戦略的創造研究推進事業　CREST
研究領域「「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」
研究課題「ポストペタスケール時代のスーパーコンピューティング向けソフトウェア開発環境」

研究終了報告書

研究期間　平成23年10月～平成29年3月

研究代表者：千葉　滋
（国立大学法人東京大学
情報理工学系研究科、教授）

§１ 研究実施の概要

（1）実施概要

本研究チームは、近年、特に web サービス開発の分野等で急速に発展した現代的なソフトウェア開発技術の恩恵を高性能計算分野にもたらし、ポストペタスケール時代に予想される高度なソフトウェアの開発に貢献すべく研究を進めてきた。ポストペタスケール時代には、異種混在（heterogeneous）型のハードウェア上で、省エネルギーを意識して（energy-aware）動作するソフトウェアが求められる。その一方で、アプリケーション開発の視点から考えると、並列規模は増大し、複雑なハードウェアを活用するような高度に複雑なソフトウェアを高い計算精度、品質を保ちながら開発しなければならなくなる。ポストスケール時代には、益々そのような困難なソフトウェア開発をしなければならないことは明白でありながら、歴史があり成熟してはいるが、 開発人件費・期間ともに高コストな開発手法を用いて、高性能なソフトウェアを生み出し続けなければならない状況にある。性能に対する要求水準が極端に高く、職人芸的な技法を駆使した高コストなソフトウェア開発が許容される環境にあること、従来からの膨大なライブラリの資産があること、また多くのアプリケーションプログラムが長い年月をかけて少しずつ拡張を加えながら開発されることなどから、新しい開発手法に移ることが現実的に困難であるためである。

このようなソフトウェア開発の状況にあって、本研究チームはドメイン専用言語を用いた開発手法に基づく高性能計算のためのソフトウェア基盤、開発ツールを研究した。それにより、そのような開発手法に基づいて開発すると、アプリケーションプログラムを実際に高い抽象度で記述でき、開発容易性や保守性が改善されること、その一方、従来手法で職人芸的な技法を駆使して開発したアプリケーションプログラムに大きくは劣らない実行性能を達成できることを、実証的に示した。具体的には従来手法で多く用いられる Fortran や C、C++といったプログラミング言語ではなく、Java と Ruby という比較的新しいと言われる言語をもとにしたソフトウェア基盤、開発ツールを本研究チームで実現した。これは分散並列型スーパーコンピュータや GPU マシンで動作するアプリケーションソフトウェアの開発に用いることができる。

国際的視点から他の類似の研究活動の成果を考慮しても、現時点で、ここで述べているような新しい開発手法が、高性能計算のアプリケーション開発の現場に浸透しているとは言いがたい。まだまだ従来型の開発手法の優位性が強く信じられている状況である。しかしながら、本チームの研究では実アプリケーションの開発に耐える品質のソフトウェア基盤、開発ツールを実際に生み出し、そのような手法の可能性を実証的に示すことを達成できた。これは、それを足がかりとして次の研究段階に進むことを可能にする成果である。次の研究段階では、本研究で提示するような新しいソフトウェアの開発手法を実際に普及させることを目指すソフトウェア基盤、開発ツールの研究開発が期待できる。

近年のソフトウェア開発では、書かれたプログラムの正しさを、 完全ではないにせよ、ある程度機械的に静的あるいは動的に検証する手法が広く用いられている。我々の研究チームは、この課題にも取り組み、分散並列計算に必要とされる正しさを研究し、それを実際に検証するための基礎技術を開発した。プログラムの正しさと一口に言っても、それを厳密に表現することは自明ではなく、応用目的ごとにどのような正しさを検証するべきかを考えること自体も研究である。我々の研究チームは、分散並列計算において、メモリアクセスを高速化するためのアクセス順序の最適化の正しさを検証することの重要性を示し、そのような検証をおこなうための技法を静的および動的の両面から研究し、それが実現可能であることを示した。

（2）顕著な成果

＜優れた基礎研究としての成果＞
1． ホーア論理による並列プログラムの動作自動検証のための基礎技術
概要：
CUDA アーキテクチャなどの SIMT 計算モデルにおけるプログラムの正当性を示すためのプログラム論理の構築を行い、逐次の命令型言語の意味論を Habermaier と Knapp の手法による SIMT 拡張を行い対象言語とし、逐次命令型言語に対するプログラム論理であるホーア論理を拡張し、対象言語の意味論に対する健全性・相対完全性を示した。また注釈付きの C 言語プログラムの動作を半自動検証する処理系を開発した。

2．GPGPU プログラムの仕様検証を行うための論理体系
概要：
GPGPU プログラムがメモリ競合のような誤りを持たないこと、また期待した仕様通りの計算を行っていることを確信することは容易でない。そのような GPGPU プログラムの正しさを検証するための論理として Twente 大学の Blom らの並行分離論理の拡張(GPUCSL) を定理証明支援器 Coq を用いて形式化し、その健全性証明を再構築した。この成果により第一著者の朝倉が 2016 年度情報処理学会コンピュータサイエンス領域奨励賞を受賞した。

3．埋め込み型ドメイン専用言語のための拡張可能構文機構
概要：
ホスト言語上のライブラリとして実現されるドメイン専用言語の構文上の自由度を増し、かつ実用的な時間での構文解析を可能にする手法を提案し、プログラミング言語分野の主要国際会議で発表することができた。この技術はライブラリによって簡易に HPC の特定ドメイン向けの DSL の開発できるようにし、ポストペタスケール時代の複雑化するプログラムの開発を大きく支援する可能性をもつ。

＜科学技術イノベーションに大きく寄与する成果＞
1．Java言語向け埋め込み型ドメイン専用言語開発基盤 Bytespresso
概要：
MPI、GPU を用いた高性能計算向けドメイン専用言語を、Java 言語をホスト言語としてそこに埋め込む、埋め込み型ドメイン専用言語として開発するためのソフトウェア基盤 Bytespresso を開発した。Bytespresso は、埋め込まれたドメイン専用言語のプログラムを抜き出し、効率のよい C あるいは CUDA プログラムに変換して実行するための処理系である。これにより lambda 式などを駆使する Java 風の構文をもち高速実行可能なドメイン専用言語を容易に開発できるようになった。

2．Ruby 言語で向け GPGPU プログラミング基盤 Ikra
概要：
Ruby 言語で GPGPU プログラミングをおこなうための言語基盤 Ikra を開発した。これにより Ruby 言語が提供する高い抽象度で容易に GPU プログラミングをおこなうことが可能になる。動的言語のための型推論系を研究開発し、それを Ikra に組み込むことで、基本的なステンシル計算をおこなう Ruby プログラムから、ほぼ変更なしに型推論した上で単一 GPU および複数 GPU を対象とした CUDA コードを生成することを可能にした。

3．並列分散処理向け実行時検証器 HPCUnit
概要：
大規模な並列分散プログラムでは、性能向上のため、個々の計算順序を手作業で入れ替えるが、この入れ替えによって一部の計算が実行されなくなっていないか、あるいは二重に実行されていないか、を実行後に確認するシステムを開発した。このシステムを用いると、ドメイン専用言語を用いて検査したい計算の内容を記述するだけで、それが自動的に実行時に検査

され、実行後に検査結果がレポートされる。このシステムを用いて実際に既存のプログラムの誤りを発見し、有用性を実証した。

§２ 研究実施体制

(1)研究チームの体制について

① 「千葉」グループ

研究参加者

氏名	所属	役職	研究参加期間			
			開始		終了	
			年	月	年	月
千葉　滋	東京大学	教授	23	10	30	3
佐藤　芳樹	東京大学	特任講師	24	7	28	3
莊　永裕	台湾国立中央大学	助理教授	24	4	30	3
伊尾木　将之	東京工業大学	博士課程学生	23	10	25	9
赤井　駿平	東京工業大学	博士課程学生	24	4	25	3
武山　文信	東京工業大学	学術支援職員	24	4	26	3
Maximilian Scherr	東京大学	学術支援職員	25	4	28	3
市川　和央	東京大学	学術支援職員	25	4	30	3
山口　洋	東京大学	博士課程学生	25	4	28	3
夏　澄彦	東京大学	修士課程学生	26	4	27	3

研究項目

- スーパーコンピューティングのための静的言語処理系

② 「増原」グループ

研究参加者

氏名	所属	役職	研究参加期間			
			開始		終了	
			年	月	年	月
増原　英彦	東京工業大学	教授	23	10	29	3
当山　学	東京大学		23	10	26	3
青谷　知幸	東京工業大学	助教	25	4	29	3
華井雅俊	東京工業大学	博士課程学生	26	4	28	3
朝倉泉	東京工業大学	博士課程学生	26	4	29	3
Springer, Matthias	東京工業大学	博士課程学生	27	10	29	3

研究項目
・ スーパーコンピューティングのための動的言語処理系
「鵜林」グループ
研究参加者

氏名	所属	役職	研究参加期間			
			開始		終了	
			年	月	年	月
鵜林 尚靖	九州大学	教授	23	10	29	3
久住 憲嗣	九州大学	准教授	23	10	29	3
亀井 靖高	九州大学	助教	23	10	29	3

研究項目
・スーパーコンピューティングによるリポジトリマイニング

③ 「五十嵐」グループ
研究参加者

氏名	所属	役職	研究参加期間			
			開始		終了	
			年	月	年	月
五十嵐 淳	京都大学	教授	23	10	29	3
小島 健介	京都大学	研究員	24	4	29	3
奥村 健太郎	京都大学	博士課程学生	23	10	29	3
Qi Tan	京都大学	博士課程学生	26	4	28	9

研究項目
・スーパーコンピューティングのための検証技術

(2)国内外の研究者や産業界等との連携によるネットワーク形成の状況について

分散並列処理を用いた社会シミュレーションの研究をしている同一領域の野田チームと連携し、我々の研究を適用すべきアプリケーションとして社会シミュレーションを取り上げて研究した。また我々のソフトウェア技術を野田チームに提供し、野田チームの研究を支援した。

ドイツ The University of Potsdam、Hasso Plattner Institute の Robert Hirschfeld 教授、SUNY Binghamton の Yu David Liu 准教授、スイス Universita della Svizzera italiana の Walter Binder 教授らとの研究協力をすすめ、各氏を当研究チームの研究拠点に招き、研究ネットワークの形成に努めた。

また日独共同研究プロジェクトである SPPEXA に参加し、ExaStencil 研究チームのメンバーとして共同研究をおこない、同様に研究ネットワークの形成に努めた。

§３ 研究実施内容及び成果

３．１ スーパーコンピューティングのための静的言語処理系（東京大学　千葉グループ）

(1)研究実施内容及び成果

千葉グループは静的言語をホストとするドメイン専用言語を主たる研究対象として研究を進めたが、同時にチームの中核として、他のグループとの連携にも力を注いだ。検証技術については静的検証を研究する京大の五十嵐グループに対し、補完する研究として千葉グループは実行時検証の研究をおこなった。また九大の鵜林グループとはスーパーコンピュータの利用の面で技術的な協力をおこなった。またチーム外の連携として、同じ領域の野田チームと連携し、野田チームのシミュレータの並列化技術について研究協力をおこなった。

① Java 言語向け埋め込み型ドメイン専用言語開発基盤 Bytespresso, WootinJ

Java 言語をホスト言語として、MPI、GPU を用いた高性能計算向けドメイン専用言語を埋め込み型で開発するためのソフトウェア基盤 Bytespresso, WootinJ を開発した。これらは埋め込まれたドメイン専用言語のプログラムを抜き出し、効率の良い C あるいは CUDA プログラムに変換して実行するための処理系である。これにより例えば、分散ノード型のスーパーコンピュータ上の行列計算を行列オブジェクトを用いて抽象度高く記述できるドメイン専用言語を、Java 言語埋め込み型の言語として容易に実現できるようになった。ドメイン専用言語の利用者は、特に必要のない場合は分散メモリの存在を特に意識せずに計算を記述することができる。またそのようなドメイン専用言語で書かれたプログラムは、従来型の手法により実行速度重視で開発されたプログラムに匹敵する性能を達成できる。我々はまず WootinJ と呼ぶ処理系をプロトタイプとして開発し、後にその開発を通して得られた知見をもとに、Bytespresso をより実用性の高い処理系として開発した。

Bytespresso は埋め込み型ドメイン専用言語のための開発基盤であるので、Bytespresso 上に作られたドメイン専用言語のプログラムは Java 言語に埋め込まれる形で記述される。従来、埋め込まれたプログラムはコンパイル時に抜き出されて外部の専用コンパイラでコンパイルされるか、あるいは Java 言語のプログラムとして Java コンパイラによってコンパイルされるか、であった。後者の場合、Java 仮想機械によって実行されるのでプログラムの意味は Java 言語と同じように解釈される。ドメイン専用言語とはいっても、それで書かれたプログラムの実態は、見かけが専用言語であるかのような API (Application Programming Interface) をもったライブラリを使った Java プログラムである。Bytespresso の場合、埋め込まれたプログラムは実行時に取り出されるのが特徴である。そのとき、埋め込まれたプログラムが参照する外側の Java オブジェクトも、あたかもクロージャのように一緒に取り出される。ドメイン専用言語の処理系は、取り出されたプログラムと Java オブジェクトを用いて、その取り出されたプログラムが意味するところを独自の解釈で実行することができる。あるいは、その意味する処理を実行する C あるいは CUDA プログラムを生成して、外部のコンパイラでコンパイル、実行することができる。このような構成は、Java 仮想機械上で動く Java プログラムの一部を CUDA プログラムに変換して実行する、いわゆるオフローダーの構成とよく似ている。

ドメイン専用言語を Bytespresso の上に作る場合、必要な作業は計算の詳細を抽象化した Java クラスのライブラリを提供することだけである。Bytespresso はそのような Java クラスを、通常の Java の意味論にしたがってほぼ等価な C プログラムに変換する。ドメイン専用言語の開発者は、ライブラリのクラスにアノテーションを加えたり、Bytespresso が提供する特別なクラスを利用したりすることで、変換後の C プログラムを高速動作するものにすることができる。例えば Bytespresso は DoubleArray2D というクラスを提供するが、このクラスは C 言語の2次元配列を利用するためのクラスである。このオブジェクトは変換後 double 型の2次元配列となり、要素にアクセスするための get メソッドの呼び出しは、直接要素にアクセスする C 言語の a[i][j] のような式に変換される。また

Bytespresso は MPI クラスも提供しており、例えば MPI.isend メソッドの呼び出しは、MPI_Isend 関数の呼び出しに変換される。

我々は Bytespresso の記述力、実行時性能を調べるため、いくつかの簡単なドメイン専用言語、つまり Bytespresso 上の Java クラスライブラリ、を作成した。例えば MPI 環境向けのベクトル行列ライブラリを用いると、分散メモリ型のスーパーコンピュータ上の計算を、メモリの分散を気にせずプログラムに書けるようになる。例えば以下の図 3.1.1 は NAS Parallel Benchmark CG を Bytespresso 用のライブラリを用いて書き直したものの一部である。

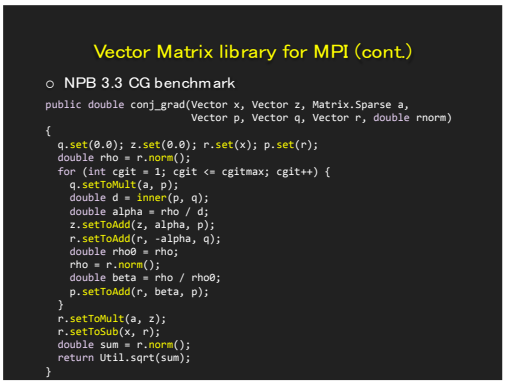

図 3.1.1 Bytespresso を使ったプログラム例

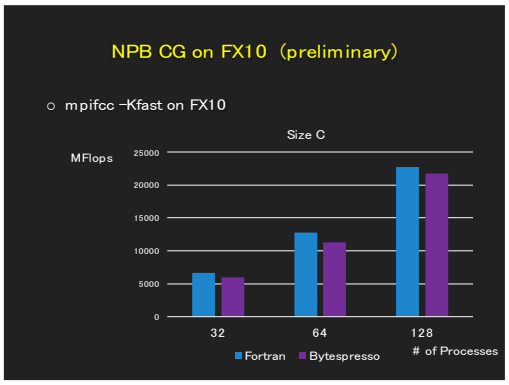

図 3.1.2 CG ベンチマークの実行性能

元の Fortran プログラムでは行列やベクトルは配列を使って表現されているが、このプログラムでは Vector オブジェクトや Matrix オブジェクトを用いて表現されている。分散メモリに分割して配置された行列とベクトルの積も setToMult メソッドを呼ぶだけである。そのような簡潔なプログラムであるが、図 3.1.2 に示すように実行時性能が元の Fortran プログラムと比べても大きく劣るわけではなく、十分に実用的な性能がでているといえる。

② 並列分散処理向け実行時検証器 HPCUnit

分散並列型のスーパーコンピュータ向けの典型的な科学技術計算のプログラムでは、反復処理の計算空間を分割したり、その処理順序を変更することで、ノード間通信の遅延の隠蔽、キャッシュヒット率の改善などの最適化実施し、実効性能を高めている。しかしながら、計算空間の分割により、分割された部分空間の境界判定などがプログラム中で必要になり、プログラムの誤りの温床になっている。例えば計算空間内の一部の領域に対して計算が行われない、あるいはその領域に対して同じ計算が二度以上実行されてしまう、などの誤りを引き起こす。

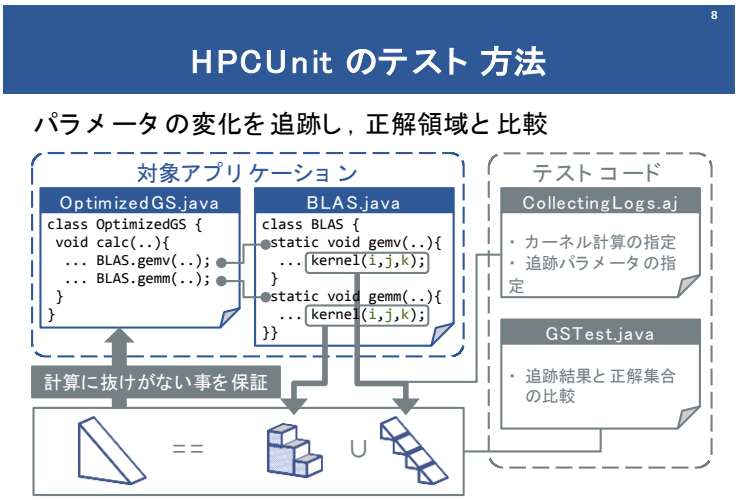

図 3.1.3 HPCUnit

そこで、そのような誤りが起きていないか、プログラムの実行時に検査をおこなう実行時検証器 HPCUnit を開発した(図 3.1.3)。この検査器は実行時に、計算空間内の各要素に対する計算の実行を記録し、目的とする計算が各要素に対して1回ずつ実行されているかをテストする。我々はこれを calculation coverage tests と名付けた。HPCUnit は Java で書かれた大規模分散並列プログラムを対象とする。HPCUnit が提供する Java 言語の埋め込み型ドメイン専用言語を用いて、計算空間の要素に適用される計算を表現するメソッドや、AspectJ 言語風のポイントカット記述で実行時にどのようなデータを記録するかを宣言的に指定する。例えば MPI のランクやスレッド識別子、またその要素の空間内での位置などを記録できる。さらに得られた記録を実行後に集合演算を用いて検査することで正しく計算がおこなわれたかを調べることができる。ステンシル計算における袖領域の存在や、分散メモリの存在により、それらの検査は必ずしも容易ではない。HPCUnit によって検査の記述を簡素化し、検査に必要なデータを記録するためのコードの挿入も自動化できた。我々は HPCUnit をよく知られたベンチマークテストに適用し、あるベンチマークプログラムに計算が二重に実行される誤りがあることを実際に発見した。

③ 埋め込み型ドメイン専用言語のための拡張可能構文機構

Java 言語などのプログラムの中に、埋め込む形で使われるドメイン専用言語を埋め込み型ドメイン専用言語という。この中でも特に、ホスト言語のライブラリとして実現されるドメイン専用言語は、開発が容易であり、小規模なドメインであっても開発してアプリケーション・プログラムの開発に役立てることが費用対効果の面からも可能である。しかしあくまでホスト言語のライブラリであるので、ホスト言語の構文の範囲内でしかドメイン専用言語のプログラムを記述できず、表現力の面で難がある。

我々はこの問題を軽減するために、記号を含む任意の文字列を演算子としてユーザが定義できる言語機構を開発し、それを用いることでホスト言語の文法にしばられない自由な構文をもったドメイン専用言語をホスト言語のライブラリとして容易に開発できるようにした。またそれを実証するために Java 言語でその言語機構を使えるようにした言語処理系 ProteaJ を開発、公開した。ProteaJ を使うと、通常 Java 言語では不可能なような構文をもつライブラリを作成できる。例えば

```
readas Regex r "+" a? (Regex r, Anno a = Anno.greedy)
  : priority = 250 {
      return new RegexPlus(r, a);
  }
```

```
 readas Anno "+" () : priority = 300 {
   return Anno.possessive;
}
```

のような定義をライブラリ中に書けば

```
using RegexOperators;
  …
Regex r = hel+o;
```

のようなプログラムを利用者が書くことができる。このライブラリは正規表現を使うためのライブラリである。上のプログラムを実行すると、Regex 型の変数 r に“hel+o”という正規表現を表すオブジェクトが代入される。通常の Java 言語では、上のようなプログラムを書くことはできず、例えば次のようなプログラムになってしまう。

```
Regex r = new Regex.make(“hel+o”);
```

右辺に正規表現のパターン hel+o だけを書いてすませるわけにはいかず、全体に冗長な表現になってしまう。

通常の Java 言語では右辺に正規表現のパターンだけを書くわけにはいかないのは、hel+o は変数 hel と変数 o の加算 + と見なすこともできるからである。当然 ProteaJ では、hel+o が正規表現なのか、加算なのか、判別しなければならない。ProteaJ では式に要求される静的な型を使ってこれを判別する。上の例では、代入 = の左辺は Regex 型の変数 r なので、右辺の値も Regex 型の値であることが求められる。その場合、ProteaJ は Regex 型の値を戻り値とする演算子を優先的に選択するので、意図したとおりの意味となる。ProteaJ は、従来技法では構文解析が終わった後に決定される静的型を、構文解析をしている途中に決定していき、それを用いて構文解析をする。従来技法で上のような例を処理しようとすると、まず hel+o が正規表現の場合と加算の場合と両方の場合を想定して構文木を2種類作り、その後、意味解析によって望ましい方を選択する。このような技法では、演算子の組み合わせによっては可能性がある構文木の数が指数爆発するので、実用的ではない。図 3.1.4 に示すように、場合によっては入力プログラムの長さに対して構文解析時間が指数関数的に増えてしまう。

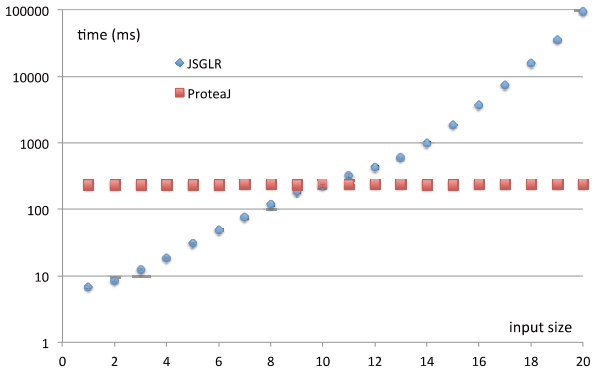

図 3.1.4 ProteaJ と JSGLR 構文解析器との速度差

④ Java 言語実行系 OpenJDK の移植

我々は開発したソフトウェア基盤、ツールを様々なスーパーコンピュータ上で動かせるようにするた

め、Oracle 社の Java 言語実行系 OpenJDK を、K computer および富士通 FX10 向けに移植した。移植の成果は富士通株式会社に提供され、同社によって K computer および FX10 上で Java 言語が一般ユーザからも利用可能になった。

④ エージェント・シミュレーションのための並列処理

社会シミュレーションの研究をおこなっている同じ研究領域の野田チームと研究協力を進め、同チームがもつ社会シミュレーションのソフトウェアの並列化をおこない、その知見をもとにエージェント・シミュレーションを効率よく並列実行するための言語機構の研究をおこなった。野田チームでは災害時における地域住民の避難経路の検討をおこなうためのシミュレーション・ソフトウェアを研究開発している(図 3.1.5)。社会科学者にとって自然な形でそのようなシミュレーションを記述しようとすると、避難民をエージェントとして、その動きをプログラムの形で記述することになる。それを並列実行しようとすると、各エージェントにスレッドを1つ作り動かすようにしたくなるが、これはエージェント数が多くなると並列度が過剰になり実行時性能が低下しがちで、またメモリを大量に消費するので大規模な並列化にそぐわないことがある。

図 3.1.5 避難シミュレータ CrowdWalk

これを避けるためには、社会科学者がプログラムを書く際に、エージェントの動作を複数に分割する、エージェント間の動作の進捗を調整するバリア同期のタイミングを工夫する、などして、過剰な並列化をおさえて効率よくプログラムが動くようにしなければならない。あるいは避難民一人一人をエージェントと考えてその動きをプログラムするという直感的な方法をあきらめ、地域をいくつかの領域に分割して、領域ごとにスレッドを割り当てて並列計算する、という並列計算機に習熟したソフトウェア技術者であれば当然であるが、社会科学者にとっては必ずしも直感的ではない方法でプログラムを書かなければならない。

我々はユーザである社会科学者ができるだけ負担なくシミュレーションのプログラムを書けるよう、新たに軽量で抽象度の高い条件付バリア同期機構 waitless を開発した。これを用いることで、社会科学者は避難民一人一人の動作をエージェントとしてプログラムしても、比較的大規模な並列処理を効率よく行えるようになる。この研究は、アプリケーション・プログラムの書き手の負担を減らすための研究である。Waitless は Java 言語向けのバリア同期ライブラリであり、抽象度の高い all-to-all 型の記述でありながら point-to-point 型(あるいはグループ型)に近い柔軟な同期制御が可能になる。Waitless では、オブジェクトの状態変更に対する条件式としてバリア同期を記述させることで、同期処理を他から分離し、散在する同期のためのコードを一カ所に集約できる。Waitless を用いたプログラムはロード時にプログラム変換され、同期のためのコードが必要な箇所全てに自動的に挿入される。またバリア同期の実行前後でプログラムが自動的に分割され、無駄な並列化をおさえてスケーラブルな性能を達成する。その一方、変換前のプログラム、すなわち利用者が書くプログラムでは同期処理が一カ所にまとまって記述されるので、モジュラリティがよく、保守しやすいプログラムとなる。

Waitless を用いると、従来なら手動で施さなければならなかった高速化のための技法が自動的に

適用されるので、プログラムを書くことは容易になる。一方、図 3.1.6, 3.1.7 に示すように waitless によって得られる性能改善で、他の同期機構を使って手動で性能を改善した場合と同等の性能改善が得られることがわかった。ポストペタスケール時代には、アプリケーション・プログラムを書く際の難易度は、現在よりもさらに高くなると思われるが、本研究はそのような難易度を下げる技術のひとつとして期待できる。

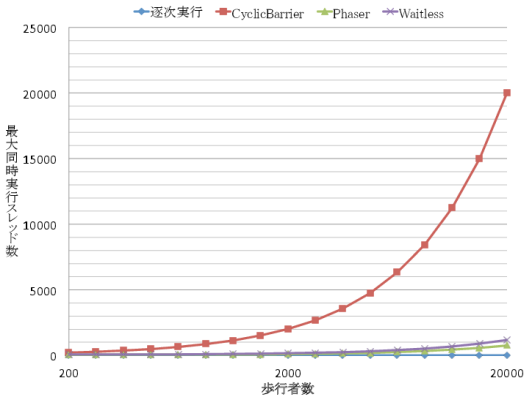

図 3.1.6 最大同時実行スレッド数

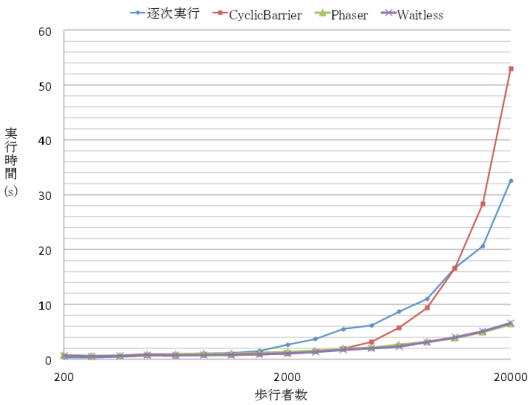

図 3.1.7 実行時間

3．2 スーパーコンピューティングのための動的言語処理系
（東京工業大学 増原グループ）
(1)研究実施内容及び成果

増原グループも、千葉グループ同様に並列計算を行うためのソフトウェア開発環境・基盤を研究したが、千葉グループとは異なり、動的言語である Ruby 言語に基づいた開発環境・基盤を研究した。動的言語は、型情報が静的に利用できない代わりに、より柔軟にソフトウェアの改変が可能であり、それによって優れたソフトウェア生産性を達成できからである。特に Ruby 言語は組み込み型ドメイン専用言語に適した非常に柔軟な構文を提供することで知られる。mた生物学をはじめとして今後 HPC 技術の利用が高まると思われるアプリケーション分野では、Ruby のような静的に型付けされない言語が好んで用いられている。このため Ruby 言語を対象とする変換器を開発することは実用上の意義が大きい。増原グループは、動的言語の望ましい性質を発展させつつ、静的型情報が得られないことによる性能劣化を型情報を静的に推測する技術を駆使することでおさえる研究をおこなった。

① Ruby 向け GPGPU プログラミングのためのプログラミング基盤

Ruby 言語で GPGPU プログラミングをおこなうための言語基盤 Ikra を研究開発した。まず Ruby 言語そのものを使ってプロトタイプを開発し、その後、プロトタイプ開発の知見を元に関数型言語 OCaml を用いて実用的に動作する版のソフトウェアを開発した。動的言語のための型推論系を、いわゆる soft typing を拡張する形で研究開発し、それを用いることで、基本的なステンシル計算をおこなう Ruby プログラムに対し型推論により各式に静的に型をつけ、それを単一 GPU および複数 GPU を対象とした実行効率の良い CUDA コードに変換することに成功した。またオブジェクト指向機能をサポートするためのコード生成手法および GPGPU を対象とした最適化手法を研究し、Ikra に組み入れた。作成した Ikra 処理系は Ruby 言語のパッケージ集の一部として https://rubygems.org/gems/ikra にて公開している。

またデータ並列実行 DSL におけるオブジェクトサポート方法の研究をおこない、特にエージェントシミュレーションのような (1) スレッドごとに振舞いが異なる可能性があり、(2) 多くの属性を持った

オブジェクトに対してデータ並列的な計算を行う状況における GPGPU 実行の性能について研究した。実験から（1）に関してはワープ発散による性能低下が問題となり得ること、(2) に関しては素朴な実現では memory coalescing が阻害されることを明らかにした。この知見に基づき、ワープ発散を低減するためのスレッド割当て方式と structure-of-arrays 型レイアウト方式を可能にするための言語処理系の設計と実現を行った。これによって、従来からワープ発散が多発するような問題も、効率的なスレッド割当て要求をオブジェクト指向的なモデル化で表現できるようになった。

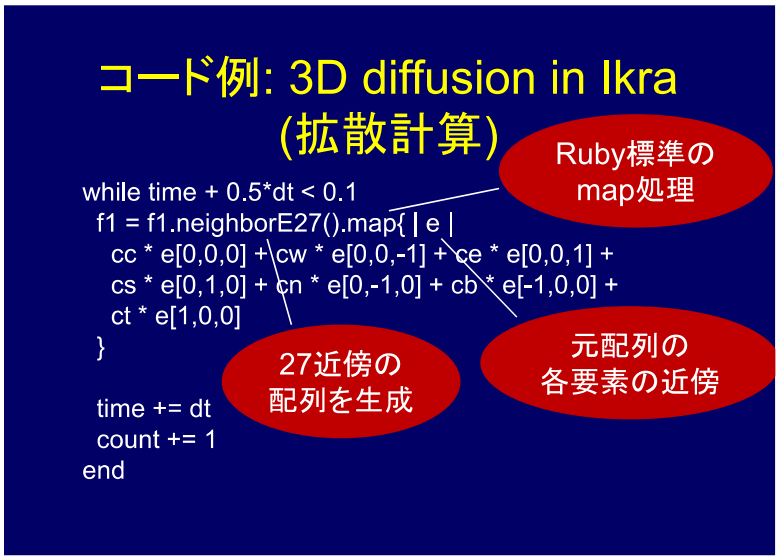

図 3.2.1 Ikra のコード例

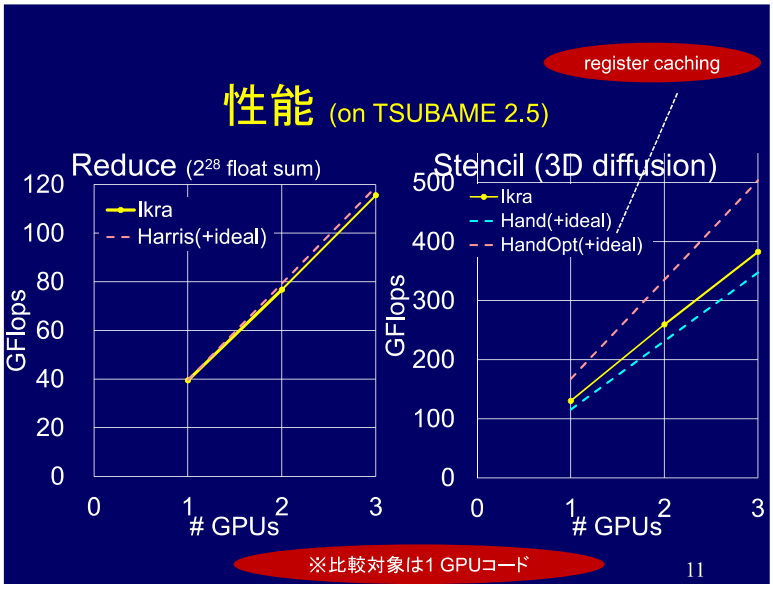

図 3.2.2 Ikra の実行時性能

② 文脈試行によるソフトウェアのモジュール機構の基礎研究

動的に振舞いを変えるソフトウェアのモジュール性を向上させるための言語技術として文脈指向プログラミングに関する基礎的な研究をおこなった。Bytespresso や Ikra のようなライブラリに基づくドメイン専用言語の構築を進めていくと、言語処理系内部を容易かつ安全に拡張可能とするためのモ

ジュール化機構がますます必要になる。そこで、プログラム中の複数のモジュールにまたがった制約・検査・記録・連携などの処理をモジュール化するためのアスペクト指向プログラミングと、実行中にプログラムの振舞を変化させる場合に限定した文脈指向プログラミングについて基礎研究を進めた。前者に関しては特に、モジュールの型を変更する場合の型安全性と分割コンパイルについて、後者に関しては計算モデルの整理についての研究成果を得た。またこれらに加え、並列離散事象シミュレーションの分散メモリ実行フレームワークに関して、time-warp アルゴリズムを利用した新しい差分実行方式を考案した。

3．3 スーパーコンピューティングのリポジトリマイニングへの応用

(九州大学 鵜林グループ)

(1)研究実施内容及び成果

昨今、ソフトウェア工学研究ではビッグデータ解析を用いた実証的研究が主流になっている。全世界のオープンソースソフトウェア開発プロジェクトで蓄積された大規模リポジトリデータを高速かつ効率良く解析するための環境作りは喫緊の課題である。鵜林グループでは、スーパーコンピュータを用いた大規模な解析をソフトウェアリポジトリマイニング(MSR, Mining Software Repogitories)の分野における重要な新しい研究手法とするべく環境整備を行った。さらに、スーパーコンピュータ FX10 上で大規模なソフトウェアリポジトリマイニングを実際に実施した。以下に主な成果を示す。

① ソフトウェアエンジアリング向け HPC 環境の整備

MSR 向けシェル (Shell) Argile を開発した。Argyle は図 3.3.1 に示すような MSR 支援に特化した DSL で、スーパーコンピュータ上でのソフトウェアリポジトリマイニングを容易にし、HPC 技術の利用を促進する。リポジトリからのデータの取り込み、データの複数ノードへの自動振り分けと実行など、MSR 共通の機能を提供する。さらに、ソフトウェア欠陥予測、メーリングリスト解析・ソーシャルネットワーク解析、コードクローン解析などリポジトリマイニング向け部品群をパッケージングしている。機能拡張が可能であり、Argyle に新規に MSR 向けのコマンドを追加することが可能である。ソフトウェア工学の研究者は必ずしもスーパーコンピューティングの専門家ではない。そのため、Argyle では、既存の MSR ツールをできるだけ少ない手間でスパコンを使って如何にスケールさせるかという点に力点を置いている。

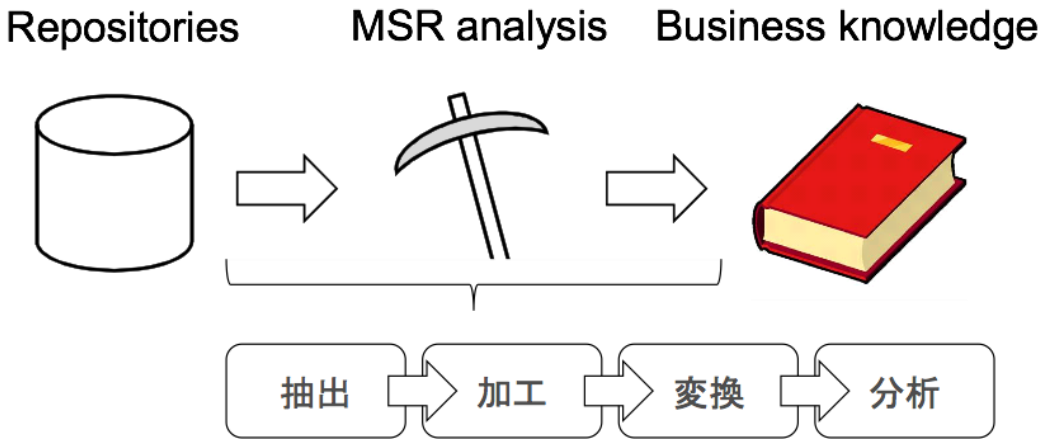

図 3.3.1 MSR のプロセス

②コードクローン解析を対象とした大規模実証実験

ソフトウェアリポジトリマイニングの分野でも最も解析コストの高いコードクローン解析（図 3.3.2）を対象に大規模な実証実験を実施した。コードクローン検出アルゴリズムの中でも最も計算負荷の高いタイプ 3（図 3.3.3 に示すように抽象構文木に対してコードクローン検出をおこなう）を、代表的な大規模ソフトウェアリポジトリの一つである Apache 全 131 プロジェクト（700 万ソースファイル以上）に適用した。コードクローン解析器として Java で記述されたソフトウェア Scorpio（大阪大学で開発）を用い、図 3.3.4 に示すように、FX10 上で並列分散処理をするためにファイル群の分割の仕方などを検討した上で実験した。この実験には、既存のスパコンで Apache プロジェクト群がどこまで解析できるのか、という目的がある。

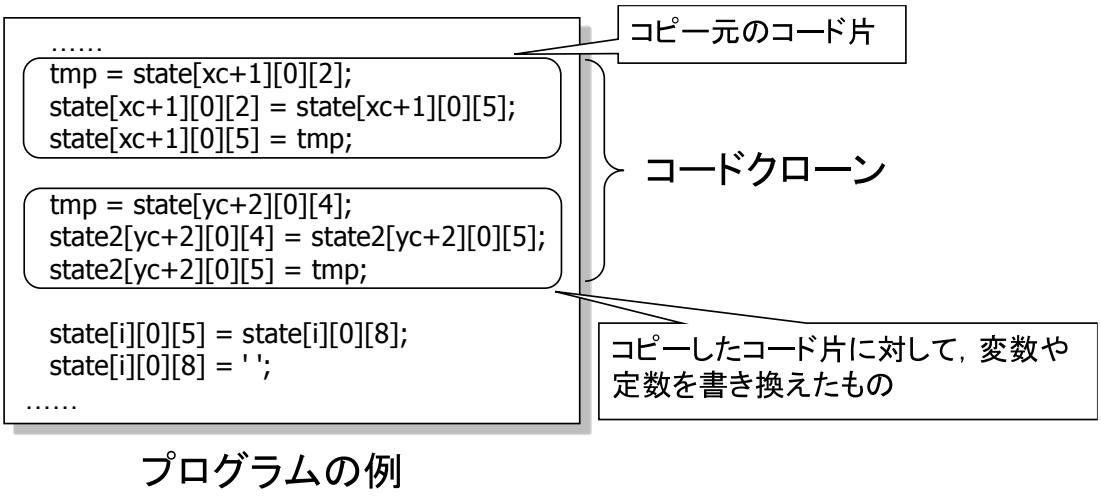

図 3.3.2 コードクローンとは

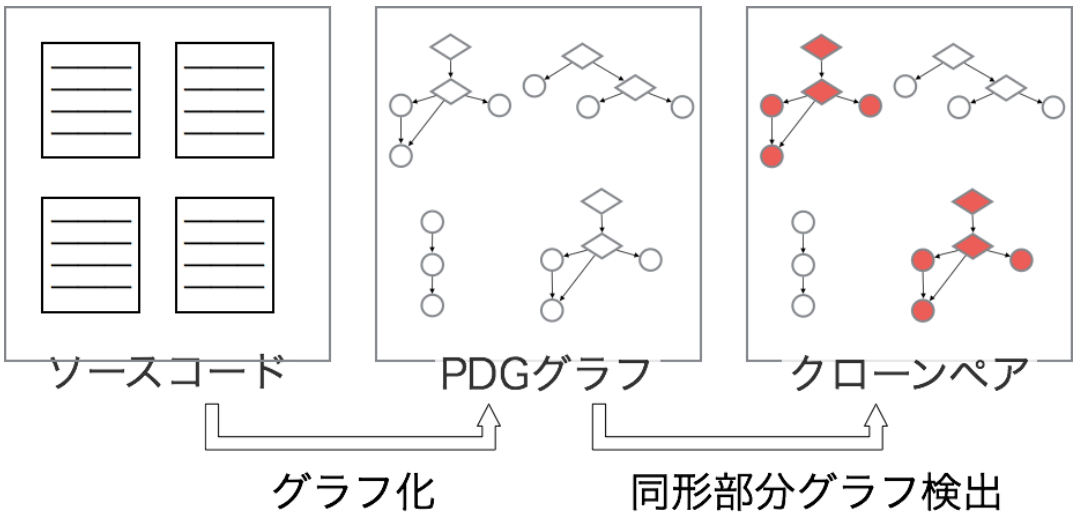

図 3.3.3 コードクローン解析の手順

(1) プロジェクト単位の解析

プロジェクト単位のコードクローン解析は、同じプロジェクト内に同じコード片がないかを解析するため、ソフトウェアの保守性を調べる際に有効である。該当するコード片にバグが存在すると、他のコード片も同様に修正しなければならないからである。このような修正はコストがかかる上に、修正ミスを誘発しやすい。コードクローン解析を活用することにより、このような問題を未然に防ぐことが可能となる。

図 3.3.5 に示すように、Apache131 プロジェクト中、約 8 割のプロジェクトが 1 ノードのみで解析できた。36 ノードを用いると、最大サイズのプロジェクトでも解析を行うことができた。図 5 では、コードサイズ（単位はバイト）の小さい順にソートして結果を表示している。

(2) プロジェクト間の解析

コードクローン解析にはプロジェクト間にまたがる解析がある。応用領域として、コード盗用の検出（あるプロジェクトのコードが他のプロジェクトで無断に使用されているケース）、コード記述パターンの検出（多くのプロジェクトで頻出するコーディングパターン）などがある。後者は、ソフトウェア工学研究の実用化という観点から非常に重要である。Apache プロジェクトなどのオープンソースソフトウェアプロジェクトでは、様々な開発ノウハウが暗黙的な群衆知（Crowd Knowledge）として蓄積されている。コードクローン解析を用いることにより、コーディングパターンなどの明示的な群衆知が生成できる可能性がある。コーディングパターンの具体例として、API 利用のコーディング方法、プログラミングミスの典型例などがある。後者については、リポジトリマイニングの一分野である ITS（Issue Tracking Systems）の解析とコードクローン解析を組み合わせることにより検出することが可能である。

プロジェクト単位のコードクローン解析は、図 3.3.5 に示したように既存の HPC 環境で解析可能であるが、プロジェクト間解析については、計算量が膨大になるため、既存の HPC 環境では全世界のオープンソースソフトウェアの解析をすることは不可能である。Apache プロジェクトですら難しいと考えられる。そこで、鵜林グループでは、既存の HPC 環境でどこまで解析できるのか、どこに限界があるのかを調べた。その結果、プロジェクト間分析については、図 3.3.6 に示すように、190 ノードを用いて、93 プロジェクト（サイズが小さい順）までが解析できることが判明した（制限時間は 24 時間に設定）。190 ノードに限定したのは、計算ジョブの待ち時間を考慮してのことである。一般の研究者や技術者がスパコンを利用する際の状況に近い形で実験した。Apache プロジェクトを小さい順に並べて 93 プロジェクトまで解析できたので、かなりの規模で解析できたと言える。しかしながら、全 131 プロジェクト丸ごとのコードクローン解析には至らず、ソフトウェア工学研究サイドからすると現状の HPC 環境は必ずしも満足の行くものではないと言える。現在、公開されている最大規模のリポジトリとして、UCI（University of California, Irvine）Source Code データセット（プロジェクト数は 13,000 プロジェクト、対象言語は Java、総行数が約 4 億行）がある。また、Github には 6,000 を超える開発プロジェクトが登録されている。Apache プロジェクトと比較すると桁違いに大きい。

ソフトウェア工学研究における HPC 活用の事例は極めて少ない。その一方で、上記で述べたように、解決しなければならない問題は極めて大きい。今回、鵜林グループでは、実際の HPC 環境を用いた実証実験を実施したが、このような研究事例は世界的にみてもほとんど存在しない。その意味で、今回の実証実験の意義は大きい。今後の研究の発展の方向として、ヒューリスティックを用いたコードクローン解析の高速化、解析規模の拡大などが重要だと考えている。実際、この数年の間に、ソフトウェア工学の研究コミュニティでは、コードクローン解析をスケールして解析するヒューリ

スティックがいくつか提案されている。今後、これらを Argyle に組み入れていくことが考えられる。

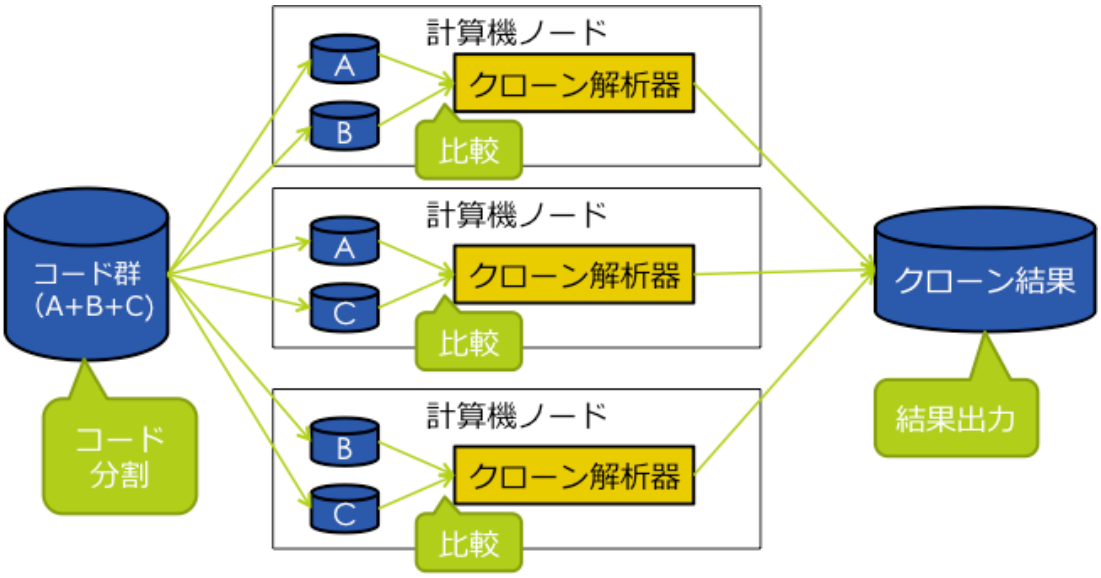

図 3.3.4 並列分散によるコードクローン解析

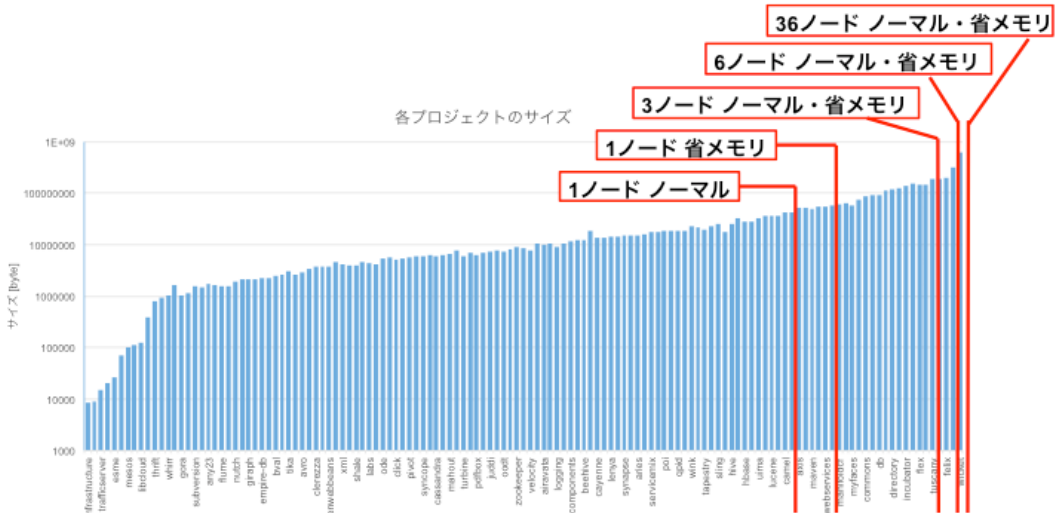

図 3.3.5 タイプ 3 の解析結果(プロジェクト単位)

• 93プロジェクトまでは解析できた

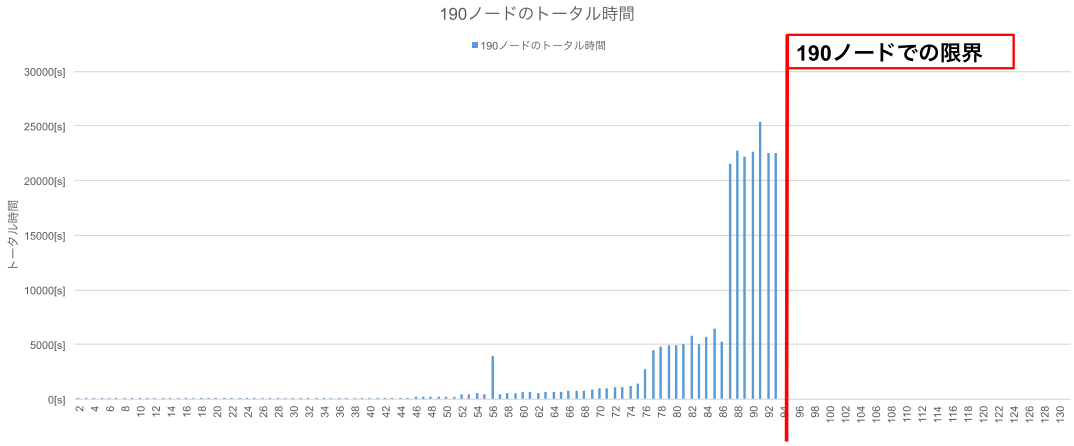

図 3.3.6　タイプ 3 の解析結果(プロジェクト間解析)

3. 4　スーパーコンピューティングのための検証技術
(京都大学　五十嵐グループ)
(1)研究実施内容及び成果

典型的な HPC プログラムである科学技術計算は、計算内容は数学的に簡潔に記述できるものであるが、ハードウェアの性能を引き出すために、プログラム上は非常に難解な記述になることが多い。そのような最適化された複雑・難解なプログラムが、正しく意図された計算を実行していること確認するためには、機械的な検証技術がかかせない。そこで我々は、ホーア論理の考えを適用することで、実行が高速になるよう最適化された並列プログラムの正しさを検証する検証器の開発を進めてきた。

ホーア論理は、命令型プログラムを検証するための論理体系で、プログラムが仕様ーこれはプログラムの入力に関する条件(事前条件)とプログラムの出力に関する条件(事後条件)として与えられるーを満たすかどうかを検証するための枠組みである。プログラムの構文毎に、事前条件・プログラム・事後条件の三つ組が満たす関係が公理として与えられており、これらを組み合わせて大きなプログラムについての証明を行う。証明が完成すれば、そのプログラムは事前条件を満たす任意の入力について(停止すれば)事後条件を満たす出力が得られることが理論的に保証されている。ホーア論理は、任意の入力について網羅的に、かつ、プログラムを実行することなく検査することができるので、代表的とはいえ限定されたケースに限られるテストなどに比べて、よりプログラムの品質を高いレベルで保証することができる。一方で、自動でホーア論理の証明を構築することは難しい。逐次プログラムについては技術が進んできて、ある程度大規模なプログラムを計算機で自動的に検証することができるようになってきたものの、並列プログラムについては難しかった。

我々は、並列プログラムの中でも GPGPU プログラムのカーネル関数(プログラムのうち、CPU ではなく、GPU で実行される部分)に着目し、ホーア論理による自動検証器の構築を目指して研究を行ってきた。検証対象として GPGPU プログラムのカーネル関数に着目したのは、カーネル関数が比較的小規模であることもあるが、GPGPU プログラムのロックステップ実行(複数スレッドが同一命令を同じタイミングで実行すること)のおかげでスレッド間干渉の可能性が限られ、スレッド毎に実行箇所が違うような一般の並列プログラムほどは検証が困難ではないだろうという見込みがあったためである。

研究は、まず GPGPU プログラムのためにホーア論理の理論を拡張することから始め、理論的の基

盤が整ったところで自動検証器の構築に取り組んだ。以下、理論的な研究と自動検証器の構築についてそれぞれ説明する。

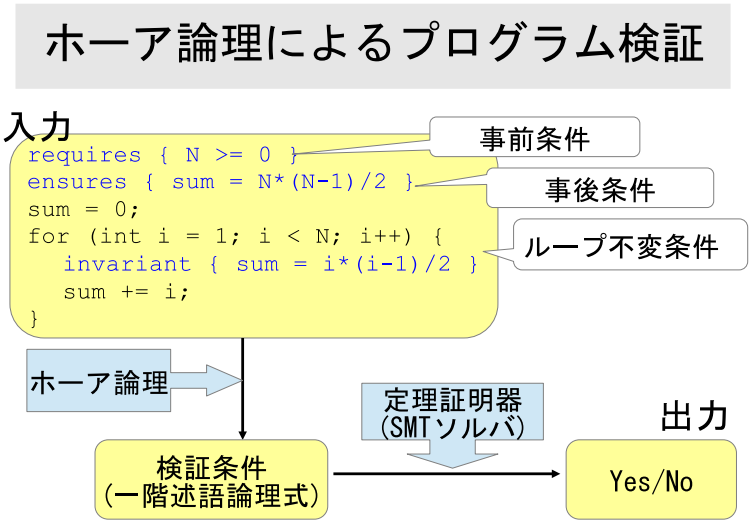

図 3.4.1 プログラム検証の流れ

① GPGPU プログラムのためのホーア論理

ホーア論理は、ホーア論理の公理系を与えることによって作られるが、その公理系が「まとも」でかつ「十分強力である」ことを示す必要がある。「まともさ」は、公理を使って得られた結論が実際のプログラムの実行に合致しているという健全性、「十分な強力さ」は、プログラムが実際にある仕様を満たすのであれば、そのことが公理系から証明できる、という相対完全性と呼ばれホーア論理の重要な理論的性質である。

我々は、まず、全てのスレッドがロックステップ実行を行うという仮定のもとでの GPGPU カーネルの実行過程の数学的モデルとホーア論理公理系を与え、その健全性・相対完全性を示した。これによりスレッド数が Warp のサイズを越えないような GPGPU プログラムについての検証の理論ができたことになる。次に、スレッドがロックステップ実行ではなくインターリーブ実行を行うモデルを構築し、ロックステップ実行モデルとの比較を行った。その結果、競合状態に陥らないプログラムについては、前スレッドがロックステップ実行を行うという計算モデルの下の実行結果と、全プロセスが interleave しながら実行を行う計算モデルの下の実行結果が一致することを数学的な定理として証明することができた。これによって、競合状態がない、という仮定つきながら、ロックステップ実行と interleave 実行が混在する現実的な GPGPU プログラムについても、検証の理論が構築できた。

一般に、並列プログラムが競合状態に陥らないことを確かめるのは簡単ではないが、GPU プログラムのための競合状態・バリア同期検出のためのツール GPUVerify がイギリス・インペリアル大のグループによって開発されているため、それらと組み合わせれば十分に意味のある検証を行うことができる。

② CUDA C プログラムのための半自動検証システム VeriCUDA

上記(1)の理論をもとに、CUDA C プログラムの検証システム VeriCUDA の構築を行った。検証シ

ステムは CUDA-C で記述したカーネル関数に仕様としての事前条件・事後条件、またループに関する条件を表すループ不変条件を注釈として付加したものを入力とし、ホーア論理の公理を使って証明ができるかを、(プログラムを実行せずに)判定するものである。内部的には、入力プログラムを解析して、それに対するホーア論理による証明が存在するための必要十分条件を一階述語論理式として得て、それを既存の SMT ソルバに与えて証明できるかを(複数のソルバを用いて)調べるという構造になっている。「半自動」とあるのは、論理式注釈は人手で付加する必要があるためで、一度注釈がついたものについては自動で検証が行われる。

具体的な対象プログラムとして、ブロック化や shared メモリの利用など、GPU のメモリ階層を考慮して高速化した行列積プログラム(CUDA C でコメント・仕様記述を除いて 30 行程度)を用いた実験を行った。図 3.4.1 にプログラムを示す。/*@ ... */ に囲まれた部分が仕様記述である。最初の仕様記述コメントにある requires ... が関数呼出し時の変数に関する事前条件で、ブロックサイズが 16 であるなどが指定されている。ensures が関数の実行終了時に成立することが想定されている事後条件で、配列 C の各要素が A と B の適当な部分の内積になっていることが記述されている。途中の for 文にはループ不変条件という各繰り返しの先頭で成立している条件が記述されている。

当初は、30 行のプログラムとはいえ検証に 1 時間程度かかっていたが、SMT ソルバに出力する論理式を前処理によって簡単化する方法を開発し、検証時間を数秒程度まで短縮することに成功している。その他、一次元の拡散方程式のステンシル計算などの検証に成功している。

また、本ツール VeriCUDA はオープンソースソフトウェアとして公開している。(https://github.com/SoftwareFoundationGroupAtKyotoU/Vericuda)

関連研究として、既に述べた GPUVerify のように特定の性質に限った検証器が開発されているが、行列積プログラムが本当に行列積を計算していることなどのプログラムの機能についての性質の検証が可能な検証器は、我々が知る限り初めてのものである。ただし、我々の検証器は対象プログラムには競合状態エラーやバリア同期エラーがないことを仮定しているので関連研究の検証器とは相補的な関係にある。

③ CUDA C プログラムの対話的検証のための Coq ライブラリ

上述の VeriCUDA は、プログラムに仕様の注釈を付加した後は人間が介在しないで自動で検証を行うシステムだが、より複雑なプログラムに関して自動証明を行うのは現状では困難な場合も多く、その場合には人間が介在する必要がある。そのような人間が介在するプログラム検証のためのツールとして証明支援系 Coq を用いて、CUDA プログラムの正当性証明を行うための Coq による検証ライブラリを構築した。

Coq は計算機上で形式的証明を記述するためのソフトウェアである。検証ライブラリは、CUDA C のサブセットの構文やホーア論理の公理系の定義、ホーア論理による証明が存在するための必要十分条件となる一階述語論理式を抽出するアルゴリズムなどからなっている。基本的にはユーザがホーア論理の証明を記述すると、それが本当に公理系に沿って記述されているかどうかを Coq が検査してくれるシステムとなっているが、抽出アルゴリズムを使うことでユーザは生成された条件の証明だけを行えばよいようになり、証明記述の負担を軽減することができる。この Coq ライブラリを使って、Bleloch による並列 prefix sum アルゴリズムの正当性証明を行い、このツールで、より複雑なプログラムの検証ができることを示した。

```
template <int BLOCK_SIZE> __global__ void
matrixMulCUDA(float *C, float *A, float *B, int wA, int wB) {
  /*@ ghost int m;
```

```
        ghost int hA;
        requires blockDim.x == 16;
        requires blockDim.y == 16;
        requires 0 <= m;
        requires wA == 16 * m;
        requires wB == gridDim.x * 16;
        requires hA == gridDim.y * 16;
        ensures forall i. forall j.
          0 <= i && i < hA && 0 <= j && j < wB ->
          C[wB * i + j] == sum(k, A[wA * i + k] * B[wB * k + j], 0, wA-1); */

  int bx = blockIdx.x;
  int by = blockIdx.y;
  int tx = threadIdx.x;
  int ty = threadIdx.y;
  int aBegin = wA * 16 * by;
  int aEnd = aBegin + wA - 1;
  int aStep = 16;
  int bBegin = 16 * bx;
  int bStep = 16 * wB;
  float Csub = 0;

  /*@ loop invariant loop_count <= m;
        loop invariant a == aBegin + aStep * loop_count;
        loop invariant b == bBegin + bStep * loop_count;
        loop invariant forall t : thread.
          Csub@t == sum(k, A[wA * (16 * by@t + ty@t) + k] * B[wB * k + 16 * bx@t + tx@t],
                        0, 16 * loop_count - 1); */
  for (int a = aBegin, b = bBegin; a <= aEnd; a += aStep, b += bStep) {
    __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
    __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

    As[ty][tx] = A[a + wA * ty + tx];
    Bs[ty][tx] = B[b + wB * ty + tx];
    __syncthreads();

#pragma unroll
    /*@ loop invariant k == loop_count;
          loop invariant loop_count <= 16;
          loop invariant forall t : thread. Csub@t ==
            sum(k, A[wA * (16 * by@t + ty@t) + k] * B[wB * k + 16 * bx@t + tx@t],
                0, 16 * loop_count_2 + loop_count - 1); */
    for (int k = 0; k < 16; ++k) {
      Csub += As[ty][k] * Bs[k][tx];
    }
    __syncthreads();
  }
  int c = wB * 16 * by + 16 * bx;
  C[c + wB * ty + tx] = Csub;
}
```

図 3.4.2　CUDA による行列積プログラム

§４ 成果発表等

（1)原著論文発表 （国内(和文)誌 7 件、国際（欧文）誌 51 件）

[1] 青谷知幸, 紙名哲生, 増原英彦, ″オブジェクト毎の層遷移を宣言的に記述できる文脈指向言語 EventCJ″, コンピュータソフトウェア, Vol.30, No.3, pp.130-147, 2013.

[2] Izumi Asakura, Hidehiko Masuhara, and Tomoyuki Aotani. ″Proof of soundness of concurrent separation logic for GPGPU in Coq.″ Journal of Information Processing, Vol. 23, No. 1, pp. 132-140, January 2016.

[3] Robert Hirschfeld, Hidehiko Masuhara, Atsushi Igarashi, and Tim Felgentreff. ″Visibility of context-oriented behavior and state in L.″ Computer Software, Vol. 32, No. 3, pp. 149-158, August 2015.

[4] 小須田 光, 亀井 靖高, 鵜林 尚靖,クラッシュレポートの送信頻度と不具合との関連付けに関する実証的評価,日本ソフトウェア科学会誌 コンピュータソフトウェア, vol.32, no.4, pp.131-140, 2015.

[5] 夏澄彦・佐藤芳樹・千葉滋、軽量で抽象度の高い条件付きバリア同期とその実装方法、情報処理学会論文誌プログラミング、vol.8 no.3、情報処理学会、pp.11-21、2015

[6] 板橋晟星・佐藤芳樹・千葉滋、並列分散処理向けプログラミング言語 X10 向けの対話的に>フィルタリング可能な挙動可視化ツールの開発、情報処理学会論文誌プログラミング、vol.8, no.3、情報処理学会、pp.22-32、2015

[7] Shumpei Akai and Shigeru Chiba, “Method Shelters: Avoiding Conflicts among Class Extensions Caused by Local Rebinding,” MODULARITY: aosd 2012, ACM, 2012.

[8] YungYu Zhuang, Shigeru Chiba, ″Method Slots: Supporting Methods, Events, and Advices by a Single Language Construct″, Modularity:AOSD13, pp.197-208, 2013, DOI 10.1145/2451436.2451460

[9] Masayuki Ioki, Shigeru Chiba , A Framework for Multiplatform HPC Applications, Proc. of Programming Models and Applications on Multicores and Manycores (PMAM2014 workshop), ACM, pp.61-69, Feb. 15, 2014, DOI: 10.1145/2560683.2560693

[10] Tetsuo Kamina, Tomoyuki Aotani, Hidehiko Masuhara, and Tetsuo Tamai, ″Context-oriented software engineering: A modularity vision″, In Proceedings of International Conference on Modularity (Modularity'14), Track on Modularity Visions, April 2014. DOI: 10.1145/2577080.2579816 (to appear)

[11] Tomoyuki Aotani, Tetsuo Kamina, and Hidehiko Masuhara, ″Context holders: Realizing multiple layer activation mechanisms in a single context-oriented language″, In Proceedings of the Workshop on Foundations of Aspect-Oriented Languages (FOAL'14), April 2014. DOI: 10.1145/2588548.2588552 (to appear)

[12] Robert Hirschfeld, Hidehiko Masuhara, and Atsushi Igarashi, ″L: Context-oriented programming with only layers″, In COP'13: Proceedings of the International Workshop on Context-Oriented Programming, Article No. 4, New York, NY, USA, 2013. ACM. DOI:

10.1145/2489793.2489797

[13] Tetsuo Kamina, Tomoyuki Aotani, and Hidehiko Masuhara, "A unified context activation mechanism", In COP'13: Proceedings of the International Workshop on Context-Oriented Programming, Article No. 2, New York, NY, USA, 2013. ACM. DOI: 10.1145/2489793.2489795

[14] Changyun Huang, Kazuhiro Yamashita, Yasutaka Kamei, Kenji Hisazumi, and Naoyasu Ubayashi: Domain Analysis for Mining Software Repositories ---Towards Feature-based DSL Construction---, 4th International Workshop on Product LinE Approaches in Software Engineering (PLEASE 2013), pp.41-44, 2013 (DOI: 10.1109/PLEASE.2013.6608663)

[15] Changyun Huang, Yasutaka Kamei, Kazuhiro Yamashita, and Naoyasu Ubayashi: Using Alloy to Support Feature-Based DSL Construction for Mining Software Repositories, 5th International Workshop on Model-driven Approaches in Software Product Line Engineering; 4th Workshop on Scalable Modeling Techniques for Software Product Lines (MAPLE/SCALE 2013), pp.86-89, 2013 (DOI: 10.1145/2499777.2500714)

[16] Kensuke Kojima and Atsushi Igarashi. A Hoare logic for SIMT programs. In Proceedings of the Asian Symposium on Programming Languages and Systems (APLAS2013), volume 8301 of Lecture Notes in Computer Science, pages 58-73, Melbourne, Australia, December 2013.
Springer-Verlag.

[17] Tetsuo Kamina, Tomoyuki Aotani, Hidehiko Masuhara, and Tetsuo Tamai, "Context-oriented software engineering: A modularity vision", In Proceedings of International Conference on Modularity (Modularity'14), Track on Modularity Visions, April 2014. DOI: 10.1145/2577080.2579816.

[18] Tomoyuki Aotani, Tetsuo Kamina, and Hidehiko Masuhara, "Context holders: Realizing multiple layer activation mechanisms in a single context-oriented language", In Proceedings of the Workshop on Foundations of Aspect-Oriented Languages (FOAL'14), April 2014. DOI: 10.1145/2588548.2588552.

[19] Kazuhiro Ichikawa and Shigeru Chiba, "Composable User-Defined Operators That Can Express User-Defined Literals," In Proc. of the 13th international conference on Modularity (MODULARITY '14), pp. 13-24, April 2014. DOI: 10.1145/2577080.2577092

[20] Maximilian Scherr and Shigeru Chiba, "Implicit Staging of EDSL Expressions: A Bridge between Shallow and Deep Embedding," In Proc. of the 28th European Conference on Object Oriented Programming (ECOOP 2014), LNCS 8586, Springer, pp. 385-410, 2014. DOI: 10.1007/978-3-662-44202-9_16

[21] Tomoyuki Aotani, Tetsuo Kamina, Hidehiko Masuhara, "Unifying Multiple Layer Activation Mechanisms Using One Event sequence", Proceedings of 6th International Workshop on Context-Oriented Programming, COP'14, pp. 2:1-2:6, 2014. DOI: 10.1145/1869459.1869489

[22] Hiroaki Inoue, Atsushi Igarashi, Malte Appeltauer, and Robert Hirschfeld. Towards type-safe JCop: A type system for layer inheritance and first-class layers. In Proc. of the International Workshop on Context-Oriented Programming (COP'14), July 2014. pp. 7:1-7:6. DOI: 10.1145/2637066.2637073

[23] Tetsuo Kamina, Tomoyuki Aotani, and Atsushi Igarashi. On-demand layer activation for type-safe deactivation. In Proc. of the International Workshop on Context-Oriented Programming (COP'14). pp. 4:1-4:6. DOI: 10.1145/2637066.2637070

[24] Changyun Huang, Ataru Osaka, Yasutaka Kamei, and Naoyasu Ubayashi. Automated DSL Construction Based on Software Product Lines. In Proc. of the 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2015), pp.247-254 (2015).

[25] Tetsuo Kamina, Tomoyuki Aotani, and Hidehiko Masuhara. Generalized layer activation mechanism through contexts and subscribers. In Proc. of International Conference on Modularity (Modularity'15), pp.14-28, March 2015. DOI:10.1145/2724525.2724570

[26] Raffi Khatchadourian, Awais Rashid, Hidehiko Masuhara, and Takuya Watanabe. "Detecting broken pointcuts using structural commonality and degree of interest.", In Proceedings of 30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015), p. 641-646, November 2015.

[27] Hiroaki Inoue, Atsushi Igarashi, "A Sound Type System for Layer Subtyping and Dynamically Activated First-Class Layers", Proc. of 13th Asian Symposium on Programming Languages and Systems (APLAS 2015), Springer LNCS 9458, pp. 445-464.

[28] Masatoshi Hanai, Toyotaro Suzumura, Georgios Theodoropoulos, and Kalyan S. Perumalla, "Towards Large-Scale What-If Traffic Simulation with Exact-Differential Simulation", In Proceedings of the 2015 Winter Simulation Conference (WSC'15), pp.748-756, 2015

[29] Masatoshi Hanai, Toyotaro Suzumura, Georgios Theodoropoulos, and Kalyan S. Perumalla, "Exact-Differential Large-Scale Traffic Simulation", In Proceedings of ACM SIGSIM Conference on Principles of Advanced Discrete Simulation 2015 (PADS'15), pp.271-280, 2015

[30] Yungyu Zhuang and Shigeru Chiba, Enabling the Automation of Handler Bindings in Event-driven Programming, IEEE COMPSAC 2015, IEEE Computer Society, pp.137-146, 2015

[31] Laurie Hendren, Hidehiko Masuhara, Mary Sheeran, and Jan Vitek, editors. "Proceedings of the 2nd ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming (ARRAY'15)", ACM, June 2015.

[32] Yoshiki Sato, Shumpei Hozumi and Shigeru Chiba, Calculation Coverage Testing for Scientific Applications, International Symp. on Software Testing and Analysis (ISSTA'15), ACM, pp.350-360, 2015

[33] Shigeru Chiba, YungYu Zhuang, and Maximilian Scherr, A Design of Deep Reification, Workshop on Modularity Across the System Stack (MASS'16), ACM, pp.168-171, 2016

[34] Tetsuo Kamina, Tomoyuki Aotani, and Hidehiko Masuhara. "Toward fluent module interactions.", In Proceedings Workshop on Modularity Across the System Stack (MASS 2016), pp. 158-159. 2016

[35] Matthias Springer, Hidehiko Masuhara, and Robert Hirschfeld. "Hierarchical layer-based class

extensions in Squeak/Smalltalk.", In Proceedings of the Workshop on Live Adaptation of Software Systems (LASSY 2016), pp. 107–112. 2016.

[36] Hirotada Kiriyama, Tomoyuki Aotani, and Hidehiko Masuhara. "A lightweight optimization technique for data types à la carte.", In Proceedings of the Workshop on Language Modularity À La Mode Workshop (LaMOD'16), pp. 86–90. 2016.

[37] Raffi Khatchadourian, Olivia Moore, and Hidehiko Masuhara. "Towards improving interface modularity in legacy Java software through automated refactoring." In Proceedings of the Workshop on Language Modularity À La Mode Workshop (LaMOD'16), pp. 104–106. 2016.

[38] Hidehiko Masuhara, Kenta Fujita, and Tomoyuki Aotani. "An advice mechanism for non-local flow control." In Proceedings of Foundations of Aspect-Oriented Langauges (FOAL2016), pp. 73–78, 2016.

[39] Matthias Springer, Fabio Niephaus, Robert Hirschfeld, and Hidehiko Masuhara. "Matriona: Class nesting with parameterization in Squeak/Smalltalk." In Proceedings of International Conference on Modularity (Modularity'16), pp. 118–129, 2016.

[40] Hiroaki Inoue, Atsushi Igarashi, "A Library-Based Approach to Context-Dependent Computation with Reactive Values", Proc. of Workshop on Constrained and Reactive Objects, Modularity 2016 Companion, pp. 50–54

[41] Kensuke Kojima, Atsushi Igarashi: A Hoare Logic for GPU Kernels, ACM Transactions on Computational Logic. (to appear) 2016.

[42] Kensuke Kojima, Akifumi Imanishi, Atsushi Igarashi, Automated Verification of Functional Correctness of Race-Free GPU Programs, Proceedings of the 8th Working Conference on Verified Software: Theories, Tools, and Experiments (VSTTE 2016), Lecture Notes in Computer Science Vol. 9971, pp. 90–106, Springer-Verlag, 2016.

[43] Kensuke Kojima, Minoru Kinoshita, Kohei Suenaga, Generalized Homogeneous Polynomials for Efficient Template-Based Nonlinear Invariant Synthesis, Proceedings of the 23rd International Symposium on Static Analysis (SAS 2016), Lecture Notes in Computer Science Vol. 9837, pp. 278–299, Springe-Verlag, 2016.

[44] Yasutaka Kamei, Takafumi Fukushima, Shane McIntosh, Kazuhiro Yamashita, Naoyasu Ubayashi, Ahmed E. Hassan, Studying just-in-time defect prediction using cross-project models, Empirical Software Engineering, 21(5), pp.2072–2106, 2016.

[45] Kazuhiro Yamashita, Changyun Huang, Meiyappan Nagappan, Yasutaka Kamei, Audris Mockus, Ahmed E. Hassan and Naoyasu Ubayashi, Thresholds for Size and Complexity Metrics: A Case Study from the Perspective of Defect Density, Proceedings of the 2016 IEEE International Conference on Software Quality, Reliability & Security (QRS 2016), pp.191–201, 2016.

[46] Keisuke Miura, Shane McIntosh, Yasutaka Kamei, Ahmed E. Hassan, and Naoyasu Ubayashi, The Impact of Task Granularity on Co-evolution Analyses, Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2016), 47:1–47:10, 2016.

[47] Kazuhiro Yamashita, Yasutaka Kamei, Shane McIntosh, Ahmed E. Hassan, Naoyasu Ubayashi, Magnet or Sticky? Measuring Project Characteristics from the Perspective of Developer Attraction and Retention, Journal of Information Processing, vol.24, no.2, pp.1-10, 2016.

[48] Shigeru Chiba, YungYu Zhuang, Maximilian Scherr, Deeply Reifying Running Code for Constructing a Domain-Specific Language, Proc. of the 13th International Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools (PPPJ'16), Article No. 1, August 2016.

[49] Yung Yu Zhuang and Shigeru Chiba, Expanding Event Systems to Support Signals by Enabling the Automation of Handler Bindings, Special Issue of Applications and the Internet in Conjunction with Main Topics of COMPSAC 2015, Journal of Information Processing, vol. 24, no. 4, pp.620-634, 2016.

[50] Thanh-Chung Dao and Shigeru Chiba, HPC-Reuse: efficient process creation for running MPI and Hadoop MapReduce on supercomputers, 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, pp.342-345, May 2016. (short paper)

[51] Springer, Masuhara, Hirschfeld, "Classes as Layers: Rewriting Design Patterns with COP: Alternative Implementations of Decorator, Observer, and Visitor", In Proceedings of the 8th International Workshop on Context-Oriented Programming (COP 2016), pp.21-26, July 2016. DOI: 10.1145/2951965.2951968

[52] Springer, Masuhara. "Object Support in an Array-based GPGPU Extension for Ruby", In Proceedings of the 3rd ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming (ARRAY 2016), pp.25-31, 2016. DOI: 10.1145/2935323.2935327

[53] Tetsuo Kamina, Tomoyuki Aotani, and Hidehiko Masuhara, "Generalized layer activation mechanism for context-oriented programming", Transactions on Modularity and Composition (TOMC), Volume I, pp.123-166. September 2016. DOI: 10.1007/978-3-319-46969-0_4

[54] Matthias Springer, Hidehiko Masuhara, and Robert Hirschfeld, "A layer-based approach to hierarchical dynamically-scoped open classes", Journal of Information Processing, Vol. 25 (2017) pp. 296-307. DOI: 10.2197/ipsjjip.25.296.

[55] Ruochen Huang, Hidehiko Masuhara, and Tomoyuki Aotani, "Improving Sequential Performance of Erlang based on a Meta-tracing
Just-In-Time Compiler", in Post-proceedings of the 17th Symposium on Trends in Functional Programming, 2016, to appear.

[56] Tim Felgentreff, Robert Hirschfeld, Maria Graber, Alan Borning, and Hidehiko Masuhara.
Declaring constraints on object-oriented collections.
Journal of Information Processing, Vol.24, No.5, pp. 917-927, November 2016.

[57] Izumi Asakura, Hidehiko Masuhara, and Tomoyuki Aotani. CertSkel: a verified compiler for a Coq-embedded GPGPU DSL. In Proceedings of The Third International Workshop on Coq for Programming Languages (CoqPL 2017), colocated with PoPL'17, January 2017.

[58] Marcel Taeumel, Stephanie Platz, Bastian Steinert, Robert Hirschfeld, and Hidehiko Masuhara.
Unravel programming sessions with THRESHER: Identifying coherent and complete sets of fine-granular source code changes. Computer Software, Vol.34, No.1, pp. 103-118, 2017.

(2)その他の著作物(総説、書籍など)

[1] Agha, G., Igarashi, A., Kobayashi, N., Masuhara, H., Matsuoka, S., Shibayama, E., Taura, K. (Eds.) Concurrent Objects and Beyond. Springer-Verlag, 2014.

(3)国際学会発表及び主要な国内学会発表
① 招待講演 (国内会議 0 件、国際会議 5 件)

[1] Shigeru Chiba, "Modularity for Supercomputing," International Workshop on Peta-Scale Computing Programming Environment, Languages and Tools (WPSE 2012), Kobe, 2012 February 29.

[2] Shigeru Chiba, "Software composition for high-performance computing", ACM SPLASH FREECO - 3rd Workshop on Free Composition, ACM SPLASH2012, Tucson, October 22, 2012.

[3] Atsushi Igarashi, "Type Systems for Dynamic Layer Composition," International Workshop on Foundations of Aspect-Oriented Languages (FOAL2013), Fukuoka, March 24, 2013.

[4] Shigeru Chiba, "How can we apply software product lines to high-performance computing?" Software Product Line (SPL) Symposium, Modularity:AOSD13, Fukuoka, March 28, 2013.

[5] Shigeru Chiba, "To be destructive or not to be, that is the question on modular extensions," FOAL'14 workshop (Foundations of Aspect-Oriented Languages 2014), Modularity 2014.
http://aosd.net/2014/keynotes

② 口頭発表 (国内会議 37 件、国際会議 11 件)

[1] 山下 一寛・山本 大輔・亀井 靖高・久住 憲嗣・鵜林 尚靖、「リポジトリマイニング向けドメイン専用言語の設計と実装」、電子情報通信学会 信学技報 SS2011-81、pp.145-150、那覇、2012 年 3 月 14 日

[2] 紙名哲生, 青谷知幸, 増原英彦, 文脈指向言語 EventCJ への合成層の導入, 情報処理学会第 89 回プログラミング研究会発表:2012-1-(3), June 2012.

[3] 中村 央記, 永野 梨南, 久住 憲嗣, 亀井靖高, 鵜林尚靖, 福田晃, "GPGPU を用いたリポジトリマイニングのための外部ドメイン専用言語 QORAL の提案," 電子情報通信学会技術報告, Vol.112, No.23, 2012 年 5 月 10 日

[4] 黄 長贇, 中城 亮祐, 山下 一寛, 亀井 靖高, 久住 憲嗣, 鵜林 尚靖, "Alloy によるリポジトリマイニング向けドメイン専用言語の構築支援," 電子情報通信学会技術報告, Vol.112, No.164,

2012 年 7 月 28 日

[5] 山下 一寛, 亀井 靖高, 久住 憲嗣, 鵜林 尚靖, "リポジトリマイニング向けドメイン専用言語 ArgyleJ の開発と実証的評価," 情報処理学会 ソフトウェアエンジニアリングシンポジウム 2012 (SES 2012), 東京, 2012 年 8 月 29 日

[6] 永野 梨南, 中村 央記, 亀井 靖高, ブラム アダムス, 久住 憲嗣, 鵜林 尚靖, 福田 晃, "GPGPU を用いたリポジトリマイニングの高速化手法 ープロセスメトリクスの算出への適用 ー," 情報処理学会 ソフトウェアエンジニアリングシンポジウム 2012 (SES 2012), 東京, 2012 年 8 月 29 日

[7] 宗 桜子・武山 文信・千葉 滋, "メソッド間の依存関係から適切な計算順序を生成する言語", 日本ソフトウェア科学会第 29 回大会, 小金井市, 2012 August 22-24.

[8] 穂積 俊平・伊尾木 将之・千葉 滋, "HPC アプリケーションにおける OOP を用いたパフォーマンスチューニング", 日本ソフトウェア科学会第 29 回大会, 小金井市, 2012 August 22-24.

[9] 大坂 陽, 山下 一寛, 亀井 靖高, 鵜林 尚靖, "リポジトリマイニングに対する Hadoop の性能評価,"情報処理学会 第 179 回ソフトウェア工学研究会, 東京, 2012 年 3 月 12 日

[10] 宗桜子・佐藤芳樹・千葉滋, 処理の差異と順序を考慮した並列コレクション向け Java 言語拡張, SWoPP 北九州 2013(2013 年 並列/分散/協調処理に関する『北九州』サマー・ワークショップ), 2013-HPC-140, no.9, 情報処理学会 HPC 研究会, pp.1-6, 2013.

[11] 穂積俊平・佐藤芳樹・千葉滋, HPC プログラム向け計算網羅性・計算順序をテストするツール, SWoPP 北九州 2013(2013 年 並列/分散/協調処理に関する『北九州』サマー・ワークショップ), 2013-HPC-140, no.23, 情報処理学会 HPC 研究会, pp.1-7, 2013.

[12] Shigeru Chiba, On modularity for high-performance computing, Symposium on Program Comprehension, Univ. Twente, 2013 Dec. 5.

[13] 山口洋・千葉滋, より柔軟な非同期実行記述を可能にするライブラリのための新しい限定継続演算子に向けて, 第 16 回プログラミングおよびプログラミング言語ワークショップ(PPL2014), 2014 Mar. 5-7.

[14] Kazuhiro Ichikawa, Shigeru Chiba, Composable User-Defined Operators That Can Express User-Defined Literals, 第 16 回プログラミングおよびプログラミング言語ワークショップ(PPL2014), 2014 Mar. 5-7.

[15] 穂積 俊平・佐藤 芳樹・千葉 滋, 科学技術計算における最適化に伴う分割の正しさを検査するユニットテストフレームワーク HPCUnit, 第 16 回プログラミングおよびプログラミング言語ワークショップ(PPL2014), 2014 Mar. 5-7.

[16] Robert Hirschfeld, Hidehiko Masuhara, and Atsushi Igarashi, "Layer and object refinement for context-oriented programming in L", 95th IPSJ Workshop on SIG Programming, 2013-2-(2), August 2013.

[17] Manabu Toyama, Tomoyuki Aotani, Eric Bodden, Hidehiko Masuhara, and Eric Tanter, "Aspect interfaces: Towards separate type-checking of aspect-oriented programs with inter-type

declarations″, 94th IPSJ Workshop on SIG Programming, 2013-1-(4), June 2013.

[18] 大坂 陽, 山下 一寛, 亀井 靖高, 鵜林 尚靖: リポジトリマイニングに対する, Hadoop の導入に向けた性能評価, 情報処理学会 ソフトウェアエンジニアリングシンポジウム 2013 (SES 2013), 東京, 2013 年 9 月 9-11 日. [最優秀論文賞受賞]

[19] 大坂 陽, 伊原 彰紀, 亀井 靖高, 鵜林 尚靖: OSS 開発におけるパッチレビュープロセス追跡技術の提案, 情報処理学会ソフトウェア工学研究会 ウィンターワークショップ 2014・イン・大洗, pp.19-20, 大洗, 2014 年 1 月 23-24 日.

[20] 山下 一寛, 亀井 靖高, 鵜林 尚靖: リポジトマイニング研究への高速化手法適用に向けた検討, 情報処理学会ソフトウェア工学研究会 ウィンターワークショップ 2014・イン・大洗, pp.27-28, 大洗, 2014 年 1 月 23-24 日.

[21] 黄 長贇, 亀井 靖高, 鵜林 尚靖: DSL ラインエンジニアリング支援環境の設計, 電子情報通信学会 信学技報 SS2013-89, pp.103-108, 那覇, 2014 年 3 月 11-12 日.

[22] 川島 関夫, 亀井 靖高, 鵜林 尚靖: 開発メーリングリストマイニングの前処理システムの開発, 情報処理学会 研究報告 2012-SE-183, No.19, 東京, 2014 年 3 月 19-20 日.

[23] 関山太朗, 西田雄気, 五十嵐淳. 顕在的契約計算における代数的データ型. 第 14 回プログラミングおよびプログラミング言語ワークショップ (PPL2014) オンライン論文集, 熊本県阿蘇市, March 2014. 日本ソフトウェア科学会.

[24] 大坂 陽, 亀井 靖高, 堀田 圭佑, 鵜林 尚靖. コードクローン解析に対するスーパーコンピュータ導入に向けた試行実験. 電子情報通信学会 信学技報 SS2014-35, pp.13-18 (2015).

[25] 川島 関夫, 亀井 靖高, 鵜林 尚靖. 社会としての OSS プロジェクトを解析するソーシャルネットワークマイニングツール Coli. 電子情報通信学会 信学技報 SS2014-64, pp.55-60 (2015).

[26] 夏澄彦, 佐藤芳樹, 千葉滋. 依存関係をもった並列タスクのための動的グループバリア同期とその効率的な実装. 日本ソフトウェア科学会第 31 回大会, 2015.

[27] 奥村健太郎、小島健介、五十嵐淳、”SIMT のための Hoare 論理の Coq を用いた形式化と並列 prefix-sum アルゴリズムの検証” 第 18 回プログラミングおよびプログラミング言>語ワークショップ(PPL2016)論文集

[28] YungYu Zhuang, Shigeru Chiba、Better abstraction for efficient code in HPC programs、2015 年並列／分散／協調処理に関する『別府』サマー・ワークショップ (SWoPP2015)、大分県別府市、2016/08/04

[29] Stephanie Platz, Marcel Taeumel, Bastian Steinert, Robert Hirschfeld, and Hidehiko Masuhara. Unravel programming sessions with THRESHER: Identifying coherent and complete sets of fine-granular source code changes 日本ソフトウェア科学会大会、早稲田大学、2015/09/11

[30] Hidehiko Masuhara, ″Context-oriented programming and units of adaptation″, NII Shonan Meeting on Engineering Adaptive Software Systems (EASSy) 湘南国際村, 2015/09/07

[31] 朝倉 泉, 増原 英彦, 青谷 知幸、GPGPU 向けデータ並列コードテンプレートの形式検証、第 18 回プログラミングおよびプログラミング言語ワークショップ、岡山県玉野市、2016/03/07

[32] 小須田 光, 亀井 靖高, 鵜林 尚靖 ″ユーザ障害情報によるソースコード欠陥箇所予測ツール″, 情報処理学会ソフトウェア工学研究会, JR 博多シティ会議室, 2015/12/16

[33] 山下 一寛, 江 冠達, 亀井 靖高, 鵜林 尚靖、コミットログを用いた OSS 開発における不確かさに関する実証分析、電子情報通信学会ソフトウェアサイエンス研究会、石川県政記念しいのき迎賓館、2016/01/26

[34] Thanh-Chung Dao, Shigeru Chiba, Improving Hadoop MapReduce on supercomputers with JVM reuse and MPI shuffling, 2015 年並列／分散／協調処理に関する『別府』サマー・ワークショップ (SWoPP2015), 大分県別府市, 2016/08/04

[35] Masayuki Ioki, Shumpei Hozumi, and Shigeru Chiba, “Writing a modular GPGPU program in Java,” 2nd Workshop on Modularity In Systems Software (MISS2012), AOSD’12, Potsdam, Germany, 2012 March 27.

[36] Manabu Toyama, Tomoyuki Aotani, and Hidehiko Masuhara, “A Per-type Instantiation Mechanism for Generic Aspects,” 3rd International Workshop on Variablity and Composition (VariComp’12), AOSD’12, Potsdam, Germany, 2012 March 26.

[37] Hidehiko Masuhara and Yusuke Nishiguchi, ″A data-parallel extension to Ruby for GPGPU: Toward a framework for implementing domain-specific optimizations″, Proceedings of the 9th Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE’12), pp. 3-6. ACM, June 2012.

[38] Tetsuo Kamina, Tomoyuki Aotani and Hidehiko Masuhara, ″Bridging real-world contexts and units of behavioral variations by composite layers″, Proceedings of the Workshop on Context-Oriented Programming (COP’12), pp. 4:1-4:6, June 2012.

[39] Atsushi Igarashi, Robert Hirschfeld and Hidehiko Masuhara, ″A type system for dynamic layer composition″, Proceedings of 19th International Workshop on Foundations of Object-Oriented Languages (FOOL 2012), October 2012.

[40] Hiroki Nakamura, Rina Nagano, Kenji Hisazumi, Yasutaka Kamei, Naoyasu Ubayashi and Akira Fukuda, ″QORAL: External Domain-Specific Language for Mining Software Repositories,″ International Workshop on Empirical Software Engineering in Practice (IWESEP2012), Osaka, Japan, 2012 October 27.

[41] Tetsuo Kamina, Tomoyuki Aotani, and Hidehiko Masuhara, ″Mapping context-dependent requirements to event-based context-oriented programs for modularity″, Workshop on Reactivity, Events and Modularity (REM 2013), colocated with OOPSLA’13, October 2013.

[42] Kazuhiro Yamashita, Yasutaka Kamei, Kenji Hisazumi, and Naoyasu Ubayashi: E-CUBE: An Analysis Tool to Support Three Types of Evolution in Mining Software Repositories, 2013 International Workshop on ICT, Beppu, 2013 年 12 月 12-14 日.

[43] Changyun Huang, Naoyasu Ubayashi, and Yasutaka Kamei: Towards Language-Oriented

Software Development, 2nd International Workshop on Open and Original Problems in Software Language Engineering (OOPSLE 2014), pp.20-23, Antwerp, 2014 年 2 月 3 日.

[44] Seisei Itahashi, Yoshiki Sato and Shigeru Chiba, “Toward a profiling tool for visualizing implicit behavior in X10,” The 2014 X10 Workshop (X10'14) co-located with PLDI'14, Edinburgh, UK, 2014 June 12. http://x10-lang.org/workshop/workshop14/x10-2014-program.html

[45] Shigeru Chiba, “Embedded domain specific languages for HPC”, JST/CREST International Symposium on Post Petascale System Software, 2014.

[46] Thanh-Chung Dao, Shigeru Chiba, In-memory Hadoop on supercomputers using external memory of additional nodes, 日本ソフトウェア科学会第 33 回大会, 東北大学, 2016 年 9 月 6 日～9 日.

[47] 岩間 雄太, 市川 和央, 千葉 滋, 動的型付き言語上での構文拡張手段の提供, 日本ソフトウェア科学会第 33 回大会, 東北大学, 2016 年 9 月 6 日～9 日.

[48] 蟹暁, 朝倉泉, 増原英彦, 青谷知幸, バリア同期と共有メモリを備えた GPGPU プログラム合成器 Kani-CUDA. 情報処理学会第 113 回プログラミング研究会発表: 2016-5-(6), March 2017.

③ ポスター発表 （国内会議 39 件、国際会議 5 件）

[1] 西口裕介・増原英彦、「GPU 向けプログラムを Ruby で記述できる言語処理系 Ikra」、第 12 回プログラミングおよびプログラミング言語ワークショップ（PPL2012)、和歌山県白浜町、2012 年 3 月

[2] 山下 一寛, 亀井 靖高, 久住 憲嗣, 鵜林 尚靖, ″リポジトリマイニングにおける 3 つの進化に対応した分析ツール E-CUBE の構築,″ 日本ソフトウエア科学会 第 19 回ソフトウェア工学の基礎ワークショップ（FOSE 2012）ポスター・デモ, 湯布院, 2012 年 12 月 13 日

[3] 黄 長贇, 亀井 靖高, 久住 憲嗣, 鵜林 尚靖, ″Alloy を用いたリポジトリマイニング向け DSL の構築支援,″ 日本ソフトウエア科学会 第 19 回ソフトウェア工学の基礎ワークショップ（FOSE 2012）ポスター・デモ,″ 湯布院, 2012 年 12 月 13 日

[4] 永野 梨南, 亀井 靖高, 久住 憲嗣, 鵜林 尚靖, 福田 晃, ″リポジトリマイニングにおける GPGPU の利用とその効果,″ 日本ソフトウエア科学会 第 19 回ソフトウェア工学の基礎ワークショップ（FOSE 2012）ポスター・デモ, 湯布院, 2012 年 12 月 13 日

[5] 中村 央記, 亀井 靖高, 久住 憲嗣, 鵜林 尚靖, 福田 晃, ″リポジトリマイニングのための外部ドメイン専用言語の提案,″ 日本ソフトウエア科学会 第 19 回ソフトウェア工学の基礎ワークショップ（FOSE 2012）ポスター・デモ, 湯布院, 2012 年 12 月 13 日

[6] 小島健介・五十嵐淳,「SIMT プログラムのためのホーア論理」第 15 回プログラミングおよびプログラミング言語ワークショップ（PPL2013）プログラム, 福島県会津若松市, 2013 年 3 月

[7] 井上裕昭, 五十嵐淳, Robert Hirschfeld, 増原 英彦「動的レイヤー合成のための型システム」, 第 15 回プログラミングおよびプログラミング言語ワークショップ（PPL2013）プログラム, 福島県会津若松市, 2013 年 3 月

[8] Maximilian Pascal Scherr, Shigeru Chiba, Applications of Redefining EDSL Expression Semantics at Load-Time, 第 14 回プログラミングおよびプログラミング言語ワークショップ(PPL2014), 熊本県阿蘇市, March 2014. 日本ソフトウェア科学会.

[9] 板橋晟星・佐藤芳樹・千葉滋, PGAS プログラムのためのプロファイリングツールの開発, 第 14 回プログラミングおよびプログラミング言語ワークショップ(PPL2014), 熊本県阿蘇市, March 2014. 日本ソフトウェア科学会.

[10] 夏澄彦・佐藤芳樹・千葉滋, 社会シミュレーションを並列化するための新しい言語機構の研究へ向けて, 第 14 回プログラミングおよびプログラミング言語ワークショップ(PPL2014), 熊本県阿蘇市, March 2014. 日本ソフトウェア科学会.

[11] 汐田徹也・佐藤芳樹・千葉滋, メモリレイアウトを切り替えられるグラフ解析プログラム向け Java コンパイラ, 第 14 回プログラミングおよびプログラミング言語ワークショップ(PPL2014), 熊本県阿蘇市, March 2014. 日本ソフトウェア科学会.

[12] 当山学, 青谷知幸, Eric Bodden, 増原英彦, Éric Tanter, ″Interfaces for separate compilation of aspect-oriented programs with inter-type declarations″, 第 16 回プログラミングおよびプログラミング言語ワークショップ(PPL2014), March 2014.

[13] 紙名哲生, 青谷知幸, 増原英彦, 玉井哲雄, ″COSE: 文脈に依存した振る舞いをモジュール化するソフトウェア開発手法″, 第 20 回ソフトウェア工学の基礎ワークショップ(FOSE2013), November 2013.

[14] 小島 健介, 五十嵐 淳. Generic GluonJ に向けて. 第 14 回プログラミングおよびプログラミング言語ワークショップ (PPL2014), 熊本県阿蘇市, March 2014. 日本ソフトウェア科学会.

[15] 井上 裕昭, 五十嵐 淳. 文脈指向言語 JCop への型検査器の構成と実装. 第 14 回プログラミングおよびプログラミング言語ワークショップ(PPL2014), 熊本県阿蘇市, March 2014. 日本ソフトウェア科学会.

[16] 西田 雄気, 関山 太朗, 五十嵐 淳. 型に基づく実行時契約検査機構の実装. 第 14 回プログラミングおよびプログラミング言語ワークショップ(PPL2014), 熊本県阿蘇市, March 2014. 日本ソフトウェア科学会.

[17] 樹下 稔, 末永 幸平, 五十嵐 淳. MapReduce フレームワークにおける Combiner の自動生成. 第 14 回プログラミングおよびプログラミング言語ワークショップ (PPL2014), 熊本県阿蘇市, March 2014. 日本ソフトウェア科学会.

[18] 上田 宗一郎, 関山 太朗, 五十嵐 淳. 限定継続を備えた計算体系へのソフトウェア契約の導入. 第 14 回プログラミングおよびプログラミング言語ワークショップ (PPL2014), 熊本県阿蘇市, March 2014. 日本ソフトウェア科学会.

[19] 黄 長贇, 鵜林 尚靖, 細合 晋太郎, 亀井 靖高(九州大学), 対話的な DSL 構築環境 Argyle, 情報処理学会 ソフトウェアエンジニアリングシンポジウム 2014 (SES 2014), 東京, 2014 年 9 月 1 日.

[20] 大坂 陽, 亀井 靖高(九州大学), 堀田 圭佑(大阪大学), 鵜林 尚靖(九州大学), ソフトウェア工学研究に対する HPC 利用の効果 – コードクローン研究の事例を通して, 情報処理学会

ソフトウェアエンジニアリングシンポジウム 2014 (SES 2014), 東京, 2014 年 9 月1日.

[21] 奥村健太郎, 小島健介, 五十嵐淳. SIMT のための Hoare 論理の Coq での形式化. 日本ソフトウェア科学会 PPL 2015 ワークショップ. 2015 年 3 月 4-6 日.

[22] Ruochen Huang, 増原 英彦, 青谷 知幸. ``Pyrlang: RPython を用いた高性能 Erlang BEAM 仮想マシン.'' 第 17 回プログラミングおよびプログラミング言語ワークショップ(PPL2015), ポスター発表, March
2015.

[23] 朝倉 泉, 増原 英彦, 青谷 知幸. ``GPGPU カーネル検証のための分離論理の Coq による形式化.'' 第 17 回プログラミングおよびプログラミング言語ワークショップ(PPL2015), ポスター発表, March 2015.

[24] 桐山 裕匡, 青谷 知幸, 増原 英彦. ``Haskell のモジュラリティと効率の両立のためのコンパイル時データ合成.'' 第 17 回プログラミングおよびプログラミング言語ワークショップ(PPL2015), ポスター発表, March 2015.

[25] 当山 学 (東京大学), 青谷 知幸 (東京工業大学), Eric Bodden (EC SPRIDE, Technische Universitaet Darmstadt), 増原 英彦 (東京工業大学), Eric Tanter (Universidad de Chile). ``Interfaces for Separate Compilation of Aspect-Oriented Programs with Inter-Type Declarations.'' 第 16 回プログラミングおよびプログラミング言語ワークショップ(PPL2015), ポスター発表, March 2014.

[26] 板橋 晟星, 佐藤 芳樹, 千葉 滋. PGAS 言語 X10 向けのプロファイリングツール X-Eye の開発. 第 16 回プログラミングおよびプログラミング言語ワークショップ(PPL2015), ポスター発表, March 2014.

[27] 荘 永裕, 千葉 滋. Developing and running DSL programs on FX10 and Tsubame with a code translation framework. 第 16 回プログラミングおよびプログラミング言語ワークショップ(PPL2015), ポスター発表, March 2014.

[28] Matthias Springer and Hidehiko Masuhara, Ikra: Leveraging Object-oriented Abstractions in a Ruby-to-CUDA JIT Translator, 第 18 回プログラミングおよびプログラミン>グ言語ワークショップ, 岡山県玉野市, 2016/03/07

[29] 黄 若塵, 増原 英彦, 青谷 知幸, RPython を用いた Erlang 仮想機械 Pyrlang における JIT コンパイル方針の改良, 第 18 回プログラミングおよびプログラミング言語ワークショップ,>岡山県玉野市 2016/03/07

[30] 奥河 諒, 増原 英彦, 青谷 知幸, Featherweight Java の Meta-Theory a la Carte を利用> した拡張可能な形式化フレームワーク, 第 18 回プログラミングおよびプログラミング言語 ワークショップ, 岡山県玉野市, 2016/03/07

[31] 渡邉 恵大, 増原 英彦, 青谷 知幸, 多次元的文脈指向言語 Korz のメソッドディスパッチ> の改善, 第 18 回プログラミングおよびプログラミング言語ワークショップ, 岡山県玉野市
, 2016/03/07

[32] 山崎 徹郎・佐藤 芳樹・千葉 滋, 包含文字列の内部 char 配列を共有化する拡張文字列を>

部分的に利用した時の性能を調べる実験，第 18 回プログラミングおよびプログラミング言語ワークショップ，岡山県玉野市, 2016/03/08

[33] Kazuhiro Yamashita, Guanda Jiang, Takuya Fukamachi, Yasutaka Kamei, and Naoyasu Ubayashi, An Empirical Study of Uncertainty in OSS Projects, 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016) , 大阪大学, 2016/03/15

[34] Keisuke Miura, Shane McIntosh, Yasutaka Kamei, Ahmed Hassan, and Naoyasu Ubayashi: An Empirical Study of the Impact of Task Granularity on Co-evolution Analyses, 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016), Poster, Osaka, Japan, March 15, 2016.

[35] Masayuki Ioki, Shumpei Hozumi, and Shigeru Chiba, Modularity for HPC --WootinJ--, Student Poster Session, Modularity:AOSD 2012, March 26, 2012.

[36] Rina Nagano, Hiroki Nakamura, Yasutaka Kamei, Bram Adams, Kenji
Hisazumi, Naoyasu Ubayashi and Akira Fukuda, ″Using the GPGPU for Scaling Up Mining Software Repositories,″ International Conference on Software Engineering (ICSE2012), Poster Session, Zurich, Switzerland, 2012 June 7.

[37] Hidehiko Masuhara and Tomoyuki Aotani. ``A Dynamically-typed Language for Prototyping High-Performance Data Parallel Programs.'' JST/CREST International Symposium on Post Petascale System Software, December 2014.

[38] Matthias Springer, Peter Wauligmann, and Hidehiko Masuhara. Iterative stencil computations in Ruby on GPUs. Poster presentation at PPL'17, March 2017.

[39] 朝倉泉，増原英彦，青谷知幸．検証済みコンパイラ CertSkel による GPGPU プログラム開発．第 19 回プログラミングおよびプログラミング言語ワークショップ(PPL2017), March 2017．ポスター発表．

[40] 蟹暁，朝倉泉，増原英彦，青谷知幸．Kani-CUDA による GPGPU プログラムの合成．第 19 回プログラミングおよびプログラミング言語ワークショップ(PPL2017), March 2017．ポスター発表．

[41] 中丸 智貴・市川 和央・山崎 徹郎・千葉 滋．Java 用 Fluent API 生成システム ″B2F″ の> 設計と開発．第 19 回プログラミングおよびプログラミング言語ワークショップ(PPL2017), March 2017. ポスター発表．

[42] 山崎 徹郎・市川 和央・千葉 滋．自己反映的な Garbage Collector の実現に向けた処理系の実装方式の提案．第 19 回プログラミングおよびプログラミング言語ワークショップ(PPL2017), March 2017．ポスター発表．

[43] 小林佑樹・千葉滋．統計的機械翻訳を利用したソースコードエディターに向けて．第 19 回プログラミングおよびプログラミング言語ワークショップ(PPL2017), March 2017．ポスター発表．

[44] Wei Zhang. Introducing a Faster Inline-caching Mechanism for Method Seal. 第 19 回プログラミングおよびプログラミング言語ワークショップ(PPL2017), March 2017．ポスター発表．

(4)知財出願

なし

(5)受賞・報道等

① 受賞

[1] Shigeru Chiba, 2012 IBM Faculty Award

[2] 大坂 陽, 山下 一寛, 亀井 靖高, 鵜林 尚靖, 情報処理学会 ソフトウェアエンジニアリングシンポジウム2013 (SES 2013) 最優秀論文賞, リポジトリマイニングに対するHadoopの導入に向けた性能評価

[3] 大坂 陽, 2014 年度情報処理学会 CS(コンピュータサイエンス)領域奨励賞, リポジトリマイニングに対する Hadoop の導入に向けた性能評価

[4] 小須田 光, IEEE Computer Society Japan Chapter FOSE Young Researcher Award, クラッシュレポートの送信頻度が不具合との関連付けに与える影響, 日本ソフトウエア科学会 第 21 回ソフトウェア工学の基礎ワークショップ (FOSE 2014)

[5] 朝倉泉, 情報処理学会コンピュータサイエンス領域奨励賞,2016.「GPGPU のための並行分離論理の Coq による健全性証明」

[6] Thanh-Chung Da, 日本ソフトウェア科学会第 33 回大会学生奨励賞, 2016.

②マスコミ(新聞・TV等)報道(プレス発表をした場合にはその概要もお書き下さい。)

③その他

(6)成果展開事例

なし

§5 研究期間中の活動

5．1 主なワークショップ、シンポジウム、アウトリーチ等の活動

特に記載無し

§6 最後に

全体として4つの大学に分かれて、それぞれのテーマに沿っての研究活動であったが、チーム内ミーティングにて意見交換を重ねることで、随時テーマの見直し、相互の技術協力ができたように思う。結果としてそれぞれのグループが元々もつ強みや、本研究を通じて得られた知見を互いにフィードバックしあうことで、チーム内で相乗効果が生まれたと考える。