3P139

関数型プログラミング言語による振動波束計算法の開発

（慶大院・理工）　　○菅原道彦

Application of functional programming language to quantum mechanical vibrational wavepacket calculation

(Graduate School of Science and Technology, Keio Univ.)　○M. Sugawara

【序】近年、ソフトウェア資産のモジュール化・並列プログラミングへの親和性などの理由により、関数型プログラミング言語が注目されている。一方で、科学技術計算ではパフォーマンスにおける最適化、数値計算用ライブラリなどを含むレガシーコードの活用が優先されるため、古くからある FORTRAN、C などの手続き型言語が使用されることが多い。しかし、これらの言語は後発の言語と比較するとライブラリのモジュール化、ビルド/テストツールの充実度、並列計算への適応性などにおいてはやや時代遅れである点は否めない。純粋関数型言語は一切の状態変数を持たず、入力に対してユニークな出力を返す関数のみを使用して問題を解決する。このような参照透過性を有するため、単体テストに重きを置いた開発が可能であり、バグが少なくかつ簡潔なコードを書けるといった特徴がある。特に、強く静的型付けされた関数型言語である Haskell などはロジックに関する潜在的なバグのほとんどをコンパイラによる型チェックで検出する事が可能であるとされている[1]。しかしながら、この様な関数型言語の優位性は純粋な数値計算能力やメモリ資源の有効活用とトレードオフの関係にある。FORTRAN、C のように言語自体にメモリに対する操作が直接反映されているような手続き言語を使用した場合は効率的なループ制御や適切に代入文を使用することによって、ソースコードレベルでメモリに対するアクセス方法やリソース管理の最適化が可能である。しかし、多くの純粋関数型言語ではメモリ管理やパフォーマンス最適化のための文法が明示的に用意されているわけではない。このため、関数型言語を科学技術計算に利用する際は、メモリ管理を含めたパフォーマンス最適化に特別な配慮が必要である。本研究では、純粋関数型言語として普及している Haskell を実用的な数値計算に適用する場合に直面する上述の問題の解決方法を提示し、実際に量子力学的振動状態波束ダイナミクス計算に適用する。

【方法論】一般的な波束ダイナミクス計算は、適当な基底系を用いて状態ベクトル $\mathbf{c}(0)$ として表現された初期状態に微少時間発展演算子 $U(\Delta t)$、すなわち $\mathbf{c}(t+\Delta t)=U(\Delta t)\cdot\mathbf{c}(t)$ を繰り返し作用させていく手続き、$\mathbf{c}^{(n)}\equiv[U(\Delta t)]^n\cdot\mathbf{c}(0)$ によって実現される。$U(\Delta t)$ はハミルトニアン行列 $\mathbf{H}$ のべき乗、もしくは適当な直交多項式で

展開し適当な次数で打ち切ることによって、有限回の**H**演算として実現される。

$$\hat{U}(\Delta t) = \exp[-\frac{i}{\hbar}\mathbf{H}\Delta t] \cong 1 - \frac{i}{\hbar}\mathbf{H}\Delta t + \frac{1}{2!}\left(\frac{i\Delta t}{\hbar}\right)^2 \mathbf{H}^2 + \cdots \qquad (1)$$

この際、手続き型言語では状態ベクトル$\mathbf{c}^{(n)}$をメモリ上に保持し、そこに**H**を作用させてメモリ内容を上書きしていくことによって状態変化を容易に表現できる。一方、関数型言語では状態変化が許されないため、新しい状態を格納するメモリ領域を確保してしまう（図１参照）ため、メモリ資源の有効利用が難しい【問題点１】。また、波束ダイナミクスの計算結果から情報として相関関数や物理量を計算する際に、状態ベクトルに対する内積計算が必要とされる。手続き型言語では値の蓄積変数 S を導入することによって、中間配列を必要としないメモリ利用が可能である。一方で、関数型言語で内積計算は一旦中間配列を介する形式で表現される（図２参照）ため、実行時に無駄なメモリ領域を確保してしまう【問題点 2】。そこで、本研究では State モナドを利用し、状態計算であることを明示することによってメモリ利用の最適化を促し【問題点１】を解決した。また、Fusion[2]を利用出来る形式のベクトル型を採用することにより、GHC(The Glasgow Haskell Compiller)コンパイラの最適化処理を通して中間配列の生成を回避し【問題点２】を克服した。さらに、配列に対する並列操作を提供する Repa（Regular Parallel arrays）ライブラリ[3]を利用したコア内並列計算の適用可能性も併せて検討する。

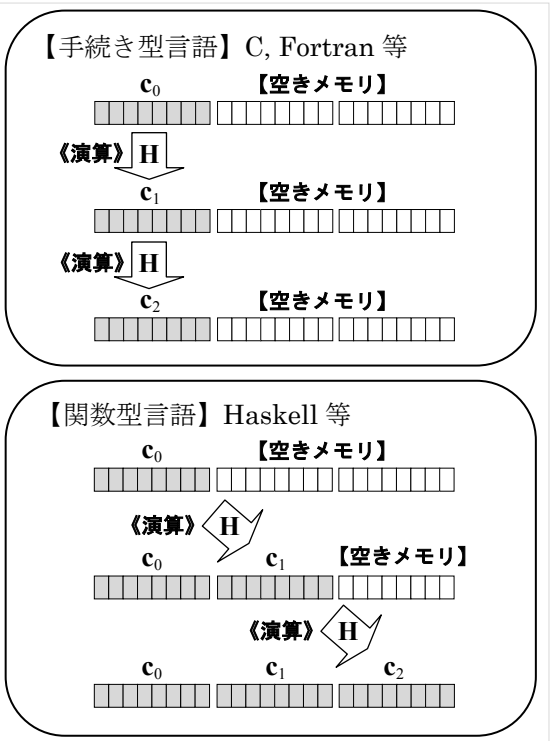

図１　状態計算におけるメモリ利用のスキーム

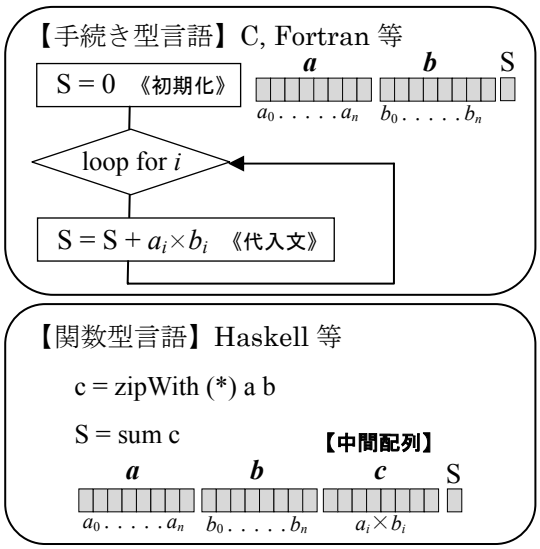

図２　内積計算におけるメモリ利用のスキーム

【参考文献】

[1] B. O'Sullivan, J. Goerzen, D. Stewart, "*Real World Haskell*" , O'Reilly Media, 2008.

[2] G. Mainland, R. Leshchinskiy, S. P. Jones, ACM SIGPLAN Notices - ICFP '13, **48**, 37-48, (2013).

[3] S. Marlow, "*Parallel and Concurrent Programming in Haskell*", O'Reilly Media, 2013.