロックがあふれない（15 種類）

(B) SSD からキャッシュブロックがあふれるが，本提案方式によりエネルギー効率が向上する（12 種類）

(C) SSD からキャッシュブロックがあふれ，本提案方式を適用してもエネルギー効率が向上しない（9 種類）

表 6.6.5 において HDD+SSD との比率と HDD 単体との比率に着目すると，SSD キャッシュからブロックがあふれない場合 (A) は，すべて本提案方式を適用することでエネルギー効率が向上している．また，表 6.6.5 には掲載していないがエネルギーの予測値も HDD+SSD，HDD 単体より上回っていた．この結果は 100GB でブロックがあふれない 10 種類についても同じ傾向であった．

SSD キャッシュからブロックがあふれる場合のうち分類 (B) では，ブロックはあふれるが本提案方式を適用することによりエネルギー効率が向上している．これは 100GB でブロックがあふれていた fileserver の ext3，nilfs の 2 種類にもあてはまる．また，分類 (C) では本提案方式を適用してもエネルギー効率が向上しない．そこで，分類 (C) については，以下，別表を用いて考察する．

事前実行の段階で得られるデータである本提案方式のエネルギー効率の予測値 (予測値) と SSD キャッシュミス率 (miss 率) の値に着目して，実測値との関係を検証する．表 6.6.6 に，予測値，三つの実測値（提案方式のエネルギー効率 (提案方式)，SSD キャッシュをスピンダウンなし時のエネルギー効率 (HDD+SSD)，HDD 単体時のエネルギー効率 (HDD 単体)），SSD キャッシュミス率の項目を示し，

(1) 事前実行で得られるエネルギー効率の予測値の段階で，HDD 単体より下回るケース

(2) 事前実行で得られるエネルギー効率の予測値は効率向上を示すが，実際には HDD+SSD や HDD 単体を上回れなかったケース

に分類した結果を示す.

エネルギー効率の予測値が HDD 単体を明らかに下回る場合 (1) は，本提案方式を適用してもエネルギー効率を向上できないという傾向にある．また，事前実行で得られるエネルギー効率の予測値は効率向上を示しても，SSD キャッシュミス率が高い場合 (2) には，本提案方式を適用しても HDD をスピンダウンできる機会が元々少ないため，予測どおりのエネルギー効率の向上はできない傾向にある．なお，分類 (1) に属する 10GB の webserver ext3，5GB の webserber ext3，xfs において SSD キャッシュミス率は 70%以下の値を示しているが，表 6.6.5 の分類 (B) に属する項目においても 70%以下の値を示していた．しかし，分類 (B) ではエネルギー効率の予測値が常に高い傾向を示している点が分類 (1) とは異なっている．

以上により，ブロックがあふれるケースであっても本提案方式を適用してエネルギー効率を向上させることは可能であり，ワークロードの事前実行時のデータを参考にすれば，本提案方式適用時にエネルギー効率を向上できるかどうかを事前に予測できることを示した．傾向としては，想定するワークロードでスピンダウンを行わない SSD キャッシュを適用した事前実行において，SSD キャッシュミス率が低い状況においては本提案方式を適用することが有効であると考える．

6.7 まとめ

本研究では，SSD を HDD のディスクキャッシュとして利用するシステムにおいて，HDD のスピンダウンを行うことで省電力化を行う手法を提案した．また，電力モデルとプロファイリングにより，電力的に損となるスピンダウンを抑制し，省電力化を行った．HDD のスピンダウンを行うことで，処理性能を大きく落とすことなく，HDD 単体での利用よりも平均約 21 倍のエネルギー効率向上を実現した．また，SSD キャッシュのみを適用する手法と比較した場合に約 2.5 倍以上のエネルギー効率を実現した．特に nilfs ファイルシステムの場合には，処理性能を落とすことなく 3 倍以上のエネルギー効果を達成し，本提案手法の有用性を示すことができた．

一方で，同じアプリケーションプログラム実行中においても，HDD へ発行される I/O パタンが数分間隔のような短い期間で変化するような場合には，本手法適用により電力的な無駄が生じる可能性がある．そのため，今後はシステム挙動に応じたスピンダウン閾値 T_{th} の動的設定方法が課題となる．またその他の課題として，複数のワークロードが一つの計算環境で混在実行する場合の閾値決定方法の検討，複数の HDD を搭載するシステムのディスクキャッシュとして SSD を導入する方式の検討などがあげられる．

第7章　結言

本章では，本研究の成果，結論，知見を述べ，今後の課題についてまとめる．

7.1　研究成果の結論

本研究では，省電力制御を隠蔽するシステムソフトウェアによる省電力化を目指し，アクセラレータとストレージにおいて省電力化を実現した．これらの成果を下記にまとめる．また，これらの成果の関係について図 7.1.1 に示す.

- 実行制御オーバヘッドをハードウェアにて高速化
  メニーコアやマルチコアアクセラレータのコア数の増加により，コアの制御やコア間制御，データ転送制御がオーバヘッドとなっている．そこで，同期制御，実行制御，データ転送制御，FIFO 制御をハードウェア実行支援機構の RCH にて高速化することで，演算時間を短縮した．演算時間の短縮により，エネルギー効率を大きく改善した．具体的には，エネルギー効率を 5 割程度改善することができた.

- 煩雑なパイプライン制御の隠蔽
  パイプライン並列を行うためには，それぞれのコアの実行制御や同期制御，FIFO 制御，データ転送制御を行う必要がある．これらの制御はデータ並列等の並列化と比較し煩雑である課題があった．さらに，近年広く普及している並列プログラミング環境の OpenCL においても，パイプライン並列をサポートしていない課題があった．そこで，OpenCL のタスク並列モデルにパイプライン並列を隠蔽した．具体的には OpenCL のタスク並列モデルを用いてプログラミングを行うことで，OpenCL ライブラリが自動的にパイプライン情報を抽出し，合わせてパイプライン制御まで行う．これにより，アプリケーションプログラマはパイプライン制御の詳細を知らずとも，OpenCL の言語仕様を理解するだけで，簡単にアクセラレータを利用することが可能となった.

- HDD のオーバヘッドを考慮した電源制御
  HDD 単体の省電力化機能では，性能が悪化する恐れがあった．そのため，HDD の電力と性能を考慮した電源制御を目指した．具体的には，HDD の電力モデルとプロファイリングを用いて，HDD のエネルギー効率を予測し，エネルギー的に損となるスピンダウンの発生を抑制しエネルギー効率の改善を実現した．さらに，SSD を HDD のディスクキャッシュとして用いる環境にて，HDD のスピンダウンを行うことにより，HDD 単体と比較しエネルギー効率を 21 倍に改善した.

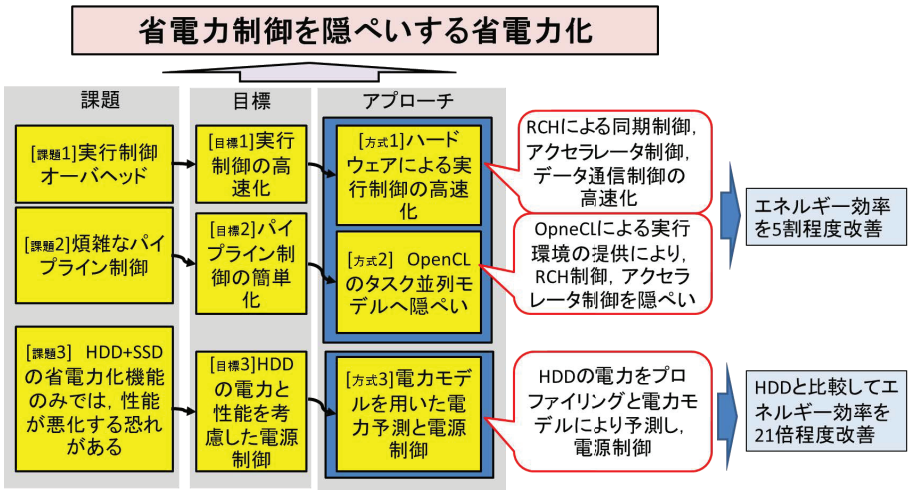

図 7.1.1 研究成果の概要

このように，課題に対して，システムソフトウェアによる省電力化制御の隠ぺいと省電力化の実現を達成した．

7.2 本研究の成果

本研究では，ハードウェアによる実行制御の高速化と，システムソフトウェアによる省電力制御の隠ぺいを達成した．その中でも，ハードウェアを用いた高速化による省電力化を実現し，汎用並列プログラミング言語環境を実現した，さらに，電力モデルを用いた電源制御を実現することで，計算機の省電力化を実現した．次に具体的な本研究の結果を示す．

7.2.1 RCH と OpenCL によるアクセラレータの省電力化

近年，プロセッサの省電力化を目指し，1 つのプロセッサ内に複数のコアやアクセラレータを搭載した，メニーコアプロセッサやマルチコアアクセラレータが普及している．多数のコアにより，並列して演算を行うことで，高い電力効率を達成することを目指している．さらに，コア間でデータを受け渡すパイプライン並列を行うことで，メモリボトルネックを解消している．

一方で，ソフトウェアによる同期や実行制御オーバヘッドが大きな課題となっている．プロセッサ内部のそれぞれのコアやアクセラレータに対する実行制御やコア間での同期制御，データ転送制御，FIFO 制御などがあげられる．これらのオーバヘッドにより，実行性能が低下し，エネルギー効率が悪化している．また，パイプラインの実行制御は非常に煩雑であり，アプリケーションプログラマに対して汎用性が低い課題がある．

そこで，実行制御オーバヘッドを改善し，高速化によってエネルギー効率を改善することを目指した．これらのオーバヘッドの改善により，プロセッサ全体の演算効率を大きく改善する

ことが可能である．あわせて，煩雑なパイプライン制御を隠蔽することを目指し，汎用性の高い実行環境を目指した．このようにすることで，アプリケーションプログラマがアクセラレータを簡単に利用できるようにする．

実行制御オーバヘッドの改善のためにハードウェアによる高速化を行った．具体的には，コアの実行制御，コア間同期制御，DMAC に対するデータ転送制御，FIFO 制御を行う，専用ハードウェアの RCH にて高速化することにより，実行制御オーバヘッドを改善し高速化を実現した．演算の高速化により，エネルギー効率を改善した．さらに，煩雑なパイプライン制御の隠蔽のために OpenCL による開発環境を実現した．OpenCL のタスクモデルに対し，パイプライン並列を隠蔽した．これにより，アプリケーションプログラマに対して汎用性の高いプログラミング環境を提供した．

評価では，RCH による高速化によって，制御を含む総演算時間を 5 割程度改善した．これにより，エネルギー効率も 5 割程度改善することを確認した．新たなハードウェアを追加したことにより，回路面積が増加し，消費電力は若干増加したが，性能が大きく向上したため，エネルギー効率を大きく改善することができた．

7.2.2 HDD の省電力化機能のオーバヘッドを考慮した HDD と SSD を用いた省電力階層キャッシュストレージ

HDD の省電力化機能を利用し，計算機の電力を削減する試みが多く行われている．一方で，単純な省電力化機能の制御では，エネルギー効率が悪化する恐れがある．HDD の省電力化では，ディスクのスピンダウンを行うことにより電力を削減可能である．一方でスピンアップには大きなエネルギーが必要であり，頻繁にスピンダウンとスピンアップを繰り返した場合，エネルギー的に損となる．

これらの HDD の省電力化へのオーバヘッドは，多くの論文や研究にて問題視されているが，具体的に，これらの問題に取り組んだ研究はなかった．そのため，これらオーバヘッドを考慮した省電力化方式を提案した．オーバヘッドとなる省電力化制御を極力抑えることにより，計算機システムの省エネルギー化を目指した．

そこで，電力の予測を行い，予測に基づき省電力化を行った．このようにして，オーバヘッドとなる省電力化制御を回避する．具体的には，プロファイリングと電力モデルにより，エネルギーを予測する．プロファイリングにて HDD の I/O 要求を記録し，電力モデルによりエネルギー効率を予測する．予測に基づきスピンダウンによる省電力化制御を行う．これにより，エネルギー的に損となるスピンダウンを極力抑える．

これらの手法を，SSD と HDD を用いたストレージシステムに適応した．SSD により，HDD への I/O を抑制することで，スピンダウンを行う時間を確保する．さらに，提案手法による電力予測によってスピンダウンを行うタイミングを決定した．これらによって，ストレージのエネルギー効率を約 21 倍改善した．

7.3 本研究より得られた知見

本研究では実行制御の高速化と簡単化，HDD の電力特性を考慮した省電力ストレージについて示した．以下に，これらの研究から得られた知見を示す．

- 汎用のパイプライン環境における実行制御の高速化と簡単化
  汎用のパイプラインプロセッサに対する，実行環境について明らかとした．パイプラインプロセッサではソフトウェアによる，各コアや FIFO, DMAC などに対する実行制御がオーバヘッドとなる．本研究では，汎用のパイプライン向けのハードウェア タスクディスパッチャを提案することで，パイプライン制御をハードウェアにて高速化した．また，煩雑なパイプライン並列を汎用の並列プログラミング環境に抽象化することにより，制御の簡単化を実現した．具体的には，FIFO と get/put を用いたパイプライン制御を，OpenCL のタスク並列モデルへ抽象化することにより，パイプラン制御の簡単化方法を明らかとした．

- 電力モデルを用いた HDD の電源制御方式
  省電力化機能のオーバヘッドを考慮した，電力モデルによる省電力制御方法を明らかとした．HDD の省電力化機能であるスピンダウンを行うことにより電力を削減できるが，HDD へ再度アクセスを行う際には，ディスクをスピンアップさせる必要があり，大きなエネルギーが必要となる．そのため，頻繁なスピンアップとスピンダウンを繰り返した場合に，エネルギー的に損となる．そこで，これらを回避するために，プロファイリングと電力モデルを用いた，省電力制御を提案した．プロファイリングと電力モデルによりオペレーションあたりの，エネルギー効率を予測することにより，スピンダウンまでのタイミングを遅らせるようにした．これにより，電力的に損となるスピンダウンを回避し，エネルギー効率を改善する方法を明らかとした．

7.4 今後の課題

今後の課題として以下のような課題を上げることができる．

- FPGA 向け OpenCL 環境への適用
  本研究で示した汎用のパイプライン環境における制御の高速化と簡単化の手法は，FPGA 環境への適用が期待できる．FPGA は本来パイプライン演算を得意とする特徴を持つ．専用の演算を行う回路を FIFO を利用して接続することにより，高いスループットにて演算を行うことができる特徴をもつ．また，FPGA 向けの OpenCL 環境は提案されているがこれらは，データ並列を主に対象としており，本来 FPGA が得意とするパイプライン処理は行うことができない欠点がある．そこで，これらの FPGA 向け OpenCL 環境に対し，パイプライン環境の制御の高速化と簡単化を適用することにより，FPGA が本来得意とするパイプライン処理を，OpenCL から効率よく利用できるようになる．

- 多数の HDD を使う環境への適応
  本研究での SSD を HDD のディスクキャッシュとして用いる省電力化環境では，1 台の HDD に対して，1 台の SSD を利用していた．そこで，多数の HDD を用いる環境への適用が考えられる．データセンタ等では多数の HDD が利用されており，HDD によって多くの電力が消費されている．そのため，本手法を多数の HDD に対して利用できるようにすることにより，さらに多くの電力を削減できると予想できる．

謝辞

本研究を行うにあたり，多くの方々からご助言とご指導をいただきました．ここに深く感謝申し上げます．

修士生から博士後期課程までの 5 年間の長きにわたり，ご指導いただいた並木美太郎教授に感謝申し上げます．先生には，研究に関する様々なご指導を頂いたことはもちろん，生活面での相談にも乗っていただきました．特に，休日平日，昼夜問わずアドバイスを頂いたことや，研究において様々な機会を与えて頂いたことは，筆者が研究者の卵として成長するために不可欠なものでした．いまさら言うまでもありませんが，先生のご指導無しではここまでたどり着くこと，そして研究者としてのスタートラインに立つことは到底不可能だったと思います．ここに深く感謝申し上げます．

今回の博士論文の審査をお忙しい中，引き受けてくださった山井成良教授，藤田欣也教授，近藤敏之教授，山田浩史准教授に感謝申し上げます．様々な分野からの先生方の数々のご指摘は，幅広い視点で研究を見据える良いきっかけとなるとともに，論文の質を高めることに繋がりました．心より感謝申し上げます．

佐藤未来子特任助教には，ゼミでのアドバイスを頂いたほか，論文のチェックなどをしていただき，そのアドバイスは筆者の研究において大きな参考となりました．また，博士課程への進学や就職先についてのお話は筆者の進路選択の参考になりました．ここに心から感謝の意を表します．

慶應義塾大学の天野英晴教授，芝浦工業大学の宇佐美公良教授，東京大学の中村 宏教授，近藤正章准教授，および，各研究室の学生の方々など，CREST プロジェクトでご一緒した方々に感謝申し上げます．様々な分野の先生方や学生が集まり，議論を行い，一つのものを作り上げることは，筆者にとって大きな刺激となりました．

最後に，27 年間筆者を支え続けてきてくれた両親に感謝します．本論文が完成したのも，この進路を選ぶことができたのも，思い返してみれば多くの方々のご支援はもちろん，両親の支えによるところが大きいと思います．特に，博士後期課程へ進むことを相談した時をはじめとして，筆者を理解し，応援をしてくれたことは常にとても心強い思いでした．心から感謝の意

を表すと共に，筆者の博士の学位取得というこの結果が，少しでも両親への恩返しになればとても嬉しく思います.

参考文献

[1] Andreas Olofsson, Roman Trogan, and Oleg Raikhman,“ A 1024-core 70GFLOPS/W Floating Point Manycore Microprocessor, ”http://www.adapteva.com/wp-content/uploads/2013/02/sc11_publish.pdf

[2] Adapteva, Introduction, http://www.adapteva.com/introduction/

[3] Intel, “ Introducing Intel Many Integrated Core Architecture, ”http://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html

[4] Intel,“ インテル Xeon Phi 製品ファミリー, ”http://www.intel.co.jp/content/www/jp/ja/processors/xeon/xeon-phi-detail.html?wapkw=xeon+phi

[5] Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan and Pat Hanrahan,“ larrabee: A Many-Core x86 Architecture for visual Computing, ”ACM Trans. Graphics, vol. 27, no. 3, 2008, pp. 18:1-18:15.

[6] KALRAY, “ Our MPPA MANYCORE Products, ”http://www.kalray.eu/products/mppa-manycore/

[7] CAVIUM, “ OCTERON Fusion Processor Family, ”http://www.cavium.com/OCTEON-Fusion.html

[8] Seungjin Lee, Jinwook Oh, Junyoung Park, Joonsoo Kwon, Minsu Kim, and Hoi-Jun Yoo, “ A 345 mW Heterogeneous Many-Core Processor With an Intelligent Inference Engine for Robust Object Recognition, ”IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 46, NO. 1, JANUARY 2011

[9] Hoeseok Yang and Soonhoi Ha, “ ILP based data parallel multi-task mapping/scheduling technique for MPSoC, ” SoC Design Conference, 2008. ISOCC ’08. International, vol. 01, pp.I-134 - I-137, nov. 2008.

[10] Michael Bedford Taylor, Jason Kim, Jason Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffman, Paul Johnson, Jae-Wook Lee, Walter Lee, Albert Ma Arvind

Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe and Anant Agarwal , “ The Raw microprocessor: a computational fabric for software circuits and general-purpose programs, ”Micro, IEEE (Volume:22 , Issue: 2)

[11] Tilera, “ Tilepro64 multicore processor product brief, ”http://www.tilera.com/

[12] Freescale, “ C-5 network processors, ” http://www.freescale.com/

[13] Y. Koizumi, E. Sasaki, H. Amano, H. Matsutani, Y. Take, T. Kuroda, R. Sakamoto, M. Namiki, K. Usami, M. Kondo and H. Nakamura, “ CMA-Cube: A scalable reconfigurable accelerator with 3-D wireless inductive coupling interconnect, ” International Conference on Field Programmable Logic and Applications, 2012

[14] Thies William, Karczmarek Michal, Amarasinghe, and Saman P. “ StreamIt: A Language for Streaming Applications, ”Proceedings of the 11th International Conference on Compiler Construction, 2002

[15] Ran Ginosar, “ The Plural Architecture: Shared Memory Many-core with Hardware Scheduling, ”IEEE 8th International Symposium on Embedded Multicore/Many-core SoCs 2013.

[16] Itai Avron and Ran Ginosar, “ Performance of a Hardware Scheduler for Many-core Architecture, ” IEEE 14th International Conference on High Performance Computing and Communication IEEE 9th International Conference on Embedded Software and Systems, 2012.

[17] Sanjeev Kumar, Christopher J. Hughes ,and Anthony Nguyen, “ Carbon: architectural support for fine-grained parallelism on chip multiprocessors, ”Proceeding ISCA ’07 Proceedings of the 34th annual international symposium on Computer architecture, Pages 162-173

[18] Daniel Sanchez, Richard M. Yoo, and Christos Kozyrakis, “ Flexible architectural support for fine-grain scheduling, ” Proceeding ASPLOS XV Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems, Pages 311-322

[19] Oracle Corporation: “ Solaris ZFS Administration Guide, ”Oracle Corporation (online), Part No: 817-2271, available from http://docs.oracle.com/cd/E19082-01/817-2271/ (accessed 2013-05-01).

[20] Srinivasan, M, “ Facebook Flashcache, ” https://github.com/facebook/flashcache

[21] Thanos Makatos, Yannis Klonatos, Manolis Marazakis, Michail D. Flouris and Angelos Bilas, “ Using transparent compression to improve SSD-based I/O caches, ”Proc. of the 5th European conference on Computer systems (EuroSys ’10), pp.1-14 (2010).

[22] Timothy Pritchett and Mithuna Thottethodi, “ SieveStore: a highly-selective, ensemble-level disk cache for cost-performance, ”Proc. of the 37th annual international symposium on Computer architecture (ISCA ’10), pp.163-174 (2010).

[23] Jeanna Matthews, Sanjeev Trika, Debra Hensgen, Rick Coulson and Knut Grimsrud, “ Intel Turbo Memory: Nonvolatile disk caches in the storage hierarchy of mainstream computer systems, ”ACM Trans. on Storage(TOS), Vol.4, No.2, pp.1-24 (2008).

[24] Xiaoyu Yao and Jun Wang, “ RIMAC: a novel redundancy-based hierarchical cache architecture for energy efficient, ” high performance storage systems, Proc. of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006 (EuroSys ’06), pp.249-262 (2006).

[25] Dong Li and Jun Wang, “ EERAID: energy efficient redundant and inexpensive disk array, ” Proc. of the 11th workshop on ACM SIGOPS European workshop (EW 11), pp.174-180 (2004).

[26] Athanasios E. Papathanasiou and Michael L. Scott, “ Energy efficient prefetching and caching, ”Proc. of the annual conference on USENIX Annual Technical Conference (ATEC ’04), pp.255-268 (2004).

[27] 引田 諭之，HieuHanh Le，横田 治夫, “ ストレージ省電力化手法の電力削減におけるシステム構成の影響, ” DEIM Forum 2012 D6-2

[28] 坂本龍一，佐藤未来子，小泉佑介，近藤正章，天野英晴，中村宏，並木美太郎, “ 組込み向け CMA アクセラレータ用 OpenCL ライブラリと OS の設計, ” 情報処理学会第 76 回全国大会講演論文集, 2K-7 (2014-03-11).

[29] Khronos OpenCL, https://www.khronos.org/opencl/

[30] Alan G.Yoder, “ Green Storage Technologies CAPEX and OPEX, ”“ Storage Networking World Spring 2011 (SNW Spring 2011)(online), available from https://www.eiseverywhere.com/file_uploads/201592e05a06cffedbfcb3a9d30ca726_Yoder_Tuesday_0305_SNWS11.pdf (accessed 2013-05-01).

[31] 仁科圭介, 佐藤未来子, 並木美太郎, “ SSD をディスクキャッシュとして用いる Linux デバイスドライバの実装, ”情報処理学会プログラミング・シンポジウム予稿集, Vol.53, pp.29-36 (2012).

[32] FTDI,“ FT232R USB UART IC, ”http://www.ftdichip.com/Products/ICs/FT232R.htm

[33] ヒューマンデータ,“ ACM/XCM シリーズ共通資料, ”http://www.hdl.co.jp/ftpdata/acm_xcm/

[34] Texas Instruments, “ INA226, ”http://www.tij.co.jp/product/jp/INA226

[35] リニアテクノロジー, “ LTC3447, ”http://www.linear-tech.co.jp/product/LTC3447

[36] Xilinx, “ Platform Studio と EDK(Embedded Development Kit), ”http://japan.xilinx.com/tools/platform.htm

[37] http://japan.xilinx.com/products/boards-and-kits/ek-v6-ml605-g.html

[38] sourceware.org, “ Device-mapper Resource Page (online), ” available from http://sourceware.org/dm (accessed 2013-05-01).

[39] Sourceforge, “ Filebench (online), ” available from http://sourceforge.net/apps/mediawiki/filebench/index.php?title=Main_Page (accessed 2013-05-01).

[40] Priya Sehgal, Vasily Tarasov and Erez Zadok, “ Optimizing energy and performance for server-class file system workloads, ”ACM Trans. Storage, Vol.6, No.3, pp.10:1-10:31 (2010).

付録　業績一覧

論文誌

(1) 坂本 龍一, 仁科 圭介, 佐藤 未来子, 並木 美太郎：キャッシュとして利用するディスク I/O の省電力化手法, 情報処理学会論文誌, Vol.55, No.4, pp.1389-1403, ISSN:1882-7764 (2014-04-15).

国際会議

(1) Ryuichi Sakamoto, Mikiko Sato, Yusuke Koizumi, Hideharu Amano, Mitaro Namiki: An OpenCL Runtime Library for Embedded Multi-Core Accelerator, IEEE 18th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2012), pp.419-422, Aug. 19-22, 2012, Seoul, South Korea.

(2) Masaaki Kondo, Hiroaki Kobyashi, Ryuichi Sakamoto, Motoki Wada, Jun Tsukamoto, Mitaro Namiki, Weihan Wang, Hideharu Amano, Kensaku Matsunaga, Masaru Kudo, Kimiyoshi Usami, Toshiya Komoda and Hiroshi Nakamura: Design and Evaluation of Fine-Grained Power-Gating for Embedded Microprocessors, Design, Automation and Test in Europe Conference and Exhibition (DATE 2014), pp.1-6, March 2014.

(3) Kimiyoshi Usami, Masaru Kudo, Kensaku Matsunaga, Tsubasa Kosaka, Yoshihiro Tsurui, Weihan Wang, Hideharu Amano, Hiroaki Kobayashi, Ryuichi Sakamoto, Mitaro Namiki, Masaaki Kondo and Hiroshi Nakamura: Design and Control Methodology for Fine Grain Power Gating based on Energy Characterization and Code Profiling of Microprocessors, 19th Asia and South Pacific Design Automation Conference (ASP-DAC 2014). pp.843-848, Jan 2014.

(4) Atsushi Koshiba, Motoki Wada, Ryuichi Sakamoto, Mikiko Sato, Tsubasa Kosaka, Kimiyoshi Usami, Hideharu Amano, Masaaki Kondo, Hiroshi Nakamura and Mitaro Namiki：A Fine-grained Power Gating Control using Leakage Monitor by Linux Process Scheduler, IEEE Symposium on Low-Power and High-Speed Chips(CoolChips) XVII, Poster, April 15, 2014.

(5) Noriyuki Miura, Yusuke Koizumi, Eiichi Sasaki, Yasuhiro Take, Hiroki Matsutani, Tadahiro Kuroda, Hideharu Amano, Ryuichi Sakamoto, Mitaro Namiki, Kimiyoshi Usami, Masaaki Kondo and Hiroshi Nakamura: A Scalable 3D Heterogeneous Multi-Core Processor with

Inductive-Coupling ThruChip Interface, IEEE Symposium on Low-Power and High-Speed Chips(CoolChips XVI), pp.1-3, April 2013.

(6) Motoki Wada, Jun Tsukamoto, Hiroaki Kobayashi, Akihiro Takahashi, Ryuichi Sakamoto, Mikiko Sato, Masaaki Kondo, Hideharu Amano, Hiroshi Nakamura, and Mitaro Namiki: Dalvik VM JIT Compiler for Fine-Grained Power Gating Control, IEEE Symposium on Low-Power and High-Speed Chips(CoolChips) XVI, Poster, April 18, 2013.

(7) Jun Tsukamoto, Motoki Wada, Yumi Shimada, Ryuichi Sakamoto, Mikiko Sato, Masaaki Kondo, Kimiyoshi Usami, Hideharu Amano, Hiroshi Nakamura, and Mitaro Namiki: A Basic Study on Leakage Power Reduction for FPU by Fine-grained Power Gating Control, IEEE Symposium on Low-Power and High-Speed Chips(CoolChips) XVI, Poster, April 18, 2013.

(8) Yusuke Koizumi, Hideharu Amano, Hiroki Matsutani, Noriyuki Miura, Tadahiro Kuroda, Ryuichi Sakamoto, Mitaro Namiki, Kimiyoshi Usami, Masaaki Kondo, Hiroshi Nakamura: Dynamic power control with a heterogeneous multi-core system using a 3-D wireless inductive coupling interconnect, FPT 2012, pp.293-296, December 10-12, 2012, Seoul, South Korea.

(9) Yusuke Koizumi, Eiichi Sasaki, Hideharu Amano, Hiroki Matsutani, Yasuhiro Take, Tadahiro Kuroda, Ryuichi Sakamoto, Mitaro Namiki, Kimiyoshi Usami, Masaaki Kondo, Hiroshi Nakamura: CMA-Cube: A scalable reconfigurable accelerator with 3-D wireless inductive coupling interconnect, FPL 2012, pp.543-546, August 29-31, 2012, Oslo, Norway.

(10) M.Sato, G.Fukazawa, K.Nagamine, R.Sakamoto, M.Namiki, K.Yoshinaga, Y.Tsujita, A.Hori, and Y.Ishikawa: A Design of Hybrid Operating System for a Parallel Computer with Multi-Core and Many-Core Processors, In Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers (ROSS '12), Article 9, June 29, 2012. Venice, Italy.

(11) Yumi Shimada, Hiroaki Kobayashi, Akihiro Takahashi, Ryuiti Sakamoto, Mikiko Sato, Masaaki Kondo, Hideharu Amano, Hiroshi Nakamura, and Mitaro Namiki: A Fine-Grained Power Gating Control for a Real Time Operating System, IEEE Symposium on Low-Power and High-Speed Chips(CoolChips) XV, Poster, April 18-20, 2012. Yokohama, Japan.

査読付き国内シンポジウム

(1) 高橋昭宏，小林弘明，坂本龍一，並木美太郎，佐藤未来子，中村宏，天野英晴，宇佐美公良，近藤正章，佐々木広：FPGA による省電力計算機の開発評価環境の試作, 先進的計算基盤システムシンポジウム（SACSIS 2011）ポスターセッション No.60, pp.269-270 (2011.05.26).

査読無しシンポジウム・研究会・全国大会等

(1) 坂本 龍一, 佐藤 未来子, 天野 英晴, 黒田 忠広, 宇佐美 公良, 中村 宏, 並木美太郎: 組込み向けアクセラレータ用 OpenCL ライブラリと実行制御機構の設計, SWoPP 新潟 2014, 信学技報, vol.114, no.155, CPSY2014-46, pp.215-220 (2014-07-21).

(2) 坂本 龍一 佐藤未来子，小泉佑介，近藤正章，天野英晴，中村宏，並木美太郎: 組込み向け CMA アクセラレータ用 OpenCL ライブラリと OS の設計, 情報処理学会 第 76 回全国大会講演論文集, 2K-7 (2014-03-11).

(3) 坂本龍一，望月秋人，小林弘明，高橋昭宏，佐藤未来子，天野英晴，中村宏，並木美太郎: 組込み向けメニーコアアクセラレータ用 OpenCL の設計と組込み OS の実装情報処理学会 第 121 回システムソフトウェアとオペレーティング・システム研究発表会・第 200 回計算機アーキテクチャ研究発表会合同研究会, Vol.2012-OS-121 2012-ARC-200, No.2, pp.1-10, 2012-05-07.

(4) 坂本 龍一，佐藤 未来子，天野 英晴，中村 宏，近藤 正章，並木 美太郎: OpenCL を用いたメニーコア・アクセラレータの仮想化手法と評価環境の構築, 情報処理学会 第 74 回全国大会講演論文集, 4J-2 (2012.03.07).

(5) 嶋田 裕巳, 坂本 龍一, 塚本 潤, 和田 基, 佐藤 未来子, 並木 美太郎: リアルタイム OS による細粒度パワーゲーティング制御手法の検討, SWoPP 北九州 2013, 情報処理学会「システムソフトウェアとオペレーティング・システム」第 126 回研究発表会 (2013-07-31).

(6) 小柴 篤史，塚本 潤, 和田 基, 坂本 龍一, 佐藤 未来子，小坂 翼，宇佐美 公良, 天野 英晴, 近藤 正章, 中村 宏, 並木 美太郎：Linux スケジューラによるリークモニタを用いた細粒度パワーゲーティング制御手法と実チップにおける評価, 情報処理学会 「システムソフトウェアとオペレーティング・システム」第 129 回研究発表会, Vol.2013-OS-129, No.14, pp.1-9 (2014-05-07).

(7) 塚本 潤，和田 基，嶋田 裕巳，坂本 龍一，佐藤 未来子，近藤 正章，宇佐美 公良，天野 英晴，中村 宏，並木 美太郎: FPU における細粒度パワーゲーティング制御手法の基礎的検討, 情報処理学会 第 197 回計算機アーキテクチャ・第 125 回システムソフトウェアとオペレーティング・システム合同研究発表会, Vol.2013-ARC-205(7), pp.1-8 (2013-04-18).

(8) 和田 基，塚本 潤，小林 弘明，高橋 昭宏，坂本 龍一，佐藤 未来子，近藤 正章，天野 英晴，中村 宏，並木美太郎: Dalvik VM による細粒度 PG 制御の動的コード生成, 情報処理学会 第 197 回計算機アーキテクチャ・第 125 回システムソフトウェアとオペレーティング・システム合同研究発表会, Vol.2013-OS-125(5), pp.1-8 (2013-04-18).

(9) 松永健作，工藤優，太田雄也，小西奈緒，天野英晴，坂本龍一，並木美太郎，宇佐美公良: オンチップ・リークモニタによるランタイム・パワーゲーティングの制御にむけた損益分

岐点の評価, 情報処理学会システム LSI 設計技術研究発表会，Vol.2013-SLDM-159, No.12, pp.1-6, 2013-1-9.

(10) 深沢豪，長嶺精彦，坂本龍一，佐藤未来子，吉永一美，辻田祐一，堀敦史，石川裕，並木美太郎: マルチコア・メニーコア混在型計算機における軽量 OS 向け I/O ライブラリの提案, 情報処理学会第 122 回システムソフトウェアとオペレーティング・システム研究発表会, Vol.2012-OS-122, No.6, pp.1-8, 2012-08-01.

(11) 深沢豪，長嶺精彦，坂本龍一，佐藤未来子，吉永一美，辻田祐一，堀敦史，石川裕，並木美太郎: マルチコア・メニーコア混在型計算機における演算コア側資源管理の代行方式, 情報処理学会第 134 回ハイパフォーマンスコンピューティング研究発表会, Vol.2012-HPC-134, No.7, pp.1-8, 2012-06-01.

(12) 嶋田裕巳，小林弘明，高橋昭宏，坂本龍一，佐藤未来子，近藤正章，天野英晴，中村宏，並木美太郎: リアルタイム OS における細粒度パワーゲーティング制御の設計と実装, 情報処理学会第 121 回システムソフトウェアとオペレーティング・システム研究発表会・第 200 回計算機アーキテクチャ研究発表会合同研究会, Vol.2012-OS-121 2012-ARC-200, No.16, pp.1-8, 2012-05-08.

(13) 仁科圭介，坂本龍一，佐藤未来子，並木美太郎: SSD をディスクキャッシュとして利用するディスク I/O の高速・省電力化手法の提案, 情報処理学会第 121 回システムソフトウェアとオペレーティング・システム研究発表会・第 200 回計算機アーキテクチャ研究発表会合同研究会, Vol.2012-OS-121 2012-ARC-200, No.5, pp.1-8, 2012-05-07.

(14) 嶋田裕巳, 小林弘明, 高橋昭宏, 坂本龍一, 佐藤未来子, 近藤正章, 天野英晴, 中村 宏, 並木美太郎,：リアルタイムシステムにおける細粒度パワーゲーティング制御の研究，情報処理学会 第 74 回全国大会講演論文集, 3K-5 (2012.03.07).

(15) 高橋 昭宏，小林弘明，坂本龍一，佐藤未来子，並木美太郎: Linux における演算ユニットの電力特性を考慮した細粒度パワーゲーティング制御手法, 情報処理学会「システムソフトウェアとオペレーティング・システム」第 120 回 OS 研究報告, Vol.2012-OS-120(4), pp.1-8(2012.02.28).

(16) 長嶺精彦，坂本龍一，佐藤未来子，下沢拓，吉永一美，辻田祐一, 堀敦史，石川裕，並木美太郎: メニーコア混在型並列計算機におけるスレッド管理方式, 情報処理学会「ハイパフォーマンスコンピューティング」第 132 回研究報告, Vol.2011-HPC-132(30), pp.1-8 (2011.11.21).

(17) 佐々木 瑛一, 佐々木 大輔, 天野 英晴, 松谷 宏紀, 坂本 龍一, 並木 美太郎：チップ間ワイヤレス接続を利用した三次元積層アーキテクチャの研究, 信学技報, vol. 111, No.163, CPSY2011-10, pp.7-12 (2011.07.29).

(18) 中條拓伯, 坂本龍一: スケーラブルFPGAシステムにおけるハードウェア・アクセラレーション, 電子情報通信学会技術研究報告. Vol.109, No.395 (RECONF2009-73), pp.119-124 (2010.1)

(19) 中條拓伯, 三好健文, 船田悟史, 坂本龍一: スケーラブルFPGAシステムにおけるハードウェア拡張方式, 電子情報通信学会技術研究報告. Vol.109, No.395 (RECONF2009-73), pp.125-130 (2010.1)

口頭発表

(1) 坂本 龍一, 龍一, 小柴 篤史, 和田 基, 佐藤 未来子, 並木 美太郎：ハードウェア・ソフトウェア協調によるプロセッサの省電力化, 2014年多摩合同コロキウム, (2014-11-01〜02).

(2) 坂本 龍一：回路技術とプロセッサとOS，第46回情報科学若手の会，(2013-09-15)

(3) 坂本 龍一, 高橋 昭宏, 小林 弘明, 佐藤 未来子, 天野 英晴, 宇佐美 公良, 黒田 忠広, 近藤 正章, 中村 宏, 三浦 典之, 並木 美太郎: 三次元ワイヤレス積層によるヘテロジニアスマルチコアプロセッサとシステムソフトウェアの実現, 情報処理学会第11回先進的計算基盤システムシンポジウム (SACSIS 2013), ポスター・デモセッション, pp.97-98 (2013-05-15).

(4) 坂本 龍一，小林 弘明，高橋 昭宏，佐藤 未来子，並木 美太郎，天野 英晴, 宇佐美 公良，近藤 正章，中村 宏: 省電力プロセッサとOSの開発評価環境の実現, 情報処理学会第24回コンピュータシステムシンポジウム (ComSys 2012), ポスター・デモセッション (2012.12.6).

(5) 坂本 龍一: 組込み向けマルチコアアクセラレータ用OpenCLライブラリと組込みOSの設計, JSASS2012 (2012.9.10).

(6) 坂本 龍一: 組込み向けマルチコアプロセッサ用OpenCLと組込みOSの設計, 早稲田・農工大三研究室合同研究発表会 (2012.8.25).

(7) 中條拓伯, 三好健文，坂本龍一: スケーラブルFPGAシステムにおけるハードウェア拡張プロトコル，信学技報(リコンフィギャラブルシステム研究会（RECONF）(2009.12)

受賞

(1) 情報処理学SACSIS2013 優秀ポスター賞: 坂本 龍一, 高橋 昭宏, 小林 弘明, 佐藤 未来子, 天野 英晴, 宇佐美 公良, 黒田 忠広, 近藤 正章, 中村 宏, 三浦 典之, 並木 美太郎: 三次元ワイヤレス積層によるヘテロジニアスマルチコアプロセッサとシステムソフトウェアの実現 (2013-05-15). http://sacsis.hpcc.jp/2013/program.html.ja

(2) 情報処理学第197回計算機アーキテクチャ研究会 若手奨励賞: 和田 基，塚本 潤，小林 弘明，高橋 昭宏，坂本 龍一，佐藤 未来子，近藤 正章，天野 英晴，中村 宏，並木美太郎:

Dalvik VM による細粒度 PG 制御の動的コード生成 (2013-04-18). http://sigarc.hpcc.jp/? 計算機アーキテクチャ研究会若手奨励賞受賞者一覧

(3) Best Feature Award (Poster): Yumi Shimada, Hiroaki Kobayashi, Akihiro Takahashi, Ryuiti Sakamoto, Mikiko Sato, Masaaki Kondo, Hideharu Amano, Hiroshi Nakamura, and Mitaro Namiki: A Fine-Grained Power Gating Control for a Real Time Operating System, IEEE Symposium on Low-Power and High-Speed Chips(CoolChips) XV, April 18-20, 2012. Yokohama, Japan.

(4) 情報処理学 SACSIS2011 ポスター賞: 高橋昭宏，小林弘明，坂本龍一，並木美太郎，佐藤未来子，中村宏，天野英晴，宇佐美公良，近藤正章，佐々木広：FPGA による省電力計算機の開発評価環境の試作, (2011.05.26). http://sacsis.hpcc.jp/2011/program.html.ja#AWARDP

(5) 情報処理学会第 74 回全国大会 学生奨励賞: 嶋田裕巳, 小林弘明, 高橋昭宏, 坂本龍一, 佐藤未来子, 近藤正章, 天野英晴, 中村 宏, 並木美太郎,：リアルタイムシステムにおける細粒度パワーゲーティング制御の研究，(2011.03.07).

付録　添付論文一覧

論文誌

1. 坂本 龍一, 仁科 圭介, 佐藤 未来子, 並木 美太郎: SSD をディスクキャッシュとして利用するディスク I/O の省電力化手法, 情報処理学会論文誌, Vol.55, No.4, pp.1389-1403, ISSN:1882-7764 (2014-04-15).

2. 坂本 龍一, 佐藤 未来子, 並木 美太郎: マルチコアアクセラレータのパイプライン処理のための OpenCL 実行環境, (電子情報通信学会　論文誌 D 投稿中)

国際会議

1. Ryuichi Sakamoto, Mikiko Sato, Yusuke Koizumi, Hideharu Amano, Mitaro Namiki: An OpenCL Runtime Library for Embedded Multi-Core Accelerator, IEEE 18th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2012), pp.419-422, Aug. 19-22, 2012, Seoul, South Korea.