

# **Towards Faster Combinatorial Search: Performance-Driven Workstealing Policy in YewPar**

Hao Xie

School of Computing Science  
Sir Alwyn Williams Building  
University of Glasgow  
G12 8QQ

A dissertation presented in part fulfilment of the requirements of  
the Degree of Master of Science at The University of Glasgow

24th August 2023

## 摘要

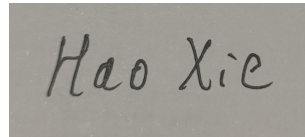
abstract goes here

## Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: Hao Xie

Signature:

A rectangular box containing a handwritten signature in dark ink. The signature is written in a cursive style and reads "Hao Xie".

## **Acknowledgements**

acknowledgements go here

# 目录

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	A section . . . . .	5
1.1.1	A subsection . . . . .	5
<b>2</b>	<b>Survey</b>	<b>6</b>
<b>3</b>	<b>Design and implementation</b>	<b>7</b>
<b>4</b>	<b>Evaluation</b>	<b>8</b>
<b>5</b>	<b>Conclusion</b>	<b>9</b>
<b>A</b>	<b>First appendix</b>	<b>10</b>
A.1	Section of first appendix . . . . .	10
<b>B</b>	<b>Second appendix</b>	<b>11</b>
	参考文献	<b>12</b>

# Chapter 1: Introduction

精确组合搜索对于包括约束编程、图匹配和计算代数在内的广泛应用都是必不可少的。而组合问题是通过系统地探索搜索空间来解决的,这样做在理论上和实践中都很难计算,其中精确搜索则是探索整个搜索空间并给出可证明的最佳答案。概念上精确的组合搜索通过生成和遍历代表备选选项的(巨大的)树来进行。结合并行性、按需树生成、搜索启发式和剪枝可以减少精确搜索的执行时间。由于巨大且高度不规则的搜索树,并行化精确组合搜索是极具挑战性的。

而其中有名为YewPar[1]的框架,这是第一个用于精确组合搜索的可扩展并行框架。YewPar旨在允许非专业用户从并行中受益;重用编码为算法骨架的并行搜索模式(to reuse parallel search patterns encoded as algorithmic skeletons);并能在多个并行架构上运行。

能并行加速搜索是YewPar的一个关键特性,这是通过各个节点的多个worker在本地任务池无任务时向其它节点的任务池窃取任务来实现节点空闲资源的利用。而YewPar在本地任务池无任务时,是随机选取节点来窃取任务的,其中Workstealing是一种被广泛研究和应用的并行调度策略,它允许空闲的处理器从繁忙的处理器中“窃取”任务。然而,很多如YewPar这样的workstealing调度器往往采用偏随机窃取的策略,这往往会导致不必要的开销和延迟,浪费了大量试探窃取任务的时间,同时导致各节点的相对负载不均衡,延长了最终完成的时间。

与此同时,随着并行计算和多核处理器的普及,有效的任务调度变得越来越重要。相比之下,本文提出了Performance-Driven Workstealing Policy,它能够搜集多项性能指标,如各节点的任务执行时间、处理器的工作/空闲时间比例和任务池获取的耗时,并通过自研的Time-Optimized Workstealing Strategy算法计算出最优窃取目标节点并缓存和定时刷新,当有worker空闲时便能直接从缓存获取最近一段时间的最优窃取目标,从而能缩短完成全部任务所需的时间。

本文剖析了具体的设计与实现细节,其中Performance-Driven Workstealing Policy在搜集性能参数时采用了平台无关设计,并不涉及具体的系统参数调用命令,而是从YewPar内部和底层的HPX[2]框架进行数据收集,从而能够在不同的硬件平台上正常运行。其中Time-Optimized Workstealing Strategy的核心思想是计算各节点执行本地任务的所需时间的预期和获取节点任务池任务的耗时的预期,并优先选取所需时间预期最大的节点同时尽量缩短不必要的获取任务的额外耗时,从而降低各节点的worker空闲率的同时缩短完成全部任务所需的时间。其中定时刷新最优窃取目标缓存的任务由一种动态调整刷新时间的自动刷新任务来主要负责,各节点都会部署一个这样的刷新器,它通过间隔一个动态时间后执行刷新性能参数信息并计算最优窃取目标最后将最优目标进行缓存来实现刷新最优窃取目标缓存的目的。而空闲的worker则会在试图获取缓存目标任务失败时进行一次称为辅助刷新的操作,帮助此时可能处在休眠的刷新器进行刷新最优窃取目标缓存的任务。这些工作结合起来便能够实现Performance-Driven Workstealing Policy的加速效果。

本文同时对改进workstealing策略后的YewPar进行了评估, 评估在具有多核机器的Beowulf集群上进行, 结果表明,改进后的YewPar在不同节点数量和线程下相比原YewPar平均能够获得更好的性能,能在不影响搜索结果的情况下不同程度的缩短执行完全部任务所需的时间.

## **1.1 A section**

### **1.1.1 A subsection**

Please note your proposal need not follow the included section headings - this is only a suggested structure. Also add subsections etc as required.

## Chapter 2: Survey

Background concepts (if required) and overview of relevant previous work (critically evaluate strengths and weaknesses).

组合搜索的历史和发展:简要回顾组合搜索的发展历史,特别是精确组合搜索的发展,并大致介绍YewPar. 并行调度的历史和发展:简要回顾并行调度策略的发展,特别是workstealing方法. 现有的workstealing方法:详细描述当前的workstealing调度方法和它们的特点,并介绍YewPar所采用的workstealing方法. 现有方法的局限性:基于文献,讨论传统方法的限制和挑战。相关的优化方法:回顾过去的研究中,针对workstealing或类似问题所做的优化尝试。研究空白:指出历史针对workstealing的不足或者空白,和YewPar的workstealing的不足。(过去的workstealing只在某种方面进行了优化尝试,而YewPar则只通过了随机策略进行窃取,本文将结合历史工作窃取思路更进一步,在YewPar上进行改进)

先介绍组合搜索背景,引出YewPar 然后介绍workstealing的背景,引出YewPar中的workstealing,并指出其存在的问题之后引出本文的工作,即改进YewPar中的workstealing

**Workstealing** 是并行编程中的一个核心概念,最早由 [引用] 提出。这种方法的优势在于其分散的性质,每个处理器都独立地管理自己的任务队列,从而减少了全局同步的开销。多年来,这种方法已被广泛应用于各种并行框架和库中,如 Cilk、TBB 和 OpenMP。

尽管 workstealing 已被证明是一种高效的调度策略,但仍有许多研究致力于进一步优化其性能。例如,Archibald 等人在 YewPar 框架中提出了一个针对精确组合搜索的并行框架,但它仍然采用了传统的随机窃取策略。

随机窃取策略的一个主要问题是它可能导致不必要的任务迁移和延迟,尤其是在不均匀的任务分布下。近年来,有一些研究开始考虑使用历史信息 and 运行时指标来指导窃取决策,从而提高调度的效率和性能 [相应的引用]。



## **Chapter 3: Design and implementation**

The content of these chapters depends on the project and should be agreed with your supervisor (e.g. description of the solution, evaluation results, etc).

## **Chapter 4: Evaluation**

## **Chapter 5: Conclusion**

Main conclusions of your project. Here you should also include suggestions for future work.

## **附录 A: First appendix**

### **A.1 Section of first appendix**

## **附录 B: Second appendix**

## 参考文献

- [1] Blair Archibald, Patrick Maier, Robert Stewart, and Phil Trinder. Yewpar: Skeletons for exact combinatorial search. In *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPOPP '20, page 292–307, New York, NY, USA, 2020. Association for Computing Machinery.
- [2] Hartmut Kaiser, Thomas Heller, Bryce Adelstein-Lelbach, Adrian Serio, and Dietmar Fey. Hpx: A task based programming model in a global address space. In *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*, PGAS '14, New York, NY, USA, 2014. Association for Computing Machinery.