科学技術計算の効率的超並列化に向けた
静的負荷分散を行うDSLの開発

西田秀之

東京大学情報理工学系研究科創造情報学専攻

１．はじめに

科学技術計算は，科学の諸原理をプログラムの形で実装し，計算機を用いて物性計算を行う。計算機科学の発展に伴って多岐に渡る分野の研究的・産業的な活用が成されている。計算資源の節約及び時間的な要求により，科学技術計算のプログラムは高速に実行できることが望ましい。しかし，スーパーコンピュータを含めた複雑化している計算機環境を十全に活かせる科学技術計算プログラムの実装は試行錯誤を要する。

実際のプログラム開発においては，対象分野（ドメイン）に関する知識と計算機に関する知識を組み合わせてプログラムの高速化・最適化が行われる。ライブラリを代表として，ソフトウェア開発効率を向上させる手段は古くから存在する。近年，ドメイン特化言語（Domain Specific Language：**DSL**）が注目されている。

多くの科学技術計算分野では，計算速度・研究の蓄積の点から，Fortran・C++が使われてきた。これらは汎用言語と呼ばれ，汎用的な用途に使えるように設計されている。これに対して，DSLは特定の領域での使用に特化している。そのため，言語の表現力に制限がかかるものの，特定領域においては開発効率性や性能が高いプログラムを書きやすくなっている。中でも，汎用言語との併用に優れる内部ドメイン特化言語（Embedded Domain Specific Language：**EDSL**）が存在する。EDSLは，汎用言語のライブラリとして使用することができ，汎用言語の機能を利用して独自の記法や機能を実現する。本研究では，スーパーコンピュータなどの超並列環境並びに特殊な計算機環境の性能を活かせるようにプログラムを書くことができるEDSLを設計し，科学技術計算プログラム開発者の支援を目指す。

２．科学技術計算

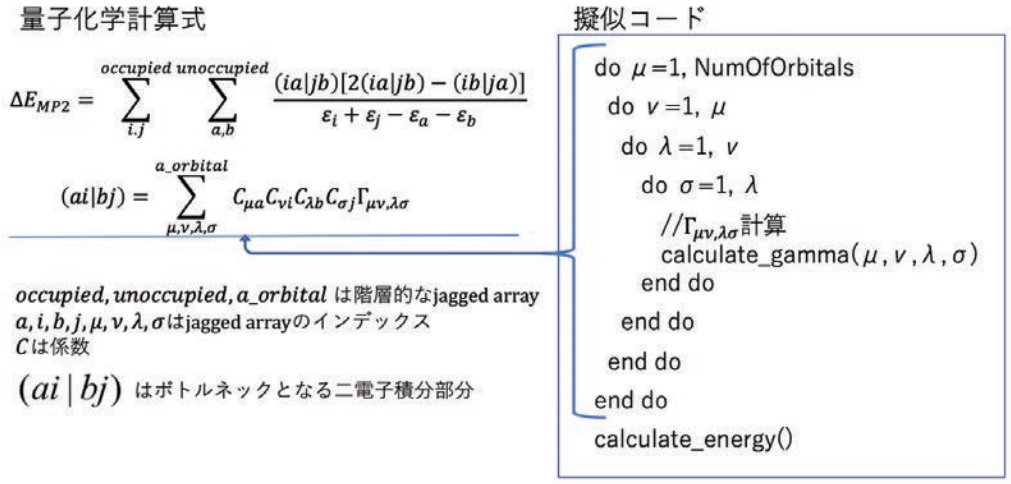
本節では，科学技術計算の一例として量子化学計算を取り上げ，その高速化における難点の一部を紹介する。

量子化学計算は量子力学を用いて物性計算を行う化学の一分野である。分野の発展と計算機技術の発展に伴い，複雑な理論の実装や高速化が行われてきた。アプリケーションプログラムを最適化する他にも，計算機環境を想定した上で理論の調整を行う取り組みも行われて来た。高速なプログラムの実装を困難にする要因の例を以下に挙げる。

(1) プログラムの複雑性により，計算量の把握が難しい。

量子化学計算では，対象物の電子を考慮して計算を行う。(厳密には電子軌道を考慮している。)ある種のN対計算である電子対の組み合わせの計算が行われ，計算手法にもよるが電子数nの累乗の計算量が必要と言われる。N対計算であるために，ある電子対を特定・計算するためには階

層的で複雑な配列（jagged array）に多数のインデックスを用いる必要がある。そのため，プログラムは入れ子が深い多重ループ構造をとり，各ループの計算量の把握は困難なものとなる。しかし，コードの高速化には，計算量に応じた最適化も求められる。



第 1 図：量子化学計算式とその擬似コード。
左側が量子化学計算中でも時間がかかる二電子積分部分の計算式の一例。
右側は積分部分を表す多重ループの擬似コード。

(2)　入力データの人的な把握管理が困難である。

プログラムの多くは，決められたフォーマットの入力ファイルを用いて計算を行う。計算量は入力データの内容に依存するため，高速性を求めるならば入力データに応じた最適化が欠かせない。以下が量子化学計算プログラムの入力データの一例である。

```
$basis gbasis=n31 ngauss=6 ndfunc=1
$end
$coordinate
H 1.0 -0.559000 -8.242000 -1.948000
O 8.0 0.302000 -7.920000 -1.673000
C 6.0 1.298000 -7.649000 -2.677000
H 1.0 0.927000 -6.886000 -3.362000
$end
```

上部には計算手法などの情報，下部には計算対象の物質情報が記述されている。しかし，このファイルには計算に必要な電子の情報は明示的に記載されず，それらは暗黙的なデータとして自動で生成される。この暗黙的なデータは原子数の数倍～数十倍の巨大なデータになるため，人的な計算量の把握は困難である。例えば，ある分子を部分系に切り分けて計算する手法においては，

部分系毎の電子データが暗黙的に生成される。部分系の計算量は電子データの大きさに依存するため，データによっては部分系毎に計算時間に大きな偏りが発生する場合がある。特に超並列計算が必要な巨大な入力データまで想定すると，明示的に記述されていた場合でも人的な把握が困難となる。そのため，入力データを考慮してのプログラムの最適化も困難である。

(3) 最適化手法の適切な選択が難しい。

多くの研究により，プログラムを高速化するための手法は多数開発されてきた。量子化学計算においても，それらの手法の活用が積極的に行われている。しかし，すでに複雑なプログラムを適切に書き換えるのは簡単ではない。例えば頻繁に用いられる，OpenMP と MPI のハイブリッド並列手法が存在する。ここでは簡単のため，OpenMP と MPI はプラグマとして多重ループに挿入されると考える。標準的な量子化学計算と部分系に分割する手法にハイブリッド並列を入れた擬似コードを以下に示す。

```
#pragma MPI
for(y=0;y<MAXLENY;y++){//電子1
  #pragma MP
  for(x=0;x<MAXLENX;x++){//電子2
    --- inside algorithm ---
  }
}
```

```
#pragma MPI
for(z=0;z<MAXLENZ;z++){//部分系
  #pragma MP
  for(y=0;y<MAXLENY[z];y++){//電子1
    for(x=0;x<MAXLENX[z];x++){//電子2
      --- inside algorithm ---
    }
  }
}
```

第２図：ハイブリッド並列の例。
左側が標準的な量子化学計算にハイブリッド並列を入れた擬似コード。
右側が部分系に分割する手法にハイブリッド並列を入れた擬似コード。

MPI は粗粒度ループに適用される場合が多いため，それぞれ異なる for 文に対してプラグマが挿入される。しかし，MPI により生じるオーバーヘッドも考慮すると，どのループに MPI を挿入するべきかは入力データに依存する。例えば，部分系の中でもインデックスが 0 と 1 の計算量が多い場合には，以下のようにループ自体を分割したコードも考えられる。

```
for(z=0;z<2;z++){//部分系
  #pragma MPI
  for(y=0;y<MAXLENY[z];y++){//電子1
    #pragma MP
    for(x=0;x<MAXLENX[z];x++){//電子2
      --- inside algorithm ---
    }
  }
}
#pragma MPI
for(z=2;z<MAXLENZ;z++){//部分系
  #pragma MP
  for(y=0;y<MAXLENY[z];y++){//電子1
    for(x=0;x<MAXLENX[z];x++){//電子2
      --- inside algorithm ---
    }
  }
}
```

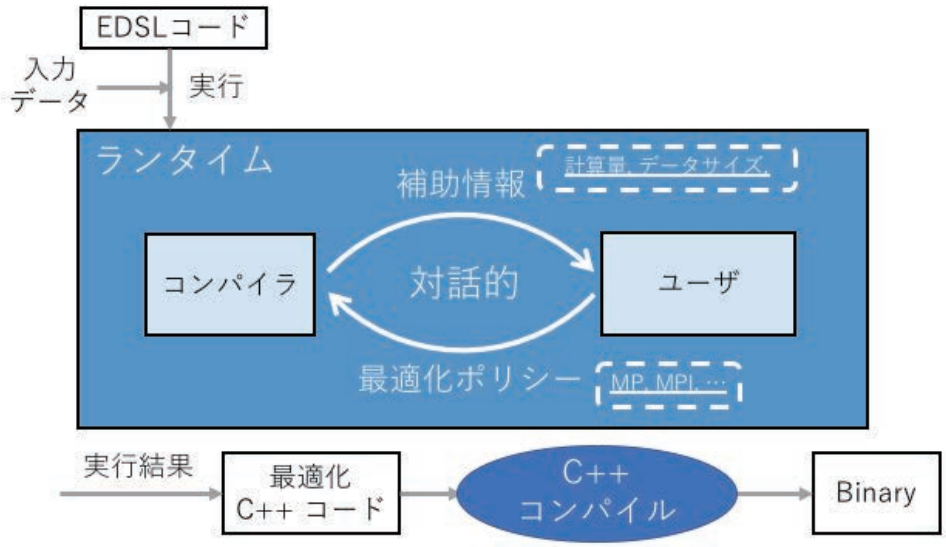
第３図：ループ分割の例。
インデックスが 0 と 1 の場合，
別のループで計算させる。

この擬似コードでは，プラグマの挿入位置が異なるループを二つ記述して，MPI が作用するループを分けている。これまでの図は擬似コードであり，実際はより大きな多重ループが用いられる。それらでは最適化可能性が増大するために，最適化がより困難なものとなる。

３．Jupyter 併用型 DSL

本研究では，ユーザとコンパイラの知見を相補的に活用してプログラムの最適化を行う環境構築を目指す。具体的には，入力データを考慮した静的な負荷分散をユーザとともに行うために，インタラクティブな環境でユーザが最適化の指示を対話的に行える DSL を設計する。最適化のための補助情報が DSL のコンパイラから提示され，それを見てユーザは適切な最適化ポリシーを指示する。ユーザが記述する DSL のプログラムは実際に科学技術計算を行うプログラムそのものではなく，科学技術計算プログラムを生成するためのプログラムとなる。

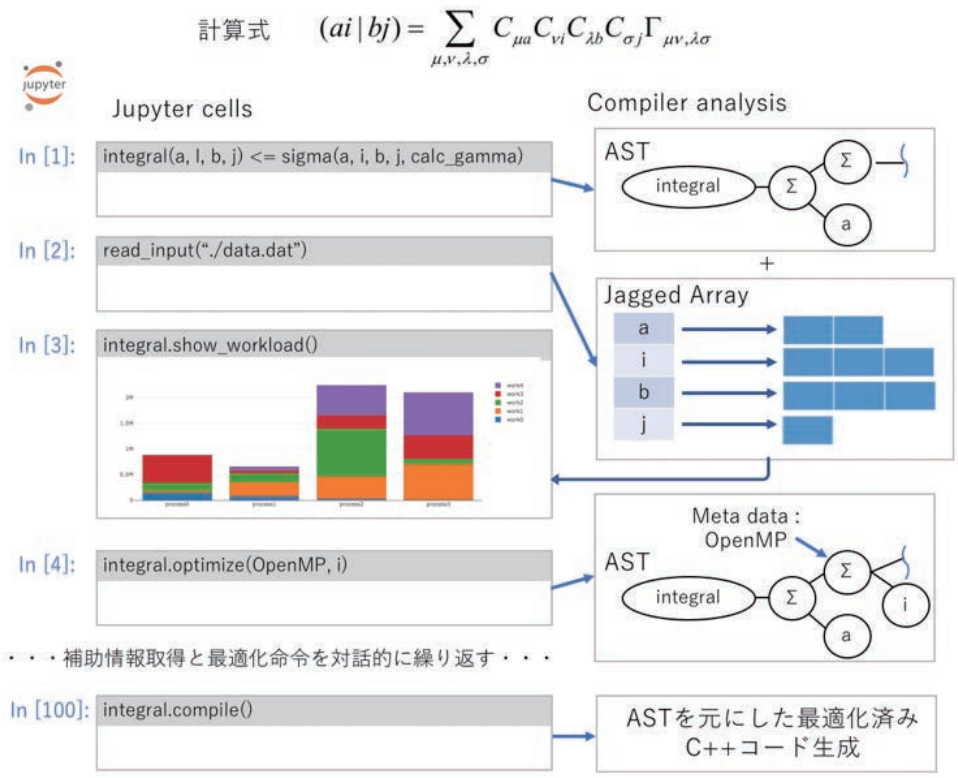
システムの概念図を以下に示す。



第４図：開発するシステムの概念図。
ユーザが記述した EDSL コードは実行時にユーザとやり取りをしながら，最後に科学技術計算用プログラムを生成する。

本システムではユーザが記述した DSL プログラムは二段階に分けてコンパイル・実行される。本システムの特徴は一段階目の実行時に，ユーザとコンパイラが対話的な環境を通じてプログラムの最適化を行うことである。コンパイラは事前に入力データを読み込んでいるため，データサイズや並列時の計算量などの補助情報を計算しユーザに提供する。これは，人的な把握が困難な部分をコンパイラが吸収・アシストする仕組みとなる。ユーザは補助情報を参考に自身の知見と合わせて，最適化ポリシーを決定する。ユーザの指示により二段階目の実行が行われ，実際に科学技術計算を行うプログラムを生成する。

本研究では，このシステムを Jupyter 上で実現した。Jupyter は広く使われる開発環境であり，対話的な操作を提供している。本研究では，EDSL のホスト言語となるのは Ruby であり，出力するのは実行速度の観点から C++プログラムである。Jupyter 上での実行の様子を次に示す。



第５図：Jupyter 上での DSL 実行の様子。
DSL の記法で記述されたプログラムと入力データから，補助情報を提供しつつプログラムの最適化を行う。

システムの実行は次の 3 ステップで行う。

(1) DSL の記述と実行（セル In[1]，セル In[2]）

プログラムの開発者は DSL をライブラリとして読み込み，DSL で定められた記法でプログラムを書く。この時は MPI などの最適化は考えずに計算の本質のみを記述する。このセルの実行時点では，deep embedding の手法に基づき，記述した計算式を示す抽象構文木(Abstract Syntax Tree: AST)が生成される（セル In[1]）。次に入力データの読み込みを行い（セル In[2]），暗黙的データが構築される。このステップにより，コンパイラは AST と入力データを組み合わせて計算量などの補助情報を生成できることになる。ここでは，一般的なプロファイリングと異なり，予測値が求められる。

(2) 対話的最適化（セル In[3]，セル In[4]）

ユーザは別のセルで，本システムへの補助情報取得命令を実行することができる。これにより計算式や入力データ情報，各種サポート情報を確認する（セル In[3]）。例えば，あるループ部分の計算量を取得する操作を行うと，ループ一回毎にどの程度の計算量がかかるかをグラフとして取得することができる。これらを基にユーザは最適化ポリシーを選択する（セル In[4]）。このセルの実行時には，最適化ポリシーが AST へのメタデータとして埋め込まれる。このステップを繰り返すことによりユーザが望む最適化ポリシーを含んだ AST が構築される。

(3) コード出力と計算の実行

対話的な最適化が終了した後，別のセルで特別な命令を実行することで AST を元に C++の最適化科学技術計算コードが生成される（セル In[100]）。

ここでは細かい実装方法の記述は避けるが，超並列計算に向けた静的負荷分散に関連する事項を説明する。簡単のため，MPI を 4 並列で行うケースについて考える。本 DSL では，計算式の構築と入力データの読み込みの時点でコンパイラに計算量を計算させることができるため，MPI 並列計算時のそれぞれのプロセスの計算量も計算することができる。それを活用して静的に負荷分散を行う例を以下に示す。

第 6 図：MPI 並列を単純に導入したケースと静的負荷分散を行ったケース。
左側は，あるループ部分に MPI を単純に導入した場合における各プロセスの計算量を示す。
右側は，同じループ部分に MPI を導入して静的に負荷分散をおこなった場合の計算量を示す。

ユーザは DSL が提供するメソッドを用いて特定のループ部分に MPI のディレクティブを挿入することができる。また，該当ループのプロセス毎の計算量を可視化することができ，このケースでは棒グラフが表示され，色分けされた四本の棒が 4 並列のプロセスに対応する。色分けされた部分の大きさが各プロセスが持つループ一回分（タスク）の計算量に相当する。この補助情報により，単純に MPI を導入した場合(左側)ではプロセス 0 と 1 のタスクの総計算量が小さく，プロセス 2 と 3 のタスクの総計算量が大きいことが視覚的に理解できる。このまま実行するとプロセス 0 と 1 の待機時間が大きくなることが想定されるため，ユーザはプロセス 2 と 3 から一部のタスクをプロセス 0 と 1 に移動させる命令を行う。その結果が第 6 図右側であり，このグラフにより計算量の平均化が図られていることが確認できる。

以上が対話的最適化の一連の流れである。補助情報は可視化だけでなく数値としても取得できるため，Ruby スクリプトによって，独自のスケジューラを構築することができる。
例えば，以下のようなスケジューラの構築ができる。

```
def scheduler_MPI(func)
  tasks = func.tasks.sortBy(0, UPPER)
  func.parallel(MPI, tasks)
end
```

```
def scheduler_complex_parallel(func)
  func1, func2 = func.divide(func.arg[0].lengthOf("child_array") > 1000)
  func.sequential(func1, func2)
  inter_func1 = function("inter_calc").at("sigma").at(func1)
  inter_func1.parallel(MPI)
  func2.parallel(MPI)
end
```

第 7 図：Ruby スクリプトによるスケジューラの例。
関数の引数に，AST が構築された計算式を入れることで，破壊的にメタデータを注入する。

左の関数は AST が構築された計算式を引数にとり，1 ループの計算量毎に昇順に並べ替えて MPI 並列化をすることで，計算量の平衡化を図るものである。右の関数は第 3 図で述べたようなルー

プの分割を行い，別のループにMPIのディレクティブを挿入するスケジューラである。

４．おわりに

本研究では，従来のDSLを参考にして，科学技術計算の超並列化を含めたプログラム最適化のための DSL を設計し，Jupyter と共に用いることで対話的に最適化をするシステムを提案した。科学技術計算例として量子化学計算を取り上げ，入力データの暗黙性や巨大さ，計算式の多重ループ性により生じる最適化の難しさが存在することを示した。その難しさを本DSLは吸収し，可視化を含めたコンパイラによる補助機能を活用することで，人間とコンパイラが相補的に最適化に取り組むことができる。今後の課題の一つとして，DSLにどこまでの機能を持ち込むかの検討がある。例えば本DSLのスケジューラは，Rubyの記法に則って実装できることを示したが，DSLにその機能を取り込むことも一つの手である。また，生成されるASTをユーザが独自に操作して好きな最適化を組み込む機能も考えられる。本DSLの開発者が実装した最適化のみではなく，ユーザにより最適化可能性を広げられるような検討を行いたい。

５．謝辞

本研究は東京大学情報基盤センター若手・女性利用者推薦制度の助成を受けて実施された。

参　考　文　献

Arie van Deursen, Paul Klint, and Joost Visser. 2000. Domain-specific languages: an annotated bibliography. *SIGPLAN Notices*. 35, 6 (June 2000), 26-36.

Jeremy Gibbons and Nicolas Wu. 2014. Folding domain-specific languages: deep and shallow embeddings (functional Pearl). *SIGPLAN Notices*. 49, 9 (September 2014), 339-347.

Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *SIGPLAN Notices*. 48, 6 (June 2013), 519-530.

Kazuo Kitaura, Eiji Ikeo, Toshio Asada, Tatsuya Nakano and Masami Uebayasi. 1999. Fragment molecular orbital method: an approximate computational method for large molecules. *Chemical Physics Letters*. 313, 3-4 (1999), 701-706.

A.Szabo and N.S.Ostlund著，大野公男，阪井健男，望月 祐志翻訳『新しい量子化学 上 電子構造の理論入門』303 p，東京大学出版会，1987