Vol.2014-HPC-145 No.11
2014/7/28

高性能計算環境向け電力配分自動最適化のためのコンパイラ環境の構築

和田 康孝[1,a)]　稲富 雄一[2,5]　井上 弘士[2,5]　三吉 郁夫[3,5]　近藤 正章[4,5]　本多 弘樹[1]

概要：将来の HPC システムでは，消費電力がシステムの設計や実効性能を制約する最大の要因の一つになると考えられており，利用可能な電力バジェットを最重要資源とする電力志向型 HPC システムに関する研究開発が進められている．この電力志向型システムにおいてアプリケーションの実効性能を向上させるためには，与えられた消費電力の上限（電力バジェット）の制約下で，CPU や DRAM，アクセラレータ等の電力性能ノブを調節し，各要素に適切に消費電力を配分する必要がある．しかしながら，このためには，各電力性能ノブと対象アプリケーションの実行性能の関係を予め知る必要があり，そのための作業に大きな手間を要する．また，従来提案されている電力配分最適化手法は多岐にわたり，適用する手法毎に異なる情報を取得する必要がある．本稿では，これらの問題を解決し，並列アプリケーションの性能/電力解析および電力配分を自動化するコンパイラフレームワークについて述べる．

1. はじめに

従来の HPC システムの性能向上は，計算ノード数の増加および各ノードの性能向上によって支えられてきた．しかしながら，将来の HPC システムでは，消費電力がシステム設計や実効性能を制約する最大の要因の一つになると考えられており，これまでと同様の方式でシステムの性能を向上させていくことは難しい．本稿執筆時点で世界最高性能を誇る Tianhe-2 は 18MW 近い消費電力を必要としており，京は 12MW 超の電力を用いて 10 ペタフロップスの性能を実現している [1] が，2020 年頃の実現を目指すエクサスケール級のシステムにおいても，20MW から 30MW 程度の電力供給が限界であると言われている [2]．つまり，ほぼ同じ消費電力で現在の世界トップクラスのスーパーコンピュータの 30 倍から 50 倍近い性能を達成することが求められることになる．また，システムのピーク消費電力（熱設計消費電力）が消費電力の制約を超えないことを保証する形で従来の HPC システムは設計されてきたが，このような設計方法では，様々な特性を持つアプリケーションを今後の大規模システムに対してスケールさせることは難しく，限られた電力を有効に活用できるとは言い難い．

このような背景のもと，我々はシステムのピーク消費電力が制約を超過することを積極的に許容し，ハードウェアが持つ電力性能ノブを調整することで限られた電力資源を計算・記憶・通信・アクセラレータ等の各要素に適応的に配分し，実効電力を制約以下に制御しつつ高い実効性能を得る電力制約適応型システムが有効であると考え，研究開発を進めている [3]．この電力制約適応型システムにおいては，システム全体およびシステム上で実行される各ジョブに対して消費電力の上限（電力バジェット）が設定されるため，これを超過しない範囲で各ジョブ（アプリケーション）の実効性能を向上させる必要がある．

HPC システム上で動作させるアプリケーションの特性は多様化しており [4]，個々のアプリケーションを電力制約適応型システム上で最適化するためには，予め与えられた電力バジェットを超えない範囲で，各ハードウェアが持つ電力性能ノブを適切に調整し，それぞれに適切に消費電力を配分する必要がある．このような最適化を行う際には，アプリケーション自身の特性はもとより，各電力性能ノブと対象アプリケーションの実行性能の関係を予め調査・解析し，それに基づいて電力の配分を行わなくてはならない．この解析では，電力性能ノブの設定を変えて何度もアプリケーションを実行する必要があるうえ，取得すべき情報はそれぞれの消費電力配分最適化手法によって異なって

[1] 電気通信大学
The University of Electro-Communications
[2] 九州大学
Kyushu University
[3] 富士通株式会社
Fujitsu Limited
[4] 東京大学
The University of Tokyo
[5] 科学技術振興機構/CREST
JST/CREST
[a)] wada@is.uec.ac.jp

おり，最適化には大きな手間を要する．

このような問題に対応するため，本稿では，並列アプリケーションの性能・電力解析および電力配分を自動化するコンパイラ環境を提案する．このコンパイラ環境では，アプリケーション中の適切な箇所に，電力性能ノブの制御や消費電力の計測，パフォーマンスカウンタ等の性能情報を取得する処理が自動的に追加される．これによりアプリケーションの解析にかかる手間を大幅に削減できる．さらに，同じプログラムを用いて電力配分最適化の結果に基づいたアプリケーション実行が可能である．このようにして，電力志向型 HPC システムに対する並列アプリケーションの最適化を簡単に行うことができる．

以降，2 節では高性能計算機における電力配分最適化手法の研究動向と電力調整ノブの種類および制御方法について，3 節では電力配分最適化に関する手順について，4 節ではコンパイラ環境プロトタイプの実装について，5 節では実際の電力配分最適化手法に対する適用とその評価について，それぞれ述べる．

2. 並列アプリケーション内での電力配分最適化

ここでは，アプリケーション内の電力配分最適化手法の研究動向および各手法で用いられる電力制御のための手段について述べる．

2.1 既存の電力配分最適化手法

並列アプリケーションを対象とした電力配分最適化手法は，これまでに複数提案されている．これらの共通した目的は，ある与えられた消費電力の上限を超えない範囲で，システム内の各要素間で電力を適切に配分することである．これにより，与えられた消費電力下でアプリケーションの実効性能を最大化する．

CPU 間の電力配分最適化: 並列アプリケーションでは各スレッドあるいはプロセス間の負荷不均衡により，同期ポイントにおける待ち時間が多く発生する．通常，同期ポイントではスピンウェイトによって他のスレッド・プロセスの到着を待つため，無駄な電力消費が発生してしまう．同期ポイントにおける待ち時間を考慮して各ノード・CPU の動作周波数を設定することで，アプリケーションの性能を低下させずに消費電力を削減する手法が提案されている [5].

CPU-DRAM 間の電力配分最適化: Intel のプロセッサでは，RAPL インタフェース [6] を用いることで，CPU および DRAM の消費電力測定及び電力制御を行うことができるようになっている．この機能を利用して，使用ノード数とともに CPU と DRAM 間で消費電力の配分を行う手法 [7] が提案されている．また，CPU および DRAM に対して適切な電力配分を行った際の性能向上に関する報告 [8, 9] がなされているほか，実際の物理メモリ量を増減して CPU・DRAM 間の電力配分を最適化する手法 [10] 等が提案されている．

CPU-GPU 間の電力配分最適化: 近年では，GPU などのアクセラレータを搭載したシステムが広く用いられるようになってきている．GPU の消費電力は CPU と比較しても大きいが，消費電力あたりの性能は高く，この特性を考慮した最適化が必要である．CPU と GPU 間の処理割当およびそれぞれの動作周波数をあわせて変更することで，与えられた消費電力制約下で GPU アプリケーションの性能を最大化するための手法が提案されている [11].

CPU-ネットワーク間の電力配分最適化: 特に大規模高性能計算機環境において，システム内の各ノードを接続するインタコネクトは性能向上のための重要な要素であり，年々複雑さを増している．これに従ってネットワークに関する消費電力は増大する傾向にあるが，通信を行わない間，低消費電力モードに移行して消費電力を削減することが可能になってきている．これにより削減された消費電力を CPU に配分する手法 [12] が提案されている．

その他，蓄電池を用いて時間方向で消費電力の再配分を行う手法 [13] なども提案されている．また，将来的に，多数の要素間で消費電力を配分し合う複合的な手法が考案されることも考えられる．

2.2 電力性能ノブの操作手段

2.1 節で紹介したとおり，各種の消費電力配分最適化手法では，主として CPU，DRAM，GPU が電力配分最適化の対象として扱われている．これは，各手法において，ユーザアプリケーションから明示的に電力性能ノブの操作が必要なためであり，そのためのインタフェースが利用可能であることが求められるためである．また，高性能計算機環境における消費電力の内訳においても，CPU，DRAM，GPU およびインタコネクトによって大部分が占められているため，これらの要素を最適化対象とすることで大きな性能向上が見込める．表 1 に代表的な電力性能ノブとその操作手段をまとめる．

表 1 代表的な電力性能ノブとその操作手段

電力性能ノブ	ノブの操作手段
CPU	動作周波数の変更（CPUFreq 等）
	消費電力上限指定（RAPL）
DRAM	ホットプラグによるモジュール数変更
	消費電力上限指定（RAPL）
GPU	動作周波数の変更（NVML 等）

CPU については，従来の CPUFreq 等を用いた動作周波数の変更がまず挙げられる．さらに，Sandy Bridge 以降の Intel 製プロセッサにおいては，RAPL（Running Average Power Limit）インタフェースを用いた操作も可能である．

このRAPLインタフェースを介することで，プロセッサのみならずDRAMの消費電力に関する情報を取得したり，消費電力の上限を設定することができる [6, 14]．また，RAPLでは消費電力を観測・制御する単位が3種類定義されており，サーバ環境では，チップ全体（Package, PKG），チップ上のコア部分（Power Plane 0，PP0）およびメモリ（DRAM）がそれにあたる．Linux OS上では，MSR（Model Specific Register）を介して消費電力の取得，消費電力の上限設定などの操作を上記のドメイン毎に適用することが可能である [15]．RAPLによる消費電力の情報は精度が高く，小さい時間粒度で取得可能であるため [16]，その点では電力配分の最適化に適していると言えるが，与えられた消費電力の上限と実行性能の関係が必ずしも明確ではないという問題点もある．

DRAMについては，メモリホットプラグ [17] を前提としたモジュール数の変更，あるいは上記のRAPLを用いた消費電力上限の設定が考えられる．本稿執筆時点においては，メモリホットプラグはOSやハードウェアのサポートがまだ十分でないことなどから，RAPLによる操作が現実的には利用しやすい手段であると言える．

GPUについては，例えばNVIDIA製のものであれば，NVML [18] が提供するAPIにより，動作周波数の設定や消費電力の上限を設定することが可能である．

3. 電力配分最適化の手順

ここでは，電力配分最適化の手順を一部自動化し，様々な最適化手法を簡単に適用することを可能とするためのコンパイラ環境を構築することを目的として，2節で述べた電力配分最適化手法に共通する手順についてまとめる．

2節で述べた通り，高性能計算機環境を対象としたアプリケーション内電力配分最適化手法では，多くの場合，対象アプリケーションおよび対象システムの消費電力・実行性能に関する情報を予め取得しておく必要がある．

このため，最適化の手順としてまず必要なことは，電力配分調整に用いる電力性能ノブの設定を変更しつつ，アプリケーションを複数回実行するということになる．このとき，予め対象アプリケーション内の適切な箇所に，ノブ調整と消費電力・実行性能計測のための処理を予め埋め込んでおく必要がある．また，実行ごとにノブ調整の設定を変更する必要がある．

その後，得られた消費電力・実行性能に関する情報を元に，対象とする各電力性能ノブ間での消費電力配分を最適化し，その結果に基づいてアプリケーションを実行する．これにより，与えられた電力バジェットを超過しない範囲でアプリケーションの実行性能を最大化することができる．

以上の最適化の手順をまとめると，共通して下記の3つの段階を経て，電力配分が最適化された状態でのアプリケーション実行を実現していることになる．

1. 電力配分対象のノブに対する設定と消費電力の関係を取得する．
2. 計測結果から，アプリケーション内の対象区間における電力の配分を最適化する．
3. 最適化結果に基づいて電力性能ノブを制御しつつ実行する．

また，いずれの最適化手法においても，最適化・電力性能ノブ制御の単位はほぼ共通しており，下記3種類のいずれかとなっている．これよりもさらに短期間で電力性能ノブの調整を行う場合には，制御オーバヘッドの問題が無視できないものになると考えられる．

- アプリケーション全体
- 同期ポイント間の処理
- アプリケーション内の主要関数

この電力性能ノブの制御単位をどのように選択するかによって，プログラムの最適化にかかるコスト，与えられた電力バジェット下におけるアプリケーションの実行性能は大きく異なる．例えば，アプリケーション全体を制御単位として電力分配を行った場合と比較して，アプリケーション内の主要間数単位で配分を制御した場合では，10[%] 以上実行性能が向上する場合もある [9]．アプリケーション内の主要間数単位で電力配分の最適化を行うことで，電力分配最適化手法の効果をより高めることができるが，この場合，関数毎に細かく情報を取得して制御を行わなくてはならず，消費電力・実行性能に関する情報の取得，および電力配分を最適化したプログラムを作成する作業がプログラマあるいはユーザに大きな負担となる．この問題を解決し，アプリケーション内の主要関数毎に計測あるいはノブ制御のためのコードを自動的に追加する仕組みが求められる．

4. コンパイラ環境の構築

ここでは，3節で述べた電力配分最適化の手順を自動的に適用するためのコンパイラ環境に関して，そのプロトタイプの実装・構築について述べる．本プロトタイプは，パフォーマンス解析ツールTAU [19]，およびこれと協調して動作するプログラム解析ツールPDT [20] をベースに実装された．PDTは入力のソースプログラムを解析し，プログラムの構造に関する情報を独自の形式で保持することができる．この情報を利用することで，プログラムの構造を考慮しつつ，適切な箇所に電力情報の計測・電力性能ノブの制御に関する処理を埋め込むことが可能となる．また，必要に応じて，TAUによるパフォーマンス解析の結果を消費電力配分最適化のために利用することも可能となる．

4.1 コンパイラフレームワークの全体像

本稿では，CPUおよびDRAM間で電力配分最適化を行う手法を対象としてプロトタイプの実装を行った．この際，

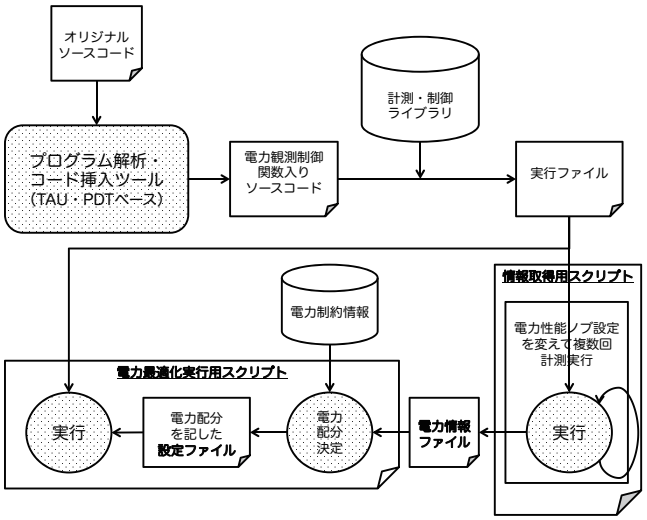

図 1 コンパイラ環境を用いた最適化プロセス全体像

最適化対象アプリケーションへの電力制御・性能計測コード埋め込みを行うコンパイラ環境の構築にあたり，HPC アプリケーション向けのパフォーマンス解析ツールである TAU [19]，およびこれと協調して動作するソースプログラム解析ツール PDT [20] をベースとして用いた．

また，この環境においては，電力配分最適化を適用する単位をアプリケーション内の関数として取り扱うこととした．これにより，3 節で述べた最適化の単位（アプリケーション全体，同期ポイント間，主要関数）をサポートし，性能計測・電力制御のオーバヘッドを抑えつつ，十分細やかな電力配分最適化を適用することができる．

図 1 に本稿で構築したコンパイラ環境を用いた最適化の全体像を示す．これは，3 節で述べた共通の手順をもとにしたものである．

図 1 では，まず TAU/PDT の機能をベースに実装したツールにより，対象アプリケーション内主要関数の開始・終了点に，電力性能ノブの制御および消費電力・実行性能の計測を行うコードを埋め込む．これらの機能は，後述のライブラリによって提供される．

その後，コード埋め込み後のソースコードとライブラリから実行ファイルを生成する．この実行ファイルは，再コンパイルすることなしに計測および最適化後の実行に用いることができる．電力性能ノブの設定を変更しつつこのプログラムを実行することで，電力配分最適化に必要な情報を順次取得する．

次に，取得した情報を用いて消費電力配分の最適化を行い，予め定められた形式でその結果を記述したファイルを作成する．ここで適用する最適化手法は，対象とするアプリケーションやシステムの環境によって適切なものを選択可能であり，最適化手法そのものは本稿では対象としていない．また，上述の通り，本稿では CPU-DRAM 間の電力配分最適化を対象として，それぞれの制御手段として CPUFreq および RAPL を用いることとした．

最後に，最適化結果を記述したファイルを読み込み，それに従って電力性能ノブを制御しつつ実行を行う．これにより，予め与えられた消費電力の上限を守りつつ，アプリケーションの実行性能を向上させることができる．

以上のような流れで，高性能計算機環境向けの電力配分自動最適化のための処理が行われる．これにより，電力配分最適化のための作業を大幅に削減することができる．

4.2 TAU および PDT をベースとしたツールによる観測・制御ポイントの埋め込み

4.1 節で述べた通り，図 1 中の最初の手順において，対象アプリケーション内の主要関数の開始・終了点に観測・制御ポイントを埋め込む．この処理を行うためには，いわゆるパフォーマンス解析ツールをベースに環境を構築するのが望ましい．

HPC システム向けのパフォーマンス解析ツールとしては，TAU の他，Scalasca [21] や Score-P [22] などが研究開発されている．いずれのパフォーマンス解析ツールにも共通する点として，アプリケーション内の各関数が実行時間に占める割合を示すプロファイリング情報と，関数の開始・終了や通信のタイミング等をイベントとして時系列で記録していくトレース情報を取得するために，1）対象アプリケーションのソースコードを解析し，情報取得に必要な機能を埋め込む，2）通信やワークシェアリングに関する情報を実行時に取得するため，MPI や OpenMP の機能に対してラッパーを用意して適用する，という点が挙げられる．

このようなパフォーマンス解析ツールの機能を用いることで，比較的容易に環境を構築することが可能である．なかでも，TAU では Program Database Toolkit（PDT）と連携することで，より詳細かつ動的に情報を取得することを可能としているため，本稿ではこれらをベースにプロトタイプを構築することとした．また，将来的には，パフォーマンス解析ツールによって取得できる様々な情報を電力配分最適化手法の為に活用することも考えられる．

図 2 に，本プロトタイプによる観測・制御ポイントの埋め込み例を示す．図中，左側のコードは埋め込み前のものであり，右側のコードが埋め込み後のものである．このコード埋め込みは TAU/PDT によるプロファイル取得コードの挿入と同時に行うことが可能であるが，本稿にて実装したプロトタイプでは消費電力・性能情報の取得は後述のライブラリ内部で行う．そのため，ここでは，TAU によるプロファイル取得コードの挿入を行わない例について述べる．

図 2 においては，まず，main 関数中のプログラム開始時に，外部から与えられた設定を読み込んで初期化を行う（PomPP_Init()）．ここでは，消費電力・性能情報を取得するための実行であるのか，最適化後の実行であるのかの判断や，計測・制御対象の関数を判別するための情報取

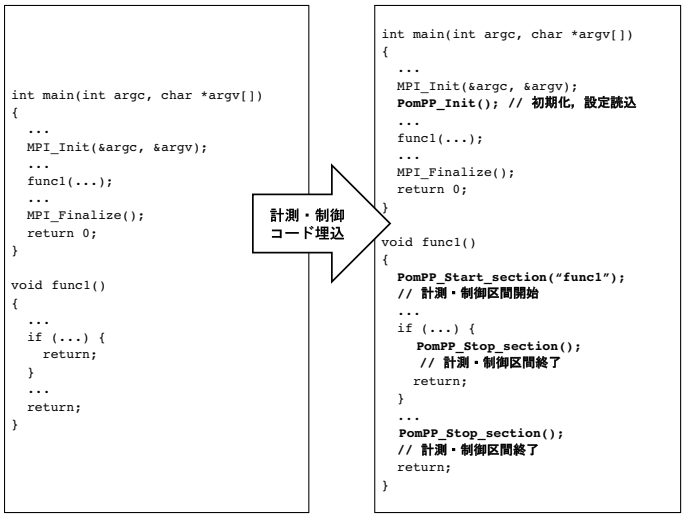

図 2 計測・制御ポイント埋め込みの例（左：埋め込み前，右：埋め込み後）

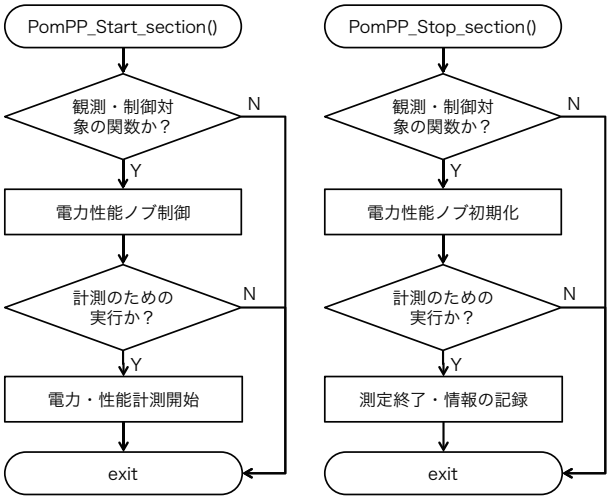

図 3 ライブラリの動作概要

得等が行われる．また，図中の関数（func1）では，関数の開始点に計測あるいは制御を開始するためのライブラリコール（PomPP_Start_section()）が埋め込まれる．初期化時に読み込まれた設定に基づき，実行時間や消費電力の計測，電力性能ノブの制御が行われる．最後に，関数の終了点に計測あるいは電力性能ノブに対する操作を終了するためのライブラリコール（PomPP_Stop_section()）が埋め込まれる．この際，関数の終了点は一意に定まらないことがあるため，関数の出口となりうる全ての点に同様のコードを埋め込む必要がある．

4.3 観測・制御ライブラリ

4.1 節で述べた通り，電力計測および制御のためのコードを挿入されたプログラムは，最適化のために必要な情報を収集する段階と最適化後の実行で共通して用いる．そのため，消費電力および実行性能の計測，および電力性能ノブの操作を行うためのライブラリ内で，機能の切り替えを行う．

情報収集時の電力性能ノブに対する設定は，環境変数を用いて指定することができる．今回実装したプロトタイプでは上述の通り CPU および DRAM を制御対象としており，CPU においては動作周波数を CPUFreq あるいは RAPL を用いて設定し，RAPL を用いて計測対象区間の CPU および DRAM の消費電力を取得する．CPUFreq を用いて動作周波数を設定して計測することにより，動作周波数と消費電力の関係を明確にすることが可能である．また，RAPL を用いることで，消費電力の上限設定が実効性能に及ぼす影響を計測することができる．

計測・制御の対象とする主要関数については，対象関数名をファイルに記述し，このファイルを環境変数で指定することにより指定可能である．ライブラリ内で計測あるいは制御の対象であるかどうかを判断し，対象であれば電力性能ノブの制御や消費電力・実行時間の計測を行う．

図 3 に，今回実装したライブラリの動作概要を示す．観測・制御の開始時と終了時いずれにおいても，対象の関数であるかどうか，情報収集のための実行・最適化後の実行のいずれであるか，を判断している．これをライブラリ側で判断することで，一度観測・制御ポイントを埋め込んで実行ファイルを生成すれば，再コンパイルなどすることなし最適化後も利用可能となる．

また，電力配分最適化の結果は別ファイルとして記述し，初期化時にこれを読み込む形式をとっている．このファイル自体はテキストファイルであることを想定しているため，適宜修正することができる．その場合でも，アプリケーションを再コンパイルする必要は無い．

5. 電力配分最適化手法への適用と性能評価

本節では，実際に 3 つの電力配分手法を本コンパイラ環境により実現し，評価を行った例について述べる．本稿で対象とした電力配分手法はいずれも CPU および DRAM に対して適用するものである．

5.1 評価対象の電力配分最適化手法

本稿では，

- ハードウェアの特性のみを考慮したナイーブな電力配分手法（naïve-1）
- アプリケーション内主要関数の特性を考慮したナイーブな電力配分手法（naïve-2）
- CPU の動作周波数と消費電力の関係を考慮した電力配分手法（3-points）

の 3 種類の電力配分最適化手法を対象として提案コンパイラ環境への適用と評価を行った．以下，5.1.1 節でナイーブな電力配分手法について，5.1.2 節において動作周波数と消費電力の関係を考慮した手法について述べる．

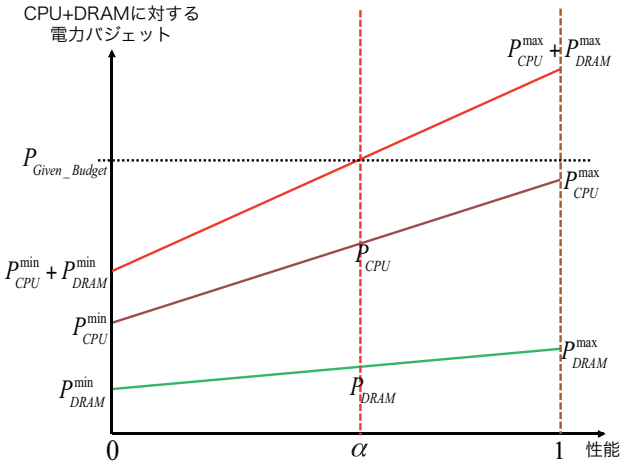

図 4 ナイーブな電力配分手法（naïve-1 および naïve-2）における電力・性能モデル

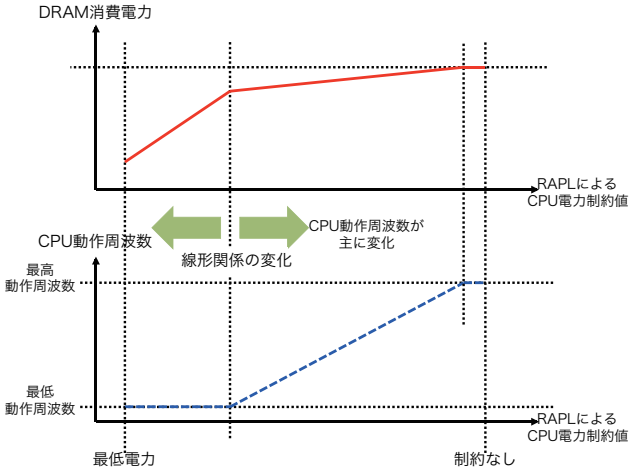

図 5 動作周波数と消費電力の関係を考慮した電力配分手法（3-points）における電力・性能モデル

5.1.1 ナイーブな電力配分手法

本稿において，ナイーブな電力配分手法とは，図 4 のように，CPU あるいは DRAM の性能と消費電力が線形関係にあるモデルを基にした手法であるとする [9]．つまり，与えられた電力バジェットに対して，CPU に配分される電力および DRAM に配分される電力は常に一定の割合を保つ．図 4 において，P^{min}_{CPU} および P^{min}_{DRAM} は対象システムにおける CPU および DRAM の最低消費電力を，P^{max}_{CPU} および P^{max}_{DRAM} は最高消費電力を示す．このモデルでは，P^{min}_{CPU} と P^{max}_{CPU}，P^{min}_{DRAM} と P^{max}_{DRAM} を結んだ直線に基づいてそれぞれの消費電力が推移するものとし，CPU と DRAM の消費電力の合計が与えられた電力バジェット（P_{Given_Budget}）を超えないようにそれぞれの電力配分を決定する．

このモデルにおいて最低消費電力および最大消費電力をどのように考えるかによって，同様のモデルに基づいて下記の 2 通りの手法（naïve-1 および naïve-2）を適用することができる．

ハードウェアの特性のみを考慮したナイーブな電力配分手法（naïve-1）： この手法は，アプリケーションの特性に関わらず CPU および DRAM の消費電力特性の間では一定の関係が保たれることを前提にしたものである．つまり，図 4 において，P^{min}_{DRAM} は対象システムにおいて予め経験的に求められた CPU および DRAM の最低消費電力，P^{max}_{CPU} および P^{max}_{DRAM} はそれぞれの定格電力（TDP）となる．これは HW の構成のみに依存するため，実行するアプリケーション全体に対して同じ電力配分を適用することになる．

アプリケーション内主要関数の特性を考慮したナイーブな電力配分手法（naïve-2）： 上記の naïve-1 では，CPU と DRAM の消費電力の関係は HW にのみ依存するモデルを用いたが，実際は，アプリケーション毎，あるいはアプリケーション内の処理毎に特性が異なる．そこで，この手法では，予め CPU の最低消費電力（P^{min}_{CPU}）を求めておき，CPU の消費電力の上限を P^{min}_{CPU} に制限してアプリケーションを実行した際の DRAM の消費電力を P^{min}_{DRAM} とする．また，消費電力の制限を行わずにアプリケーションを実行した際の CPU・DRAM の消費電力をそれぞれ P^{max}_{CPU}・P^{max}_{DRAM} とする．これにより，アプリケーションの特性に応じて図 4 のモデルを適用することができる．本手法では，これをアプリケーション内の主要関数毎に適用して電力配分を決定する．

5.1.2 動作周波数と消費電力の関係を考慮した電力配分手法（3-points）

実際のシステムでは，5.1.1 節で述べたナイーブな電力配分手法が前提とする消費電力と性能の関係と異なり，CPU 消費電力の制限に対し，その動作周波数が変化することによって消費電力が削減される範囲と，動作周波数以外の要因によって消費電力が変化する範囲とに大別できることがわかっている [23]．

図 5 に本節で述べる手法が前提とする電力・性能モデルを示す．図 5 において，横軸は CPU に対する消費電力制約，縦軸は DRAM の消費電力および CPU の動作周波数である．図 5 上は CPU に対する消費電力制約値を変化させたときの DRAM の消費電力，図 5 下は CPU に対する消費電力制約値を変化させたときの CPU の動作周波数を表している．経験的に，CPU に対する消費電力の制約値が比較的高い範囲では，CPU の消費電力制約に従って動作周波数が変化し，最も動作周波数が低くなる点以降は動作周波数一定のまま，他の要因によって消費電力が削減・制御される．また，それぞれの範囲において，CPU に対する消費電力制約値と DRAM の消費電力は異なる関係にある．

この手法では，上記の関係に基づき，1)CPU の消費電力上限を最低消費電力（P^{min}_{CPU}）に設定した場合，2)CPU の

表 2 評価環境

プロセッサ: Intel Xeon E5-2620	
コア数	6
L1 キャッシュ（コア毎）	32KB 命令，32KB データ
L2 キャッシュ（コア毎）	256KB
L3 キャッシュ（チップ毎）	15MB
マザーボード: SuperMicro X9DRL-iF	
CPU ソケット数	2
DIMM スロット数	8
メモリチャネル数	4
チップセット	Intel C602
LAN	Intel 82574L × 2
メモリ: Century 社製 PC3-12800 DDR3-1600	
容量	16GB × 8 （計 128GB）
レイテンシ	11-11-11
ECC	Registered ECC
OS: CentOS 6.4 64bit（Kernel: 2.6.32）	
コンパイラ	Intel Compiler 13.1.3
数学ライブラリ	Intel MKL 11.0.5

動作周波数を設定可能な最低値に設定した場合，3) 電力制約を適用しない場合，の 3 点において性能・電力に関する情報を取得する．取得した情報を基に図 5 のモデルに当てはめて CPU および DRAM への電力配分を決定する．このようにすることで，情報取得のためにアプリケーションを予め実行する回数を減らしつつ，より効果的に電力配分を最適化できる．

本稿では，上記の情報取得および電力配分最適化を，アプリケーション内の主要関数毎に適用する．

5.2 評価環境

本評価で用いた環境を表 2 に示す．この実験環境において，P_{CPU}^{min} は 20[W]，naïve-1 で用いる P_{DRAM}^{min} は 10[W] であった．

本評価では，CPU および DRAM による消費電力の合計に対し，その上限を 70[W]，60[W]，50[W] と変化させ，アプリケーションの実行時間を比較した．なお評価アプリケーションとして，HPCC ベンチマーク [24] より RandomAccess, STREAM（copy）, STREAM（scale）, STREAM（add），STREAM（triad）および DGEMM を用いた．

各アプリケーションは OpenMP を用いたスレッド並列化が適用されており，CPU1 基上の 6 コアを用いて 6 スレッドでの並列処理を行う．

5.3 評価結果と考察

図 6 に本評価の評価結果を示す．図中，横軸は与えられた電力バジェットおよび評価対象アプリケーション，縦軸は naïve-1 に対する速度向上率である．

naïve-2 では，アプリケーション内の主要関数毎に特性を

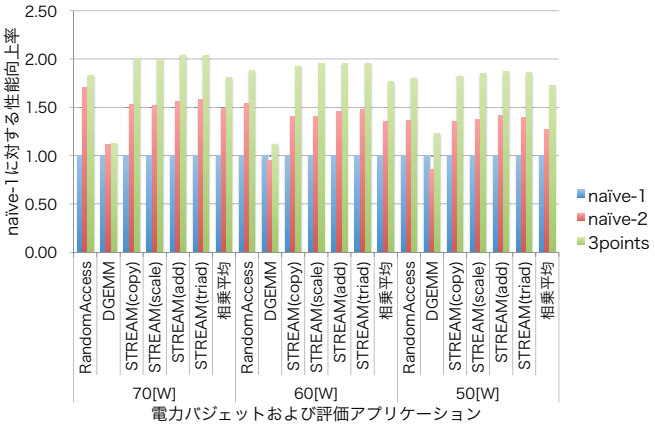

図 6 評価結果

考慮することにより，多くの場合で naïve-1 よりも高い実効性能を示している．対象アプリケーションの相乗平均では，電力バジェット 50[W] において 1.27 倍，60[W] において 1.35 倍，70[W] において 1.49 倍の性能向上が得られており，図 4 のような単純なモデルにおいても，アプリケーション内の特性変化に追従することで，同じ電力バジェットにおいても性能を向上させることができる．

3points は，性能と消費電力の関係が常に一定であると仮定するナイーブな手法と比較して，より実際の性能・消費電力の関係に近いモデルに基づいた手法である．さらに，アプリケーション内の主要関数毎に電力配分を変更することで，同じ電力バジェット内でより高い性能を実現している．対象アプリケーションの相乗平均では，電力バジェット 50[W] において 1.72 倍，60[W] において 1.77 倍，70[W] において 1.81 倍と naïve-1 に対して 2 倍近い性能向上が得られている．ナイーブな手法と比較して性能・電力に関する情報を得るための事前実行を数多く行う必要があるが，本稿で提案・実装したコンパイラ環境を用いることでその多くを自動化することができており，簡単により効果的な電力配分を実現することが可能である．

なお，評価アプリケーション中，DGEMM のみ手法ごとの性能変化が小さいが，これは，CPU に対する負荷が高く電力の消費傾向が偏っており，電力バジェット内での電力配分最適化の自由度が小さいためと考えられる．

6. まとめ

将来の高性能計算機システムでは消費電力が最大の制約になり，性能とともに消費電力を意識したアプリケーション最適化を行うことが重要である．本稿では，与えられた消費電力上限を考慮しつつ，並列アプリケーションの電力性能最適化を行うためのコンパイラフレームワークについて述べた．本稿では，特に CPU および DRAM を消費電力制御・電力配分の対象として選定し，対象アプリケーションに対して実行性能・消費電力の観測や消費電力制御を行

うコードの埋め込みを行うコンパイラ環境，および実際に観測や制御を行うライブラリによってフレームワークのプロトタイプを実装・構築した．この環境を用いることにより，最適化のための消費電力と実行性能に関する情報収集，最適化後の電力配分制御を簡単に行うことができる．

今後の仮題としては，より多くの電力配分最適化手法への対応，大規模なアプリケーションや HPC システムでの評価，GPU やネットワークカード等の電力制御対象への対応等が挙げられる．

謝辞 本研究の一部は JST CREST 研究課題「ポストペタスケールシステムのための電力マネージメントフレームワークの開発」の助成を受け行われた．

参考文献

[1] Top 500 Supercomputer Sites: http://www.top500.org/.
[2] Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., Denneau, M., Franzon, P., Harrod, W., Hiller, J., Karp, S., Keckler, S., Klein, D., Lucas, R., Richards, M., Scarpelli, A., Scott, S., Snavely, A., Sterling, T., Williams, R. S., Yelick, K., Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., Denneau, M., Franzon, P., Harrod, W., Hiller, J., Keckler, S., Klein, D., Kogge, P., Williams, R. S. and Yelick, K.: ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems (2008).
[3] PomPP (POwer Management Framework for Post Peta-Scale Computing) Project: http://www.hal.ipc.i.u-tokyo.ac.jp/research/pompp/.
[4] 石川　裕，丸山直也ほか：HPCI 技術ロードマップ白書 (2012).
[5] 會田　翔，三輪　忍，中村　宏：ロードバランスを考慮した電力制約下における CPU の DVFS 制御，情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2014-HPC-143, No. 23, pp. 1–8 (2014).
[6] David, H., Gorbatov, E., Hanebutte, U. R., Khanna, R. and Le, C.: RAPL: Memory Power Estimation and Capping, *Proceedings of 2010 ACM/IEEE International Symposium on Low-Power Electronics and Design*, pp. 189–194 (2010).
[7] Sarood, O., Langer, A., Kalé, L., Rountree, B. and de Supinski, B.: Optimizing Power Allocation to CPU and Memory Subsystems in Overprovisioned HPC Systems, *Proceedings of 2013 IEEE International Conference on Cluster Computing*, pp. 1–8 (2013).
[8] 吉田匡兵，佐々木広，深沢圭一郎，稲富雄一，上田将嗣，井上弘士，青柳　睦：CPU と主記憶への電力バジェット配分を考慮した HPC アプリケーションの性能評価，情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2013-HPC-141, No. 21, pp. 1–8 (2013).
[9] 稲富雄一，吉田匡兵，深沢圭一郎，上田将嗣，青柳　睦，井上弘士：電力指向型次世代スーパーコンピュータを想定した HPC アプリ ケーションの性能最適化 ～ 量子化学計算の場合 ～，情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2013-HPC-142, No. 30, pp. 1–6 (2013).
[10] 米澤亮太，會田　翔，三輪　忍，中村　宏：物理メモリの増減による電力制約下での HPC システムの性能向上，情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2014-HPC-143, No. 24, pp. 1–8 (2014).
[11] Komoda, T., Hayashi, S., Nakada, T., Miwa, S. and Nakamura, H.: Power Capping of CPU-GPU Heterogeneous Systems Through Coordinating DVFS and Task Mapping, *Proceedings of 2013 IEEE 31st International Conference on Computer Design*, pp. 349–356 (2013).
[12] 會田　翔，三輪　忍，中村　宏：電力制約下における CPU とネットワークの電力制御協調手法，情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2013-HPC-140, No. 1, pp. 1–8 (2013).
[13] 酒井崇至，薦田登志矢，三輪　忍，中村　宏：電力制約下における蓄電池を用いた HPC システムの性能向上，情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2014-HPC-143, No. 25, pp. 1–6 (2014).
[14] Rotem, E., Naveh, A., Rajwan, D., Ananthakrishnan, A. and Weissmann, E.: Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge, *IEEE Micro*, Vol. 32, No. 2, pp. 20–27 (2012).
[15] Intel Corporation: Intel® 64 and IA-32 Architectures Software Developer's Manual (2013).
[16] カオタン，和田康孝，近藤正章，本多弘樹：RAPL インタフェースを用いた HPC システムの消費電力モデリングと電力評価，情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2013-HPC-141, No. 20, pp. 1–8 (2013).
[17] Ishimatsu, Y.: Memory Hotplug, LinuxCon Japan 2013 (2013).
[18] NVIDIA Corporation: *NVML Reference Manual* (2014).
[19] Shende, S. S. and Malony, A. D.: The TAU Parallel Performance System, *International Journal of High Performance Computing Applications*, Vol. 20, No. 2, pp. 287–331 (2006).
[20] Lindlan, K. A., Cuny, J., Malony, A. D., Shende, S., Mohr, B., Rasmussen, C. and Rivenburgh, R.: A Tool Framework for Static and Dynamic Analysis of Object-Oriented Software with Templates, *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, p. 49 (2000).
[21] Geimer, M., Wolf, F., Wylie, B. J. N., Ábrahám, E., Becker, D. and Mohr, B.: The Scalasca Performance Toolset Architecture, *Concurrency and Computation: Practice and Experience*, Vol. 22, No. 6, pp. 702–719 (2010).
[22] Knüpfer, A., Rössel, C., an Mey, D., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A., Nagel, W. E., Oleynik, Y., Philippen, P., Saviankou, P., Schmidl, D., Shende, S., Tschüter, R., Wagner, M., Wesarg, B. and Wolf, F.: Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir, *Proceedings of the 5th International Workshop on Parallel Tools for High Performance Computing*, pp. 79–91 (2011).
[23] 稲富雄一，深沢圭一郎，井上弘士：電力制約下における分子積分プログラムの性能最適化，日本コンピュータ化学会 2014 年春季年会， No. 2P08 (2014).
[24] HPC Challenge Benchmark: http://icl.cs.utk.edu/hpcc/index.html.