

コンパイラ技術の重要性を再認識しよう

図書館情報大学 中田 育男

コンパイラ技術は古いか

今はインターネットの時代であるのに、日本はネットワークの分野で遅れているから、大学の情報系の学科でもネットワークの教育をちゃんとやって欲しい、という産業界からの要請は分からぬわけではない。しかし、それに続いて、コンパイラのような古いものは教育しなくてもいいから、とまで言う人が情報処理学会を担っている方々の中にもいることは理解できない。

コンパイラは確かに古い。コンピュータというものが生まれて間もなくの、1950年代の後半からコンパイラが開発されるようになり、その後多くの研究・開発が積み重ねられてきている。コンパイラを構成する基本要素のうち、字句解析と構文解析などの前半の部分については、その技術は1970年代にほぼ確立し、それ以後あまり大きな変化はない。その意味では確かに古い。

しかし、コンパイラの後半の部分、特に最適化に関しては、最初のFortranコンパイラから始まって、現在に至るまで研究・開発が続けられているし、今後も続けられることは間違いない。たとえば、最適化技術の基本要素となるプログラム解析の技術は現在も研究が続けられているし、VLIWマシン、スーパースカラマシン、ベクトル計算機、並列計算機などの計算機アーキテクチャの

変化に対応して、それに応じたコンパイラの最適化技術が開発されてきた。すなわち、それらの新しいアーキテクチャを生かしてきたのはコンパイラの最適化技術である。今後のアーキテクチャの進歩もコンパイラの技術の進歩と相まって達成されていくものである。その意味でコンパイラ技術は決して古くない。

コンパイラ技術は計算機科学の基本である

情報処理学会が1997年にまとめた『大学の理工系学部情報系学科のためのコンピュータサイエンス教育カリキュラムJ97』の中には、そこで取り上げた各科目とそれらをどのように組み合わせで学科や専攻のカリキュラムを組み立てるのがよいかという提案がある。そのどれをとってみてもほとんど必ず入っている科目には、計算機アーキテクチャとオペレーティングシステムとコンパイラがある。すなわちこの報告書ではこの3つが計算機科学の基本の科目であるとしていると考えることもできる。これら3つの科目は、計算機をブラックボックスとしてでなく、計算機の中で各種の機能がどのように実現され、どこにどれくらいの時間がかかるかということが理解するために必要不可欠である。これらの理解なしに、あるいはこれらの基礎的な技術を身に付けずに、これら

の上に構成されていくネットワークの機能や性能を理解したり、新しい提案をしていくのは困難であろう。

計算機科学の基本科目としてのコンパイラの役割は、簡単なコンパイラなら自分で設計したり開発したりできるようにすることであると考えられるが、それだけでなく、計算機システムを理解する、ソフトウェアの「科学」を学ぶ、といった役割がある。

計算機科学に携わる人や、優れたプログラムを開発しようとする人は、計算機をブラックボックスとしてでなく、計算機内部での動きをホワイトボックスとして理解できることが必要である。それは、プログラム言語とアーキテクチャやオペレーティングシステムとのかかわりの理解にもつながる。そのためにはコンパイラによって何が行われ、プログラムが最終的にどのような形で実行されるのかの理解が不可欠である。それがあれば、たとえば、Java言語で書いたプログラムがバイトコードと呼ばれる目的コードに変換され、JVMというバーチャルマシンで実行されるということを聞いただけで、そのことの意味が分かり、自分の書くプログラムにそのことがどのように影響されるかが分かる。さらに、JITと呼ばれる実行時コンパイラがあることや、それがどのような最適化の機能を持っているか、な

どが、実行時間に与える影響を考慮することもできる。

ソフトウェアの設計はアートであるといわれるが、その中で、コンパイラに関しては、その設計法が理論的に最もよく研究され実際にも適用されてきた。いわば、コンピュータ科学の見本になっている。たとえば、正規表現を認識する有限オートマトンの理論が、コンパイラの中の字句解析のプログラムを機械的に生成するのに使われている。この有限オートマトンの理解はコンパイラだけではなく、各種のプログラムの設計によく使われる状態遷移図の理解につながる。構文解析の理論は文脈自由文法の理論から構成され、LL(1)文法に対する再帰的下向き構文解析 (recursive descent parsing) や LR(1)文法に対するLR構文解析などが開発され、それにしがつて構文解析のプログラムを自動生成するyaccなどのシステムが実用化されている。

これらの理論はコンパイラだけでなく、より一般のプログラムの設計にも応用することができる。たとえば、事務処理のプログラムでは入力データの構造が正規表現で表現できるものが多いが、その正規表現から機械的にプログラム構造を作り出すこともできる。それが、初期のジャクソン法と呼ばれるプログラム設計法である。コンパイラの中でよく使われるスタックというデータ構造や後置記法 (postfix notation) という表現法も応用範囲が広い。

コンパイラ技術が計算機システムの進歩を支える

最初にも述べたように、新しいアーキテクチャを生かしてきたのはコンパイラの最適化技術である。日本はベクトル計算機では世界をリードしてきたが、それはハードウェアが優れていただけでなく、ベクトル化

コンパイラと呼ばれるコンパイラ (ベクトル化技術は最適化技術の一種である) がベクトル計算機の機能を最大限に生かすような目的コードを生成するように、コンパイラ開発者が努力したからである。

筆者は数年前に筑波大学の計算物理学研究センタに設置された超並列計算機CP-PACSの設計に参加した。しかし、そのノードマシンとして使う予定のマシン (ワークステーション用のチップ) では、計算物理学の計算のような大規模な配列演算に関してキャッシュが有効に働かず、目標とする性能が出ないことが分かった。そこでアーキテクチャのグループから、スライドウィンドウが提案された。それはレジスタを128個持ち、32個のレジスタが見える窓 (ウィンドウ) がその上をスライドしていく方式で、まだ見えないレジスタにあらかじめデータをロードしておくことによって、ロードの遅れを隠ぺいするものである。しかし、それを生かすように人間がプログラムするのは困難である。幸い、コンパイラのグループでそれを生かすアルゴリズムを考え出すことができて目標とする高い性能が実現できた。このように、最近の新しいアーキテクチャはコンパイラによる最適化がなければ意味のないものが多い。インテル社の新しいチップIA-64のアーキテクチャでもさまざまな機能が付加されているが、それらを生かして意味あるものとするのもコンパイラの最適化技術である。

現在は並列計算機の時代であるともいわれる。上に述べたのは、機械語の命令レベルの並列機能とそれを生かす話であるが、数台のプロセッサでメモリを共有する並列計算機から、数千台の分散メモリ型の並列計算機まで、さまざまな並列計算機が開発されている。しかし、コンパイラの技術はまだそれを生かすところ

までは行っていない。

命令レベルの並列化やベクトル化の場合は、プログラムの中でその対象とする範囲が一番内側のループとなることが多く、コンパイラで解析すべき範囲が比較的狭い。したがって、たとえば1社で何年か努力することによって成果を挙げることが可能であった。日本はそうやってベクトル計算機分野で世界をリードすることができた。しかし、プロセッサにまたがった並列化の場合は格段と難しくなる。並列化はできるだけ大きなかたまりでした方がよいので、プログラム全体に渡ったデータのアクセスのパターンなどさまざまな解析が必要になるし、並列化に伴うオーバーヘッドもいろいろあって並列化の効果もベクトル化の場合のように簡単には分らない。それはマシンによっても違うから他のマシンでよかったことをそのまま使うこともできない。何か1つのいいアルゴリズムによって解決する問題ではなく、多くの技術の積み重ねが必要であり、個々の企業の努力だけでは解決が困難である。

並列化の研究ではずっと先行しているアメリカでは、このような問題に対処するためにコンパイラの各種の部品が共通に使えるように、産官学共同の国家プロジェクトとしてNCI (National Compiler Infrastructure) の構築が進められている。

コンパイラのような基礎技術をおろそかにすると、新しいアーキテクチャや新しいプログラム言語を生み出す技術もなくなり、情報技術に関する競争力全体が弱くなってしまう。その結果、日本の計算機産業は、よその国で作ったものを使ってサービスするだけになってしまうのではないだろうか。

(平成12年2月10日受付)