4J-1

ハイブリッドマルチコア環境における関数型言語の実現手法

新井 一郎, 渡部 卓雄
東京工業大学・情報理工学研究科・計算工学専攻

概 要

GPU 上でプログラムを効率的に実行させるためには、データの配置やアクセス順序等に関するハードウェア依存の最適化が必要であるが、そのような最適化記述はアプリケーション記述から独立にモジュール化できることが望ましい。本研究では、関数型言語による GPU プログラムの開発を容易にすることを目的とした GPU 向けの最適化記述用ドメイン特化言語 (DSL) を提案する。提案手法では、Scheme によるアプリケーション記述と DSL による最適化記述から最適化された CUDA コードを自動生成する。これにより、最適化記述のモジュール性向上と GPU による高速化の両立させる。

1 CUDA

GPU 上でプログラムを効率的に実行させる際には、グローバルメモリのコアレッシング、シェアードメモリのバンクコンフリクト、通信の隠蔽などを意識する必要がある。実際の CUDA のプログラムでは

0- シェアードメモリにデータをコピー
1- シェアードメモリ上の読み込み位置を計算
1 データの取得
2 データを使って計算
3- シェアードメモリ上の書き込み位置を計算
3 結果の書き込み

といった動作をするプログラムを生成する必要がある。- のついた項目が最適化のための記述であり、ついてない項目は計算アルゴリズムの本質的な記述である。計算の記述と最適化のための記述が混在しているコードになっている。また、大量のデータを処理するプログラムを CUDA で作成する際にはデータの分割・転送・退避を繰り返しながら計算を進める必要があるが、これらは、値を求める計算では本質的な処理ではなく、もしも演算ユニットやメモリが無限にあれば必要なくなる処理である。CUDA を用いたプログラミングでは、このようなハードウェア依存の記述と計算アルゴリズムの本質的な記述（ハードウェア非依存）を混在させることになる。

Implementation of a functional language on Hybrid Multi-core environment,
Ichiro Arai, Takuo Watanabe, Graduate School of Information Science and Engineering, Tokyo Institute of Technology

2 DSL の方向性

前述したデータの分割や転送のパターン、最適化のパターンなどは、他の数値計算のアルゴリズムでも同じようなパターンで出現することがあるが、CUDA では計算のアルゴリズムとデータの転送や最適化の記述などが密接に関わりあったコードになるため、それらのパターンだけを分離することができずに何度も似た記述をして見通しや保守性や再利用性が低くなる。

そこで本研究では、計算のアルゴリズムを Scheme で記述し、提案する DSL でハードウェア依存の記述を行うことを考える。これにより、Scheme の記述は見通しがよくなり、ハードウェア依存記述の再利用性も向上できる。

3 関連研究

Skeleton という用意された部品を組み合わせることでプログラムを作成する研究がある。 [1] [2] この方法の利点は、実際のアーキテクチャを意識することなくプログラムを作成できる点である。また、Skeleton の組み合わせによっては自動で最適化できるケースがあり、見通しと移植性と効率の良いプログラムを作成できる。これに似た最適化の方法の実装例として、GHC の書き換え規則

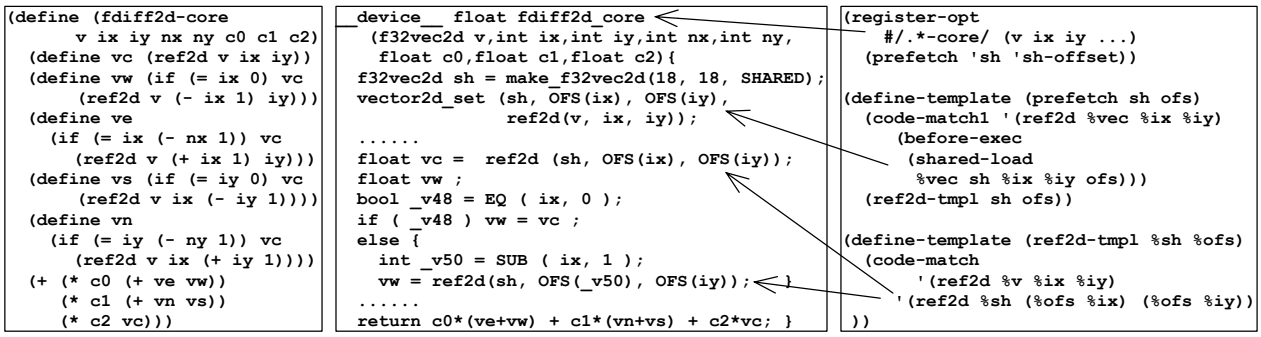

図 1: テンプレートの適用例

や融合変換がある。

AspectJ などのアスペクト指向プログラミングでは、横断的関心事をメソッド中に埋め込むことなく別の箇所に記述する。これにより再利用性を高めている。

4 DSL

図1に、Schemeで記述したコードと自動生成されたCUDAコードとDSLで記述したテンプレートを示す。左は拡散方程式を差分法で解くコードの、1格子分を計算するSchemeのコードで、右のDSLで記述したテンプレートでは、引数として高階関数をとる形式で、最適化の記述をしてある。CUDAコードに変換する直前にテンプレートが適用され、中央のコードが生成される。

・`register-opt` fname arg-list body

変換対象のコードの関数名と引数がfnameとarg-listにマッチする場合にbodyを実行する。fnameには正規表現を用いることもできる。

・`define-template` (tname . args) body

bodyの各コードが実行されるたびに変換対象のコードを置き換える。

・`code-match` tmpl expand-pattern

CUDAに変換中のコード中にtmplにマッチするコードが現れた場合には、その箇所をexpand-patternで指定した通りに展開する。%で始まるシンボルは、マッチしたコードに置き換えられる変数になる。`code-match1`は最初にマッチしたコードだけを対象とする。

これ以外にも、並列実行させる際のブロック数・スレッド数やデータの初期化のタイミング、どのデータをどこに転送するか、データの転送のタイミングなどが指定もできるべきである。

5 まとめ

「どのデータが必要か」や「どのデータを退避させるか」といった自動的な判断は難しい。Skeletonプログラミングでは、不意に離れたインデックスにあるデータが必要になるケースも考慮しないければならず、最適化が十分できない可能性もある。

本研究のDSLでは、どの範囲のデータが必要になるのかをプログラマが判断し、関数名をヒントとして与えることで適切なテンプレートを適用するように指定できるため、より細かい最適化を行えるようになっている。これにより見通しや効率や保守性の高いプログラムを生成が可能になる。

参考文献

[1] Murray Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation*. MIT Press, Cambridge, MA, USA, 1989.

[2] John Darlington, A. J. Field, Peter G. Harrison, Paul H. J. Kelly, D. W. N. Sharp, and Q. Wu. Parallel programming using skeleton functions. In *Proceedings of the 5th International PARLE Conference on Parallel Architectures and Languages Europe*, PARLE '93, pp. 146–160, London, UK, 1993. Springer-Verlag.