高知工科大学における HPC 基盤の導入と将来展望
－ハードウェア性能とソフトウェア生産性の追求－

松崎公紀 [1,*]，山際伸一 [1,2]

（受領日：2011 年 4 月 11 日）

[1] 高知工科大学　情報学群
〒 782-8502　高知県香美市土佐山田町宮ノ口 185
[2]JST さきがけ

E-mail: *matsuzaki.kiminori@kochi-tech.ac.jp

要約：本学情報学群では、平成 23 年度に NVIDIA 社の GPU Tesla を 2 つ搭載する計算機ノード 16 ノードからなる GPU クラスタと合計 256 コア以上からなる CPU クラスタとを導入し、ハイパフォーマンスコンピューティング（HPC）拠点としてのスタートを切ろうとしている。特に研究面では、世界トップクラスの HPC 研究拠点を目指した 2 つの研究プロジェクトが昨年度始動したところである。

本稿では、まず、スーパーコンピュータの歴史の概観を通じて HPC 研究の重要性を確認する。続いて、ハードウェア性能の抽出とソフトウェア生産性の向上という本学の HPC 研究における 2 つの中心軸について、それぞれ最近の研究成果を紹介する。さらに、中四国の HPC 拠点として、研究と教育の両面について将来展望を述べる。

1. はじめに

日本において、ハイパフォーマンスコンピューティング (HPC) 分野の研究・開発は旧帝大を中心とした関東・近畿地方に集中しているのが現状である。特に、中四国地方には、これまで大規模な計算に耐えうる HPC 基盤はほぼなかったと言える。そのため、比較的小規模な計算資源で対処するか、関東・近畿地方の計算リソースをレンタルするという、どちらかといえば消極的なやり方で研究が行われてきた。

情報学群では平成 23 年度に、NVIDIA 社の GPU Tesla を 2 つ搭載する計算機 16 ノードからなる GPU クラスタと、合計 256 コア以上からなる CPU クラスタとを導入することとなった。これらの計算クラスタによる HPC 基盤の導入は、著者らの目指す世界トップレベルの HPC 研究拠点に向けた活動を促進するものと期待される。また、これらの HPC 基盤は、高度 ICT 分野において今後需要が見込まれる HPC スペシャリストの育成にも大きな役割を担うものである。本稿では、HPC 研究について著者らの最近の成果を紹介し、さらに、これらの HPC 基盤を利用した研究・教育の将来展望について述べる。

本稿の構成は以下の通りである。第 1 節の残りの部分では、HPC 研究の重要性をスーパーコンピュータの歴史を交え確認する。第 2 節では、高性能並列計算機の中で使われている技術について説明し、その高いハードウェア性能を抽出して得られる研究成果について述べる。第 3 節では、並列プログラムのソフトウェア開発の観点から現在の問題点と近年の研究動向を説明し、正しさの保証された並列プログラミング手法に関する研究成果を述べる。最後に第 4 節では、本学への HPC 基盤の導入をふまえ、今後の研究・教育に関する将来展望を述べ、本稿のまとめとする。

1.1. HPC 研究の重要性

昨年、事業仕分けに取りざたされ、その存在意義を国民に投げかけた「スーパーコンピュータ」という言葉が日本を駆け巡ったのは記憶に新しいと思う。スーパーコンピュータに対する事業仕分けの結果に対して学会から声明が出されるなど、注目を集めたが、残念ながらマスコミ等では「一体スーパーコンピュータとは何だろうか？また、なぜ必要なのだろうか？」というような突っ込んだところの議論がなされることがなかったように思われる。まず、これに対してハイパフォーマンスコンピューティング (HPC) と呼ばれる高性能計算技法を研究する研究者として意見を述べる。

近年、日本はかつての高度経済成長期のような高い成長率を示しておらず、また不景気感の広がりから、日本の成長が止まったとか、日本の成長率が他国に追い越されたとかの声を聞くことが多い。広い分野をまたいだ議論についてはその専門家や政治家に譲るが、HPC 分野の研究・開発について言えば日本はごく近年まで世界トップを争ってきたことは事実である。

HPC とは、ひとことで言うと、非常に巨大な計算力を提供することである。では、ここで言う計算力とは何だろうか。計算力は、目に見えない力であり一般的な生活の中では感じ取ることができないものであるが、間接的に科学の発展をサポートするものである。計算機科学における最も有名な賞であるチューリング賞受賞者の Fran Allen 氏は講演 [7] の中で、「非常に巨大な計算力は、社会や環境が直面するグランドチャレンジを解決してきた」と述べている。

実際、自然科学・社会科学の各分野において計算力により問題を解いていく計算○○学と呼ばれる分野[1]が発展してきている。その代表的なものは、計算化学、計算物理学、計算経済学であろう。危険な化合物を実際作ることなくコンピュータ上で新たな物質を設計したり、リニアモーターカーに使われる超伝導物質の仕組みを計算機上のモデルをもとに解明したり、経済の動向を計算機上のシミュレーションにより予測し経済危機を避けるなどである。これらの、いずれも新たな特許の獲得や国力の増加にも密接に関係することが可能となっているのは、実は計算力のたまものである。

現在、米国が全世界の約半分の計算力を持っている。近年急速に計算力の向上に取り組んでいる中国がそれに続き 13%の計算力を持つ。さらに、日本・フランス・ドイツがそれぞれ 6–7%程度となっている [5]。この計算力を自国で作ることを放棄し、購入するなど外国に頼ることは危険な考え方である。上述したとおり、これらの大規模な計算力は英知を生み出す計算をするため、問題を解くための土台として計算力を他国に差し出す国はまずないであろう。国力の増加に繋がる研究・開発を支える計算力は自国で研究・開発しなければならないのである。

計算機自体は何も生まない。しかし、研究者が生きている間に価値ある結果を得る。そのために手元の超高速計算リソースを独占利用して問題に取り組むことは、スーパーコンピュータを持つことの多大なる恩恵なのである。そして、国際間での競争に打ち勝つにあたり一番の計算機を持つことは、それだけメリットがあるのである。

[1]計算機科学 Computer Science に対して、これらの分野を Computational Science と呼ぶこともある。

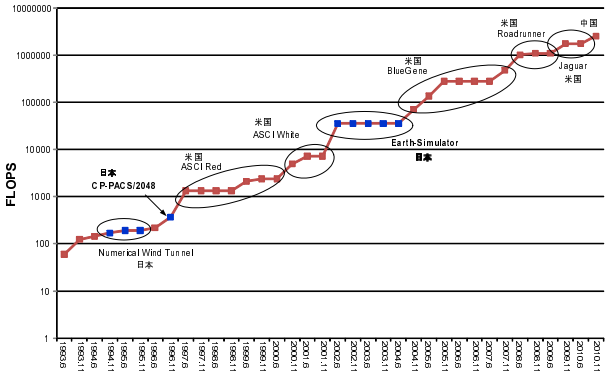


図 1　世界最速スーパーコンピュータの遷移

1.2. スーパーコンピュータの歴史

スーパーコンピュータは Top500[5] と呼ばれるランキングによって、世界最高性能を競われている。このランキングは線形代数の演算ライブラリの処理性能、より詳しくは Linpack[2] と呼ばれるライブラリの実行最高性能を 1 秒間あたりの浮動小数点数演算数 (FLOPS 値) で表したもので競われる。

Top500 ランキングは 1993 年の International Supercomputer Conference で初めて披露された。毎年 6 月と 11 月の 2 回、最新ランキングが発表される。このランキングで一位をとることが国力を示すことにつながっている。

「世界で一番速い」という栄光を勝ち取った計算機をみてみる。図 1 には Top500 が記録してきた世界で最も速い計算機の性能を縦軸に、発表年月を横軸にとり、現在までの推移を示す。縦軸は 10 を基数とする対数表示である。

1993 年から 2010 年までの 17 年間で 4 万倍もの性能向上を示している。特に、ここ 5 年間での性能の向上は著しい。2010 年 11 月での最高性能は 2.5 ペタ FLOPS (ペタは 10^{12}) である。

ランキング初期のスパコンは主に日本が独占していた。富士通は Numerical Wind Tunnel と呼ばれる計算機で、連続の 1 位を獲得していた。また、筑波大学はそれに続き、日立製作所とともに開発した CP-PACS で 1 位を獲得している。その後、ASCI Red/Blue/White と呼ばれるマシンが台頭してきた。これには実は米国の国家戦略があり、ASCI プロジェクトと呼ばれるスパコン開発のためのプロジェクトを新進していたためである。

ASCI (Accelerated Strategic Computing Initiative) プロジェクトは米国エネルギー省の関連研究所を中心に 1995 年に開始され、10 年間で延べ、1400 億円を投入し 100 テラ FLOPS (テラは 10^9) を目指した。ASCI Red は 1997 年から 2000 年まで、ASCI White

は 2000 年から 1 年、ランキングのトップを走り続けた。しかし、最終的には 10 テラ FLOPS ほどの性能を示す ASCI White と呼ばれる計算機でプロジェクトは終焉を迎えた。

ASCI プロジェクトに続けと、日本も 1997 年から 5 年間で 400 億円を投入し地球シミュレータ計画を実施した。地球シミュレータは地球の環境に関する全ての状況をシミュレーションしようという試みであった。2002 年 6 月から 2004 年 6 月までの間、地球シミュレータが 1 位をキープしている。

2004 年後半からは IBM の時代がやってくる。地球シミュレータが 40 テラ FLOPS を達成することを国家挙げてのプロジェクトとして実施していたにも関わらず、IBM は一気にこれまでの倍の性能を達成する BlueGene をリリースし、2007 年までトップを独占した。その勢いは止まらず、ストリームプロセッサと呼ばれる新型の計算機構を備えた Cell プロセッサ [22] を採用した RoadRunner が IBM により開発され、2009 年までその地位を譲らなかった。しかし、IBM のトップからの衰退は突然訪れる。

Oak Ridge National Laboratory に導入された Jaguar がトップの地位をさらった。Jaguar は RoadRunner のようなストリームプロセッシングという新たな計算手法を導入することなくこれまでの Von Neumann スタイルのプロセッサを 25 万個並べることでトップを奪った。そして、この計算機は Cray[2] というスーパーコンピュータの老舗が開発したことはとても涙ぐましい。

そして、現在、ストリームコンピューティングは GPU が主体となり、その性能は後述するように著しく向上している。これを利用した中国の計算機が現在のトップであるというのは、経済大国に迫る勢いの中国の現在を反映している。

地球シミュレータを最後として、日本のスーパーコンピュータは上位から姿を消した。しかし、東工大の TSUBAME が巻き返しを図っている。現在 4 位である TSUBAME は 4200 個の GPU を備え、超大規模計算をターゲットに稼働している。

また、2012 年の稼働を目指す、事業仕分けの対象にもなった、理化学研究所が富士通とともに神戸に建設中の「京」は世界初の 10 テラ FLOPS を達成することが期待されている。64 万プロセッサという人類が体験したことのない並列度を達成するために、そのハードウェアのみならず、システムソフトウェアからアプリーケーションまで、チャレンジングなプラットフォームになることは確かであろう。このような超大規模並列計算環境におけるソフトウェア技術は日本の研究戦略目標としても閣議決定されており、科学技術振興機構の CREST の課題としても取り上げられている。

2. パフォーマンスの追求

2.1. 計算機の起源と計算力の指標

HPC の歴史はコンピュータの歴史でもある。コンピュータはチューリングマシンと呼ばれる、状態をテープに記録された信号の連続により変更していく仮想的な機械によって、概念がもたらされ、それを Von Neumann により現代のコンピュータの形に完成させたわけであるが、その行く先は性能への挑戦であった。

1946 年の ENIAC と呼ばれる世界最初の汎用コンピュータの出現から、65 年にも及ぶ歳月の間、人類は飽くなき計算力を求め続けている。しかし、人類の最初の計算力の対象は、戦争であった。爆弾の弾道計算であった。当時の歴史的背景を見れば、この事実は理解できるわけだが今日のような平和利用が第一となる時代ではなかったと言える。

計算力を競う基点となったのは残念ながら核爆弾の開発であったのは否めない。仮想的な核実験をコンピュータという箱の中で実行することで、人的、環境的ダメージをもたらさず、最強の核兵器を作成しようというのは冷戦がもたらした HPC の最高技術ではある。この技術を平和利用する方向へシフトしていくのはその後のことでであった。日本がその主導権を握ることができたことは今日の技術大国となるための要因であったとも言える。

HPC 技術を競う上で、「計算力が向上した」「どのコンピュータが速い」といった性能を比較する際には指標が必要となる。よく使われる指標として、MIPS (ミップス) と FLOPS (フロップス) が挙げられる。MIPS (Mega Instruction Per Second) は 1 秒間に実行できる命令数を 10^6 を単位として比較する。1 MIPS は 1 秒間に 100 万命令を実行できることを示す。1977 年に DEC 社 (現在の Compaq 社) が開発した VAX 11/780 は 1MIPS マシンとして基準となる性能を示していたため、この計算機の性能を基準に比較が行われた。当時、Dhrystone や SPEC と呼ばれる性能を計測するためのベンチマークソフトウェアにより、MIPS 値を計測することが主流であった。MIPS が性能の単位として用いられる時代は整数演算が主流であった。

しかし、今日では、性能はアプリケーションにより異なり、さらには浮動小数点演算による高精度な計算

[2] Cray はその創始者の名前を社名にとっているが、彼は 1996 年に不運にも Jaguar の勇姿を見ることなく、自動車事故で死去してしまった。

が行われるのが主流となったため、FLOPS (FLoating point number Operations Per Second) が使われるようになる。FLOPS は 1 秒間に浮動小数点演算が何命令実行できるかを示す指標である。この指標は現代のスーパーコンピュータの性能を計測する指標となっている。

2.2. 計算機性能を左右するテクノロジー

高い計算力を維持するため、スーパーコンピュータのハードウェアとソフトウェアには特殊な技術が取り入れられていることは想像できると思う。しかし、これらの技術は、その価格性能比が市場の満足する時になれば、万人の手元に降りてくるのが歴史背景である。F1 カーで使われている技術が民間の自動車でも使われ始めるのに似ている。

では、そのような高速化技術はどのような観点で開発されているか説明する。以下の 3 つの観点が主なものであり、これらをアプリーケーションの特性であったり、ハードウェアの新技術にあわせる形で、最高速をめざしている。

2.2.1. プロセッサ性能

プロセッサ性能は計算の心臓部でもあるが、これを高速化することはプロセッサと同じくらいの歴史があり、今でも多数のベンダがしのぎを削る。プロセッサの歴史の初期の段階では大学や研究所がその舞台であった。プロセッサがどれだけ多くの命令を実行できるか、また、メモリをどれだけ高速に読み書きできるか、といった技術がアカデミックの場から提案されてきた。例えば、1 命令あたりのクロック数を減らすためにパイプライン処理が発明され、とてもうまくいくと 1 クロックサイクルあたり 1 命令実行できる。また、この処理パイプラインを複数組み合わせたスーパースカラーと呼ばれる並列実行の方法も提案された。これらの技術は現在のプロセッサに搭載されることもあり、その基盤となる技術として君臨している。このようにプログラムを構成する命令列をどれだけ速く処理できるかを、どれだけ多くの命令を並列実行するかの最適化法で競う方法は ILP (Instruction Level Parallelism) とよばれる。しかし、ILP の追求はハードウェアの複雑化を伴い、クロック周波数の向上に困難をきたしていた [30]。

そこで、プロセッサのハードウェアをいくつも 1 つの LSI として集積させてしまうマルチコア技術が発明され、近年の高速化手法として、発展している。この手法は、プログラムを複数、同時実行する方法に考えを移行したため TLP (Thread Level Parallelism) と呼ばれている。現在のマルチコア技術は 8 個のプロセッサコアを集積するものが市場にも出回り Corei7 といった製品で簡単に手に入れることが可能となっている。TLP の発展はその後のストリームコンピューティング方式に発展し、GPU を代表する超並列プロセッサを輩出している。GPU では、ストリームプロセッサと呼ばれるデータストリームを順次処理するプロセッサを数百という数で集積している。これは、計算方式が超並列に向けたものであるため大規模な集積が可能となっている。GPU のようなストリームコンピューティング方式を導入し多数のコアを持つようなプロセッサをメニーコアアーキテクチャとも呼ぶ [24]。

2.2.2. ネットワーク性能

TLP を追求すると、マルチコアプロセッサで動作するプログラム同士が話し合いをすることでタイミングを合わせプログラムを実行していく。また、プログラム実行の中でデータを交換する必要があるが (例えば、計算データの分配と結果の収集など)、これはプロセッサ間の通信性能が全体の性能を左右することになる。そこで、ネットワークを最適化し、プロセッサ間でいかに高速にデータ交換ができるかが計算性能を左右することになる [41]。通信ハードウェアの研究は進み、現在では Ethernet は 10Gbps を達成することができ、さらに導線での通信から光ファイバーをつかったテラバイト通信が可能となってきた。例えば、Infiniband[1] と呼ばれる規格が策定され、40Gbps を達成するネットワークが市場に投入されている。

計算力を競うスーパーコンピュータの通信は、インターネットで使われるような遠距離通信を保証するような技術ではなく、超高速にプロセッサ間でデータ交換可能な技術が必要となる。つまり、セキュリティや過剰なデータの正しさを保証する技術は無駄なものであり、それらをそぎ落とした技術が用いられる。例えば、異なるプロセッサのメモリからネットワークが自律的にデータを送受信するゼロコピー通信 [36, 31, 40] と呼ばれる技術が用いられている。ゼロコピー通信は送信元と送信先のメモリを直接操作するため、データの機密性は低くなるが、広域通信で用いられる TCP/IP のような無駄にデータをコピーしてさらに細切れにして送受信する方式と異なり、送受信するデータを全くコピーすることなく転送することが可能であり、現代の多くの構成の計算機で用いられている。

2.2.3. 制御ソフトウェア

複数のプロセッサコアで同時に実行されるプログラムのことをスレッドと呼ぶことにする。前述の通りスレッド間ではデータを共有する、初期値を配る、結果を集める、といったデータ交換が必要となる。このとき、スレッド間での通信はそのプログラムコードで制御しなければならない。この制御に関しては、2 つの方

式がある。一つは通信を明に指定するメッセージパッシング方式で、スレッド間で交換されるデータの量、送り先、またそのタイミングを正確に指定する。メッセージパッシング方式は細かな通信制御ができるのではあるが、プログラムの可読性が悪くなるのと、異なるスーパーコンピュータにプログラムを持ち込むと、指定した通信タイミングがとたんにずれてしまい、意図した高速化ができないという問題がある。その一方で、明示的に通信タイミングを記述しなくてよい共有メモリ方式と呼ばれるものがある。こちらは、全てのプロセッサコアが共通してアクセス可能な共有領域を介してデータを交換する方式である。この方式は、プログラムの見通しはとてもよくなる反面、「共有」の領域をつくるためのハードウェア、または、ソフトウェアでのサポートが必要になる。

並列に実行されるプログラムの記述性に関しては、スーパーコンピュータの歴史と同じくらい議論され続け、共有メモリによる解決方法を必死に探っていた時期もあった。しかし、共有メモリはハードウェアにより実装しても、その低いスケーラビリティ (つまり、プロセッサコアを増やしていっても、プロセッサギブのネットワークアクセス遅延などにより、メモリのアクセス許容量に限界があり、それに反比例する形でメモリへのアクセス遅延が増えていく) によって、プロセッサコア数を向上できないのが問題となっていた。そこに、マルチコア技術が出現し、共有メモリの機構を 8 コア程度と少ないコア数ではあるが、共有メモリプログラムをプロセッサ単体で実行できるようになってきた。この背景によって、2 つのソフトウェア技術が開発され、現在、スーパーコンピュータを始め、分散処理の世界で用いられている。

メッセージパッシングに対しては、MPI (Message Passing Interface)[19, 3] と呼ばれる通信ライブラリ標準が策定され、ほとんどのスーパーコンピュータがこのライブラリを備えている。従って、通信部分を MPI ライブラリを使ったプログラムはどのスーパーコンピュータでも動くことになる。MPI は実装が階層化されており、最下位層の通信機構をソフトウェアで取り替えることで、チューニングしたり異なる仕様に変更することができる。また、共有メモリに関しては、マルチコアプロセッサを対象にした OpenMP[4] と呼ばれるコンパイラ仕様が策定され、フリーの GCC であってもサポートしている。OpenMP はプログラム中の独立実行性をプログラマから明に指定することで、コンパイラにヒントを与え、複数のスレッドを自動生成し、並列化する。この際、データは全てのプロセッサから共通に参照可能な共有空間にあるものとしてコンパイルする。したがって、マルチコアアーキテクチャにおける外部メモリがこの共有メモリと見なすことができる。しかし、OpenMP はマルチコアプロセッサ外部までの通信は面倒見てくれないため、OpenMP と MPI を組み合わせて並列プログラムを書くことで、可読性を少しでも高くしようと試みているプログラマも多い。

以上のスーパーコンピュータに用いられる基礎技術を駆使することで高い計算力を得る努力が続けられている。どの分野も終わりのない性能追求の研究課題である。そこで、多くの研究機関ではどのようなサイズで、どのような構成の計算環境がこれまで開発されてきたのか、解説する。

2.3. 現代の高速計算機

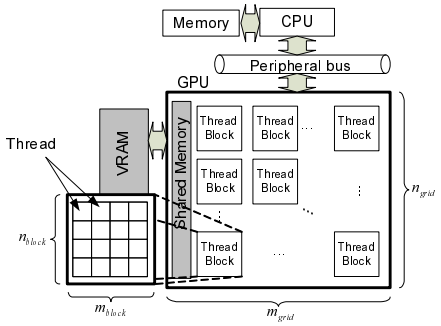
計算機の歴史は、汎用機と呼ばれる種類のものから始まり、ミニコン (汎用機に比べて小さいという意味だが、ものすごく大きい) へと続き、DEC 社の PDP-11 で UNIX が誕生し、計算機の一般利用に拍車がかかった。

現代の計算機の主流であるパーソナルコンピュータ (PC) と呼ばれているものは、IBM が策定した PC/AT 互換と呼ばれる規格に準じており、この仕様に準じる MS-DOS により一世風靡し、現在に至る。しかし、この用途は、計算性能と言うよりも共通の使い勝手を追求した。しかし、スーパーコンピュータにはこのような標準というものがなく、独自に発展している。

しかし、近年、CPU 性能は急激な成長を遂げ、また、価格がかなり下がり、複数個所有するのは容易なことになった。これを利用したのが PC クラスタと呼ばれる、PC を Ethernet のような、これまで単につなぐだけにしか使っていなかったネットワークを使い、並列計算機に見なした構成である。この構成は爆発的に普及し、現代でも 1 研究室に 1 台といったような普及を見せる。この技術変遷により、HPC というものが普及したことは確かである。

CPU はマルチコアになり、マルチコアのプロセッサをつかった PC クラスタが現在では主流になった。さらに、スーパーコンピュータでもこの構成を用いることがコストパフォーマンスを向上する最善策であるとして、採用されることが多くなった。

そして、GPU の出現により、一桁高い性能を実現することができるようになってきた。近年の CPU の最高性能は数十ギガ FLOPS ではあるが、GPU は 1 テラ FLOPS に近づこうとしている。つまり、約十数倍の性能の差がひらき、この先、それはもっと顕著になると考えられている。



(a) Execution model of kernel threads in CUDA

```
int main(){
  float *A, *B, *C;
  ...
  dim3 dimBlock( m_block, n_block );
  dim3 dimGrid( m_grid , n_grid );
  KernelFunc<<< dimGrid , dimBlock >>>(A, B, C);
  ...
}

__global__ void KernelFunc(float *a, float *b, float *c){
  int i = blockDim.x * blockDim.y *
          (gridDim.x * blockIdx.y + blockIdx.x) + threadIdx.x;
  c[i] = a[i] + b[i];
}
```

(b) Example CUDA code for array summation

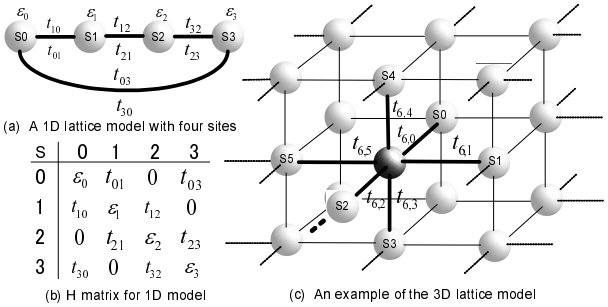
図 2 (a) GPU の構成と (b) CUDA のカーネルプログラム

2.4. GPU の台頭

GPU は元々、グラフィックの補助的なハードウェアであった。従って、PCI Express と呼ばれる CPU の周辺バスに接続されている。つまり、CPU は GPU を操作することは可能であるが、プログラムの実行にはデータの移行など、多くの手間を必要とする。しかし、GPU はそのオーバヘッドを打ち消す性能を保持しているため、多くの数値計算に用いられている [29]。

GPU の技術を広く知らしめ、応用の発展に貢献したのは NVIDIA 社が提供する CUDA[27] の出現である。CUDA はカーネルと呼ばれる特殊な関数を記述することで、GPU 上でのプログラムを記述する。カーネルはストリームコンピューティングのスタイルに従って実行されるものと見なし、プログラムを記述する必要がある。またカーネルに入力されてくるデータストリームのうちそれぞれの要素に関して、カーネルの操作が適用される。その際にカーネルのプログラムは全てのストリームプロセッサで異なるデータを対象に動作する。このカーネルの実行単位をスレッドと呼ぶ。スレッドの数を制御するのは図 2(a) に示すようなモデルである。カーネルは CPU が GPU にダウンロードすることで実行を促される。その際にスレッド数も設定する。最近の GPU の同時実行可能なスレッド数は数百という単位であるから、GPU のチップの中で超並列な実行環境が実現されている。

カーネルは図 2(b) のような構成を取る。このプログラムは 2 つのデータストリームの和を計算し出力する。担当するデータのインデックスを得て、データの位置を指定しているのがわかる。このように、各カーネルは担当する入力ストリームのインデックスを使い、入力データを特定することで、また、出力データストリームのインデックスを決定し、担当する出力ストリーム中の 1 要素を計算している。



(a) A 1D lattice model with four sites

S	0	1	2	3
0	ε_0	t_{01}	0	t_{03}
1	t_{10}	ε_1	t_{12}	0
2	0	t_{21}	ε_2	t_{23}
3	t_{30}	0	t_{32}	ε_3

(b) H matrix for 1D model

(c) An example of the 3D lattice model

図 3 (a) 1 次元と (c) 3 次元の原子モデルのシステム図、そして、(a) のハミルトン行列の様子 (b)

以上のように、GPU は超並列なプログラム実行が行える上に、その計算は各操作が 1 クロックで実行してしまうように設計されているため、高い性能を維持できる。今後の HPC の動向としては、この GPU を各プロセッサに抱き合わせることで、それをネットワークで接続した GPU クラスタが主流になると考えられ、実際、東工大の TSUBAME をはじめとするスーパーコンピュータがその構成をとっている。

2.5. 計算性能の応用

計算力を使った最前線の研究として物性物理学の応用を紹介する。本研究は本学の情報学群山際研究室と理化学研究所の柚木計算物性物理研究室の共同研究で推進しているものである。

物性物理学では物質構造を解析することで、新たな物性を発見し、その物性を持った物質を設計するマテリアルデザインと呼ばれる分野がある。例えば、カーボンナノチューブの構造解析や、強磁性体、そして、高温超伝導物質を設計するための基礎を原子構造をシミュレーションすることで、新たな物性を探る [28]。

図 3(a) には原子を 1 次元に並べた例を示している。この例では、それぞれの原子からの電子の移動を示した相関を t_{ij} で示している。この相関を図 3(b) のようなハミルトン疎行列に表し、これを対角化することでエネルギー状態を求める。この行列の大きさはシミュレーションする原子の個数を行と列に設定したサイズとなるため、図 3(c) に示すような 3 次元構造を持つ物質を対象にする場合は、その行列サイズは爆発することになる。例えば一辺を 100 個の原子からなる場合、1000000×1000000 の行列を対角化することになるた

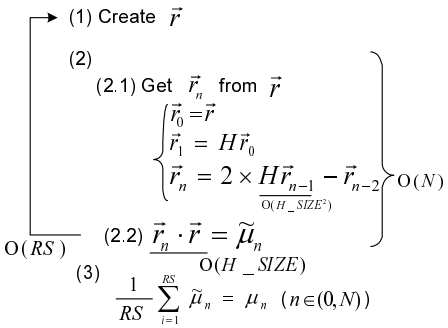


図 4 KPM のアルゴリズムと複雑さ

め、膨大な計算量が必要になる。

密行列を単純に対角化する場合、行または列の長さを H_SIZE とすると、計算の複雑さは $O(H\_SIZE^3)$ となるためこの計算量を減らす手法が必要になる。ここで使う行列の特徴して、疎行列であることを利用し、さらに近似的に対角化する方法がある。Kernel Polynomial Method (KPM)[38] と呼ばれる方法が近似的に対角化をすることで、さらに疎行列を対象にすることを利用し、$O(H\_SIZE)$ まで計算の複雑さを減らすことができる手法である。

KPM の計算ステップを簡略化したものを図 4 に示す。前述の行列 H はランダム生成されたベクトル $\vec{r}$ とハミルトン行列 H を掛け合わせることで、再帰的にベクトル $\vec{r}_n$ を求める操作を繰り返し実行する。この計算は直前の計算で得られた結果を使って次の計算をすることから、並列化が難しく、全てのプロセッサにベクトル $\vec{r}_k$ を再配分する必要があり、CPU のようなマルチコア環境においては通信を隠蔽することが困難となり、高い性能を維持することができない。そこで、この部分を含め、このアルゴリズム全体を GPU で実装することで、GPU の中で再帰的計算を閉じ込め、ストリームプロセッサで高い並列度を維持することで性能を向上することを狙うことを試みた [42]。

ハミルトン行列 H を大きくしていったときの暫定的な性能評価によると、図 5 に示すように、CPU 単体に比べ GPU (NVIDIA C2050) の約 4 倍近くの性能向上が見られる。つまり、GPU を使うと約 4 倍速い性能が得られることが確認できる。ここで注目すべき点は実行時間である。たったの 4096 個の原子をシミュレーションするだけで CPU では 7 時間ほどかかるのが読み取れる。すなわち、要素数の増大によって爆発的に計算時間が増加していくのがわかる。このような強烈なシミュレーション時間が必要なアプリケーションに GPU を適用することは高速化が期待できるため、有効である。

さらに、情報学群山際研究室が導入した GPU クラスタシステム (図 6) で、より洗練されたアルゴリズムを適用し、16 個の C2050 を使って、R、S に関して単純に並列化すると、図 7 に示すように、CPU コアを 32 個使った並列化に比べ約 11 倍の性能を得ることができる。

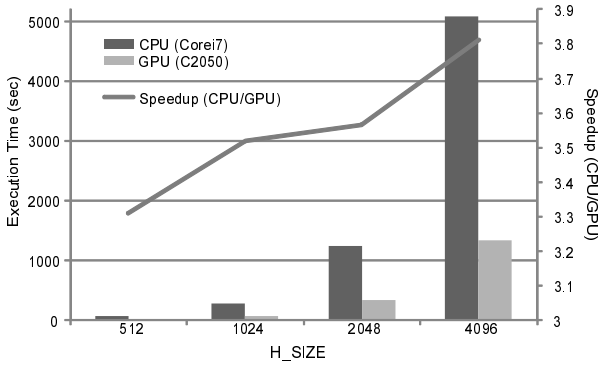


図 5 ハミルトン行列 H を増加させ、C2050 の KPM の性能 (棒グラフ) と CPU 単体性能に対する高速化度合 (線グラフ)。

すなわち、ここで問題となっている強相関格子系の物性物理シミュレーションを CPU によって実行するよりも、GPU の潜在計算力を引き出すことで 11 倍速く新しい状態の発見が可能であることを示しており、より多くの新たな物理理論を見つけることができることを示している。

以上のように、計算機の最適な構成を追求していくことで計算力を引き出す研究は日々、躍進しており、この潮流に乗ることが新たな発見をもたらすことになる。従って、一番を目指すスーパーコンピュータを構築することが国家としての明暗を分けると言っても過言ではない。

3. 並列プログラムの生産性向上の追究

3.1. 並列プログラミングの現状

スーパーコンピュータに限らず、並列計算機を効率良く動かすには、そのためのソフトウェア、すなわち、適切な並列プログラムが必要である。しかし、そのような並列プログラムを作成することは、これまで一般的であった逐次プログラムを作成することよりも複雑で難しい作業である。なぜならば、逐次プログラミングにおいて考えていたことに加えて、並列アルゴリズム、計算資源やデータの分散、プロセスやスレッド間の通信や同期、などのことを考慮する必要があるからである。

これまで、並列プログラム開発は大きく次の 3 つの方法で行われてきた。

図 6 平成 22 年度に本学に導入した GPU クラスタ。NVIDIA 社 C2050 を 16 個搭載し、ノード間を 40Gbps の Infiniband でつなぐ。

- 低レベルの並列性の直接的なコーディング
- 特定分野の並列ライブラリの利用
- 共有メモリモデルの上でのプログラム

現在最も広く用いられている方法は、プロセスやスレッド間の通信や同期といった低レベルな並列性を明示的にコーディングする手法である。この手法では、対象となる環境に応じてプログラミングを行う必要がある。例えば、PC クラスタなどの環境では、プロセス間通信を記述できる MPI (Message Passing Interface) が良く用いられている。また、GPU を用いた汎用計算 (GPGPU) では、CUDA などの専用のライブラリが用いられることが多い。低レベルな並列性を指定することにより、適切にチューニングを行うとハードウェアの持つ性能を引き出すことができる。しかし、そのような低レベルな並列性を意識したプログラミングは非常に困難である。また、並列計算機環境間で移植性がないという問題も起こってしまう。

ハイパフォーマンスコンピューティングが特に必要とされる数値計算の分野では、大規模な行列の演算がその計算の大部分を占めている。そのため、そのような行列演算を提供する数値計算ライブラリ (BLAS や LaPACK) について、さまざまな並列計算機環境向けにチューニングされたものが開発されている [39]。そのようなチューニングされたライブラリを用いると、並列計算についての知識がなくても効率の良い並列プログラムを作成することができる。ただし、数値計算分野以外の分野では、そのようなライブラリ開発が行われているのは遺伝子解析 [25] などの非常に限られた分野に限られているのが現状である。

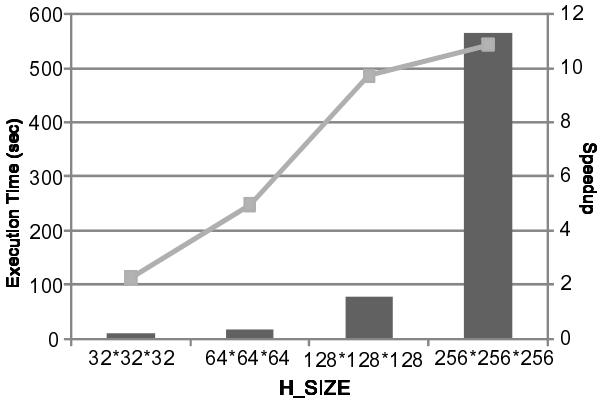


図 7 ハミルトン行列 H を増加させ、C2050x16 のときの KPM の性能 (棒グラフ) と CPU 単体性能に対する高速化度合 (線グラフ)。

それほど知識や経験がない人でも行うことができる並列プログラミング手法として、共有メモリモデル上でのプログラミングがある。共有メモリモデルでは、データ分散やプロセッサ間通信について考慮する必要がないため、比較的容易にプログラムを開発することができる。例えば、OpenMP では、要素ごとの計算が独立である繰り返しや、タスクプールに入れられた多数のタスクを並列実行することができる。これまで、PC クラスタなどの分散メモリ環境ではこの手法だけでは並列化することができなかったが、近年それを可能とするような新しいライブラリとコンパイラの開発も行われている [45]。

3.2. 近年の研究動向：高生産性並列プログラミング

第 2 節で述べた通り、今後、計算機の性能向上において様々な段階での並列性を向上することは避けて通ることができず、したがって、並列プログラムは必須となる。しかし、これまでの並列プログラムの開発手法だけでは、そのような並列プログラムの需要に応じることは非常に困難である。ゆえに、並列プログラム開発の生産性向上が必要となってきている。

実際、並列プログラム開発の生産性向上は世界的なグランドチャレンジとして捉えられている。例えば、米国国防高等研究計画局 (DARPA) は High Productivity Computing Systems (HPCS) プロジェクト (Productivity であり Performance ではない) において新世代のスーパーコンピューティング環境の研究開発を推進してきた。その中で、並列プログラム開発の生産性を向上させるプログラミング言語として、

X10 (IBM 社)[44]、Fortress (Sun Microsystems 社 (当時))[6]、Chapel (Cray 社)[12] などが開発されている。また、Google 社の開発した MapReduce[16] は近年で最も成功した大規模データ処理のためのフレームワークであり、実装 [9, 20] や応用 [13, 14, 33] などの観点でその後も多くの研究開発が続けられている。

これらの新しい高生産性並列プログラミング言語・フレームワークは次の特徴を持つ。

- 高レベルでの並列性の構造化
- 並列性を自然に記述するための記述法

これまでの MPI (Message Passing Interface) では、プロセス間の通信を直接記述することが必要であった。これに対して、より高レベルな並列性をユーザが記述できるようになっている。例えば、X10 言語では、Partitioned global address space (PGAS) という、分割はされているもののひとつのアドレススペースを共有するというモデルが採用されている [44]。これを利用すると、共有メモリモデルのプログラミングの容易さで分散メモリ環境のプログラミングを行うことができる。MapReduce フレームワークでは、ユーザは 2 種類の逐次的関数を記述するだけで大規模データ処理で扱われれる多くのアプリケーションを記述することができる。この考え方は、松崎が参加している研究グループで研究してきたスケルトン並列プログラミング [32] と共通するものである。

これまでのプログラミング言語では、そのプログラムが動作するハードウェアの制約から逐次的に処理を記述するようになっており、現在でもそれがプログラミングの基本となっている。しかし、自然に並列性を記述することができることがこれからの並列プログラミングにおいて重要になってくるものと考える。これについて挑戦的な取り組みを行っているプログラミング言語が Fortress[6] である。Fortress では、for 文や組の各要素の計算が暗黙的に並列に処理されるようそのセマンティクスが定められており、逐次的に処理をする必要がある場合は明示的にそのように指定しなければならない。近年、関数プログラミングの考え方がスクリプト言語を中心によく議論されるようになっているが、その先にはこのような並列性を自然に記述できるようにするための方法が出てくるだろうと考えている。松崎らは、このような並列性を自然に記述するための方法についての研究のひとつとして、「並列性忘却プログラミング」[43] についての研究も進めている。

3.3. 正しさの保証された並列プログラムの導出

第 3.1 節で述べたように、並列プログラムを開発するためには、逐次プログラムを開発するよりも考慮すべき点が多い。したがって、並列プログラムを正しくバグなく作ることは逐次プログラム以上に難しいことになる。並列プログラムを開発する際に、そのプログラムの正しさの保証することができれば、並列プログラム開発の生産性が大幅に向上することが期待される。

ここでは、国立情報学研究所 (NII) の胡振江氏、フランスのオルレアン大学の Frédéric Loulergue 氏らとの共同研究で得られた、正しさの保証が可能な並列プログラムの導出についての研究成果 [18] について、その概要を示す。本研究は、良く研究されてきた並列計算モデルのひとつである Bulk Synchronous Parallel (BSP)[35] による並列プログラミングについて、その正しさを保証することができるような系統的プログラミング手法を与えるものである。とくに、定理証明系 Coq[8] を用いてプログラムを導出することにより、正しさの証明ができることが特色である。本研究の重要の貢献は次の 3 つである。

- 並列計算モデル BSP において扱われる計算を抽象化した新しい計算パターン「BSP リスト準同型」を提案したこと。
- ユーザに使いやすい仕様記述法を提供し、そこから BSP リスト準同型を導出するための定理を Coq 上のライブラリとして与えたこと。
- BSP リスト準同型の実装を与え、その実装の正しさを Coq により証明したこと。

3.3.1. BSP リスト準同型とそれに関する定理

理論的な研究成果について抜粋して説明する。

リストに対する並列計算パターンの 1 つとして、リスト準同型 [34] がある。

定義 1 (リスト準同型) ある関数 f と結合的な二項演算子 $\odot$ が存在して関数 h が次の形で再帰的に定義できるとき、関数 h はリスト準同型であると言う。

$$\begin{aligned} h([a]) &= f(a) \\ h(x +\!\!+ y) &= h(x) \odot h(y) \end{aligned}$$

ただし、$+\!\!+$ は 2 つのリストを結合する演算である。このとき、$h = \mathsf{Hom}(f, \odot)$ と記述する。 □

リスト準同型はその定義から、分割統治法による並列計算と親和性が高い。そのため、高レベル並列プログラミングの分野において、リスト準同型に基づく並列プログラムの導出法についての研究が多く行われてきた [15, 23, 26]。

本研究で新しく提案した BSP リスト準同型は、「データ間の依存関係 (プロセッサ間通信) を明示的に記述することができる」という BSP の特徴をとらえて、リスト準同型を拡張したものである。

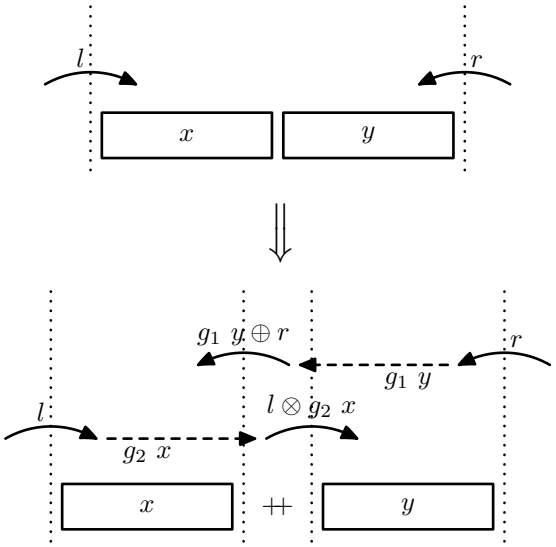


図 8 BSP リスト準同型の計算

定義 2 (BSP リスト準同型) 関数 k、リスト準同型 g_1 と g_2、および、結合的な二項演算子 $\oplus$ と $\otimes$ があるとする。関数 bh が次のように定義されるとき、関数 bh は BSP リスト準同型であると呼ぶ。

$$bh([a], l, r) = [k(a, l, r)]$$
$$bh(x +\!\!+ y, l, r)$$
$$= bh(x, l, g_1(y) \oplus r) +\!\!+ bh(y, l \otimes g_2(x), r)$$

このとき $bh = \mathsf{BH}(k, (g_1, \oplus), (g_2, \otimes))$ と記述する。□

図 8 は、BSP リスト準同型の計算がどのようなものかを図示したものである。

BSP リスト準同型は、その分割統治法による計算の定義から、BSP 計算モデルを含む様々な計算機環境で効率的に実現することができる。また、BSP 計算モデルで扱うことができる計算の多くをこの BSP リスト準同型によって定式化できる。しかし、BSP リスト準同型では、再帰関数で渡される引数 l や r の更新としてデータ間の依存関係を陽に記述する必要がある。そのため、BSP リスト準同型を用いてプログラムを開発することはそれほど容易であるとは言えない。この問題を解決するために、ユーザが仕様記述を行うための別の計算パターン *mapAround* を定義する。

定義 3 (*mapAround* 関数) 関数 *mapAround* は次のように定義される。

$$mapAround(f, [x_1, x_2, \ldots, x_n])$$
$$= [f([], x_1, [x_2, \ldots, x_n]), f([x_1], x_2, [x_3, \ldots, x_n]),$$
$$\ldots, f([x_1, \ldots, x_{n-1}], x_n, [])]$$

□

ここで、パラメタ関数 f は、リストのある要素の値 x、要素の左側の部分リスト l、要素の右側の部分リスト r の 3 つを受け取り、新しい値を $x' = f(l, x, r)$ によって計算する。計算パターン *mapAround* は、このパラメタ関数 f を、リストのすべての要素に対して適用するものである。この計算パターンを素朴に実装すると計算コストが非常に大きくなってしまうものの、この計算パターンを利用してプログラムを開発することは容易である。計算パターンを利用したプログラム開発については、後で具体例を通して見る。

BSP モデルに基づく効率的な並列計算のためのパターンとして BSP リスト準同型を、ユーザにとって書き易い仕様記述のパターンとして *mapAround* 関数をそれぞれ提案した。これら 2 つをうまく結びつける必要がある。我々は、これら 2 つのパターンをつなぐための定理を示した。

定理 4 (BSP リスト準同型の導出) 関数 h が計算パターン *mapAround* を用いて記述できる、すなわち、$h(x) = mapAround(f, x)$ であるとする。このとき、ある関数 k とリスト準同型 g_1 と g_2 が存在して、関数 f が $f(l, x, r) = k(g_1(l), x, g_2(r))$ と分解できたとする。もし、$g_1(x) = p_1(\mathsf{Hom}(h_1, \oplus))$、および、$g_2(x) = p_2(\mathsf{Hom}(h_2, \otimes))$ であるならば、関数 h は次のように BSP リスト準同型である。

$$h(x) = \mathsf{BH}(k', (h_2, \otimes), (h_1, \oplus))(x, \iota_\oplus, \iota_\otimes)$$

ただし、$k'(l, x, r) = k(p_1(l), x, p_2(r))$ であり、$\iota_\oplus$ と $\iota_\otimes$ はそれぞれ $\oplus$ と $\otimes$ の単位元である。
証明: 定理証明系 Coq[8] による。Coq を用いた証明はプロジェクトのページ [37] より得られる。 □

ユーザが定義した *mapAround* による仕様がこの定理が求める条件を満たすことを確認できれば、*mapAround* によるプログラムから BSP リスト準同型による効率的な並列プログラムを得ることができる。その条件のうちリスト準同型については、その導出についての多くの研究 [15, 23, 26] を利用することができ、その導出にかかるユーザの負担は少なくて済む。

3.3.2. 具体例：塔建設問題

本手法を用いることでどのように並列プログラムを開発できるかを具体例を通じて紹介する。ここでは、塔建設問題 (Tower Building Problem) を解く並列プログラムを扱う。この問題は、有名な見通し問題 (Line of Sight Problem)[10] の拡張である。

問題 5 (塔建設問題) 地図上のある一直線上に $n + 2$ 個の点 $(x_L, h_L), (x_1, h_1), \ldots, (x_n, h_n), (x_R, h_R)$ が存在する。ただし、x_i は i 番目の点の x_L からの水平距

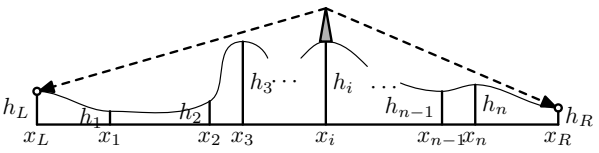


図 9 塔建設問題の問題設定

離、h_i は i 番目の点の高さとする (図 9)。このとき、点 (x_i, h_i) のうち、高さ h の塔を建てたときにその頂点から両端の点 (x_L, h_L) および (x_R, h_R) が見えるような点をすべて求めよ。 □

この問題を解く逐次プログラムを作成することは、(その効率を考えないとすると) それほど難しくはない。しかし、データ間の依存関係が強いため、効率の良い並列プログラムを作成することは難しい。一方、*mapAround* パターンを用いて問題の仕様を記述することは容易である。ある点が塔建設問題の条件を満たすかどうかを判定するには、その左側および右側にそれぞれ視界を遮る障害物がないかどうかを判定すれば良く、*mapAround* 関数に渡すパラメタ関数は次のように定義することができる。

$$f(l, x, r) = \mathit{visibleL}(l, x) \wedge \mathit{visibleR}(r, x)$$
$$\mathit{visibleL}(l, (x_i, h_i)) = \mathit{maxAngleL}(l) < \frac{h + h_i - h_L}{x_i - x_L}$$
$$\mathit{visibleR}(r, (x_i, h_i)) = \mathit{maxAngleR}(r) < \frac{h + h_i - h_R}{x_R - x_i}$$

ただし、*maxAngleL* は左側のリストに対して仰角の最大値を求める関数であり

$$\begin{aligned}&\mathit{maxAngleL}([]) = -\infty\\ &\mathit{maxAngleL}([(x_i, h_i)] +\!\!+\, x)\\ &\quad = \mathit{max}(\tfrac{h_i - h_L}{x_i - x_L}, \mathit{maxAngleL}(x))\end{aligned}$$

と定義される。*maxAngleR* は右側のリストに対して仰角の最大値を求める同様の関数である。

計算パターン *mapAround* を用いて問題の仕様を記述することができたので、それから BSP リスト準同型を導出するための定理を適用できるかどうかを考えれば良い。ここでは、*maxAngleL* および *maxAngleR* 関数に対して、それらがリスト準同型であることを示せば十分である。実際、それらの関数はリスト準同型であり、したがって BSP リスト準同型を導出することができる。そのようにして得られた BSP リスト準同型のプログラムはやや複雑であるためここでは省略する。詳細については文献 [18] を参照されたし。導出において重要なことは、定理証明器 Coq [8] の機能を利用することによって導出の作業を半自動化することができることである。実際、*maxAngleL* 関数で用いる *max* 関数が結合的であることを Coq に伝えることで、BSP リスト準同型の導出に必要な作業は Coq の上でインタラクティブに行うことができる。

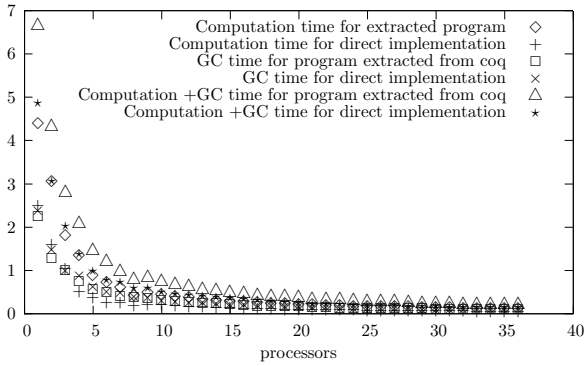
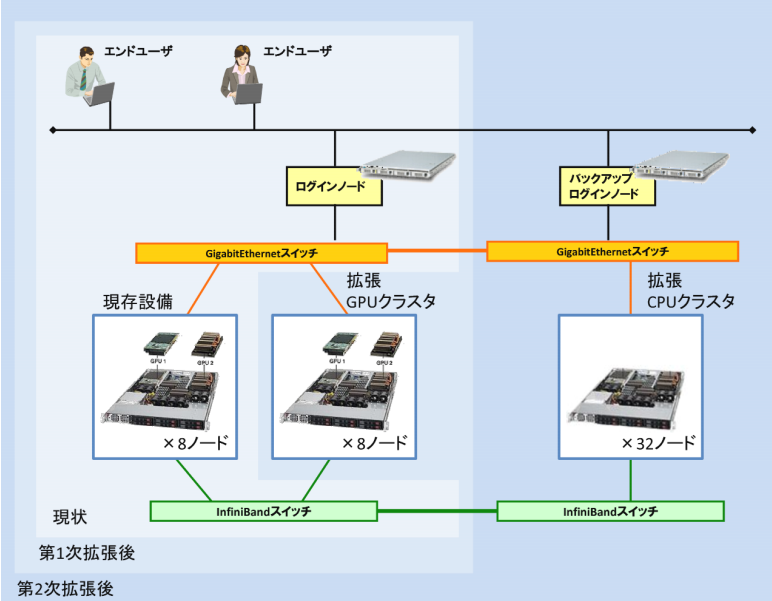


図 10 塔建設問題に対する実験結果

BSP リスト準同型のプログラムを得ることができれば、それを実装したライブラリを用いて BSP リスト準同型のプログラムを並列プログラムとして動かすことができる。我々は、BSML 言語 [17] によって実装した BSP リスト準同型ライブラリを提供している。特に、この BSP リスト準同型ライブラリは、その実装の正しさを定理証明系 Coq により証明しているという特長を持つ。プログラムの導出および BSP リスト準同型ライブラリの実装の両方が定理証明系 Coq によって証明されていることにより、最初の問題の仕様と導出に使うヒントの正しささえ保証できれば、ユーザが得る最終的な並列プログラムもその正しさが保証できることになる。とくに、定理証明系 Coq の場合には、プログラムの正しさの証明から、実際に計算機上で動く (BSML) プログラムを生成することができ、これによって実装の途中でのバグの混入も避けることができる。

最後に、塔建設問題のプログラムを用いた実験の結果を報告する。実験環境には、Intel Xeon E5450 の CPU 2 つと 8GB のメモリを持つブレード PC からなる PC クラスタを用いた。ただし、1CPU あたり 1 コアのみを利用し、36CPU (18 ノード) までの範囲で実験を行った。使用したプログラムは、正しさの証明されたプログラムと得られた BSP リスト準同型プログラムから手実装を行ったものの 2 種類である。実験データは、536 万要素からなるリストを用いた。

図 10 に実験の結果を示す。グラフの横軸は使用したプロセッサ数、縦軸は実行時間である。どちらのプログラムもプロセッサを増やすことにより、実行時間は短くなり、台数効果によるスピードアップが得られている。ただし、Coq により生成されたプログラムは、同じアルゴリズムを手実装したものに比べて 40%程度のオーバーヘッドがある。このオーバーヘッドは、データ取り扱い方などによると推測している。



導入済みクラスタ：8 ノード
CPU: 2 × Intel Dual-Core CPU
Mem: 12GB
GPU: 2 × Tesla M2050
SSD: 80GB

拡張 GPU クラスタ：8 ノード
CPU: 2 × Intel Hexa-Core CPU
Mem: 12GB
GPU: 2 × Tesla M2050
SSD: ≥ 80GB

拡張 CPU クラスタ：≥ 32 ノード
CPU: 2 × Intel Hexa-/Quad-Core CPU
Mem: 12GB
SSD: ≥ 80GB

図 11 HPC 基盤の拡張計画

文献 [18] では、塔建設問題を対象として並列化の議論を行っている。この問題の他にも、配列パッキング問題 (Array Packing Problem)[10]、疎行列ベクトル積、All Nearest Smaller Values 問題 [21] などの問題が、BSP リスト準同型を用いたプログラム導出によって並列化できることを得ている。

3.4. 証明付き並列プログラムに向けて

第 3.3 節で述べた、正しさの保証付きで並列プログラムを導出するという開発手法をさらに発展させるべく、(独) 科学技術振興機構 (JST) の支援により、オルレアン大学 (フランス) との共同研究プロジェクトを進めている。本共同研究では、並列スケルトンと呼ばれる並列計算パターンの考え方と定理証明系を利用することで、正しさの証明された並列プログラムを導出するための手法を開発することを目標としている。

このプロジェクトの中で、日本側 (高知工科大学、東京大学、国立情報学研究所からなるグループ) は、より実践的な高水準並列プログラミングのための定理証明系との親和性の高い並列スケルトンの提案を中心に研究を行っている。これまでは、比較的単純な計算に対してプログラム変換手法の議論が行われてきたが、これをより複雑に組み合わされた場合や実用的なデータ処理に耐えうるレベルでの定式化を行う。具体的には、並列パターンの複雑な組み合わせに対するプログラム変換には、並列言語 NESL[11] で研究されてきたコンパイル技術からそのアイデアを得、また、実用的なレベルでの並列スケルトンの定式化には、MapReduce[16] の根底にある考え方からアイデアを得ることができると期待して研究を行っている。このような、より実践的なレベルにおいて並列計算パターンの抽象化や変換手法を議論することにより、証明付き並列プログラム導出のための新しいフレームワークとして拡張ができると見込んでいる。

4. 将来展望

4.1. 中四国の HPC 拠点となるべく

HPC の研究・開発は、旧帝大を中心とした関東・近畿地方に集中しているのが現状である。他の地域での研究は、それらのリソースをレンタルする形で行われていることが多い。特に、中四国の地域に至っては、大規模な計算科学をサポートするための計算基盤・計算リソースは皆無であると言える。本学を中心とした中四国の計算基盤を構築し、HPC 研究の柱を築く努力を行うことは、基礎研究拠点を築く上での重要な動向であると考える。

本学情報学群では、平成 23 年度に 2 段階での HPC 基盤の拡張を行う予定である (図 11)。第 1 段階の拡張では、GPU Tesla を 2 つ搭載するノード 16 基から構成される GPU クラスタへと拡張する。さらに、第 2 段階では、GPU クラスタに加えて、合計 256 コア以上からなるよりノード数の大きな CPU クラスタを導入する。これらの拡張により、ピーク性能で言うと 18 テラ FLOPS の大規模クラスタが導入されることになる[3]。

これらの並列計算リソースは、まずは、著者らを中心とする研究プロジェクトのための研究基盤として利

[3]参考までに、最新の Top500 における 500 位の計算機の Linpack 性能は 30 テラ FLOPS 程度である。

用されることとなるが、近い将来により広い分野の研究および教育への展開を考えている。最後に、HPC 基盤を利用することによる、HPC 研究拠点の構築および HPC スペシャリスト育成について将来展望を述べ、本稿のまとめとする。

4.2. HPC 研究の 2 つの軸と分野の拡大

著者らは、本学における HPC 研究を世界トップレベルの研究となるものにすべく、2 つの研究プロジェクトを開始している。山際は、(独) 科学技術振興機構 (JST) の支援により、GPU を中心として新たな計算基盤を確立するプロジェクト「高性能ストリーム・コンピューティング環境の構築」を開始している。松崎は、上述の正しさの保証できるソフトウェア開発手法の構築を目指したフランスとの共同研究「並列スケルトンを用いた並列プログラム開発に関する研究」を (独) 科学技術振興機構 (JST) の支援により開始した。これらの 2 つの研究プロジェクトは、本学における HPC 研究の 2 つの重要な軸となるものである。

今後、これらの研究プロジェクトを中心とした誘導のもと HPC 拠点としてさらに展開していくために、HPC 研究の対象となる分野を広げていくことになる。そのような研究分野として可能性の高いものとして、3 つ例を挙げてみる。

- シミュレーション分野のアクセラレーション
- 多量のテキスト・画像・映像データからの知識マイニング
- 生体信号のリアルタイムフィードバックのための高性能計算バックボーン

シミュレーションおよび知識マイニングは、これまでも HPC 分野において応用がなされてきた工学の重要な分野である。本学内においても、これらの技術を要する・望む研究分野の教員・研究者は多いはずである。そのような方との連携を進めることにより、互いの研究に良い刺激と成果をもたらすことが重要な責務となるだろうと考えている。また、3 つ目の研究分野は、教授会における学長の方針演説にもあった「医工連携」を見すえた研究内容である。このように、HPC 基盤を用いて新しい応用を切り開いていくことは、中四国における主たる HPC 拠点のプレゼンスを高めることにつながると考える。

4.3. HPC スペシャリストの育成

HPC リソースの教育への利用について議論する。

並列性の向上と HPC 分野の重要性は今後ますます高くなることが予想され、大学という高等教育の場に HPC を組み込むことは、将来の人的リソースを育成する上で必須の事項と言える。しかし、HPC を題材とした講義を扱うことはとても難しいとよく言われる。HPC の技術の多くは、コンピュータサイエンスの多岐にわたる分野に広くかつ深く関連している。したがって、大学院修士課程以降の学生でやっと受講可能となるが、基礎的な知識を完全に習得していない学生に対しては、必要となる知識を補いながらの教育が必要となる。また、並列プログラムの実践を演習として行うには、そのような目的のため比較的自由に利用できる HPC 環境があることが最低条件となる。言い換えると、座学として理論を説くことも、ましてや、演習を用いた超並列プログラムの実践を行うことも大変難しいことである。しかし、このような難しい科目であっても、アメリカにおける HPC の拠点となる大学 (および日本の一部の上位大学) では、前述の MPI を用いたメッセージパッシングパラダイムでの並列計算の演習講義が行われている。

このような HPC 拠点の大学のみでしか受けられない講義を本学では受講できる、という特色を今後打ち出していくのが責務であろうと考えている。つまり、本学の HPC 基盤の計算リソースを利用することで、四国では皆無と言える HPC プロフェッショナル育成のための講義を行うということである。このような講義を受講可能であることは、高知発の新たな ICT 人材育成に大きな役割を果たすと考えられる。

具体的な内容についての提言を以下に 2 つ挙げる。まず、本学の大学院科目「並列分散処理システム論」が、HPC を教育できる場の 1 つである。昨年の講義では、分散処理に定評のある教科書 [30] を用い並列分散処理の過去の歴史と、最新の GPU プログラミングの教科書 [24] を使って現代の GPU を中心として計算手法を解説した。次回の来年度の講義においては、上述の 256 コア CPU クラスタを使った実践演習が可能であると考えている。次に、学部生への HPC 利用にも挑戦してみようと考えている。前述の OpenMP をつかったコンパイラによる自動並列化技術を「オペレーティングシステム」の講義で体験させる演習を考えている。この演習は現代ではオペレーティングシステムの力を借りた当然の高速化技術となったマルチスレッドを体験することができ、HPC とはどのようなものなのかを、学部の段階から触れさせることができると考えている。

謝辞

本稿を仕上げるにあたり、本学客員研究員でもある (独) 理化学研究所准主任研究員　柚木清司先生に貴重なご意見をいただいた。

本稿の第 3 節で示した研究成果は、(独) 科学技術振興機構「さきがけ」の研究成果の一部である。

文献

(1) Infiniband trade association. http://www.infinibandta.org/.

(2) Netlib Linpack site. http://www.netlib.org/linpack/.

(3) Message passing interface forum. http://www.mpi-forum.org/.

(4) OpenMP homepage. http://openmp.org/.

(5) Top500 supercomputing sites. http://www.top500.org/.

(6) E. Allen, D. Chase, J. Hallett, V. Luchangco, J.-W. Maessen, S. Ryu, G. L. Steele Jr., and S. Tobin-Hochstadt. The Fortress language specification version 1.0, 2008. http://labs.oracle.com/projects/plrg/fortress.pdf.

(7) F. Allen. The challenge of the multicores. 情報処理学会第 72 回全国大会招待講演, 2010.

(8) Y. Berlot and P. Casteran. *Interactive Theorem Proving and Program Development — Coq'art: the Calculus of Inductive Constructions*. Springer Berlin Heidelberg, 2004.

(9) A. Bialecki, M. Cafarella, D. Cutting, and O. O'Malley. Hadoop: A framework for running applications on large clusters built of commodity hardware, 2005. http://lucene.apache.org/hadoop/.

(10) G. E. Blelloch. *Prefix Sums and Their Applications*. Morgan Kaufmann, 1991. Chapter 1 of *Synthesis of Parallel Algorithms*.

(11) G. E. Blelloch. NESL: A nested data-parallel language. Technical Report CMU-CS-95-170, School of Computer Science, Carnegie Mellon University, 1995.

(12) B. L. Chamberlain, D. Callahan, and H. P. Zima. Parallel programmability and the Chapel language. *International Journal of High Performance Computing Applications*, 21(3):291–312, 2007.

(13) H. chih Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker. Map-Reduce-Merge: Simplified relational data processing on large clusters. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pp. 1029–1040, 2007.

(14) C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, and K. Olukotun. Map-Reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems (NIPS 2006)*, pp. 281–288, 2007.

(15) M. Cole. Parallel programming with list homomorphisms. *Parallel Processing Letters*, 5(2):191–203, 1995.

(16) J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI2004), December 6–8, 2004, San Francisco, California, USA*, pp. 137–150, 2004.

(17) L. Gesbert, F. Gava, F. Loulergue, and F. Dabrowski. Bulk synchronous parallel ML with exceptions. *Future Generation Computer Systems*, 26(3):486–490, 2010.

(18) L. Gesbert, Z. Hu, F. Loulergue, K. Matsuzaki, and J. Tesson. Systematic development of correct bulk synchronous parallel programs. In *The 11th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2010)*, pp. 334–340, 2010.

(19) W. Gropp, E. Lusk, and A. Skjellum. *Using MPI*. The MIT Press, 1999.

(20) B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang. Mars: a MapReduce framework on graphics processors. In *17th International Conference on Parallel Architecture and Compilation Techniques (PACT 2008)*, pp. 260–269, 2008.

(21) X. He and C.-H. Huang. Communication efficient BSP algorithm for all nearest smaller values problem. *Journal of Parallel and Distributed Computing*, 61(10):1425–1438, 2001.

(22) H. Hofstee. Power efficient processor architecture and the Cell processor. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 258–262, 2005.

(23) Z. Hu, H. Iwasaki, and M. Takeichi. Formal derivation of efficient parallel programs by construction of list homomorphisms. *ACM Trans-*

actions on Programming Languages and Systems, 19(3):444–461, 1997.

(24) D. B. Kirk and W. mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, 2010.

(25) H. Lin, P. Balaji, R. Poole, C. Sosa, X. Ma, and W. chun Feng. Massively parallel genomic sequence search on the Blue Gene/P architecture. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pp. 33:1–33:11, 2008.

(26) K. Morita, A. Morihata, K. Matsuzaki, Z. Hu, and M. Takeichi. Automatic inversion generates divide-and-conquer parallel programs. In *Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI2007)*, pp. 146–155, 2007.

(27) NVIDIA Corporation. CUDA: Compute unified device architecture programming guide. http://developer.nvidia.com/cuda.

(28) K. Ohno, K. Esfarjani, and Y. Kawazoe. *Computational Materials Science*. Springer, 1999.

(29) J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. In *Eurographics 2005, State of the Art Reports*, pp. 21–51, 2005.

(30) D. A. Patterson and J. L. Hennessy. *Computer Architecture, Fourth Edition: A Quantitative Approach*. Morgan Kaufmann, four edition, 2006.

(31) L. Prylli, B. Tourancheau, and R. Westrelin. Modeling of a high speed network to maximize throughput performance: the experience of BIP over myrinet. In *Proceedings of Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, 1998.

(32) F. A. Rabhi and S. Gorlatch eds. *Patterns and Skeletons for Parallel and Distributed Computing*. Springer, 2002.

(33) A. Sarje and S. Aluru. A MapReduce style framework for computations on trees. In *39th International Conference on Parallel Processing (ICPP 2010)*, pp. 343–352, 2010.

(34) D. B. Skillicorn. *Foundations of Parallel Programming*, Vol. 6 of *Cambridge International Series on Parallel Computation*. Cambridge University Press, 1994.

(35) D. B. Skillicorn, J. M. D. Hill, and W. F. McColl. Questions and answers about BSP. *Scientific Programming*, 6(3):249–274, 1997.

(36) H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato. Pm: An operating system coordinated high performance communication library. In *High-Performance Computing and Networking (HPCN Europe 1997)*, pp. 708–717, 1997.

(37) The SDPP Development Team. Systematic development of parallel programs. https://traclifo.univ-orleans.fr/SDPP/.

(38) A. Weiße, G. Wellein, A. Alvermann, and H. Fehske. The kernel polynomial method. *Review of Modern Physics*, 78(1):275–306, 2006.

(39) R. C. Whaley, A. Petitet, and J. Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001.

(40) S. Yamagiwa, K. Aoki, and K. Wada. Active zero-copy: A performance study of non-deterministic messaging. In *Proceedings of the The 4th International Symposium on Parallel and Distributed Computing*, pp. 325–332, 2005.

(41) S. Yamagiwa, K. Ferreira, K. Aoki, M. Ono, K. Wada, and L. Sousa. Maestro2: Experimental evaluation of communication performance improvement techniques in the link layer. *Journal of Interconnection Networks*, 7(2):295–318, 2006.

(42) S. Zhang, S. Yamagiwa, M. Okumura, and S. Yunoki. Performance acceleration of kernel polynomial method applying graphics processing units. In *IPDPS/APDCM 2011*. IEEE CS, 2011.

(43) 松崎. 並列性忘却プログラミングの研究推進に向けて. 高知工科大学紀要, 7(1):91–100, 2010.

(44) 河内谷. マルチコア時代のプログラミング言語「X10」. 情報処理, 52(3):342–356, 2011.

(45) 李, 朴, 佐藤. 分散メモリ向け並列言語 XcalableMP コンパイラの実装と性能評価. 情報処理学会論文誌：コンピューティングシステム, 3(3):153–165, 2010.

Advanced Challenges on the HPC Platform at KUT
— Exploiting Hardware Performance and Enhancing Software Productivity —

Kiminori Matsuzaki [1,*] and Shinichi Yamagiwa[1,2]

（Received : April 11th, 2011）

[1]Kochi University of Technology
185 Miyanokuchi, Tosayamada-cho, Kami-shi, Kochi 782-8502, JAPAN
[2]JST PRESTO

E-mail: *matsuzaki.kiminori@kochi-tech.ac.jp

Abstract: The School of Information at KUT has begun to act as one of the most important academic institutes for the high performance computing (HPC) research and education, introducing an HPC environment of a 16 node GPU-based cluster where the node embeds two NVIDIA Tesla GPUs, and a CPU-based cluster with no less than 32 nodes that includes 256 or more processor cores. Especially in the research field, two major research projects have accepted in the last year to decide the destiny of our faculty to become an academic institute of the world top class in the HPC field.

This paper discusses importance of research activity for the super computing through the history and the technological overview. Moreover, this paper introduces two successful research examples performed at KUT ; one exploits the hardware potential performance and another enhances productivity of parallel programs of super computer. Finally, this paper draws the future blueprints of HPC research and its education at KUT as a main HPC research institute in the Chu-Shikoku area in Japan.