非均一分散環境における並列性の仮想化

平 澤 将 一[†1,†2] 本 多 弘 樹[†1,†2]

マルチコア CPU，クラスタ，グリッドなど，並列実行環境がますます広がりを見せている．共有メモリから分散メモリ，均一 CPU コアから非均一 CPU コアへの広がりが システムアーキテクチャを複雑化させている．各並列実行環境でそれぞれ別のプログラミングインタフェース，プログラミングモデルが用いられており，ユーザに負担となるためそれら並列実行環境の普及を妨げている．本稿では，これらのシステムアーキテクチャをメモリ資源，CPU 資源で統一的に扱う実行モデルを探ることにより非均一で分散メモリ型のシステムアーキテクチャを対象とする統一的なプログラミングインタフェース，プログラミングモデルの実現方法および問題点の考察を行う．

Parallelism abstraction of distributed heterogeneous computing eivironment

Shoichi Hirasawa[†1,†2] and Hiroki Honda[†1,†2]

Parallel execution environment, such as the multi-core CPU, a cluster, and a grid, shows a spread increasingly. A spread from a shared memory to the distributed memory and the homogenious multi-core CPU to the heterogenious multi-core CPU has made system architecture complicate. The respectively different programming interface and the programming model are used in each parallel execution environment, and since it becomes a burden to users and prevents these parallel execution environment spread. In this paper, we consider distributed memory and hetero-genious architecture as a general programming model and consider how to realize a unified programming interface for the distributed heterogenious system architecture.

1. はじめに

中央演算装置 (CPU) の目覚ましい発展によりかつてないほどの計算能力を個人が享受できるようになった．一方計算機システムとしてはかつてないほど多岐にわたるシステムが登場している．それにはグリッド，クラスタ，GPU を計算に用いる GPGPU，Cell プロセッサなどの非均一マルチプロセッサ (以下，MP)，ノート PC ですら当然のように内蔵されている均一マルチプロセッサなどが含まれる．それぞれの環境に対して，その環境の持つ能力を十分に発揮させることは重要であり，そのためのプログラミングインタフェースが用意されている．

しかしながら，新しい環境においてユーザが計算を行いたいと考えた場合には，各環境で異なるプログラミングインタフェースでユーザが実行しようとするプログラムを再実装する必要がある．ユーザが当該プログラミングインタフェースに習熟していない場合には学習する必要が生じる．ユーザが本来行いたい計算とプログラミングインタフェースは無関係であり，これはユーザには無意味な負担である．過剰な負担はさまざまな並列実行環境に対するユーザ離れの原因となり，並列実行環境の普及を阻害する．

一方で，新たな並列化言語 (Fortress[1)] など) も登場しているが，一部の例外を除いて広く普及しているとは言えない．以上から，本稿ではさまざまな並列実行環境において統一的に用いることができるプログラミングインタフェースの実現可能性について考察を行う．

2. 発散，収束していくシステムアーキテクチャ

専用システムや組み込み用途を除いて CPU アーキテクチャの種類が減少する一方で，廉価で高速なネットワーク環境の普及および，消費電力の問題により単一 CPU の性能向上が限定的になってきた結果以下のようなさまざまなシステムアーキテクチャが登場した．

均一 MP 熱，消費電力の問題が顕在化したことにより複雑な単一 CPU コアを高速化する手法にかわって単純な CPU コア複数を同一チップに同梱させるマルチコアプロセッサが普及している．各コアはバスを通じて共有するメモリを持ち，同一種の演算コアである CPU コアを複数持つ．

非均一 MP 汎用で複雑な CPU コア複数を同一チッ

†1 電気通信大学
The University of Electro-Communications
†2 独立行政法人科学技術振興機構, CREST

	メモリ	演算速度
グリッド	分散	非均一
クラスタ	分散	均一
アクセラレータ	分散	非均一
非均一 MP	分散	非均一
均一 MP	共有	均一

表 1 システムアーキテクチャごとのメモリシステムおよび CPU 時間

プに同梱させる均一 MP に対して，非均一な CPU コアを同一チップに同梱させるプロセッサが登場している．代表的なものとして Cell BE が挙げられる．各 CPU コアでは命令セットが異なり，バスなどを通してアクセスすることができる共有メモリを持たない分散メモリ型アーキテクチャである．

アクセラレータ 非均一 MP が同一チップ内で異なる命令セットの CPU コアが協調するアーキテクチャであるのに対し， チップ外の PCI Excpress バスなどのバスを通して CPU と協調し，SIMD 型の命令処理を得意とするアクセラレータが登場している．代表的なものとして ClearSpeed 社製アクセラレータボードや GPU を汎用計算に用いる GPGPU が挙げられる．アクセラレータに含まれる演算装置は CPU とは異なるアーキテクチャであるため非均一で，共有メモリを持たない．

クラスタ 多数の計算機ノードを専用計算機よりは低速だが廉価なネットワークで接続することにより協調するシステム．計算機ノードは同種で，メモリは各ノードに分散されアドレス空間を共有しない．

グリッド LAN 内で接続されるクラスタに対し，WAN でノードを接続することにより協調するシステム．ノードは非均一で，それぞれアドレス空間が独立している．

計算機の資源としてもっとも重要な要素としてメモリと CPU 時間が挙げられる．各システムアーキテクチャごとにメモリが共有か分散か，演算速度が均一か非均一かという観点から各システムは表 1 のようにまとめることができる．

ここで，共有メモリは分散メモリの特殊形，均一の CPU 時間は非均一の CPU 時間の特殊形であると考えることが可能である．つまり，物理的，論理的に離れた位置にメモリ空間が存在する分散メモリ型システムアーキテクチャのメモリ空間数が 1 である場合が共有メモリである．また，非均一の CPU 時間を持つシステムアーキテクチャの各 CPU 時間が同一であった場合が均一の CPU 時間を持つシステムアーキテクチャであると考えることができるからである．

この見方に立てば，メモリおよび CPU 時間がともに特殊形である均一型の共有メモリ型アーキテクチャである均一 MP が各システムアーキテクチャでもっとも特殊化されたアーキテクチャであり，メモリを共有型から分散型へと一般化したシステムアーキテクチャがクラスタ，さらに CPU 時間を均一型から非均一型へ一般化したシステムアーキテクチャが非均一 MP，アクセラレータ，グリッドであると考えることが可能である．

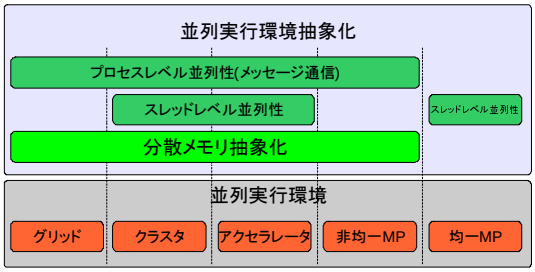

図 1 各並列実行環境と，それぞれが持つメモリ資源の仮想化および，CPU 資源の並列性の関係

以上から，分散メモリ型で非均一なシステムアーキテクチャが一般形であり，多様化していくように見えたシステムアーキテクチャは一般形に収束していると考えることが可能である．分散メモリ型で非均一なシステムアーキテクチャを対象とする実行モデルを仮定することにより，統一的なプログラミングインタフェースを実現することが可能となる．

統一的なプログラミングインタフェースが実現されれば，対象とするシステムアーキテクチャごとに異なるプログラミングインタフェースを用いることを強いられているユーザの負担を軽減することができる．システムアーキテクチャに関しても，現在の共有メモリ型で均一の逐次計算を実行モデルとするプログラムをより高速に実行させるかわりに，明示的に記述された並列性を高速，高効率に実行することに注力できる．これによりさらなるシステムアーキテクチャの高速化，高効率化が達成できる．

もっとも重要なシステムの資源には

- メモリ
- CPU 時間

があげられる．図 1 にあるように，各並列実行環境に対して共有メモリ型システムの均一 MP 以外は分散メモリシステムの仮想化を行い，共有メモリを用いた並列システムにおいてはスレッドレベルの並列性，分散メモリを用いた並列システムにおいてはプロセスレベルの並列性をメッセージ通信を用いることで仮想化を行うことでこれらのシステムの実行モデルを記述することが可能となる．

ここで，クラスタシステムおよびアクセラレータシステムは分散メモリ型アーキテクチャとして扱っているが，分散メモリで結ばれば各ノード内で共有メモリを用いた並列性が存在するためスレッドレベル並列性を用いることが可能であるとしている．

3. 実行モデルとプログラミングインタフェース

さまざまなシステムアーキテクチャに対して統一的に用いることができるプログラミングインタフェースは統一的な実行モデルを仮定し，その上で効率的な実行を可能とするプログラミングインタフェースであることが必要である．言い換えるならば，単一メモリ上で逐次計算を行うという，フォンノイマン型アーキテクチャを具現化した現在の実行モデルを一般化し，分散メモリ上で並列計算を行う新たな実行モデルを明確化した上で効率的なプログラムの実行を可能とするプログラムインタフェースを定めることが必要である．

本稿で仮定する実行モデルには以下の特徴がある．

異なる命令セット　命令セットの異なる CPU，アクセラレータが混在する．

分散メモリ　論理的に別空間となるメモリが存在する．但し共有メモリは分散メモリの特殊形とし，これらが階層型に混在することがある．

非均一の CPU 時間　計算速度が非均一の CPU，アクセラレータが混在する．

以下，それぞれの特徴をいかに抽象化，仮想化するか考察する．

3.1 命令セット

異なる命令セットでの呼び出し方の現状は以下の通りとなる．

(1) コンパイル済みオブジェクトをファイルシステムからハンドルまたはコンテクストとしてロードする
(2) ハンドルを用いて実行を開始する
(3) ハンドルを用いて実行終了を待つ

別の命令セット，別のメモリアドレス空間で実行するため同一のプログラムバイナリ内にリンクできず，単純な関数呼び出しとして実現ができない．しかしながら，上記呼び出し手順は下記のような一般的なスレッドの呼び出し，実行手順と共通であり，一般性が高いと考えられる．

(1) スレッド構造体 (またはオブジェクト) を生成する
(2) スレッドの実行を開始する
(3) スレッドの実行終了を待つ

スレッドと同一，または共通性の高い上記呼出し手順を実現するためには，異なる命令セットを持つ各実行ユニット (CPU，アクセラレータなど) を統合するため以下のような手段が考えられる．

仮想機械　プログラムバイナリを仮想機械上の中間言語として統一する方法．

Fat Binary　複数の実行イメージを同一バイナリ中にリンクせず同梱する方法．

いずれの方法も，異なる命令セットを持つ実行ユニットの命令列を統一的に扱うことができるが，仮想機械を用いた方法は性能オーバヘッドが，Fat Binary を用いた方法は扱う必要がある命令セット数が増えるにつれバイナリの容量が増加するという問題が考えられる．

3.2 メモリの仮想化

現在，CPU と OS の協調による仮想アドレス機構により物理メモリは仮想メモリとして仮想化されている．分散メモリ型のシステム上にソフトウェアにより共有メモリを実現するソフトウェア分散共有メモリ方式[2] も実現されているが，その性能は必ずしもシステム，アプリケーションに最適とはならない．

分散メモリ型システムにおいてはメッセージ通信方式が広く用いられているが，主に動作ノード数が一定のシステムが想定されている．仮想メモリ機構により，メモリモジュールを足すだけでより広いメモリ空間が活用でき自由度が高い．分散メモリ型システムにおいても同様に，アプリケーションの実装によらず分散メモリ，共有メモリを活用できる仮想化が必要である．分散メモリ型システムにはメッセージ通信方式が適応できるが，ノード数にかかわらない通信を行う仮想化が必要である．

メッセージ通信　分散メモリ型システムに対応する．アプリケーションレベルでノード数に依らない実行モデル化が必要である．また，通信の各ノードの中では共有メモリシステムが動いている可能性がある．

共有メモリ　共有メモリ型システムに対応する．共有変数を用いることができるスレッドレベルの並列性を持つシステムのモデル化が可能．共有メモリ型システムの上位層では上記メッセージ通信システムが動いている可能性があり，これを含めたモデル化が必要である．

3.3 並列性の仮想化

仮想メモリ機構により，メモリシステムではモジュールを足すだけで同一の仮想アドレス空間に対してより多くの物理メモリ資源を活用することができる．スレッド，プロセスなどの仮想化もそれぞれ物理的な実行にマッピングされるものの，それらの並列実行を統一的に扱う仮想化は存在していない．

メモリに対するスレッド，プロセスの違いはメモリ空間を共有しているか，していないかである．計算機システムにおける計算はデータとそれをもとに計算を行う命令が実行されることであるが，現在は CPU 時間をメモリに結びつけるモデル化がなされていない．共有メモリを用いた処理の並列性と分散メモリを用いた処理の並列性はそれぞれのメモリシステムの扱いが異なるためそれぞれを別に扱うことが必要である．

また，命令数をもとに暗黙的な計算の割り振りを行う方法は，OS によるプロセス，スレッドのスケジューリングに依存するため計算資源を明示的に確保できない．最終的にプログラミングを行うユーザに対して計算資源の明示的な確保を行うインタフェースを開放する必要はないが，非均一な計算環境においてはシステ

ムレベルで計算資源を明示的に確保することにより実行性能，実行効率を増大させることが可能となる．

本章では，並列性の仮想化を達成するための方法について考察する．

3.3.1 スレッドレベル並列性とプロセスレベル並列性の融合

メッセージ通信インタフェースの MPI とスレッドレベルの並列性を用いる OpenMP の融合[3] だけではすべての並列性に対応できない．これらのインタフェースは均一のマルチプロセッサを前提としたインタフェースであり，また命令セットが異なる演算ユニットが混在することを想定していない．

しかしながら，プロセスレベルの並列性を活用し分散メモリ型システムにおける並列実行を可能とするメッセージ通信インタフェースと，共有メモリを活用することによりスレッドレベルの並列性を活用する OpenMP のようなインタフェースの融合は有用であると考えられる．それは，異なるメモリ階層をまたがった並列計算を行うことができるため実行環境のモデルとして用いることができるためである．

3.3.2 明示的な CPU 時間の確保

現在の一般的なプログラミングモデルは CPU 時間を独占できるという実行モデルに基づいたプログラミングモデルである．もちろん，実際にはプロセス，スレッドという形で抽象化が行われ OS によってプリエンプションされるが，プログラミングモデルとしては CPU 時間を抽象的に扱うことができるプログラミングモデル，実行モデルとはなっていない．

メモリに関しては，多くの言語，実行システムにおいてゴミ集め (GC) が幅広く用いられているがシステムアーキテクチャとしては明示的に確保，解放が行われている．計算機にとってもっとも重要な資源であるメモリと CPU 時間でメモリのみ明示的な操作に対応できる．

均一の命令セットを持ち，実行時間も均一であったシステムにおいては暗黙的に実行時間を想定し，全体のプログラムを構成することが可能であった．しかし，多種の命令セットによるバイナリが混在し，実行時間が非均一なシステムにおいては明示的にこれらを指定して実行するモデルが有用であると考えられる．

明示的に CPU 時間を確保する場合に必要となる情報には以下のような要素がある．どの要素も，指定を省略することによりデフォルトの動作が存在し，プログラムの最低限の動作を保証するものが考えられる．

命令セット 明示的にアーキテクチャを指定可能．

優先度 プロセス，スレッドの優先度と同様，スケジューリングの際の優先度を表す．

用いるメモリ 分散メモリ，共有メモリ，フットプリントなどを表す．

実行時間 実行する時間を指定する．尺度には絶対時間 (秒) やクロック，相対時間，電力，エネルギーなどを指定する．

プログラムは実行開始時にデフォルトの CPU 資源のコンテクストを持ち，これを用いて実行する．そのため明示的に CPU 資源を確保せずに書かれたプログラムについてはデフォルトの動作をする．明示的に CPU 資源を確保するよう記述されたプログラムは，実行システムに存在する CPU 資源，メモリ環境を最大限に用いて実行できるよう，システムソフトウェアで最適化を行う．

4. おわりに

本稿では，発散していくシステムアーキテクチャに対して，最小公倍数的ではあるが一般性の高いシステムアーキテクチャを考えることにより統一的な実行モデルが存在することを示した．これは，分散メモリでかつ非均一な並列実行環境に一般化できるものである．

統一的な実行モデルに対しては，異なる命令セットの実行ユニットの統合的扱い，分散メモリと共有メモリの一般化，非均一な CPU 時間を明示的に扱うことが可能である．これにより現在の並列性記述方式と比較して一般性の高い統一的な実行モデルに基づく API が実現できる可能性を提唱した．

統一的な実行モデルを仮定し，それを前提とした API を用いたシステムソフトウェアおよびアプリケーションが普及することによりさらに CPU アーキテクチャ，システムアーキテクチャの自由度が高くなる．これにより，さらに高性能な CPU アーキテクチャ，システムアーキテクチャの登場が期待でき，ユーザはより高い計算機性能を享受できるようになる．

参 考 文 献

1) Guy Steele. Parallel programming and parallel abstractions in fortress. In *PACT '05: Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques*, p. 157, Washington, DC, USA, 2005. IEEE Computer Society.

2) Daniel J. Scales, Kourosh Gharachorloo, and Chandramohan A. Thekkath. Shasta: a low overhead, software-only approach for supporting fine-grain shared memory. In *ASPLOS-VII: Proceedings of the seventh international conference on Architectural support for programming languages and operating systems*, pp. 174–185, New York, NY, USA, 1996. ACM Press.

3) Franck Cappello and Daniel Etiemble. MPI versus MPI+OpenMP on the IBM SP for the NAS Benchmarks. Vol.00, p.12, Los Alamitos, CA, USA, 2000. IEEE Computer Society.