High-productivity Abstractions and Efficient Runtime for Dynamically Load-balanced Distributed Programs on Multi/Many-core Clusters

Finnerty, Patrick Martin

(Degree)
博士（工学）

(Date of Degree)
2022-09-25

(Date of Publication)
2024-09-25

(Resource Type)
doctoral thesis

(Report Number)
甲第8467号

(URL)
https://hdl.handle.net/20.500.14094/0100477893

（別紙様式 3）

論文内容の要旨

氏　　名　FINNERTY Patrick Martin

専　　攻　情報学専攻

論文題目（外国語の場合は，その和訳を併記すること。）

High-productivity Abstractions and Efficient Runtime for Dynamically Load-balanced Distributed Programs on Multi/Many-core Clusters
マルチコア／メニーコアクラスタにおける動的負荷分散プログラムのための高生産性抽象化と効率的ランタイム実装法

指導教員　太田　能　教授

Chapter 1: Introduction

Modern supercomputers rely on clusters of many-core processors, bringing large amount of parallelism both within a node and across nodes. These architectures revive interest in programming models and languages dedicated to supporting these new environments.

The Asynchronous Partitioned Global Address Space (APGAS) as implemented in the X10 language brings nice abstractions that allow programmers to handle the distributed nature of their program. However, this model is not perfect. Most notably, it does not support communication between processes. Neither does it inherently solve the issue of balancing the computational load across nodes.

High-level abstractions to dynamically balance the computation are therefore desired to allow non-expert programmers to

Chapter 2: Background

In the second chapter, we will recall useful background, covering existing parallel and distributed computing programming models and languages.

We will cover the existing state of the X10-style implementation of the Asynchronous Partitioned Global Address Space (APGAS) programming model. The semantics of the finish/async constructs will be demonstrated.

Chapter 3: Task Granularity Tuning for the lifeline-based multi-worker GLB

This scheme was first proposed in X10 but **suffered from implementation issues**. In the Java version discussed in this chapter, these issues were resolved by introducing a **yielding mechanism in the worker' s main routine**. The mechanism goes against the "work first" principle generally observed in pool threads and work-stealing scheme. However, it is necessary in this case to ensure that worker threads do not monopolize the resources and that steal requests coming from other nodes can be scheduled successfully.

The concept of **granularity** appears in various contexts. Among the parameters of the hybrid load-balancing scheme discussed in this chapter, this specific setting is particularly **important to achieve good performance**. This parameter determines the

number of micro tasks that worker threads process before checking on the works-stealing runtime. A value too low causes overhead through excessive runtime checking, while a value too high will cause starvation to appear. Finding a suitable compromise is necessary to achieve good performance.

The problem with this setting is that it can be quite arbitrary. On the one hand, the dynamic load balancer cannot choose a value which will work in any situation against any computation. On the other hand, users of dynamic load balancers cannot be expected to know in advance what a good value should be. In practice, users may go through a wasteful trial-and-error process to find a value which gives appropriate performance for their particular workload.

In this chapter, we will introduce **a tuning mechanism capable of dynamically adjusting the task granularity in the context of the hybrid lifeline-based global load balancer**. This mechanism monitors the most recent situation on each host and uses two criteria to determine if either excessive overhead or starvation is occurring. It then automatically adjusts the granularity as necessary, without any user input.

While observing starvation is relatively straightforward, observing overhead is more difficult. We present a criterion capable of determining if excessive runtime checking is occurring by exploiting a data race in the load-balancing scheme.

To evaluate the performance of this mechanism, we challenge it against 4 backtrack-search algorithms implemented using this load balancer. The tuning mechanism built using these criteria is capable of handling the variety of exploration searches, starting from an unfavorable setting, the granularity is adjusted to a satisfactory level within a few seconds. The tuning mechanism we develop is also **robust against variations in the problem implementation and the hardware used** to perform the computation. Moreover, this tuning mechanism **does not generate any noticeable overhead** compared to executions without the mechanism.

Chapter 4: Relocatable Distributed Collections

Modern clusters and supercomputers based on many-core processors make available to computer scientist a high level of parallelism both between nodes and within nodes. However, the complexity of such distributed systems requires dedicated programming languages for programmers to successfully harness these architectures.

The elegant abstractions of the APGAS model as implemented in X10 and Java allows for rather simple management of termination detection. However, **the lack of support for collective computation/communication across processes** is a handicap for distributed

object-oriented programming.

To fill-in this gap, we introduce "relocatable distributed collections" which complements the APGAS for Java library. Under the model we propose, a **distributed collection is defined on a group of processes**, meaning it can contain records on the set of processes it is defined on. We mimic the usual collections from the standard Java library that programmers are already familiar with. The key innovation of our library lies in the **relocatable nature of the entries recorded into our distributed collections**, which can be relocated from a process to another using high-level abstractions provided by our library.

To handle the distributed nature of the computation**, we introduce of notion of teamed methods**. Unlike regular methods, the teamed methods need to be called on each process on which collections are defined. They create synchronization points between asynchronous activities running on different processes. Common computation patterns, such as reductions, and other features of our library are implemented using this concept.

Using these features, **non-experienced programmers are capable of writing sophisticated distributed programs** that would otherwise remain out of reach, or too complex to apprehend. The most prominent example of this is the PlhamJ financial market simulator which uses the features of our library extensively. **The features and abstractions supported by our library made it possible to introduce dynamic load balancing** into this simulator. In cases where the performance of the hosts differs, it is now possible **to dynamically adjust the distribution of data objects to match the actual performance available** in the cluster.

Chapter 5: Integrated Global Load Balancer

While the facilities presented in the previous chapter allow programmers to explicitly manage the distribution of entries across processes, **implementing a load-balancing strategy for each application still comes with non-negligible effort**. It would be preferable if such entry relocation for the purpose of **load-balancing could instead be left up to the library**. However, automatic load balancing facilities imply that the distribution of entries is surrendered to the library, when up until now control over the distribution of entries was entirely left up to the programmer.

To resolve this dichotomy, **we introduce a specific context within which the integrated load balancer operates**. Within this context, the computation to perform on the entries contained in the underlying distributed collection are expressed using a staging /

submission system. Outside of this specific context, the control over the distributed collections is left entirely up to the programmer. This design choice has the advantage of providing clear boundaries for the operation of the integrated load balancer and contribute to the greater programmability of our programming model.

Internally, **we re-visit the lifeline-based global load balancer** to implement these load-balancing facilities. There are however a number of key differences between the original scheme intent and the implementation used in our integrated load balancer.

First, contrary to the original scheme where computation tasks are self-contained, the source of the computation is external to the tasks in our scheme. As a consequence, **entries of the distributed collection are relocated along with the computation tasks** when inter-host load balancing procedures occur.

Secondly, the fact that **multiple computation operating on the same collection** is permissible under our scheme **challenges the typical "single englobing finish" used in the original scheme**. Under the canonical APGAS finish/async programming model, asynchronous activities are only allowed to spawn new activities which belong to the same finish. To guarantee the correctness of our scheme, we removed this constraint and allowed asynchronous activities to participate into multiple finish constructs.

Finally, **the notion of "lifelines" takes up a new meaning** in the context of our distributed collections.

We show that our current implementation is capable of dynamically balancing the load to some degree, but that further work is needed to reduce the redundant work relocation in cases where no load imbalance actually occurs.

Chapter 6: Conclusion

The last chapter will summarize the main findings of the previous chapter and open on new avenues for research. In particular, the concept of relocatable distributed collections have the **potential to provide appropriate abstractions for elastic programs** in which processes can be dynamically dropped or added to the computation to match the evolving parallelism needs of the program.

（別紙１）　　論文審査の結果の要旨

氏名	FINNERTY Patrick Martin
論文題目	High-productivity Abstractions and Efficient Runtime for Dynamically Load-balanced Distributed Programs on Multi/Many-core Clusters マルチコア／メニーコアクラスタにおける動的負荷分散プログラムのための高生産性抽象化と効率的ランタイム実装法

審査委員	区　分	職　名	氏　　名
	主　査	教授	太田　能
	副　査	教授	大川　剛直
	副　査	教授	横川　三津夫
	副　査	准教授	鎌田　十三郎
	副　査		印

要　　旨

分散環境向け並列プログラムでは，従来，分散したデータ配置やノード間通信をプログラマが明示的に管理することが多い。また，マルチコアクラスタなどの分散・共有メモリのハイブリッド環境では，MPI と OpenMP などが併用される場合や，PGAS（Partitioned Global Address Space）モデル上での分散・並列ハイブリッドプログラミングがおこなわれる場合があるが，規則的なものを除いて分散データはプログラマにより明示的に管理されている。一方，アプリケーション分野によっては，探索問題など分割した各タスクの計算量を事前に見積もることが困難であることも多く，その際，並列化による性能向上は限定的となっている。負荷の不均等に対し，共有メモリ環境では自動動的負荷分散技術が効果を発揮している。一方，分散メモリ環境では計算とデータの関係がプログラマによって管理されており，タスク・データの自動再配置が困難であった。これに対し，Charm++ では，再配置可能なデータ構造を導入した上で，過分割したタスクの自動再配置により負荷分散を図っているものの，データ間の連想関係や各タスクのスケジューリングをシステムに全て任せる形となっており，アプリケーション領域も限定的であった。

本論文は，マルチコア/メニーコアクラスタを対象に，効率的動的負荷分散機構，要素の再配置が可能な分散集合ライブラリ，さらに両者の融合を図った研究成果をまとめたものになっている。具体的には，PGAS モデルを対象に，メニーコアクラスタ向け効率的負荷分散機構を実現するとともに，高い抽象度でデータの配置・移動を記述することができる分散集合ライブラリをデザイン・開発している。さらに，両者を融合することで，分散環境を対象とした動的負荷分散を可能とし，加えてプログラマがデータ間の連想関係や処理の流れを記述しつつ，自動負荷分散が可能なプログラミングモデルの実現を試みている。

本論文を構成する各章は次のようになっている。

第１章では，まず分散プログラミングにおける既存のプログラミングモデルの位置づけと負荷分散の必要性についてまとめられ，そのうえで本研究の目的と概要について述べられている。

第２章では，各分散プログラミング言語・モデルにおける分散データ処理の扱い，特にプログラマが容易に計算の局所性を管理できるか・データ分散を動的変更可能かについてまとめられている。

第３章では，メニーコアクラスタを対象とした動的負荷分散機構の実現法とそのタスク粒度の自動管理手法が提案されている。Saraswat らの提案した Global Load Balancer（GLB）を Yamashita らが既にハイブリッド化していたが，本研究では外部メッセージ処理を考慮した，より安定した multi-worker GLB を Java 言語上に実装することに成功している。また，本章では Oakforest-PACS という 1 ノード 68 コア構成のメニーコアクラスタを対象に，タスク粒度による影響が評価・確認されている。一般に負荷粒度が大きいと他ノードからのタスクリクエストの処理が遅れ飢餓状態が起こりやすい。一方，粒度が小さいとランタイムオーバヘッドが大きくなり，メニーコア環境では共有メモリアクセスによるコンテンションも起こりうる。本章では，飢餓状態やコンテンション発生をプロファイルから検出しタスク粒度を自動管理する手法が提案・実装され，複数のベンチマークアプリケーションにおいてタスク粒度の自動調整に成功している。なお，本章の内容は次の２件の学術論文をまとめたものになっている。

氏名	FINNERTY Patrick Martin

- Patrick Finnerty, Tomio Kamada, and Chikara Ohta, "Self-adjusting task granularity for global load balancer library on clusters of many-core processors," Proc. of the Eleventh International Workshop on Programming Models and Applications for Multicores and Manycores (PMAM2020), Association for Computing Machinery, New York, NY, USA, Article no. 4, pp. 1–10, Feb. 2020. (doi: 10.1145/3380536.3380539)
- Patrick Finnerty, Tomio Kamada, and Chikara Ohta, "A self-adjusting task granularity mechanism for the Java lifeline-based global load balancer library on many-core clusters," Concurrency and Computation Practice and Experience, 14 pages, Feb. 2021. (doi: 10.1002/cpe.6224)

第4章では，要素の再配置が可能な分散集合ライブラリのデザインと効果的な利用法が示され，加えて，本ライブラリの効率的実装法と性能評価結果が示されている。本ライブラリは，分散データ配置や要素移動に対する高い抽象度記述を目指してデザインされており，移動した要素群についてもライブラリがそのデータ配置を管理可能であり，負荷状況などに応じた要素移動を想定したプログラムを簡潔に記述可能となっている。本章では，実際に人工市場シミュレータ PlhamJ を改良することでトレーダーエージェントの再配置を可能としており，外部負荷変動のある実行環境下でタスク移動による性能向上が達成されている。なお，本章で示されたのはシステムによる自動負荷分散ではなく，プログラマにより明に記述・実現されたものである。本章では，各種データアクセスパターンに応じたライブラリ拡充についても示されている。例えば，キャッシュ可能分散配列および当該分散配列を対象とした二重ループ処理も表現可能であり，当該機能を用いた分子動力学プログラムも実現されている。当ライブラリはマルチコアクラスタを想定して設計されており，単純にノード間・ノード内データ並列を容易に記述できるだけでなく，計算結果の reduction においても，スレッド毎にデータを reduction した後，スレッド間で reduction するといった操作も容易に記述可能となっている。なお，本章の内容は下記の学術論文をまとめたものになっている。

- Patrick Finnerty, Yoshiki Kawanishi, Tomio Kamada, and Chikara Ohta, "Supercharging the APGAS programming model with relocatable distributed collections," Scientific Programming. （採録決定・出版手続き中）

第5章では，第4章で述べた分散集合ライブラリに動的負荷分散機能が統合され，加えて，複数の分散集合に対する一連のデータ並列演算に対するスケジューリングを，容易かつ柔軟に記述可能なプログラミングモデルが提案されている。これまでの分散集合ライブラリでは，データ分散や移動を抽象的に扱うことはできたが，データ分散はプログラマが明に指定するものであり，またデータと共に移動可能なタスクを表現する手段も提供されていなかった。本章では，分散集合に対する GLB 演算が導入され，集合要素に対するデータ並列演算において，負荷状況に応じた要素再配置をともなったノード間タスク移動が表現されている。GLB 演算は underGLB ブロック内で実行されるが，複数の分散集合に対する GLB 演算の並列実行や，future を用いた非同期処理も容易に記述可能であるのが特徴的である。分散プログラミングでは，クリティカルパス短縮などのため各種計算をオーバラップさせることも多いが，本提案では動的負荷分散環境下で複数のデータ並列演算のスケジューリングを簡潔に表現することに成功している。本機能は，分散集合ライブラリにおける DistCol クラスに実装されている。3章で述べた multi-worker GLB をベースに，領域単位のタスク管理・分割による効率的な実装法も提案されており，ノード内データ並列処理については，一般の繰り返し文に匹敵する性能が達成されている。本ライブラリは K-means，PlhamJ を対象とした性能評価がおこなわれており，動的負荷分散機能が有効に機能していることも示されている。第4章で示された負荷分散とは異なり，システムがノード間の負荷バランスに応じて動的に自動負荷移動を実現することに成功している。なお，本章の内容は下記の学術論文をまとめたものとなっている。

- Patrick Finnerty, Tomio Kamada, and Chikara Ohta, "Integrating a global load balancer to an APGAS distributed collections library," Proc. of the Thirteenth International Workshop on Programming Models and Applications for Multicores and Manycores (PMAM2022), Association for Computing Machinery, New York, NY, USA, pp. 55–64, April 2022. (doi: 10.1145/3528425.3529102)

第6章は結論であり，本論文の学術的貢献をまとめるとともに，今後の研究課題について述べられている。なお，本論文の内容は，2件の査読付き国際会議と2件の学術的専門誌で公表された成果に基づいている。

このように，本論文は，分散プログラミングにおいて従来おこなわれていた明示的分散データ配置・ノード間通信の管理の代わりに，プログラマがデータ分散やタスクフローを抽象的に記述することを許しつつ，計算状況や実行環境の変化に応じた自動負荷分散・移動を実現するための手法を示したものであり，学術的貢献の大きい価値のある集積である。提出された論文はシステム情報学研究科学位論文評価基準を満たしており，学位申請者の FINNERTY Patrick Martin は，博士（工学）の学位を得る資格があると認める。