密级:_____

中国科学院大学
University of Chinese Academy of Sciences

# 硕士学位论文

面向高通量处理器的 Benchmark 研究


作者姓名：苗福涛

指导教师：张志敏　研究员

中国科学院计算技术研究所


学位类别：工学硕士

学科专业：计算机系统结构

研究所：中国科学院计算技术研究所

**2015 年 5 月**

# Benchmark Research for High-Throughput Processor

**By**

Miao Futao

**A Dissertation/Thesis Submitted to**

**University of Chinese Academy of Sciences**

**In partial fulfillment of the requirement**

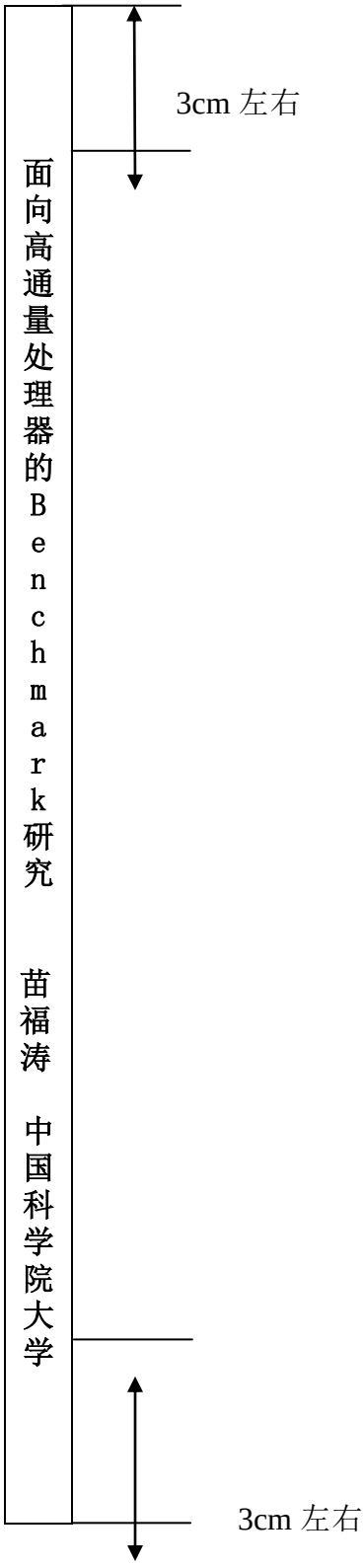
**For the degree of**

**Master of Computer Architecture**

**Institute of Computing Technology**

**May, 2015**

书脊



# 声　　明

我声明本论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，本论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

作者签名：　　日期：

# 论文版权使用授权书

本人授权中国科学院计算技术研究所可以保留并向国家有关部门或机构送交本论文的复印件和电子文档，允许本论文被查阅和借阅，可以将本论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编本论文。

（保密论文在解密后适用本授权书。）

作者签名：　　导师签名：　　日期：

# 摘 要

随着大数据时代的到来，在过去的十几年间，互联网、云计算、大数据等方面的技术飞速发展。据统计，全球产生的信息量正以每天 2.5EB 的速度增长。在此背景之下，数据中心技术迅速发展起来，而当前数据中心主要服务于 IT、互联网和电信相关的企业，数据中心所提供的应用多为高通量应用，所谓高通量应用是指主要关注指标为吞吐率的一类应用，整个应用通常是由大量松耦合的规模较小的作业组成，而不是一个规模较大的作业。高通量应用具有大数据、高并发、实时性等典型的应用特点。

为了在处理器级别更好的适应高通量应用的特征，高通量处理器的研究成为当前学术界的热门，高通量处理器主要关注的是如何提高对作业的吞吐效率，当前多为多核/众核结构。而高通量处理器的研究需要面向高通量处理器的 Benchmark 的研究作为基础，面向高通量处理器的 Benchmark 的研究主要有两方面的意义：1)指导高通量处理器的设计；2)对高通量处理器的性能进行测评。

本文对面向高通量处理器的 Benchmark 的研究主要集中于两个方面：

1. 针对目标应用类型是高通量类应用，完成了高通量应用的分类与分析工作。本文首先提出了一种基于高通量需求特点的高通量应用分类模型，此模型将高通量应用分为数据处理类、数据服务类和实时交互类三种类型。并基于此分类模型对各个典型高通量应用进行了分析与分类。然后，提取了各个典型高通量应用的核心 Workload，并分析了各个应用的基本程序特征。
2. 针对目标平台是高通量处理器，完成了 Benchmark 的实现。首先，针对“高通量处理器”这个目标平台的特点，本文提出了一种基于线程的作业处理节点并行化模型思想，并针对三类高通量应用各自的特点，说明了模型的具体结构。然后，从三类高通量应用中各选取了代表性的应用和 Workload，完成了基于上述模型的 Benchmark 的实现。

本文最后进行了对 Benchmark 的实验评估。首先，对本文实现的 Benchmark 的作业并发性、作业之间耦合性、访存需求、Cache 使用效率和访存位宽等几个指标进行了实验评估，实验结果证明了本文所实现的 Benchmark 有效的反映出了高通量应用应该具有的基本特征。然后，使用本文所实现的 Benchmark 对 TILE-Gx 和 Xeon 两种处理器的并行加速能力做了评估，评估结果说明了本文所实现的 Benchmark 能够对不同处理器进行有效的测评。

**关键词**：高通量应用，高通量处理器，基准测试程序

**Benchmark Research for High-Throughput Processor**

Miao Futao (Computer Architecture)

Directed By Zhang Zhimin

With the coming of the era of big data, in the past ten years, the technology about Internet, cloud computing, big data and other aspects have developed rapidly. 2.5EB data is being generated per day all over the world. In this background, data center technology has developed rapidly, and the current data center primarily serves the IT, the use of Internet and telecommunications-related businesses, which are high-throughput applications. The so-called high-throughput applications are applications that focus on throughput efficiency. Usually, high-throughput applications content a large number of small-scale jobs composition loosely coupled, rather than one large-scale job. The typical characteristics of high-throughput applications are large-scale data set, high concurrency and real time.

In order to better adapt to the characteristics of high-throughput applications in the processor level, the study of high-throughput processors becomes popular. High-throughput processor's main attention is how to improve the throughput efficiency of operations, currently most high-throughput processors are multi-core or many-core architecture. The study of high-throughput processors needs high-throughput processor Benchmark as a basis. The research for high-throughput processor Benchmark has mainly two meanings: 1) to guide the design of high-throughput processor; 2) to evaluate performance of high-throughput processor.

Research of this paper focuses on two aspects:

1. The target application type is high-throughput applications, so this paper completes the classification and analysis of high-throughput applications. This paper presents a high-throughput applications' classification model that will divide high-throughput applications into data processing, data service and real time three classes. Completes the classification and analysis of high-throughput applications based on this classification model. Then, this paper extracted core workloads of typical high-throughput applications, and analyzed the basic characteristics of each application.
2. According to the target platform is a high-throughput processor, this paper completed Benchmark implementation. First, against the characteristics of "high-throughput processor", this paper presents a multi processing node parralell model based on pthread, and the implementation of the model for three high-throughput applications according to their own characteristics. Then, from the three types of high-throughput applications, selected representative applications and Workloads, implement the Benchmark based on the above model.

This paper makes experiments to evaluate the Benchmark. First, the paper makes

experiments to evaluate concurrency and coupling between jobs, demand for memory access, efficiency of Cache and memory access bit width. The experimental results prove that the Benchmark of this paper effectively reflects the basic characteristics of high-throughput applications. Then, use this Benchmark to test parallel acceleration of TILE-Gx and Xeon processors and the results prove that the Benchmark implemented in this paper can be used on different processors' evaluation.

**Keywords:** high-throughput applications, high-throughput processor, Benchmark

# 目 录

# 图目录

# 表目录

# 第1章 引言

## 1.1 本文的研究背景和动机

大数据时代已经来临。在过去的十几年间，互联网、云计算、大数据等方面的技术飞速发展。据统计，全球产生的信息量正以每天 2.5EB 的速度增长[1]。而在互联网用户量方面，据国际电信联盟在 2014 年的统计，全球互联网用户量已超过 30 亿，占全球人口的 40%。

全球互联网使用人数的快速膨胀和全球数据量的大量积累，对用户服务系统的技术提出了更严峻的挑战，IT、互联网和电信企业需要更强大的数据处理系统的支撑。在这个背景下，数据中心技术迅速发展起来。数据中心为相关的企业提供了强大的基础设施，提供足够强大的服务。以 Google 数据中心为例，Google 之所以凭借强大的搜索能力成为搜索引擎乃至整个互联网行业的翘楚，与它强大的数据中心是分不开的。不仅如此，Google 将其数据中心的富裕计算能力租用给其他的公司，为这些公司提供必需的计算能力。

当今数据中心，主要服务于 IT、互联网和电信相关的企业，数据中心所提供的应用多为高通量应用，所谓高通量应用是指主要关注指标为作业吞吐量的一类应用，整个应用通常是由大量松耦合的规模较小的作业组成，而不是一个规模较大的作业。高通量应用具有大数据、高并发、实时性等典型的应用特点[2]。

当前数据中心采用面向通用计算领域的处理器，在结构设计时并未充分考虑到高通量应用的典型特征。这种以高吞吐效率为主要性能指标的新型应用场景，使得当前主流的商业处理器在处理这类新型应用时能效比更为低下。这主要体现在当前商业多核/众核处理器的结构设计原则与高通量应用程序执行行为特征的不匹配上。

在这种发展趋势之下，适用于处理数据中心典型高通量应用的高通量处理器成为学术界和工业界的研究热门。

高通量处理器的研究离不开面向高通量处理器的 Benchmark 的研究。面向高通量处理器的 Benchmark 的研究主要有两方面的意义：

**1)指导高通量处理器的设计**

高通量处理器的应用主要是针对数据中心高通量应用。这就意味着高通量应用的特点是高通量处理器结构设计的决定性因素。这里的高通量应用特点主要包括并发性、计算量、计算特点、访存量、访存特点等。

基于对面向高通量处理器的 Benchmark 的分析，可以对高通量处理器的设计做出指导。

**2)对高通量处理器性能进行测评**

Benchmark 作为计算机处理器性能评价的重要手段，已经被学术界和工业界所共同认可。对应不同的评价需求，学术界已经存在大量不同侧重的 Benchmark 作为测试评价

的工具。

而对于高通量处理器的测试，也需要一套针对其应用目标、结构特点和性能需求的 Benchmark 作为测试评价工具。

## 1.2 本文的主要贡献

由于高通量处理器与传统的高性能处理器在编程模型、性能目标和可靠性等方面的不同，因此，传统的 Benchmark 并不适合评价高通量计算机。而在高通量处理器领域，已有的 Benchmark 测试程序集都是针对整个系统，而没有公认的代表程序集作为处理器芯片微结构设计的基准程序，如面向高性能科学计算领域的如 SPEC 和 SPLASH2 等。

在[3]中提到，一个 Benchmark 集必须有特定的目标系统和特定的目标应用类型。而面向高通量处理器的 Benchmark 的目标系统是高通量处理器，目标应用是数据中心中具有高通量特性的应用。

针对上述两点以及上一节所说当前研究中的不足之处，本文的主要研究内容可以细分为以下两点：

**1)基于高通量需求模型的高通量应用分类与分析**

a) 提出基于高通量需求的高通量应用分类模型。

b)分析数据中心使用量较大的应用，依据高通量需求模型进行分类。

c)提取每一应用领域的核心 workload。

d)总结分析每一类高通量应用的程序特征，最终的 Benchmark 需要能反映出这些特征。

**2)面向高通量处理器的 Benchmark 的设计**

a)针对于目标平台是高通量处理器和目标应用是高通量应用这两点，提出一种合适的编程模式：基于线程的作业处理节点并行化模型。

b)由于三类高通量应用的每一类中各个应用的基本特征都是相同的，所以，从高通量应用的每一类中，选取使用量最多的一个应用来代表此类高通量应用，使用基于线程的作业处理节点编程模式实现其Benchmark，形成初版的面向高通量处理器的Benchmark集。

c)实验分析所设计的 Benchmark 反应出了各类高通量应用的程序特征。

## 1.3 论文的组织结构

第一章，引言，首先介绍了当今时间互联网技术和数据中心的发展现状和趋势，由此引出高通量应用作为典型的数据中心应用的重要地位，针对高通量应用，需要高通量处理器的研究，从而需要相关的 Benchmark 的研究。然后介绍了本文的主要贡献和论文的组织结构。

第二章，相关研究，介绍了当前 Benchmark 研究领域的发展现状和高通量应用发展

现状，作为本文研究的理论基础之一。

第三章，高通量应用的分类与分析，首先，根据不同高通量应用反映出的高通量需要的不同，提出了一种基于高通量需要的应用分类模型，在此基础上对典型的高通量应用进行分类；然后，对典型的高通量应用进行分析，提取出各个应用中的典型 Workload。

第四章，面向高通量处理器的 Benchmark 设计与实现，首先从每一类高通量应用中选取了一个，作为要实现 Benchmark 的应用领域，确定了哪些 Workload 是有必要实现的；然后，根据高通量处理器的固有特征，提出了每一类高通量应用在实现中需要有的并行化模型；最后，具体介绍了 Benchmark 中各个测试程序的实现方式。

第五章，实验，首先，从不同的角度进行实验，分析实验数据，证明所设计的 Benchmark 反映出了高通量应用所具有的基本的应用特征。然后，采用面向高通量处理器的 Benchmark，对 Xeon 和 Tilegx 两种处理进行测评，根据高通量应用的作业特点和需求特点，测评主要关注两种处理器在处理高通量应用 Benchmark 时并行加速比方面反映出的性能。

第六章，结束语，介绍了此论文研究结果在相关工作中的应用现状，对本文的工作进行了总结，对未来的工作做了展望。

# 第2章 相关研究

Benchmark 的研究一直是计算机领域一个热门方向，也是一个充满挑战的方向。Benchmark 的主要作用是对计算机系统或者计算机的各个组件进行测试评价，从而更好的指导计算机系统的设计或者指导消费者选择合适的计算机产品[4][5]。

而对处理器结构的研究作为计算机系统结构研究中的重点方向，当然也离不开对相应的 Benchmark 的研究的支持。Benchmark 是当今计算机界通用的和普遍接受的用于评测计算机和处理器性能的方法。

## 2.1 传统高性能计算领域 Benchmark

高性能计算领域的研究已经有几十年的历史，而高性能计算领域的 Benchmark 也已经相对成熟，业界已经有很多比较成功的基准测试集，如 SPEC 基准测试体系[5]，HPCC 基准测试集[6]，PARSEC 基准测试集[7]等。当然，每种基准测试集都有不同的侧重点。

SPEC（Standard Performance Evaluation Cooperation， SPEC）标准性能评测机构是一个全球性的、权威的第三方应用性能测试组织，它针对计算机领域的不同方向，提出了一系列的计算机方面的测评标准。例如SPEChpc是准对高性能计算领域的测评程序集，SPECweb是针对网络服务器方面的测评程序集等等，SPEC提出的这些测评标准，在业界得到了广泛的认可和应用。

SPEC CPU2006 是业界常用的一套程序集，主要是对处理器计算性能进行测试。程序集主要由 29 个程序组成，包括整型计算和浮点计算，覆盖到了大部分计算领域。测试程序集的构成如表 2.1所示。

表 2.1 SPEC CPU 2006 测试程序集

| 测试程序名 | 程序类型 | 源码语言 | 说明 |
|---|---|---|---|
| perlbench | INT | C | Perl language |
| bzip2 | INT | C | Compression algorithm based on block classification |
| Gcc | INT | C | GNU C compiler |
| Mcf | INT | C | Optimal combination of bus schedule |
| gobmk | INT | C | A program playing go |
| hmmer | INT | C | Gene sequence search |
| Sjeng | INT | C | A program playing chess |
| libquantum | INT | C | Quantum computer simulation program |
| h264ref | INT | C | A video compression algorithms |
| omnetpp | INT | C++ | Use OMNET++ discrete event simulator simulates Ethernet |
| Aster | INT | C++ | Path discovery algorithm |
| xalancbmk | INT | C++ | Transfer XML files into other file types |
| bwaves | FLOAT | Fortran | Calculating three-dimensional supersonic laminar flow transient viscous |
| gamess | FLOAT | Fortran | Quantum chemistry |
| Milc | FLOAT | C | Quantum chromodynamics |

| | | | |
|---|---|---|---|
| zeusmp | FLOAT | Fortran | Computational fluid dynamics simulation |
| gromacs | FLOAT | C，Fortran | Molecular dynamics simulation |
| cactusADM | FLOAT | C，Fortran | Evolution equations of general relativity |
| leslie3d | FLOAT | Fortran | Large Eddy Simulation of Computational Fluid Dynamics |
| namd | FLOAT | C++ | Large biomolecular simulation |
| dealII | FLOAT | C++ | An adaptive finite element and error estimates of C ++ libraries |
| soplex | FLOAT | C++ | Using the simplex algorithm and sparse linear algebra solving linear programming |
| povray | FLOAT | C++ | Images raytracing |
| calculix | FLOAT | C，Fortran | Dimensional linear and nonlinear finite element code |
| GemsFDTD | FLOAT | Fortran | Finite-difference time-domain method for solving Maxwell's equations |
| tonto | FLOAT | Fortran | Object-oriented quantum chemistry package |
| lbm | FLOAT | C | Three-dimensional incompressible fluid simulation |
| wrf | FLOAT | C，Fortran | Weather models |
| sphinx3 | FLOAT | C | Speech recognition system |

可以看出，SPEC CPU 2006 覆盖到了计算机编程、算法、人工智能、基因、流体力学、分子动力学和量子计算等各个方面，保证了测试的完整性。

PARSEC (The Princeton Application Repository for Shared-Memory Computers)是一个多线程应用程序组成的测试程序集。具有多线程（并发性）、新型负载、非针对高性能和研究性等几个典型特征。PARSEC中的选取的测试程序覆盖了多个领域，如计算机视觉、视频编解码、金融分析、动画物理学和图像处理等。PARSEC基准测试程序集的构成如表 2.2 所示。

**表 2.2 PARSEC 基准测试程序集**

| 程序名 | 程序内容 |
|---|---|
| Blackscholes | Option pricing with Black-Scholes Partial Differential Equation (PDE) |
| Bodytrack | Body tracking of a person |
| Canneal | Simulated cache-aware annealing to optimize routing cost of a chip design |
| Dedup | Next-generation compression with data deduplication |
| Facesim | Simulates the motions of a human face |
| Ferret | Content similarity search server |
| Fluidanimate | Fluid dynamics for animation purposes with Smoothed Particle Hydrodynamics (SPH) method |
| Freqmine | Frequent itemset mining |
| Raytrace | Real-time raytracing |
| Streamcluster | Online clustering of an input stream |
| Swaptions | Pricing of a portfolio of swaptions |
| Vips | Image processing |
| X264 | H.264 video encoding |

HPCC 是另一个典型处理器高性能计算能力基准测试集，它包含 HPL、DGEMM、STREAM、PTRANS、FFTE、RandomAccess 和带宽延迟测试等七个主要的测试程序，弥补了 TOP500 高性能计算机排名只使用 Linpack 的不足，通过这七个测试程序，可以对系统和处理器的各个部分得到充分的测试，包括浮点计算能力、持续内存带宽大小、处理器协作能力、随机内存访问效率和内部高速互联网络的性能等各个方面。可以看出，HPCC 的这些测试程序没有针对某种应用，而是是属于很有针对性的核心类的测试程序。

## 2.2 大数据相关领域的 Benchmark 研究现状

学术界对当今比较热门的大数据相关应用领域的 Benchmark 也已经有了很多的研究成果。

HiBench[8]是 Intel 开放的一个 Hadoop Benchmark Suit，包含 9 个典型的 Hadoop 负载，分为 Micro benchmarks、HDFS benchmarks、web search benchmarks、machine learning benchmarks 和 data analytics benchmarks 五类。用于测评运行 Hadoop 的集群的性能。Micro Benchmarks 包含 Sort、Wordcount 和 Terasort 三个基本的 Hadoop 应用，测试基本的计算性能; HDFS Benchmarks 包含一个增强的 DFSIO，用于测试 HDFS 的吞吐量; Web Search Benchmarks 用于测试 Hadoop 用于网络服务时的性能，包含一个 Nutch indexing 和一个 PageRank 算法；Machine Learning Benchmarks 用于测试 Hadoop 系统对当前比较热门的机器学习领域的支持能力，包含一个分类算法 Bayes 和一个聚类算法 Kmeans；Data Analytics Benchmarks 用于对 Hadoop 的数据处理能力进行测试，包含一个 Hive Query 用例。

YCSB（Yahoo! Cloud Serving Benchmark）[9]是 2010 年 Yahoo 研究院针对 NoSQL 系统开发的开源基准测试框架。随着大数据时代的到来，互联网和 IT 相关企业所要存储的数据量呈指数级增长，而数据的不规则性特点也变得越来越显著，这对数据库的架构提出了性的需求，传统的关系型数据库已经不能满足相关的数据存储需求。在这种背景下，NoSQL 迅速发展起来，NoSQL 是非关系型数据库的典型代表，更适合于当前大数据时代存储数据具有数据量大，分布式和数据格式多样化等特点。YCSB 为这种非关系型数据库的性能提供了一套标准测试。如表 2.3所示是 YCSB 的负载组织情况。

表 2.3 YCSB 自带的负载结构

| Workload 类型 | 操作类型比率 |
|---|---|
| workloada | update/read(0.5/0.5) |
| workloadb | read/update(0.95/0.05) |
| workloadc | readonly(1) |
| workloadd | read/insert(0.95/0.05) |
| workloade | scan/insert(0.5/0.5) |
| workloadf | read/readmodifywrite(0.5/0.5) |

DCBench[3]是一个对于数据中心负载的 Benchmark 集合。第一个发行版本含有 19 个具有代表性的负载。19 个负载时多样化的，根据应用特征的不同，可以分为 on-line 和 off-line 两类，根据编程模型的不同，可以分为 MPI、MapReduce 等。一个 Benchmark

集必须有特定的目标系统和特定的目标应用类型[3]，DCBench 的目标系统是数据中心系统，目标应用类型是数据中心系统上运行的应用。基于此，DCBench 具有的特点是：1)具有代表性的应用负载，即数据中心系统的典型应用负载；2)多样化的编程模型；3)分布式；4)使用新兴的技术

DCBench 的主要内容如表 2.4所示。

表 2.4 DCBench

| Category | Workloads | Programming model | language | source |
|---|---|---|---|---|
| Basic operation | Sort | MapReduce | Java | Hadoop |
| | Wordcount | MapReduce | Java | Hadoop |
| | Grep | MapReduce | Java | Hadoop |
| Classification | Naïve Bayes | MapReduce | Java | Mahout |
| | Support Vector Machine | MapReduce | Java | Implemented by ourselves |
| Cluster | K-means | MapReduce | Java | Mahout |
| | | MPI | C++ | IBM PML |
| | Fuzzy k-means | MapReduce | Java | Mahout |
| | | MPI | C++ | IBM PML |
| Recommendation | Item based Collaborative Filtering | MapReduce | Java | Mahout |
| Association rule mining | Frequent pattern growth | MapReduce | Java | Mahout |
| Segmentation | Hidden Markov model | MapReduce | Java | Implemented by ourselves |
| Warehouse operation | Database operations | MapReduce | Java | Hive-bench |
| Feature reduction | Principal Component Analysis | MPI | C++ | IBM PML |
| | Kernel Principal Component Analysis | MPI | C++ | IBM PML |
| Vector calculate | Paper similarity analysis | All-Pairs | C&C++ | Implemented by ourselves |
| Graph mining | Breadth-first search | MPI | C++ | Graph500 |
| | Pagerank | MapReduce | Java | Mahout |
| Service | Search engine | C/S | Java | Nutch |
| | Auction | C/S | Java | Rubis |
| Interactive real-time application | Media streaming | C/S | Java | Cloudsuite |

LinkBench[10]是 Facebook 开发的一套用于对社交网络数据库进行性能测评的工具集。社交网路中，最重要的数据组织方式是社交图谱，社交图谱用于保存社交网络中人与人之间以及相关对象的互联关系，而社交图谱最终是存放在社交网络数据库中的，因而，相关数据库架构的好坏和对社交图谱的支持能力，对整个社交网络的性能具有决定性的影响。LinkBench 作为一套专门的测试工具集，可以很有针对性的对社交网络数据库进行测试，从而给出合理的评价以及提出相应的设计指导。

CloudSuite[11]是针对当前最为热门的云计算而开发的一套测试程序集，Scale-out 应用特点是当前云计算应用的主要特征，随着云计算技术的迅速发展，此类应用越来越多，CloudSuite2.0 选取了 Data Analytics、Data Caching、Data Serving、Graph Analytics、Media Streaming、Software Testing、Web Search 和 Web Serving 八个 Applications，分别对应了数据分析、数据缓存、数据服务、图分析、流媒体、软件测试、网络搜索和网络服务等云计算中常用的负载。

BigDataBench[12][13]是由中科院计算所开发的一套互联网大数据应用相关的 Benchmark 集，此 Benchmark 集的主要特点是覆盖范围较广，覆盖到了互联网应用的大部分大数据场景。而且，此 Benchmark 集保留了真实的应用场景，使用了商业互联网系统中实际使用中的软件站，使用了真实的数据集，这使得对系统的测试更加真实可靠。

## 2.3 已有 Benchmark 相关研究的不足之处

一个 Benchmark 必须有特定的目标系统和特定的目标应用类型[3]。对于本文所做研究，面向高通量处理器的 Benchmark 的目标系统是高通量处理器，目标应用类型是高通量应用。针对这两个方面，当前已有的 Benchmark 研究尚有不足之处，不能完全满足需求。

针对传统的高性能 Benchmark，高通量处理器不同于传统的高性能处理器，高通量处理器需要运行数据中心高通量应用，数据中心的高通量应用所处理的作业，一般是大量规模较小、耦合性较低的作业的集合，而不是单个作业规模很大的应用[14]。高通量应用主要关注的是吞吐率[15]。而从 2.1 小节的介绍可以看出，传统的 Benchmark 主要是针对科学计算领域的应用，主要关注的是处理器的高性能处理能力，如整数计算能力和浮点计算能力[16]，而没有考虑到新型的大数据应用场景中的应用程序特点，因此，传统的 Benchmark 不适用于高通量处理器。

对于近年来发展较为热门的大数据相关应用领域的 Benchmark，对于高通量处理器的测评也有不足之处。从上面的介绍可以看出，这些 Benchmark 主要是用于对系统级进行性能测试和评价的，而不是对芯片级的测试和评价。例如，HiBench 用于对 Hadoop 集群性能的测试，YCBS 用于对 NoSQL 数据库系统的测试，LinkBench 用于对社交图谱数据库的测试，BigDataBench 主要关注互联网服务系统整体性能等等。系统级的评价主要关注系统运行的整体性能，如集群内部协同工作的效率，板间互连和通信的效率，系

统软件栈的性能等。而芯片级的测试和评价则主要关注芯片内部的工作情况，如众核处理器中多个核的并行处理能力，线程在处理器中的调度效率，共享存储的利用效率，cache的命中率，数据通路的使用效率等。因此，虽然这些 Benchmark 的目标应用具有高通量应用的特点，但是他们的目标系统不是处理器，因而，也不适合于对高通量处理器的性能测试和评价。

针对本文所涉及的研究领域，当前已有研究尚有如下两点明显不足之处。

1.在目标应用层面，学术界对数据中心高通量应用的研究尚不完善，虽然已经有了对高通量的定义和发展现状的研究，但是缺少一个整体的对高通量应用的归纳、分类和分析的工作，因而没有一个具有代表性的数据中心高通量应用的负载集合。

2.在目标系统层面，已有的 Benchmark 都不是面向高通量处理器平台的 Benchmark，不同的目标平台决定了 Benchmark 的具体实现有很大的不同。

## 2.4 本章小结

本章首先对国内外计算机系统 Benchmark 领域的相关研究做了比较系统的总结归纳，分析了每种 Benchmark 各自的特点。Benchmark 主要可以分为传统高性能领域 Benchmark 和大数据相关领域 Benchmark 两大类，每种 Benchmark 都有各自特定的目标系统和目标应用。通过分析可以发现，当前已有的 Benchmark 相关研究都不能满足高通量处理器的测评需求，因而，本章最后提出了面向高通量处理器的 Benchmark 的目标平台和目标系统，对相关 Benchmark 的设计提出了要求。

# 第3章 高通量应用分类与分析

首先，本文对当前数据中心中，使用量较大的几个应用领域做了汇总。汇总结果如表 3.1所示。

表 3.1 数据中心应用总结

| 应用领域 | 解释 |
|---|---|
| Big Data Analytics | 大数据分析和数据挖掘 |
| Web Search | 网页搜索应用 |
| Social Networks | 社交网络应用 |
| E-Commerce | 电子商务应用 |
| Database | 数据库应用 |
| Big Data Multimedia | 大数据多媒体应用 |
| Advertising | 广告应用 |
| Machine Learning | 机器学习 |
| Finance | 金融类应用 |
| Genomics | 基因类应用 |
| Visualization | 可视化技术 |
| Radio Network Controller | 无线网络控制器 |
| Streaming Media | 流媒体 |
| Voice over IP | 网络电话应用 |

在此基础上，本文提出了一种基于高通量需求特点的高通量应用定义和分类模型，对数据中心使用量较大的这些应用领域做高通量特性的判定和分类。然后对各个高通量应用进行分析。

## 3.1 基于高通量需求的高通量应用定义与分类

通过对上表列出的应用进行应用特征分析，总结归纳出高通量需求可以分为三种：

1)单位时间内能够处理尽量大的数据量。

2)单位时间内能够处理尽量多的请求数。

3)能够同时支持尽量多的用户在线实时处理数据。

根据这些不同的高通量需求，可以将高通量应用分为三类：数据处理类、数据服务类、实时交互类。

基于此定义基于高通量需求的高通量应用分类模型如图 3.1所示：

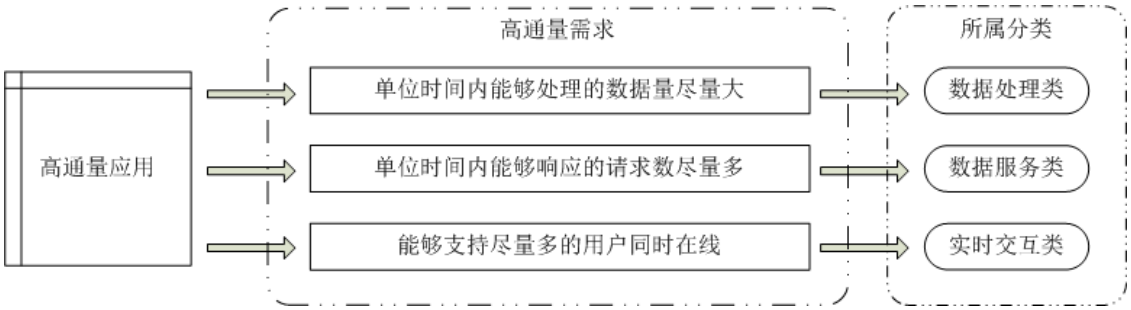



图 3.1 基于高通量需求的分类模型

高通量应用具有不同的类型，不同类型的高通量应用作业的定义和高通量需求关注指标是不同的，因而，本文对不同的高通量应用类型进行了：

1)作业的定义。

2)性能指标的定义。

对三类应用分别做相关定义：

1)数据处理类

作业：对大数据集切分后的一部分数据进行处理的过程。

指标：一定时间内能够处理的数据量，即对数据的吞吐量。

2)数据服务类

作业：对一个请求进行响应的过程。

指标：一定时间内能够及时处理和响应的请求数量，即对请求响应的吞吐量。

3)实时交互类

作业：维持一个用户的在线状态并处理此用户的数据。

指标：能够支持同时处于链接状态并保证服务实时性的用户数，即对用户接入的吞吐量。

本文分析了典型高通量应用的高通量需求，基于上述模型，完成了对典型高通量应用的分类。

### 3.1.1 数据处理类高通量应用

数据处理类的高通量应用一般都是离线工作的，对数据进行存储、计算、统计、汇总等各方面处理，从而得到有效的结果和结论。数据中心中较为典型的数据处理类高通量应用是 Big Data Analytics 和 Machine Learning。下面对这两类应用的高通量需求进行分析。

Big Data Analytics 是指大数据分析，大数据分析在当今互联网时代是应用十分广泛的一类应用，由传统的数据分析方法配合大数据的特点发展而来，尤其是随着商业智能的崛起，大数据分析进入了一个新的快速发展阶段[17]。由于需要处理的数据量呈现指数式增长，当前，大数据处理应用最关注的研究点，是如何能够高效的处理大量数据，如何在单位时间内处理尽量多的数据。大数据处理相应的软件多为分布式系统，如

Hadoop[18]等分布式处理框架，对大数据处理应用软件及相应的硬件系统，最重要的评价指标，是单位时间内所能处理的数据量，这就是大数据处理相关应用的高通量需求。所以，大数据处理属于数据处理类高通量应用。

Machine Learning 的核心思想是让计算机通过学习得到新的知识结构，从而具有人工智能[19]。主要方法是在海量已知数据的基础上，利用概率论和统计学等提供的方法进行归纳和综合，得到相应结论，然后可以将学习的结果用于新的数据，从而得出新的预测和结论。机器学习所关注的性能主要是一定时间内能够对多大的数据量进行学习，即对数据处理量的吞吐率，符合数据处理类高通量需求的特点，所以属于数据处理类高通量应用。

### 3.1.2 数据服务类高通量应用

数据服务类高通量应用主要是指互联网服务类应用，十几年的时间内，互联网服务行业迅猛发展，随着互联网用户数量的不断膨胀，各大互联网网站的访问量呈指数式上涨。

Alexa[20]对各大互联网站的 Pages Views 和 Daily Visitor 两个指标进行过统计，发现当前使用量最大的三类互联网服务网站是 Web Search、Social Network 和 E-commerce。当然，这一点也可以从当前互联网领域各大公司的市值反映出来，美国的 Google、Amazon 和 Facebook，中国的 Baidu、Tencent 和 Alibaba，都是互联网业界的领导企业。

Alexa 对各大网站的 PV 值的统计结果如表 3.2所示。

表 3.2 各大网站日均 PV 值

| 网站类别 | 网站 | 日均 PV（月平均） |
|---|---|---|
| 搜索引擎类 | google | 22929600000 |
| | yahoo | 3201870000 |
| | baidu | 3285600000 |
| 社交网络类 | facebook | 14580000000 |
| | youtube | 5767200000 |
| | twitter | 941880000 |
| 电子商务类 | amazon | 4412460000 |
| | taobao | 1858140000 |
| | ebay | 1276860000 |

可以看到，大型互联网应用网站的日均 PV 值在几十亿的量级，而 Google、Facebook 等互联网巨头可以达到百亿以上的 PV 值。如此巨大的访问密度对数据中心系统的吞吐效率提出了巨大要求。

Web Search 是当今互联网服务中使用量最大的一类服务应用，对用户的搜索请求进行解析并到数据库中查询匹配，从而给出用户匹配的查询结果。巨大的日均访问量对系统的吞吐效率提出巨大要求，这里的吞吐效率主要是指一定时间内能够并发响应的数据

访问请求数量，是 Web Search 应用系统最重要的评价指标，因而，Web Search 应用系统的高通量需求是单位时间内能够对尽量多的用户请求给出响应。

Social Network 是当前互联网应用中使用量第二大的应用。核心工作是解析用户的页面请求，分析社交图谱，为用户提供所请求的页面以及进行相关内容的推送。同样，Social Network 也面临着大量用户请求的问题，相关的软硬件系统也具有高通量需求，要求能够在单位时间内处理和响应尽量多的用户请求，属于数据服务类高通量应用需求。

E-commerce 是另外一类用户量巨大的互联网服务应用。核心工作主要有两方面，一方面是根据用户查询请求，为用户展示响应的产品页面，另一方面是订单相关的处理，订单的创建、查询、修改、删除等。无论是两方面工作中的哪个，都是要响应用户的操作请求。同样，在用户量巨大的情况下，E-commerce 的软硬件系统同样具有高通量的需求，即能够在单位时间内尽量多的处理请求，具有数据服务类高通量需求的特点。

### 3.1.3 实时交互类高通量应用

实时交互类高通量应用通常是指需要对某一用户持续提供服务的应用，此类应用一般需要在服务开始之前建立链接或者进行用户注册，一般是对具有实时性和持续性的一类应用的实现方式。如无线通信中的无线网络控制器，网络流媒体等应用。

实时交互类高通量应用与数据服务类高通量应用的不同之处在于，对于每个用户的服务具有一定的持续性需求，因而，此类应用不是针对单个请求返回单个响应，而是需要针对一个用户维护一个连接状态，并在一段时间内持续提供实时交互服务。此类应用一般会有相关的网络协议来支持相应的在线实时处理。

无线网络控制器（RNC）是 WCDMA 通信系统中陆地无线接入网部分的核心[21]，是运营商数据中心的主要工作之一。RNC 按照功能可以分为两方面，用户面和控制面。控制面主要完成用户连接状态的建立、维护和释放，为用户面的工作提供通路；用户面是负责协议的数据处理，是真正具有高通量需求的部分，用户面要求系统能够支持尽量多的用户同时接入并保持一定时间的在线状态，以保证数据的实时处理。因而，RNC 用户面应该属于高通量应用中的实时交互类应用。

网络流媒体（Streaming Media）是典型的实时交互类高通量应用。随着互联网技术的发展，传统的多媒体服务越来越多的使用互联网技术作为载体。而以在线视频服务为主要代表的网络多媒体服务有较高的实时性需求，在这种背景之下，网络流媒体技术很快发展起来。当今业界的网络流媒体服务多采用实时流式传输（Real time streaming），这种传输方式使用如 RTSP 这种实时协议。随着当前多媒体服务用户数量指数式增加，流媒体应用的软硬件系统需要同时满足尽量多的用户数量的持续性在线和实时服务，因而，其高通量需求属于实时交互类高通量应用的高通量需求。

## 3.2 核心 Workload 提取与程序特征总结

核心 Workload 是指能够反映出整个应用程序特点和硬件需求的核心算法或者核心

执行模块。本文对典型高通量应用的核心 Workload 进行了提取工作。

### 3.2.1 数据处理类高通量应用核心 Workload 提取

对数据处理类高通量应用中的两个典型应用大数据分析和机器学习做了核心 Workload 的提取工作。

#### 3.2.1.1 大数据分析（Big Data Analytics）

大数据分析是一个广义的概念，由很多独立的数据分析算法组成，这些算法可能会应用到其他很多领域。当前业界多将这些大数据分析算法运行于 MapReduce 模型之上[22]，以实现分布式存储和分布式处理，从而提高数据处理的速度和吞吐效率。此类应用所处理的数据具有的特点是数据量大，数据之间耦合性低，可以将数据分块，多个节点并行处理。

通过调研和分析，大数据分析应用中所使用的算法主要可以分为基本操作、分类、聚类、分段和回归分析等几个方面[23]。如表 3.3所示，总结出了大数据分析中常用的核心 Workload。

表 3.3 Big Data Analytics 核心 workload

| | |
|---|---|
| Basic operation | sort |
| | grep |
| | wordcount |
| Classification | Naïve Bayes |
| | SVM |
| Cluster | K-means |
| | Fuzzy k-means |
| Segmentation | HMM |
| regression analysis | regression analysis |

#### 3.2.1.2 机器学习（Machine Learning）

机器学习的主要工作是在海量数据的基础上，利用概率论和统计学等提供的方法进行归纳和综合，使计算机具有智能。从 1996 年有了对机器学习的定义以来，将近二十年的时间里，机器学习的方法论有了成熟的发展。而随着近几年大数据领域的发展，使得机器学习与大数据的结合产生了新的发展需求。

本文通过文献调研，对机器学习常用的方法进行总结归纳，得到机器学习领域核心 Workload 集合，机器学习中的核心 Workload 就是指一些典型的学习算法和模型，如表 3.4 所示。

表 3.4 Machine Learning 核心 workload

| | |
|---|---|
| 半监督学习 | 基于混合产生式模型的学习算法 |
| | 基于低密度划分的学习算法 |
| | 基于图的学习算法 |
| | 基于不一致性的学习算法 |
| | 协同训练学习算法 |
| 集成学习 | 顺序的集成学习算法 |
| | 并行的集成学习算法 |
| 概率图模型 | 贝叶斯网络模型 |
| | 马尔科夫网络模型 |
| | 隐马尔可夫网络模型 |
| 迁移学习 | 归纳迁移学习 |
| | 直推迁移学习 |
| | 无监督迁移学习 |

### 3.2.2 数据处理类高通量应用程序特征总结

通过对数据处理类高通量应用的分析，可以得到此类高通量应用的如下特性：

1) 作业具有并发处理特性，支持高并发的系统可以提高处理能力。数据处理类高通量应用中的大部分 Workload 可以通过对所处理的大数据进行切分的方式来实现并行处理，每一部分数据用一个作业来处理，这样可以多节点并发处理。此过程可以使用 MapReduce 等并行编程模型来辅助完成。

2) 作业之间耦合性低，作业节点之间通信量小。数据处理类应用中的数据都是扁平化的，数据块之间逻辑上的关系较小，即数据之间耦合性低，所以，作业处理节点之间通信量较小。

3) 访存需求比较大，较小的浮点计算量。数据处理类高通量应用多使用统计归纳和数据挖掘等方面的技术，由于需要对大量数据进行操作，而对每一数据的操作相对简单，因而访存需求相对较大。而由于很少需要进行复杂的计算，没有科学计算的需求，所以计算量比较小，几乎没有浮点计算。

4) 处理的数据多为文本数据，以字符或字符串为主，访存位宽偏小。数据中心的数据处理类高通量应用多用于文本类数据的分析，以字符或字符串为主，与传统高性能应用以 32bit 或 64bit 访存位宽为主相比，此类应用访存位宽偏小，8bit 位宽的访存量占到一定的比例。

### 3.2.3 数据服务类高通量应用核心 Workload 提取

数据服务类应用主要是指互联网服务应用，[20]基于 pages views 和 daily visitor 两个指标，统计出 Web Search、Social Network 和 E-commerce 是三类使用量最大的互联网服务。本文对三类应用典型的软件系统结构进行分析，总结出了各个应用的核心 Workload。

#### 3.2.3.1 网页搜索（Web Search）

搜索引擎是当今互联网中使用量最大的一类服务应用，巨大的日均访问量对系统的吞吐效率提出巨大要求，这里的吞吐效率主要是指一定时间内能够并发响应的数据访问请求数量，符合数据服务类高通量需求。

当今主流的搜索引擎的架构大同小异，主要包含网络爬虫、建立索引库和响应用户请求几个大的功能模块，而每个功能模块中包含一些核心的模块或算法，即核心 Workload，如表 3.5所示。其中，响应用户请求部分是典型的数据服务类高通量应用的代表。

表 3.5 Web Search 核心 workload

| | |
|---|---|
| 网络爬虫 | 广度优先遍历算法 BFS |
| | 深度优先遍历算法 DFS |
| | 调度系统（管理优先级排序） |
| | Hash Table |
| 建立索引库 | Html parser |
| | Directory iterative |
| | Index |
| | PageRank |
| | TF-IDF |
| 响应用户请求 | Query parser |
| | Database query |
| | Term weight |
| | Document weight |
| | MSet |

#### 3.2.3.2 社交网络（Social Network）

社交网络是使用量第二大的互联网服务，对常见的社交网络进行分析，主要性能指标是对用户请求页面的响应能力，符合数据服务类高通量需求。

通过分析应用，可以得出社交网络系统中核心的 Workload：

表 3.6 Social Network 核心 workload

| | |
|---|---|
| 社交网络影响力最大化[24] | Diffusion Degree 算法 |
| | Greedy 算法 |

| | |
|---|---|
| 推荐算法 | 协同过滤推荐 |
| | 基于内容推荐 |
| 数据抽样方法 | 广度优先抽样法 |
| | 点边抽样法 |
| | 用户均匀抽样法 |
| | 同伴推动抽样法 |
| | 随机行走抽样法 |
| 拓扑分析 | 计算平均路径长度 |
| | 计算聚集系数 |
| 数据分析 | 基于链接的结构分析 |
| | 基于内容的分析 |

#### 3.2.3.3 电子商务（E-commerce）

电子商务是使用量第三大的互联网服务，主要性能指标是支持大量页面请求、订单处理请求等。典型的电子商务系统通常会包含搜索系统、订单系统、数据库系统和推荐系统几大功能模块，而每个模块有包含多个小的核心模块或算法，即核心 Workload，如表 3.7所示。其中，搜索系统、订单系统都有典型的数据服务类高通量应用的特点。

表 3.7 E-commerce 核心 workload

| | |
|---|---|
| 搜索系统 | 根据商品不同的关键字建立索引(Index) |
| | 响应用户查询(Query) |
| 订单系统 | 订单建立(Create) |
| | 订单查询(Select) |
| | 订单修改(Update) |
| | 状态机(Finite State Machine) |
| 数据库系统 | 商品管理系统 |
| | 会员管理系统 |
| | 订单管理系统 |
| 推荐系统 | 协同过滤算法 |
| | 基于内容的推荐算法 |
| | 基于聚类算法的推荐 |
| | 基于产品到产品的推荐 |

### 3.2.4 数据服务类高通量应用程序特征总结

从应用特点层面对数据服务类高通量应用进行分析，可以得到其具有的程序特征：

1) 作业具有天然的并发性，支持高并发的系统可以提高处理能力。数据服务类高通

量应用的一个作业就是对一个用户请求的处理和响应，所以具有天然的并发性，可以并发处理。

2) 作业具有独立性，作业节点之间通信量小。用户的请求都是独立的，不同的作业之间不存在业务逻辑，不同的请求之间不存在交互，所以具有良好的独立性。

3) 较高的访存计算比，较小的浮点计算量。数据服务类高通量应用对用户请求的处理工作主要是根据不同的用户请求，查询相应的数据库或获取远程服务器资源，将结果返回给用户。其中，后台的处理过程可能会涉及到一些数据处理的工作。可以看出，此类应用对访存通路需求比较高，而很少有计算方面的需求，特别是浮点计算几乎没有。

4) 处理的数据多为文本数据，以字符或字符串为主，访存位宽偏小。用户请求多以文本的方式进行，以字符或字符串处理为主，与传统应用常见的 32bit 或 64bit 访存位宽相比，较小位宽的访存量占到不可忽略的比例。

### 3.2.5 实时交互类高通量应用核心 Workload 提取

与数据服务类应用不同，此类应用不是针对单个请求返回单个响应，而是需要针对一个用户维护一个连接状态，保持一段时间的实时交互服务。

此类应用一般会有相关的网络协议来支持相应的在线实时交互，因而此类应用的 Workload 是指关键协议和协议中的数据处理步骤和算法，下面对此类应用进行关键 Workload 抽取。

#### 3.2.5.1 无线网络控制器（Radio Network Controller）

无线网络控制器（RNC）是 WCDMA 通信系统中陆地无线接入网部分的核心，是运营商数据中心的主要工作之一。RNC 按照功能可以分为两方面，用户面和控制面。控制面主要完成用户连接状态的建立、维护和释放，为用户面的工作提供通路；用户面是负责协议的数据处理，是真正具有高通量需求的部分，用户面要求系统能够支持尽量多的用户同时接入并保持一定时间的在线状态，以保证数据的实时处理。因而，RNC 用户面应该属于高通量应用中的实时交互类应用。

RNC 用户面主要对数据处理协议的实现，因而是一个完整的实体，其中主要的部分是 MACD 层和 RLC 层[25]的工作，主要涉及到的 Workload 如表 3.8所示。

表 3.8 Radio Network Controller 核心 workload

| | |
|---|---|
| MACD 层 | SDU receive |
| | Schedule |
| RLC 层 | Entity query |
| | Segment |

#### 3.2.5.2 流媒体（Streaming Media）

流媒体[26]中所涉及到的 Workload 主要是指相关的协议，这些协议在整个流媒体系统中占用绝大部分资源和时间，通过文献查阅进行分析和汇总，得到流媒体中的核心

Workload 如表 3.9所示。

表 3.9 Streaming Media 核心 workload

| | |
|---|---|
| 视频编解码算法 | MPEG |
| | H264 |
| 流媒体传输协议 | 资源预留协议 RSVP |
| | 实时传输协议 RTP |
| | 实时传输控制协议 RTCP |
| | 实时流协议 RTSP |

### 3.2.6 实时交互类高通量应用程序特征总结

通过分析可以总结，实时交互类高通量具有如下程序特征：

1) 作业具有并发性。这里的一个作业是指对一个用户维持在线状态并提供服务。这种作业具有天然的并发性。

2) 作业之间低耦合。不同的用户提供的实时服务是相对独立的，因而作业之间耦合性低，交互通信较少。

3) 高访存计算比。经过分析可以发现，实时交互类高通量应用对用户进行服务的重点是数据传输，基于不同的协议保证传输质量，因而，数据的访存较多而计算较少，需要系统提供较大的访存带宽或者较多的访存通路。

4) 数据量大与不规则的数据访问导致空间局部性较差，Cache 失效率高。由于需要同时服务于不同的用户，所以访存具有不规则性，而且当大量用户接入时，数据量较大，导致了较高的 Cache 失效率。

5) 访存位宽多样化，且偏小。此类应用的用户数据以协议数据包的方式存在，而数据包中的数据多是以字节为单位来组织的，因而，与传统的应用常见的 32bit 或 64bit 访存位宽相比，此类应用访存位宽呈现多样化，且整体来说偏小，8bit 和 16bit 位宽占到不可忽略的比例。

## 3.3 本章小结

本章首先根据高通量应用的定义，从众多的数据中心常见应用中选出了典型的高通量应用，然后提出了一种基于高通量需求的高通量应用分类模型，基于此模型，对典型的高通量应用进行了分类。

然后，对典型的高通量应用进行了核心 Workload 的提取，核心 Workload 是指应用中的关键算法和功能模块。对核心 Workload 进行分析，总结出各个典型高通量应用该有的程序特征。高通量应用分类和 Workload 提取与划分结果汇总如表 3.10所示。

表 3.10 典型高通量应用 Workload 提取与划分结果汇总

| 类型 | 应用 | Workload 提取与划分 | |
|---|---|---|---|
| 数 据 | Big Data | Basic operation | sort |

| | | | |
|---|---|---|---|
| 处理类 | Analytics | | grep |
| | | | wordcount |
| | | Classification | Naïve Bayes |
| | | | SVM |
| | | Cluster | K-means |
| | | | Fuzzy k-means |
| | | Segmentation | HMM |
| | | regression analysis | regression analysis |
| | Machine Learning | 半监督学习 | 基于混合产生式模型 |
| | | | 基于低密度划分 |
| | | | 基于图的 |
| | | | 基于不一致性的 |
| | | | 协同训练算法 |
| | | 集成学习 | 顺序的集成学习方法（Boosting） |
| | | | 并行的集成学习方法（Bagging） |
| | | 概率图模型 | 贝叶斯网络 |
| | | | 马尔科夫网络 |
| | | | 隐马尔可夫网络模型 |
| | | 迁移学习 | 归纳迁移学习 |
| | | | 直推迁移学习 |
| | | | 无监督迁移学习 |
| 数据服务类 | Web Search | 网络爬虫 | 广度优先遍历算法 BFS |
| | | | 深度优先遍历算法 DFS |
| | | | 调度系统（管理优先级排序） |
| | | | Hash Table |
| | | 建立索引库 | Html parser |
| | | | Directory iterative |
| | | | Index |
| | | | PageRank |
| | | | TF-IDF |
| | | 响应用户请求 | Query parser |
| | | | Database query |
| | | | Term weight |
| | | | Document weight |
| | | | MSet |
| | Social Network | 社交网络影响力最大化 | Diffusion Degree 算法 |
| | | | Greedy 算法 |

| | | | |
|---|---|---|---|
| | | 推荐算法 | 协同过滤推荐 |
| | | | 基于内容推荐 |
| | | 数据抽样方法 | 广度优先抽样法 |
| | | | 点边抽样法 |
| | | | 用户均匀抽样法 |
| | | | 同伴推动抽样法 |
| | | | 随机行走抽样法 |
| | | 拓扑分析 | 计算平均路径长度 |
| | | | 计算聚集系数 |
| | | 数据分析 | 基于链接的结构分析 |
| | | | 基于内容的分析 |
| | E-commerce | 搜索系统 | 根据不同的关键字建立索引(Index) |
| | | | 响应用户查询(Query) |
| | | 订单系统 | 订单建立(Create) |
| | | | 订单查询(Select) |
| | | | 订单修改(Update) |
| | | | 状态机(Finite State Machine) |
| | | 数据库系统 | 商品管理系统 |
| | | | 会员管理系统 |
| | | | 订单管理系统 |
| | | 推荐系统 | 协同过滤算法 |
| | | | 基于内容的推荐算法 |
| | | | 基于聚类算法的推荐 |
| | | | 基于产品到产品的推荐 |
| 实时交互类 | Radio Network Controller | MACD 层 | SDU receive |
| | | | Schedule |
| | | RLC 层 | Entity query |
| | | | Segment |
| | Streaming Media | 视频编解码算法 | MPEG |
| | | | H264 |
| | | 流媒体传输协议 | 资源预留协议 RSVP |
| | | | 实时传输协议 RTP |
| | | | 实时传输控制协议 RTCP |
| | | | 实时流协议 RTSP |

# 第4章 面向高通量处理器的 Benchmark 设计

在对数据中心高通量应用领域进行了全面分析的基础上，可以开始设计面向高通量处理器的 Benchmark。

## 4.1 应用与 Workload 的选取

一套完整的 Benchmark 集需要对第三章所分析的所有高通量应用针对目标平台实现相应的 Benchmark，而考虑到工程量的问题，本文不会对所有高通量应用进行实现，只能选取部分具有一定代表性的应用进行实现。

**本文所实现的 Benchmark 的选取主要基于以下原则：**

1) 从三类不同的高通量应用中选取代表应用。根据分析可知，每一类高通量应用在应用特征和性能指标方面都有很大的相似性，因而，可以从每一类高通量应用中选取一个应用领域，来代表此类应用。

2) 选取的应用是使用量较大的。要保证从每一大类高通量应用中选取出来的应用具有代表性，重要的一点是此应用在数据中心中的使用量较大，才能真实反映应用对平台的需求。

3) 结合作者在实验室项目中已经做的工作。结合本人之前在实验室项目中已有的科研基础，可以更高效、高质量的完成相应的 Benchmark。

基于以上三点，本文选取了如下三个应用作为高通量应用的 Benchmark。

数据处理类选取了 Big Data Analytics 应用，因为 Big Data Analytics 中的算法都是是大数据处理中的基础算法，使用量非常广泛。而这些算法之间是相对独立的，各个算法所反映出的应用特点、程序特性和对硬件系统的需求又很相似，因而可以选取其中最为常见的几个算法来实现，而不需要实现所有算法。本文选取了 Wordcount、Terasort、Kmeans 和 Grep 四个算法来实现 Benchmark。

数据服务类选取了 Web Search 应用中的响应用户请求部分来实现 Benchmark，因为这一部分的工作特点最符合数据服务类高通量应用的特点。其中几个 Workload 构成了一个有机整体，需要同时运行才能实现一个完整的功能，因此，最终实现的 Benchmark 是这几个 Workload 构成的一个完整测试程序。

实时交互类选取了 Radio Network Controller 中的用户面数据处理流程部分，这一部分的工作具有典型的实时交互类高通量应用的特点，而且作为无线通信中的核心部分，使用量是相当大的，具有很大的代表性。此 Benchmark 主要是根据协议的定义，模拟数据包处理流程，各个 Workload 之间是一个有机的整体，最终构成一个测试程序作为 Benchmark。

表 4.1对选取的三个应用领域进行了总结。

表 4.1 高通量 Benchmark 的组成

| 应用 | 核心 Workloads | 类型 | 高通量需求指标 |
|---|---|---|---|
| Big Data Analytics | Wordcount | 数据处理类 | 单位时间内能够处理的数据量 |
| | Grep | | |
| | Terasort | | |
| | K-means | | |
| Web Search | Query Parse | 数据服务类 | 单位时间内能够响应的请求数 |
| | Database Query | | |
| | Term Weight | | |
| | Document Weight | | |
| | MSet | | |
| Radio Network Controller | SDU Receive | 实时交互类 | 能够支持的同时在线的用户数 |
| | MACD Schedule | | |
| | Entity Query | | |
| | Segment | | |

## 4.2 基于线程的作业处理节点并行化思想

经过第三章的分析可知，高通量应用的典型特点是含有大量小规模作业，而作业之间耦合性低，因而，要提高处理器对于高通量应用的吞吐效率，必须支持多处理节点并行处理应用中的作业。如 Hadoop 等系统级的软件栈，提高应用程序吞吐效率的重要手段就是采用分布式系统，并行化多节点处理作业。

高通量处理器一般是多核或者众核结构，在处理器级别能够并行工作的载体是线程，因此，要实现用于高通量处理器的高通量应用的 Benchmark，需要将不同的作业处理节点用线程来实现，以达到在处理器级别多节点并行的效果。

线程在高通量处理器上的调度，由对应于高通量处理器结构的 Runtime 来负责，而在应用层面（Benchmark），只需要保证不同作业的节点是线程即可。

因而，本文提出了一种适用于面向高通量处理器的 Benchmark 的并行化思想：基于线程的作业处理节点并行化思想。

此思想的关键点是处理作业的节点是基于线程实现的。针对三种不同的高通量应用，有不同的并行化模型。

## 4.3 数据处理类高通量应用 Benchmark 实现

首先，本小节根据基于线程的作业处理节点并行化思想，以及数据处理类高通量应用的工作特点，提出了数据处理类高通量应用的 Benchmark 的基于线程的作业处理节点并行化模型。基于此模型，实现了数据处理类 Benchmark 包含的四个测试程序。

### 4.3.1 数据处理类高通量应用多节点并行化模型

当前的数据中心数据处理类高通量应用多采用分布式处理的方式，将一个算法需要处理的大量数据分块，每块数据的处理作为一个作业，交给不同的处理节点进行处理。较常用的一种框架是 MapReduce 框架，大部分数据处理类应用都可以用这个框架来实现。MapReduce 框架中的作业就是数据分块之后形成的 Map 任务和 Reduce 任务。

数据处理类高通量应用的基于线程的作业处理节点并行化模型就是实现一个基于线程的 MapReduce 编程框架，一个处理节点就是一个线程，每个 Map 作业和 Reduce 作业的处理交由线程来完成。

模型结构如图 4.1所示

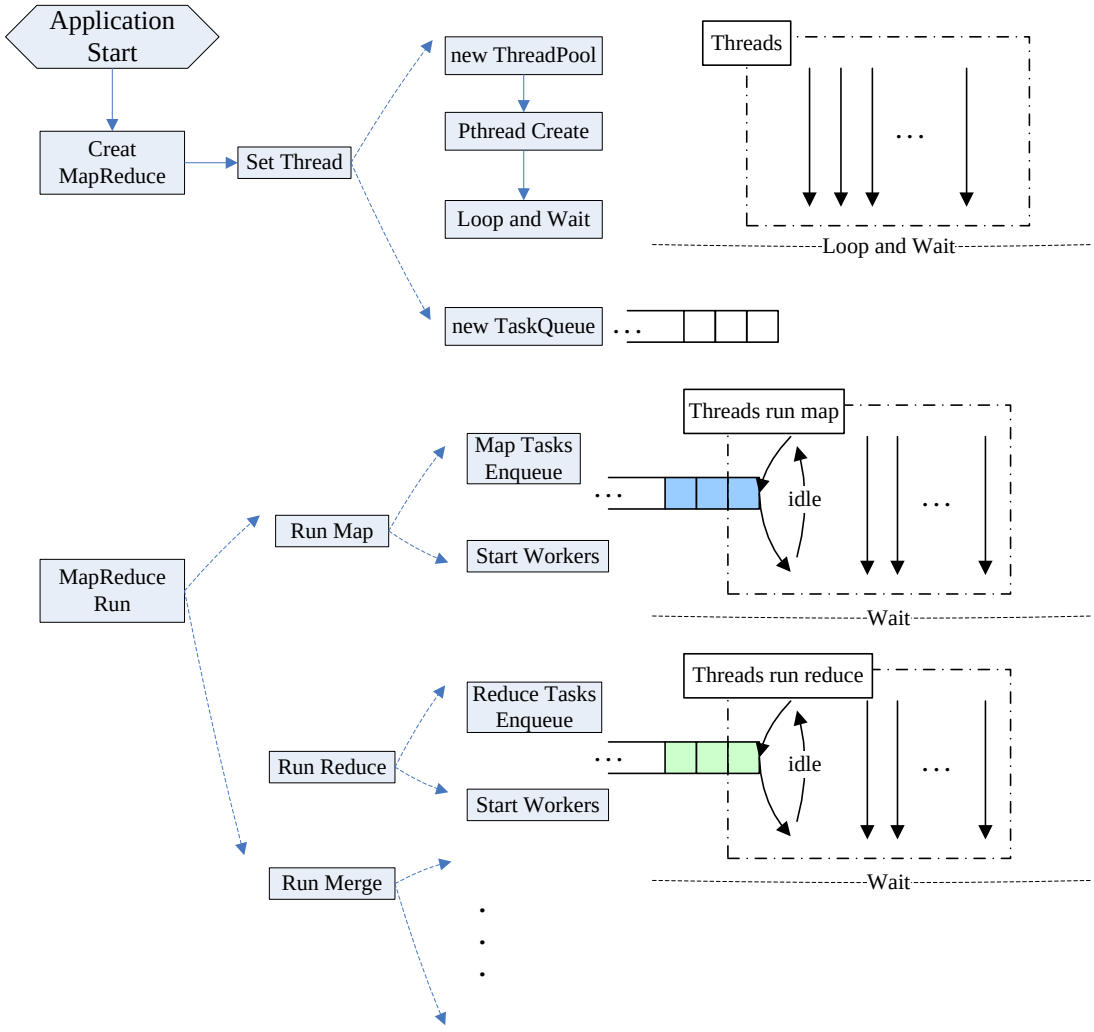



图 4.1 数据处理类编程模型

模型框架是基于 Pthread 来实现的。当测试程序开始执行时，会创建一个线程池，根据用户输入的线程数决定线程中有多少个线程。线程创建时，会运行一个线程体，被一个信号量阻塞，不执行具体的操作。当 Map 阶段的数据和运行的函数准备好，放入 Task Queue 之后，会打开信号量。这时线程就会从 Task Queue 中取任务并执行，要执行

的函数是用函数指针传递的。运行完一个任务后，线程会再从 Queue 中取一个任务来执行。直到 Task Queue 中的任务被执行完毕，线程池中的线程再次进入 Wait 状态，等待下一阶段(Reduce 或 Merge)的执行。

这种模型的好处在于，但每个线程执行的工作量是大致相同的，有利于均衡。另外，将线程处理的数据记录在 Task 中，并将线程执行的任务用函数指针表示增加了灵活性，线程可以在创建之后完成 Map、Reduce 和 Merge 多种任务直到线程被停止。

基于此模型，本文对大数据处理中的四个典型算法 Wordcount、Terasort、Kmeans 和 Grep 进行了实现，作为数据处理类高通量应用的 Benchmark。

### 4.3.2 Big Data Analytics 核心 workload

Big Data Analytics 应用的实现主要选取了四个最为典型的算法，每个算法是独立的，完成不同的大数据分析功能。

#### 4.3.2.1 Wordcount

Wordcount 算法是经常用于大数据文本处理的一个算法，统计文本中各个单词出现的次数。一个单线程的 Wordcount 程序大致可以是如下伪代码表示：

```
define wordCount as Multiset;
for each document in documentSet
{
    T = tokenize(document);
    for each token in T
    {
        wordCount[token]++;
    }
}
display(wordCount);
```

按照数据本文提出的并行化模型，实现 Wordcount 算法需要完成 split、map 和 reduce 几个功能函数。

Split 函数用于切分输入数据，将比较大的数据切分成小块数据，分发给不同的 map 线程。Map 函数是 MapReduce 中的多线程并行处理部分，多个 map 函数由不同的线程来并行执行，map 函数对各自对应的数据块进行词频统计。Reduce 函数用于合并各个线程得到的词频统计的结果。流程图如图 4.2所示：

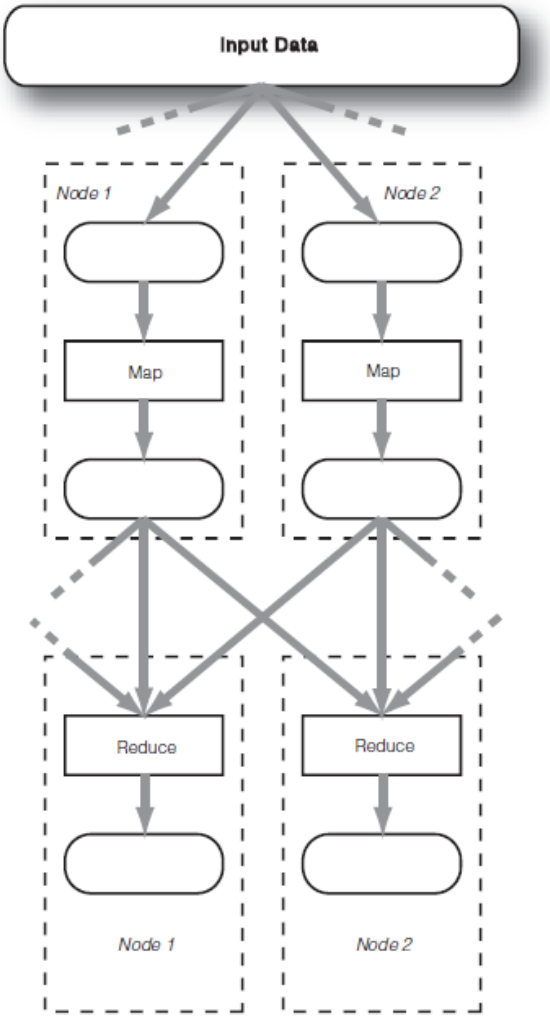



图 4.2 Wordcount 在 MapReduce 中的工作流程示意图

#### 4.3.2.2 Terasort

Terasort 是一个大规模键值对数据排序的算法，是计算机领域作为系统计算性能测评的标准算法之一。Terasort 算法的主要思想包括下面三个步骤[27]。

1) 数据采样。从需要排序的所有键值对数据中选取一部分数据，采用传统的排序方式进行排序，然后根据需要的分割点个数，从排序后的数据中等步长的距离选择一些数据点作为分割点，组织到一颗 Trie 树中，在 Map 阶段，此 Trie 树中的数据作为将每个数据分到不同 Reduce 任务中的依据。

2) Map 阶段。Map 阶段的输入数据是对源数据进行简单切分后的一块数据，对其中的每一个数据，Map 阶段的工作是根据 Trie 树中的分割点信息，决定此数据应该被分配到哪个 Reduce 任务。

3) Reduce 阶段。每个 Reduce 任务将从 Map 中得到的数据进行内部排序，然后按照 Reduce 编号，顺序输出每个 Reduce 任务内部排序后的结果，就可以得到最终的排序结果。

本文所实现的 Benchmark 中，Terasort 测试程序的实现要点有：

1) 数据采样与分割点的选择

要将所有数据分成 reducenum 个 partition，所以分割点 partition_point 的个数是 reducenum – 1；其中 reducenum 是 reduce 任务的个数；

采样数据的个数 sam_num 如下确定：如果输入数据个数小于 1000，则 sam_num= 输入数据个数；如果输入数据个数大于 1000，则 sam_num = 1000；采样就是从输入文件的第一个数据开始，顺序读入 sam_num 个数据；

将得到的采样数据进行全排序；

计算得到一个步长 step = sam_num / reducenum；

然后按照 step 从全排序后的采样数据中再进行采样得到 partition_point[reducenum-1]，这就是各个 reduce 对应的 partition 之间的分割点。

举例说明：

比如采样数据为 b，abc，abd，bcd，abcd，efg，hii，afd，rrr，mnk；

经排序后，得到：abc，abcd，abd，afd，b，bcd，efg，hii，mnk，rrr；

如果 reduce Task 个数为 4，则分割点为：abd，bcd，mnk。

2) 建立 trie 树：

设计采用两层内部结点的 trie 树，即利用 key 值的前两个字节进行组织。trie 树的作用是加快 partition_id 的计算速度，有了 trie 树，对于一个需要确定 partition_id 的 key 值，可以不用从第一个 partition_point 进行比较，而是根据 key 值的前两个字节，从中间某个部分开始与 partition_point 进行比较大小。

如图 4.3所示，是 trie 树结构。其中，四个 partition_point 分成五个 partition 对应五个 Reduce Task。

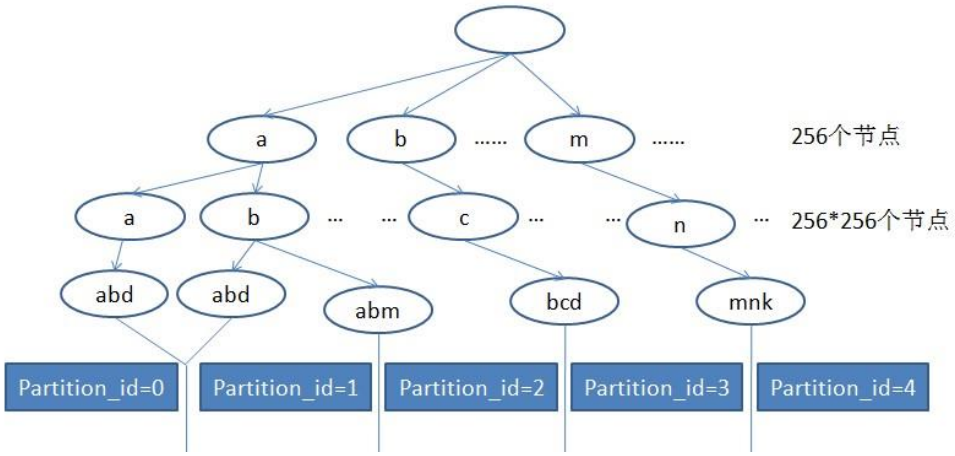



图 4.3 trie 树结构

3) Map Task 对数据记录做标记

Split 阶段将输入数据分成若干 Chunk，每个 Chunk 对应一个 Map Task；

Map Task 从各自对应的 Chunk 中一条一条读取数据，并通过 Trie 树查找每条记录所对应的 Reduce Task 编号，即 Get_Partition_Id。比如：abg 对应第二个 Reduce Task，mnz 对应第五个 Reduce Task。

Get_Partition_Id 的本质是 Key 值与 Trie 树的叶节点进行比较，得到 Partition_Id（即此数据要被分配到的 Reduce 任务号）；

具体过程是：通过 Key 值的前两个字节将比较范围缩小到所有叶节点（所有叶节点就是 Partition_Point[]，已是完全排序的）中的一部分叶节点，然后再将 Key 值与这些叶节点按顺序进行比较，确定出 Partition_Id。

4) Reduce Task 进行局部排序

每个 Reduce Task 进行局部排序，局部排序使用归并排序，最后依次输出结果即可得到整个 Terasort 算法的最终排序结果。

#### 4.3.2.3 Kmeans

Kmeans[28]是用于聚类分析的经典算法。Kmeans 算法是将多维空间中的大量数据点，按照距离分到不同的集合中。

Kmeans 算法的工作是将空间中的 M 个数据点，按照距离分到 K 个聚类之中。Kmeans 的算法流程是：

1) 随机选取 K 个种子点，作为 K 个聚类的初始中心点。

2) 对 M 个数据点中的每一个，计算到 K 个中心点的距离，然后选取距离最小的，作为此数据点在本次循环所属的聚类。

3) 对每一个聚类，计算新的所有数据的中心点。

4) 循环执行 2、3 两步，直到在执行第 2 步时，没有数据点所属的聚类发生变化，结束循环。

5) 得到了所有 M 个数据点所属的聚类和各个聚类的中心点。

本文基于上述模型，实现 Kmeans 算法，作为 Benchmark 的一个测试程序。可配置参数有四个：

-c：聚类数量（C）

-d：数据维度（D）

-p：数据量（P）

-s：数据最大值（S）

-t：线程数（T）

实现中的要点有：

1) 随机生成 D 维的数据点 P 个和 D 维的聚类中心点 C 个。对于每个聚类，定义一个 D 维的 Sum 用于记录一次循环中，属于此聚类的各个数据点各个维度之和，定义一个 Count 用于计数属于此聚类的点的个数。Sum 和 Count 用于在每次循环之后，计算新的聚类中心点。

2) 每个数据点作为一个 Map Job，每个 Map Job 会计算此数据点到所有聚类中心点的距离，然后去距离最小的，作为此数据点此次循环所属的聚类。将此数据点的各个维度的值与所属聚类的 Sum 各个维度的值求和，得到新的 Sum，Count 值加 1。

3) 所有数据点计算完成后，执行 Reduce 阶段，Reduce 阶段对每个聚类计算新的中心点，计算公式为 Sum/Count。

4) 循环执行 2、3 两个过程，直到某次执行 2 的时候，所有数据点新计算的所属聚类与上次循环计算的所属聚类相同，结束循环。得到所有聚类中心点和所有数据点所属的聚类。

#### 4.3.2.4 Grep

Grep 算法是字符串匹配算法，在大数据分析中是最基本的算法之一。算法内容是在大文本中匹配模式字符串。

基于本文模型实现 Grep 比较简单，主要流程有：

1) 自定义 split()函数对较大的源数据文件进行切分，每 1M 数据作为一个数据块，每一数据块对应一个 Map Job。

2) Map 任务在对应的数据块中查找模式字符串。

3) Reduce 阶段只是对 Map 阶段匹配的结果做简单汇总。

### 4.3.3 性能指标统计模块

根据前文的分析，数据分析类高通量应用的性能指标是单位时间内能够处理的数据量，也就是对处理数据的吞吐效率。因此，本文对所实现的 Benchmark 设计了性能统计模块，用于统计处理器运行 Benchmark 时的吞吐效率，从而评价处理器对数据处理类高通量应用的 Benchmark 时的性能。

如图 4.4所示，是 Wordcount 测试程序执行结束后，输出的一个性能统计结果。

```
                INFORMATION
Total Data Size is    :  104306760 B
Total Run Time is     :  643363133 ns
Average Throughput is:  162127350 B/s
```

图 4.4 数据处理类 Benchmark 统计信息

统计结果中各指标的含义是：

Total Data Size：本次执行本测试程序所处理的总的数据量；

Total Run Time：本次执行本测试程序总执行时间；

Average Throughput：平均单位时间内所处理的数据量，即吞吐效率。

通过以上指标，可以评价不同处理器对数据处理类高通量应用的 Benchmark 的处理能力。

## 4.4 数据服务类高通量应用 Benchmark 实现

数据服务类高通量应用的Benchmark选取的是搜索引擎中响应用户查询请求部分的核心 workload 来进行实现的。

### 4.4.1 数据服务类高通量应用多节点并行化模型

数据服务类应用主要是指 C/S 或 B/S 结构中的服务器端程序，数据服务类高通量应用的基于线程的作业处理节点并行化模型结构如图 4.5所示：

1) 一个 Listen Thread 用于监听请求，将收到的请求发送到不同线程对应的 Job Queue 中。

2) 每个线程对应一个 Job Queue，用于缓存 Listen Thread 接收到的用户请求。

3) 线程池，大量线程作为作业处理节点，循环从对应的 Job Queue 中获取 Job，然后做相应处理，可能需要查询数据库，例如搜索引擎响应用户查询就要使用数据库查询操作。

4) 将处理结果返回给用户，作为对用户相关操作的响应。

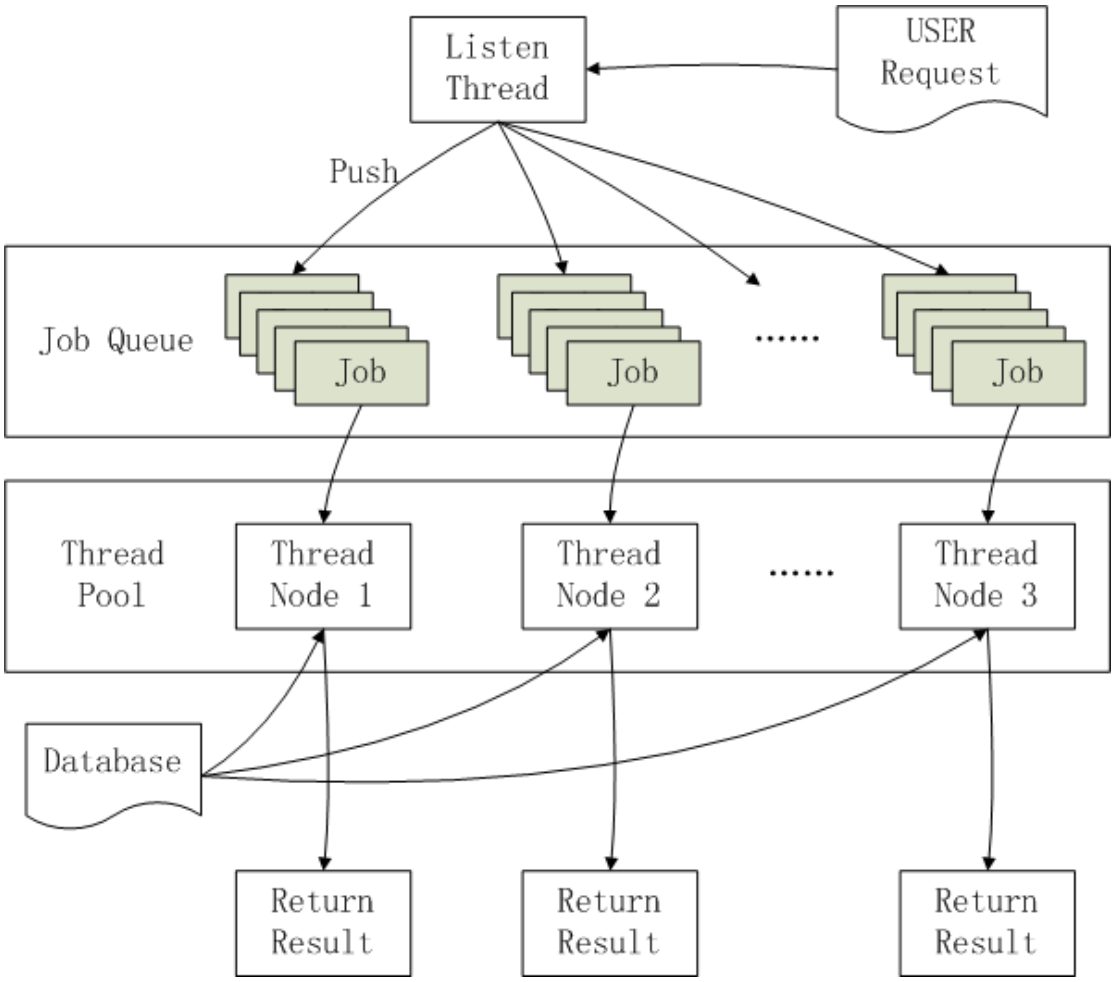



图 4.5 数据服务类编程模型

从模型结构可以看出，数据服务类高通量应用的并行化模型的要点是将用户请求分

发到不同的 Job Queue，每个 Job Queue 对应一个处理节点，处理节点由一个线程来实现。处理节点轮询对应的 Job Queue，对用户请求做相应的处理。

### 4.4.2 无锁 Job Queue 的实现

一般情况下，Job Queue 数量与 Task Thread 数量是相同的，一一对应的。在数据服务类的这种并行化模型中，Listen Thread 只负责向 Job Queue 中写数据，而 Task Thread 只会从 Job Queue 中读取数据。这种情况下，可以设计一种无锁的队列管理机制，使用这种机制可以避免在对 Job Queue 进行读写时加锁，从而加快请求处理过程。由于各个线程需要频繁的对 Job Queue 进行读写，所以，这种无锁的缓存队列设计对提高性能具有很大的效果。无锁 Job Queue 实现的几个要点：

1) 队列类定义

对列类定义代码下所示：

```
class TaskQueue
{
    uint32_t queue_elem_num;
    uint32_t mask;
    volatile uint32_t r;
    volatile uint32_t w;
    Query queue_elem[QUEUE_SIZE];
public:
    TaskQueue();
    uint32_t enq(Query data);
    uint32_t deq(Query &pDest);
};
```

1) 构造函数

```
TaskQueue::TaskQueue()
{
    Queue_elem_num = QUEUE_SIZE;
    Mask = QUEUE_SIZE – 1;
    r = 0;
    w = 0;
    for (int i = 0; i <= QUEUE_SIZE – 1; i ++)
    {
        queue_elem[i].str = “”;
    }
}
```

3) 队列写函数

```
uint32_t TaskQueue::enq(Query data)
{
    uint32_t w_pt = w;
    uint32_t r_pt = r;
    uint32_t w_pt_next;
    w_pt_next = (w_pt +1) &mask;
   /*Full*/
   if(w_pt_next == r_pt)
   {
        return 1;
   }
   /*Write*/
   queue_elem[w_pt] = data;
   w = w_pt_next;
   return 0;
}
```

3) 队列读函数

```
uint32_tTaskQueue::deq(Query &pDest)
{
    uint32_t   w_pt = w;
    uint32_t   r_pt = r;
    /*Empty*/
    if(w_pt == r_pt)
    {
        return 1;
    }
    /*Read*/
    pDest = queue_elem[r_pt];
    r_pt = (r_pt +1)&mask;
    r = r_pt;
    return 0;
}
```

从无锁 Job Queue 的实现可以看出，主要是通过 w 和 r 两个指针，来判断循环对列的空或满状态，从而保证在无锁的情况下还能保证对列使用逻辑上的正确性。

### 4.4.3 Web Search 核心 Workload

模型中的线程池部分就是作业处理节点的工作，每个节点独立运行，对用户请求进行处理和响应。

对数据服务类高通量应用的 Benchmark 的具体实现，本文选取了 Web Search 应用中的响应用户请求部分。如前文所述，这一部分包含 Query Parser、Database Query、Term Weight、Document Weight 和 MSet 几个核心 Workload。几个模块构成一个完整的处理流程，流程图如图 4.6所示。

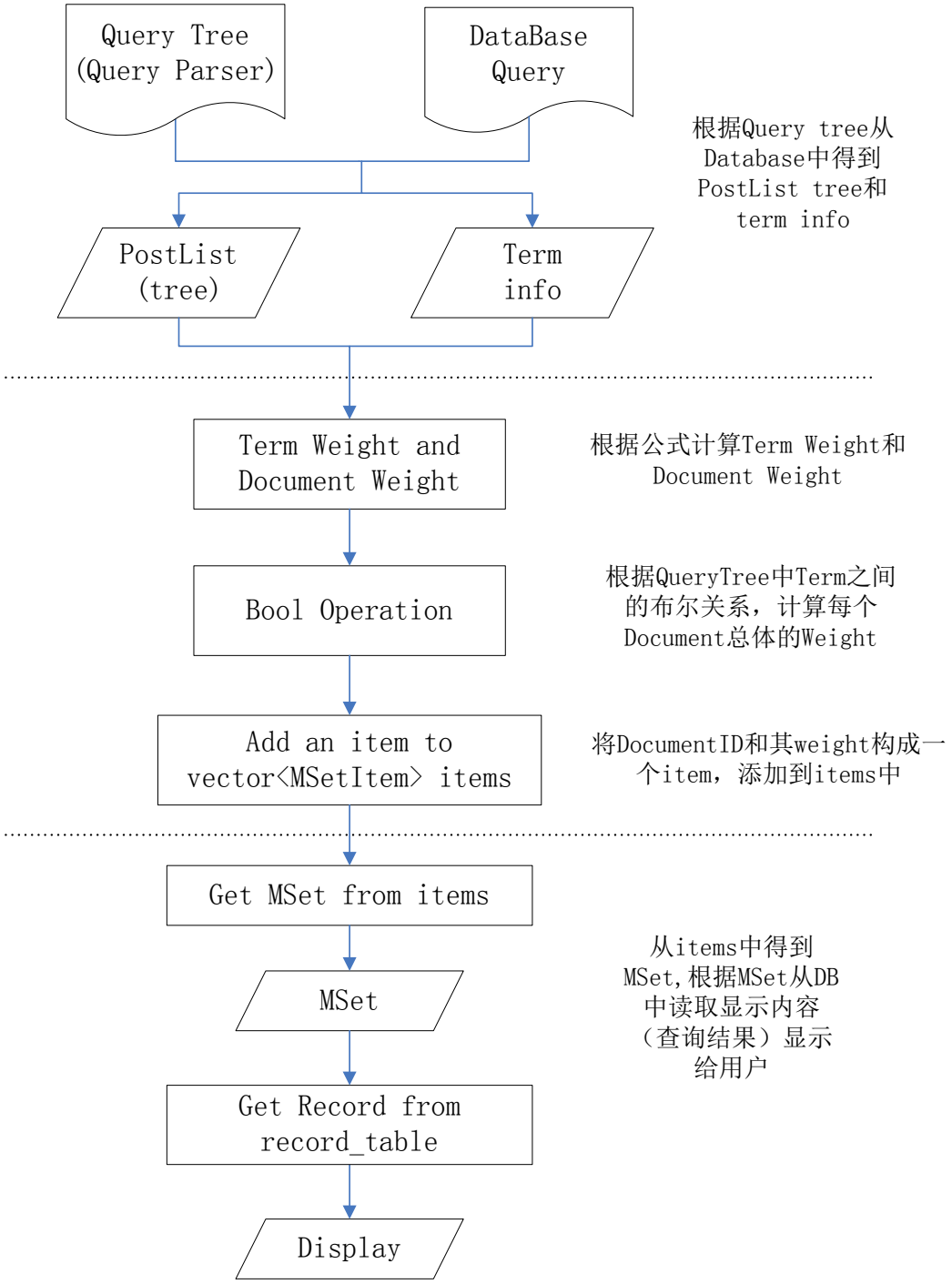



图 4.6 Search 响应用户请求部分处理流程图

首先由 QueryParser 对用户输入的 Key Words 进行 parse 得到 Query Tree，此过程主要是去掉 Key Words 中的 Stop Words，提取单词词干得到 term，组织各个 term 之间的布尔关系等，最后形成一个 Query Tree。

对 Query Tree 中每个叶节点的 Term，进行 Database Query，得到相应的数据。

计算 Term Weight，并结合 Query Tree 中的布尔关系，计算 Document Weight。

执行 MSet 过程，从查询结果中选取最符合要求的结果，返回给用户。

整个过程中，需要建立 Enquire 的一个对象，称为一个查询的实体，Query Tree 和 Database 是查询实体的成员。

此部分工作流程具有典型的数据服务类高通量应用特点，可以有效的反映出对处理器的需求。

本小节详细介绍流程中的各个 Workload 的及其具体实现方式。

#### 4.4.3.1 相关概念介绍

Document 是指查询请求要获取的文档，每一个 Document 中都会包含大量的 Term，Term 是一些单词或者短语。当一个 Document D 中含有一个 Term t，那么，D 就能被 t 索引到。在一个实际的系统中，一般会有大量的 Document 和大量的 Term，所以关系可以表示为 $t_i$ ->$D_j$。

一个 Term 索引到一个 Document，称之为一个 Posting，另外，Posting 中还带有位置信息。

一个 Document 中会有大量的 Term，用 Term List 来表示。

一个 Term 会索引到多个 Document，用 Posting List 来表示。

通常，一个 Document D 由大量单词组成：

$$D = w_1, w_2, w_3 \ldots w_m$$

许多索引 D 的 Term 应该是派生于这些词（例如它们的小写或者词根）。

如果一个 Term 派生于单词 $w_9$，$w_{38}$，$w_{97}$ 和 $w_{221}$，则这个 Term 在 D 中的 9,38,97 和 221 这些位置出现，所以，对每一个 Term，一个 Document 应该有一个 occur 位置信息的向量。这称作 t 的 within-document positions，简称 t 的 WDP。

一个 Term t 在 D 中的 within-document frequency（简称 WDF）是指索引过程中，t 在 D 中 occur 的次数。通常，WDF 是 WDP 的长度，但在实际中 WDF 可能超过 WDP 的长度，因为可以对 Document 的一些部分使用额外的 WDF。例如，经常对 Document 的 title 部分这么做，来体现 title 比 Document 其他部分的重要性。

有多种测量文件长度的方法，但是最简单的是假设这个 Document 由 m 个 word 组成，$w_1$ 到 $w_m$ 并且定义 Document 的长度是 m。

NDL(normalizeddocument length)用来归一化的表示文件的长度，一般选择一个较小的非零值作为 NDL 的标准[29]。

WQP(within-query positions)是指 Term 在 Query 中出现的位置信息。

WQF(within-query frequency)是指 Term 在 Query 中出现的频数。

#### 4.4.3.2 Query

Query 是为搜索定义的一个类，Query 中的主要内容是对用户输入的关键字进行解析后得到的要检索的内容。

**1) 只有一个 term 的 query**

只有一个 term 的 query 可以如下产生：

Query query(term);

这个构造函数还包含其他的参数，用于指示 term 的位置信息和频率信息：

Query(const string &tname_,

termcountwqf_ = 1,

termposterm_pos_ = 0)

其中：

wqf：用于指示对应 term 出现的频数，体现了此 term 的重要性，wqf 在单 term 的 query 中没有用，在多 term 的 query 中有用。

term_pos：指示此 term 在此 query 中的位置，同样，term_pos 在单 term 的 query 中也是没意义的。它用于短语查询，段落检索，和其他的需要知道 query 中各个 term 的顺序的操作。如果不关心 term_pos，可以使用默认值 0。

**2) 含有多个 term 的 query**

复合 query 可以通过将单 term 的 query 用操作符连接来构造。可以用如下构造函数来两两组合：

Query(Query::op op_,

constQuery& left,

constQuery& right)

最常用的两个操作符是 Query::OP_AND 和 Query::OP_OR，表 4.2中列出一些操作符和他们的具体含义：

表 4.2 操作符及其功能

| | |
|---|---|
| Query::OP_AND | 返回两个 Query 都索引到的 document |
| Query::OP_OR | 返回两个 Query 中任何一个索引到的 document |
| Query::OP_AND_NOT | 返回左边的 Query 索引到但是没有被右边的 Query 索引到的 document |
| Query::OP_FILTER | 返回两个 Query 都索引到的 document，但是只取左边 Query 得到的 weight 值 |
| Query::OP_AND_MAYBE | 返回被左边 Query 索引到的 document，如果此 document 也被右边的 Query 索引到，则 weight 值 |

| | |
|---|---|
| | 取两个 weight 值之和 |
| Query::OP_XOR | 返回只被两个 Query 中某一个索引到的 document |
| Query::OP_NEAR | 返回两个 Query 都索引到的 document，而且要求这两个 Query 中的 term 在此 document 中的距离符合一个指定值 |
| Query::OP_PHRASE | 返回两个 Query 都索引到的 document，而且要求这两个 Query 中的 term 在此 document 中的距离符合一个指定值，顺序符合指定情况 |
| Query::OP_ELITE_SET | 从两个 Query 中所有的 term 中选取权值最高的 term 组成一个子集，这个子集中的所有 term 取 OR，得到新的 Query |

**3) Query 的工作原理**

一个 Query 可以看做一个树状结构，每个结点是一个 Query::op 操作，左右分支也是两个 Query 树，在每个叶结点是一个 term，见图 4.7所示。查询工作过程中，根据此 Query 树和索引到的 document 可以构成一个 Postlist 树，内部结点对应的 document 集根据表 4.2所示原则从左右子树得到。根据 Postlist 树可以计算索引到的 document 的 weight 值，叶结点中的 document 的 weight 值按照 BM25 相关公式计算，内部结点中 document 的 weight 值根据表 4.2所示原则从左右子树得到。

对于查询请求：Research OR （News AND Student），Query 树如图 4.7所示。

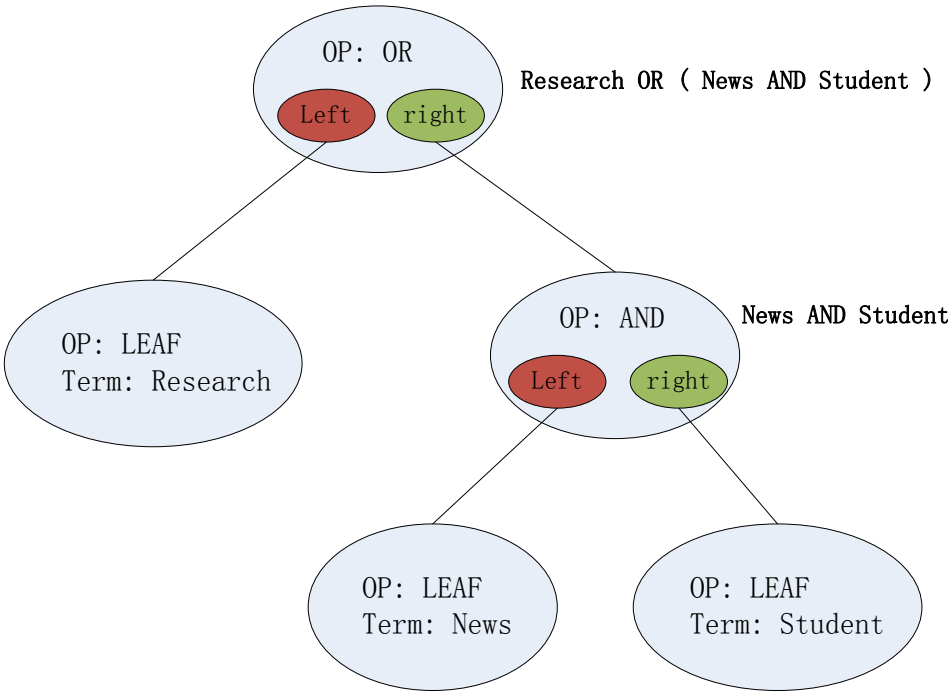



图 4.7 Query 树结构图

对应的 Postlist 如图 4.8所示：

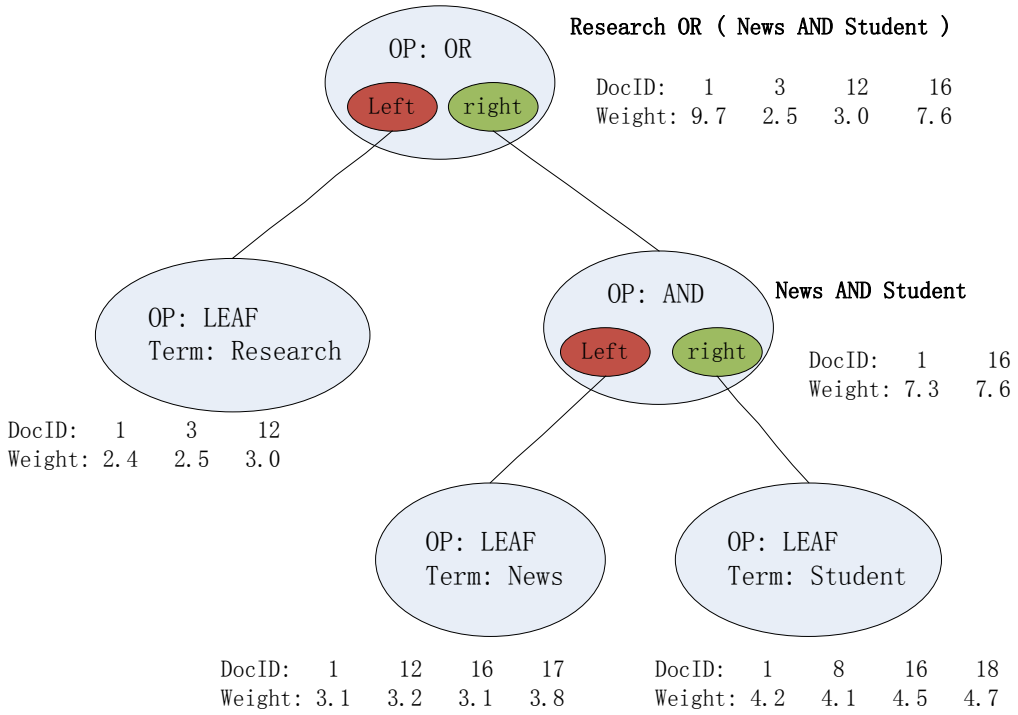



图 4.8 Postlist 树结构图

综上，一个 Query 可以看做一个由所有叶结点的 Query 的 term 产生一个 MSet 的过程。每个叶结点给出一个命中的 document 的列表及表中各 document 对应的 weight。其他各级的结点都是类似的，根结点得到最终的 document 列表及各 document 的 weight。MSet 就是根结点的 list 中的一部分，一般情况下取 weight 最高的部分。

#### 4.4.3.3 Query Parser

实现了一个 QueryParser 模块，用一个类来实现，可以将用户输入的 Key Words 字符串转换成 Query 树，生成 Query 类的一个对象。主要功能与支持的 Key Words 输入技巧有：

**1) Term Generation：**

Term Generate 的作用是将用户输入的 Key Words 中的文本进行分词，得到文档中所有的 Term。这是 Query Parser 的第一步。其中包括了单词的词干提取（Stemming），大小写归一化，以及无效词的过滤等操作。还用到一些策略，比如同一文档中重复出现的 Term 可以合并，再统计其频率。另外，也会记录 term 在 Key Words 中出现的位置。

**2) Bracketed Expressions：**

可以通过括号来指定各个运算之间的优先级。例如：

对于 apple OR pear AND dessert，QueryParser 默认的解析为 apple OR (pear AND dessert)，如果使用(apple OR pear) AND dessert 将改变表达的布尔关系。

**3) Default Operator：**

如果用户输入的 Key Words 中没有指定之间的运算符，将默认取 OP_OR。

**4) Additional operators：**

除了表 4.2列出的逻辑运算符，Queryparser 还支持其他一些，例如：

apple NEAR dessert

president "united states"

"race condition" -horse

+recipe +apple pie cake dessert

其中“+”和“-”是新的运算符，分别用于指定选择有和没有某个 term 的 document。例如：

"race condition" -horse

用于匹配所有含有短语“race condition”但是不含有 horse 的 document，即得到的 document 必须不含有 horse。再如：

+recipe +apple pie cake dessert

用于匹配所有含有 recipe 和 apple 的 document 按后三个 term 进行 weight 排序，即得到的 document 必须含有 recipe 和 apple。

**5) Ranges：**

QueryParser 支持关于 document values（document values 的介绍见 Document 一小节）的 range search。即匹配 value 在一个给定范围内的 document。Range 指定方式如下所示

$10..50

01/01/1970..01/03/1970

**6) Stop Words：**

用于指定哪些单词要从用户输入的 Key Words 中去除，例如 the，a，an 等虚词。但是在查询短语时，Stop Words 不起作用，例如：

“the green space”将得到含有这整个短语的 document。

**7) Parser Flags：**

QueryParser 具体可以实现的功能可以通过标志位来选择，通过将以下标志进行位或实现：

FLAG_BOOLEAN：使能 AND 和 OR 等基本布尔运算符和括号；

FLAG_PHRASE：使能对短语的支持；

FLAG_LOVEHATE：使能对“+”和“-”的支持；

FLAG_BOOLEAN_ANY_CASE：使能运算符的混合使用；

FLAG_WILDCARD：使能对通配符的支持。

默认情况下，FLAG_BOOLEAN、FLAG_PHRASE、FLAG_LOVEHATE 被设置。

#### 4.4.3.4 Database Query

数据库操作是数据服务类应用的核心之一，具体到 Search 中，是通过数据库查询，得到每个 Term 索引到的 Document 信息。

Search 系统使用的 Database 的索引表是按 B 树的方式管理的，可以获得高效的磁盘访问，有利于做快速的查询和修改。具体实现中，Table 数据类型负责了 B 树以及数据文件的管理。Database 的成员变量主要就是 Postlist Table，Termlist Table，Record Table 和 Position Table 这四个 Table 的实例，分别用来维护 Postlist，Term，Document Data 还有 Position 的数据文件以及索引表。这四个文件的结构分别为：

Postlist Table：

Key：Term

Tag：$WDF_1$ $DocID_1 WDF_2$ $DocID_2$ ... $WDF_{last}$ $DocID_{last}$

Termlist Table：

Key：DocID

Tag：$Term_1 WDF_1 Term_2 WDF_2$... $Term_{last} WDF_{last}$

Record Table：

Key：DocID

Tag：Document Data & Document Value

Position Table：

Key：DocID+Term

Tag：Term 在 DocID 对应的文档中的所有位置。

查询过程中，主要涉及到的两个表是 Postlist Table 和 Position Table。对于 Query 树的叶结点，只有一个 term，直接调用 Database 的相关函数：

LeafPostList * pl = db->open_post_list(term);

此函数的具体工作就是到 Postlist Table 中进行查询，以 term 作为 key，搜索相应的<key, tag>对，然后读出其 tag，tag 的内容就是此 term 索引到的所有 document ID 和相应的 WDF。然后，以 term+document ID 作为 key，到 Position Table 中查询，得到 term 在相应 document 中出现的位置信息和频率信息。

而对于内部结点，则要根据子树之间的布尔运算关系，得到相应的 document 集（Query 树如图 4.7所示和布尔运算关系见表 4.2）。

#### 4.4.3.5 Term Weight

假设有一个搜索系统共有 N 个 Documents，Q 是一个 Query，由$t_1, t_2, \ldots t_Q$组成，有 R 个 Documents 与这个 Query 相关。

1976 年，Stephen Robertson 给出了一个计算 Q 中 term t 的权重的公式。权重越高表示这个 term 越重要。如下：

$$w(t) = \log\left(\frac{p(1-q)}{(1-p)q}\right)$$

p 是 t 索引到一个相关的 document 的概率，q 是 t 索引到一个不相关的 document 的

可能性。形式化的表示：

p = P(t → D | D in R)

q = P(t → D | D not in R)

1 - p = P(t not → D | D in R)

1 - q = P(t not → D | D not in R)

假设 t 索引了 N 个 documents 中的 n 个。设相关的 documents 总数是 R，t 索引了其中的 r 个。

可以得到：

$$p = \frac{r}{R}$$

$$q = \frac{n-r}{N-R}$$

$$1-p = \frac{R-r}{R}$$

$$1-q = \frac{N-R-n+r}{N-R}$$

所以能得到：

$$w(t) = \log\left(\frac{(N-R-n+r)r}{(R-r)(n-r)}\right)$$

注意到上述公式在 r=n 时会得到无穷大，在 r=0 时会得到负无穷大，所以，Robertson 将其改为：

$$w(t) = \log\left(\frac{(r+\frac{1}{2})(N-R-n+r+\frac{1}{2})}{(R-r+\frac{1}{2})(n-r+\frac{1}{2})}\right)$$

这就是 BM25[30]中用的 Term Weighting Formula。

公示中 R 是未知量，但是，可以通过 R 的一个子集来得到 p 和 q 并计算 w(t)，这个 weight 可以用于提高查询的效率。

实际上，R 并不表示相关文件的完全集，这个完全集实际上是很难被发现的，R 是表示一个小的被判为相关的文件的集合。

如果还没有 Documents 被标记为 relevant，那么$R = r = 0$，并且 w(t)变为：

$$\log\left(\frac{N-n+\frac{1}{2}}{n+\frac{1}{2}}\right)$$

这就是搜索系统中常用的逆对数加权，本文所实现的 Benchmark 中实现了这个公式，以完成 Term Weight 的计算。

#### 4.4.3.6 Document Weight

根据 w(t)的值，可以计算 W(d)，即考虑一个 Term 时，某 Document 的 Weight。对于单个 Term，Document d 的权重计算公式为：

$$W(d) = \frac{(k+1)f_t}{kL_d + f_t} w(t)$$

其中，$f_t$是 t 在 d 中的 WDF，$L_d$是 d 的 NDL，k 是常数。(k+1)这个因子有助于方程的解释。

如果 k 设为 0，则 w(t)前的因子成为 1，WDF 被忽略；如果 k 趋向于无穷大，因子变成$f_t/L_d$，WDF 的重要性被放到很大。k 取中间值就是在两个极限之间调整 WDF 的重要性。

一般默认取 k 为 1，这对大多数系统可以得到合理的结果。W(d)被 WDF 改变了很小，用户得到一个简单的搜索模式。对于特别的搜索系统，可以调整 k 以得到最优的结果。

如下函数提供了对上述公式的支持：

BM25Weight::get_sumpart(uint64_t wdf, uint64_t len)

其中，wdf 是此 term 在此 document 中出现的频数，len 是此 document 的长度。

#### 4.4.3.7 MSet

得到每个 Term 索引到的 Document 的权重之后，根据 QueryParse 得到的 Postlist 树和表 4.2的计算原则，计算此 Document 总的 weight。计算 Document d 的权重的公式如下所示：

$$W(d) = \sum_{t \to d, t \text{ in } Q} \frac{(k+1)f_t}{kL_d + f_t} w(t)$$

如果 query 中没有 term 索引到 d，那么 W(d)将为 0。实际情况中只有 W(d)>0 的是用户感兴趣的。

如下所示是对于 Query Q 的按照 W(d)的降序排列的一个 Match Set(MSet)：

item 0: $d_0$, W($d_0$)

item 1: $d_1$, W($d_1$)

item 2: $d_2$, W($d_2$)

.....

item K: $d_K$, W($d_K$)

根据 query 中的 term 对应的 Posting Lists 中得到 Mset。

MSet 产生之后会选取前 K 个，K 的典型值在 1000 或者小于 1000。

修正后的权重计算公式将 Query 考虑在内：

$$W(d) = \sum_{t \to d, t \text{ in } Q} \frac{(k_3+1)q_t}{k_3 L' + q_t} \frac{(k+1)f_t}{kL_d + f_t} w(t)$$

其中，$q_t$ 是 t 在 Q 中的 WQF，L’是 nql，$k_3$ 是另一个常数。

本文计算权重策略是这个权重策略修正版的推广形式—BM25。在 BM25 中，L’一般被设为 1。

### 4.4.4 性能指标统计模块

根据前文的分析，数据服务类高通量应用主要的性能指标是单位时间内能够响应的用户请求数量，也就是处理器对用户请求的吞吐效率。因此，本文所实现的 Benchmark 中设计了相应的性能统计模块，用于评价处理器在处理数据服务类高通量应用的 Benchmark 时所反映出的性能。

如图所示是 search 的统计结果示例。

```
the total time is:              193259486616 nsec
the thread_average_time is:     48314871654 nsec
the query_average_time is:      1150354 nsec
the total_max_query_time is:    4873553 nsec
the total_min_query_time is:    21026 nsec
the total query num is:         168000
the total timeout num is:       0
the throughput is:              3477 /sec
```

图 4.9 Search 统计信息示意图

其中各个指标的含义是：

Total time： Benchmark 总的运行时间；

Thread_average_time：每个用户请求处理线程平局执行时间；

Query_everage_time：每个用户请求平均响应时间；

Total_max_query_time：最大的用户响应时间；

Total_min_query_time：最小的用户响应时间；

Total query num：本次执行 Benchmark 所处理的用户请求总数；

Total timeout num is：定义如果对一个用户请求的响应时间超过 10ms，就算做响应超时，此指标表示响应超时的用户请求的数量。

Throughput：平均每秒钟处理的用户请求数量，即系统的吞吐效率。

通过以上指标，可以对处理器运行数据服务类高通量应用时的吞吐效率进行评价。

## 4.5 实时交互类高通量应用 Benchmark 实现

本小节首先提出了实时交互类高通量应用的并行化模型，然后对 RNC 用户面关键数据处理流程进行模拟，基于此模型实现了 RNC 测试程序。

### 4.5.1 实时交互类高通量应用多节点并行化模型

实时交互类应用每个作业的主要工作是按照协议处理用户数据。实时交互类高通量应用的基于线程的作业处理节点并行化模型结构如图 4.10所示。

1) 每个用户注册之后都会有与具体协议相关的实例表，处理数据时需要查询。

2) 线程池，大量线程作为作业处理节点。

3) 每个线程节点对应多个用户，轮询处理每个用户的数据包 buffer，处理时需要查询相应的实例表。

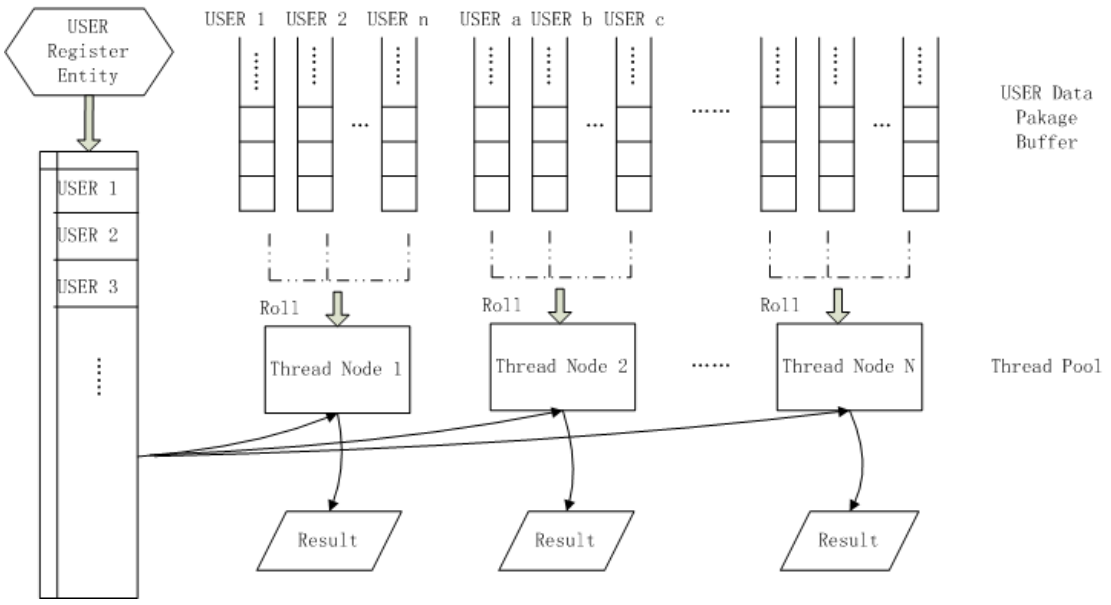



图 4.10 实时交互类并行化模型

基于上述模型，实现 RNC 测试程序。

### 4.5.2 Radio Network Controller 核心 Workload

无线网络控制器（RNC）是 3G 移动通信陆地无线接入网中数据处理的核心部分。它包含用户面和控制面。控制面主要控制初始化、资源的分配以及数据处理结束后的资源释放等；用户面负责对接入用户的数据包接收、存储、数据的协议处理和转发等，整个用户面的处理流程是按照协议来处理用户数据的。用户面是 RNC 中具有高通量需求的部分。

通过对 RNC 用户面协议进行解读和对商用代码的分析，可以提取出用户面协议处理流程中最核心的部分。主要可以分成四个 Workload：SDU Receive，MACD Schedule，Entity Query 和 Segment。本文实现的 RNC 测试程序中，主要实现这几个 Workload，用于模拟 RNC 用户名数据处理流程。

应用系统的吞吐能力体现在保证实时性的前提下，能够支持的同时在线的用户数。

#### 4.5.2.1 SDU Receive

此 Workload 的主要工作是处理器从核心网获取数据，得到含有用户业务信息的源数据，对源数据进行几个操作相对简单的协议层的协议处理。由于涉及到的 GTPU、PDCP 两个协议层数据处理协议比较简单，只是简单的 Unpack 和 Pack 的过程，所以，在此 Benchmark 中将这几个协议层的工作放在同一个 Workload 中完成。

处理之后，可以得到 RLC 层的服务数据单元（SDU），并将 RLC SDU 缓存于 SDU 缓存区。处理过程如图 4.11所示。

具体测试程序实现中，此过程对应于并行化模型中的将每个用户的数据不断放入用户数据 Buffer 的过程。

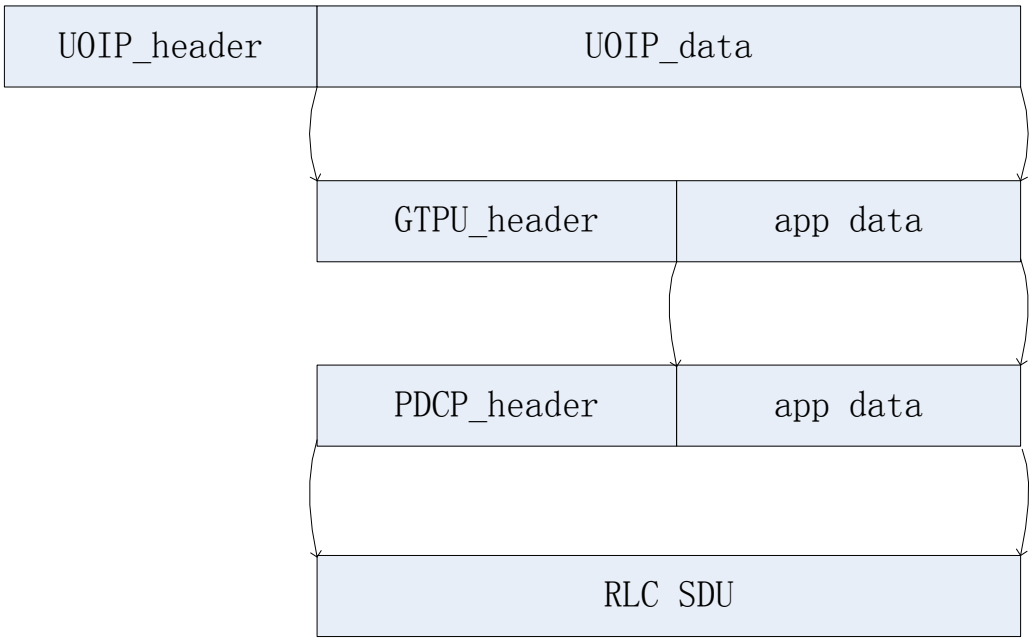



图 4.11 SDU Receive 流程图

#### 4.5.2.2 MACD Schedule

为了保证服务质量，MACD Schedule 控制每 10ms 中每个用户至少处理一个 SDU 数据包，即完成一个用户 SDU 数据包的解析和转存。

为此，系统设计一个单独的任务调度系统，用来控制处理节点从 Buffer 中读取用户数据包，配合一个定时器模块使用。

定时器模块可以使用操作系统自带的统计系统 Cycle 的函数来实现：

```
static inline unsigned long long read_tsc(void)
{
    unsigned eax, edx;
    asm volatile("rdtsc" : "=a" (eax), "=d" (edx));
    return ((unsigned long long)eax) | (((unsigned long long)edx) << 32);
}
```

使用方法：

```
before :
    unsigned long long cycle_start = read_tsc();
after :
    unsigned long long cycle_end = read_tsc();
    unsigned long long cycle_elapse = (cycle_end > cycle_start) ? (cycle_end - cycle_star
t) : (0xffffffffffffffff - cycle_start + cycle_end);
```

其中，变量 cycle_elapse 是经过的 Cycle 数，根据计算机的主频，可以计算出系统运行过的时间。

当计时完成，每个处理节点会从各自对应的所有用户数据包 buffer 中各读取一个用户 SDU 数据包，等待处理。

#### 4.5.2.3 Entity Query

每个用户在接入系统时，要注册一个用户实例，此实例中即数据处理的协议，包含业务处理的相关信息，其中主要的几部分内容有：

—用户 ID

—PDU 大小

—传输窗口大小（此用户每 10ms 中最多传送的 PDU 个数）

—已成功发送的 PDU 数量

—下一个 PDU 的序列号（Sequence Number）

等几项。

在商用的系统中，每个实例中包含大量的配置信息，每个实例所占空间约为 10KB 左右。同一个实例中不同的信息项的使用过程是相似的，对 Cache Miss 和 CPU 占用率的影响也是相同的。本文为了使用尽量少的代码实现实际应用中的程序的特性，所实现的 Benchmark 中只使用实例众多内容中的很少的几项。而为了使每个实例的大小与商用程序中的实例的大小相同，可以在实例的结构体中加入一系列填充。

实例结构体的定义如下：

```
typedef struct _entity_t
{
    char pad0[1024];
    uint16_t user_id;
    char pad1[1024];
    uint16_t pdu_size;
    char pad2[1024];
    uint8_t VT_A;
    char pad3[1024];
    uint8_t VT_S;
    char pad4[1024];
    uint8_t AM_window_size;
    char pad5[10 * 1024 – 1024 * 5 - 7];
}entity_t
```

定义一个实例表初始化函数，在程序的初始化部分为每个用户初始化一个实例，所有用户的实例构成一个实例表。

首先要申请一块内存空间，用于存放实例表：

```
entity_t *gp_entity_table;
gp_entity_table = (entity_t *)malloc(user_num * sizeof(entity_t));
```

然后，将每个用户的对应信息写入到这个用户对应的实例中去。

在对每个用户的 SDU 数据包进行解析之前，需要根据用户 ID，查询实例表，得到用户 SDU 数据包处理的相关信息，以进行 SDU 数据包的处理。

#### 4.5.2.4 Segment

对 SDU 数据包进行协议处理，处理的过程就是，解析 SDU，获取头部信息，从头部信息中读取用户 ID，根据用户 ID 查询实例表，根据实例表中的信息进行数据包的处理，然后将得到的 RLC PDU 缓存。

根据协议，SDU 分段重组成 PDU 是 RLC 协议层的主要工作。这个过程主要由一个函数来完成：

uint8_t segment_sdu(char *p_sdu, uint16_t user_num);

函数的处理过程分为以下几个步骤：

每个 SDU 的前 40B 为头部信息，包含 SDU 序号，用户 ID，SDU size 等信息，将 SDU 的头部信息取出，待用：

uint16_t user_id;

uint16_t pkt_id;

uint16_t pkt_len;

根据从 SDU 中得到的 user_id，对实例表进行查找，找到此用户的实例，然后把实例中将要用到的信息取出，待用：

uint16_t pdu_size;

uint8_t VT_A;

uint8_t VT_S;  // PDU 序列号

uint8_t AM_window_size;

计算出存放下一个 PDU 的内存块的头地址，在此 PDU 的头部写入对应的头部信息，根据 PDU 数据格式的规定，进行如下操作：

// write the head info of this PDU

p_pdu_this_head->dc = 1;  // 1 表示此 PDU 为数据 PDU

p_pdu_this_head->seq_num = VT_S++;  // 写入 PDU 的序列号

p_pdu_this_head->p = 0;  // p=0 表示不请求状态 PDU

分几种情况填充 PDU 中的 LI_E 字段，并将 data 字段写入 PDU。

根据协议，要放入 PDU 的不同大小的 SDU 片段，决定了 LI 值的不同，现在，有几种情况需要讨论，讨论之前，还需定义一个变量：

uint16_t  pkt_len_left;

此变量表示当前处理中的 SDU 还未进行处理的部分的大小。

如果用 pdu_size 来表示 PDU 的大小，PDU 的头部要占用 2 字节，因此，LI 字段与 data 字段总共的空间 pdu_size - 2。

如果(pkt_len_left > pdu_size - 2)，此 PDU 中没有 SDU 的分割片，因而不需要长度指示 LI 字段，只需将此 SDU 没处理部分的前(pdu_size - 2)个字节拷贝到此 PDU 的 data 字

段，再继续处理剩余部分；

如果(pkt_size_left = pdu_size - 2)，则此部分 SDU 将正好填满(pdu_size - 2)的空间，并且没有长度指示 LI 字段，从而，根据 LI 特殊字段的预定义，需要在下一个 PDU 的第一个 LI 中填充 000000000000000；

如果(pkt_size_left = pdu_size - 2 - 1)，则此部分 SDU 将正差一个字节填满(pdu_size - 2)的空间，并且没有长度指示 LI 字段，从而，需要在下一个 PDU 的第一个 LI 中填充 111111111111011；

如果(pkt_size_left = pdu_size - 2 - 2)，则此 PDU 中可以填入一个长度指示 LI 字段，并且，此部分 SDU 将正好填满 PDU 的剩余部分，根据协议，不再需要在下一个 PDU 的第一个 LI 中指示此 PDU 正好被一个 SDU 分割片填满；

如果(pkt_size_left = pdu_size - 2 - 3)，则此 PDU 中可以填入一个长度指示 LI 字段，并且，此部分 SDU 将差一个字节填满 PDU 的剩余部分；

如果(pkt_size_left < pdu_size - 2 - 3)，则此 PDU 中可以填入一个长度指示 LI 字段来指示 SDU 分割片的位置，还可以填入一个预定义的特殊功能 LI 字段 111111111111111，来表示 PDU 剩余部分是填充（填充的大小可以为 0）。

经过以上步骤，一个 SDU 即可被成功的分段重组成 PDU 并放入缓存区。

### 4.5.3 性能指标统计模块

根据前文的分析，实时交互类高通量应用的主要性能指标是能够支持同时在线并保证实时处理数据的用户数，即系统对在线用户数的吞吐效率。因此，本文对所实现的 Benchmark 设计了性能统计模块，用于统计和评价处理器在运行实时交互类高通量应用的 Benchmark 时所反映出的性能。

实时交互类高通量应用的 RNC 测试程序在使用时，需要用户指定两个参数：

User_num_per_thread：每个线程的用户数；

Thread_num：用户数据处理线程数。

RNC 测试程序在运行之后，会给出一个结果，判断当前处理器在上述参数设定情况下，是否能够保证对用户数据的实时处理。本测试程序将丢包率指标设定在 5%，若丢包率在 5%以下，输出结果如图 4.12所示，表示系统能够支持此参数设定情况。若丢包率大于 5%，则输出结果如图 4.13所示，表示表示此系统不能支持当前用户数和处理节点数的设定情况。

```
RESULT    ...... PASS
-------------------------------------
User Num Per Thread : 2000
Thread Num          : 4
Total User Num      : 8000
-------------------------------------
```

图 4.12 RNC PASS 输出信息示意图

```
RESULT    ...... FAILED
-------------------------------------
User Num Per Thread : 7000
Thread Num          : 8
Total User Num      : 56000
-------------------------------------
```

图 4.13 RNC FAILED 输出信息示意图

通过以上测试，可以统计出不同处理器运行实时交互类高通量应用的 Benchmark 时所反映出的性能。

## 4.6 本章小结

本章主要介绍了面向高通量处理器的 Benchmark 的具体实现方法。首先，根据所要实现的 Benchmark 的目标平台高通量处理器的结构特点，提出了一种基于线程的作业处理多节点并行化模型思想，对于三种不同的高通量应用模式，这种模型思想的具体实现也是不同的。

然后对数据处理类、数据服务类和实时交互类三种类型的高通量应用，选取了代表性的应用和 Workload，分别介绍了如何实现其 Benchmark。具体实现过程中，首先介绍了不同高通量应用类型的多节点并行模型的具体实现方式，然后详细介绍了 Benchmark 实现过程中所涉及到的算法。

最终，本文实现了 Wordcount、Terasort、Kmeans、Grep、Search 和 RNC 六个测试程序。

# 第5章 Benchmark 实验评估

本文实验部分主要分两个方面，第一方面对 Benchmark 各个测试程序自身所反映出的基本特征进行测试，通过实验，验证所实现的 Benchmark 中的几个测试程序是否能反映出高通量应用的基本特征。第二方面，使用本文所实现的 Benchmark 对 Xeon 和 TILE-gx 两种处理器吞吐率的并行加速比进行了测试，说明了本文所实现的 Benchmark 能够对不同处理器进行正确的评估。

## 5.1 Benchmark 基本程序特征实验评估

本部分实验主要关注作业并发性、作业之间耦合性、访存需求、Cache 使用效率和访存位宽几个方面。

### 5.1.1 作业的并发性评估

高通量应用的一个重要特点是能够多节点并发处理作业，高通量处理器的一个重要目的是在线程级别，支持高通量应用的大规模并发作业，所以，处理器的吞吐效率与用于处理作业的节点数量的关系，可以体现出所设计的 Benchmark 是否具有良好的并发性，从而可以验证面向高通量处理器的 Benchmark 设计的正确性和有效性。

本节实验所用平台是 Tilera 的 Tilegx 处理器。

数据处理类高通量应用的评价指标是单位时间内所能处理的数据量，因而对数据处理类各个测试程序测试指标是数据处理效率与节点数之间的关系。

各个测试程序所用数据集大小分别为 Wordcount（500MB）、Terasort（200MB）、Kmeans（12MB）、Grep（500MB），测试结果如表 5.1所示。

表 5.1 数据处理效率与线程数之间的关系（单位 B/s）

| 线程数 | Wordcount | Terasort | Kmeans | Grep |
|---|---|---|---|---|
| 1 | 9552862 | 1441490 | 80714 | 67607157 |
| 2 | 21664717 | 2745568 | 156242 | 138032576 |
| 4 | 39278326 | 4656939 | 259978 | 265910300 |
| 8 | 65812566 | 6309082 | 560337 | 504880512 |

本部分主要关注各测试程序吞吐效率的并行加速比，因而对所有测试程序得到的数据，以 1 线程的情况为标准进行归一化，得到归一化数据吞吐率与线程数之间的关系如表 5.2所示。

表 5.2 归一化数据吞吐率与线程数之间的关系

| 线程数 | Wordcount | Terasort | Kmeans | Grep |
|---|---|---|---|---|
| 1 | 1.000 | 1.000 | 1.000 | 1.000 |
| 2 | 2.268 | 1.905 | 1.936 | 2.042 |
| 4 | 4.112 | 3.231 | 3.221 | 3.933 |
| 8 | 6.889 | 4.377 | 6.942 | 7.468 |

数据服务类高通量应用所关注的需求指标是单位时间内所能处理的请求数。对 Web Search Benchmark 进行实验，实验数据如表 5.3所示。

表 5.3 Web Search 单位时间处理请求量与线程数的关系

| 线程数 | 吞吐率（单位/s） | 归一化吞吐率 |
|---|---|---|
| 1 | 112.180 | 1.000 |
| 2 | 225.209 | 2.008 |
| 4 | 450.178 | 4.013 |
| 8 | 887.872 | 7.915 |

实时交互类高通量应用所关注的高通量需求指标是，在保证对每个用户的服务质量的前提下，能够同时支持的在线用户数。

对于本文的实时交互类高通量应用的 Benchmark，RNC 用户面 Benchmark，保证服务质量是指用户数据包由于系统繁忙而导致的丢包率在一定的范围之内，本文取丢包率小于 5%为可接受的。

对 RNC Benchmark 实验得到实验数据如表 5.4所示。

表 5.4 RNC 支持用户数与线程数之间的关系

| 线程数 | 所能支持的用户数 | 归一化 |
|---|---|---|
| 1 | 2600 | 1.000 |
| 2 | 5100 | 1.962 |
| 4 | 10120 | 3.892 |
| 8 | 20080 | 7.723 |

对以上各个测试程序的归一化测试结果做图表，如图 5.1所示，能够直观的显示出 Benchmark 中各个测试程序的并行加速能力。

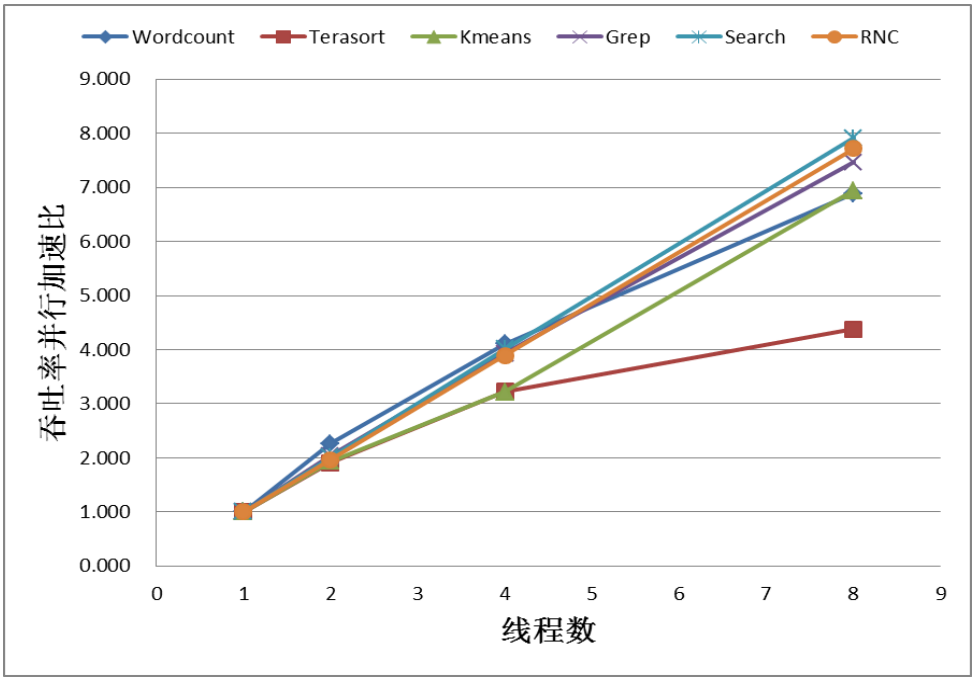



图 5.1 高通量 Benchmark 各测试程序并行加速能力

由实验结果可以很明显的看出，本文所实现的 Benchmark 中的各个测试程序均有较好的并行加速特性，即各测试程序的作业之间具有良好的并发性，能够反映出高通量应用并发性的特点。

### 5.1.2 作业之间耦合性

高通量应用的另一个显著特性是作业之间耦合性较低，这也是高通量应用与其他应用类型的一个显著区别。因此，所设计的 Benchmark 需要能够有效的反映出这个特性。

由于本文所实现的 Benchmark 是共享内存并行编程模型的，每个作业是由一个线程来处理，线程跑在多核或者众核处理器之上，因此，核间共享数据量的大小，可以反映出作业之间耦合性的大小。

本文使用 Intel 的硬件性能测试软件 VTune 进行实验，通过统计核间共享数据量占访存总量的比例，来反映作业之间的耦合性，实验结果如表 5.5所示。

表 5.5 高通量 Benchmark 各测试程序核间共享数据情况

| 测试程序 | 线程数 | 访存指令总数 | 核间共享数据数量 | 核间共享数据占访存指令数的比例（%） |
|---|---|---|---|---|
| Wordcount | 2 | 1634880000 | 9720 | 0.0006 |
| | 4 | 616440000 | 5040 | 0.0008 |
| | 8 | 2234520000 | 302040 | 0.0135 |
| Terasort | 2 | 1811400000 | 645240 | 0.0356 |
| | 4 | 1684080000 | 3079920 | 0.1829 |
| | 8 | 2279340000 | 3709680 | 0.1628 |
| Kmeans | 2 | 573600000 | 286920 | 0.0500 |
| | 4 | 2366640000 | 899640 | 0.0380 |
| | 8 | 5217900000 | 981840 | 0.0188 |
| Grep | 2 | 1376880000 | 81240 | 0.0059 |
| | 4 | 1257660000 | 101640 | 0.0081 |
| | 8 | 1495980000 | 150600 | 0.0101 |
| Search | 2 | 2266980000 | 4560 | 0.0002 |
| | 4 | 6446460000 | 146880 | 0.0023 |
| | 8 | 13591200000 | 925920 | 0.0068 |
| RNC | 2 | 2380080000 | 6600 | 0.0003 |
| | 4 | 5424360000 | 11520 | 0.0002 |
| | 8 | 9575160000 | 30120 | 0.0003 |

如图 5.2直观的显示了核间共享数据量占访存指令总数的比例。从实验结果可以看出，高通量应用的 Benchmark 核间共享数据量占访存指令总数的比例是极小的，其中 Terasort 核间共享数据量所占比例最大，这是由于 Terasort 算法中，有较大的部分需要多线程之间共享数据，但是，核间共享数据量也只有不到 0.2%，而其他几个算法所占比例几乎是可以忽略的。

所以，此 Benchmark 正确反映出了高通量应用各作业之间耦合性低的特点。

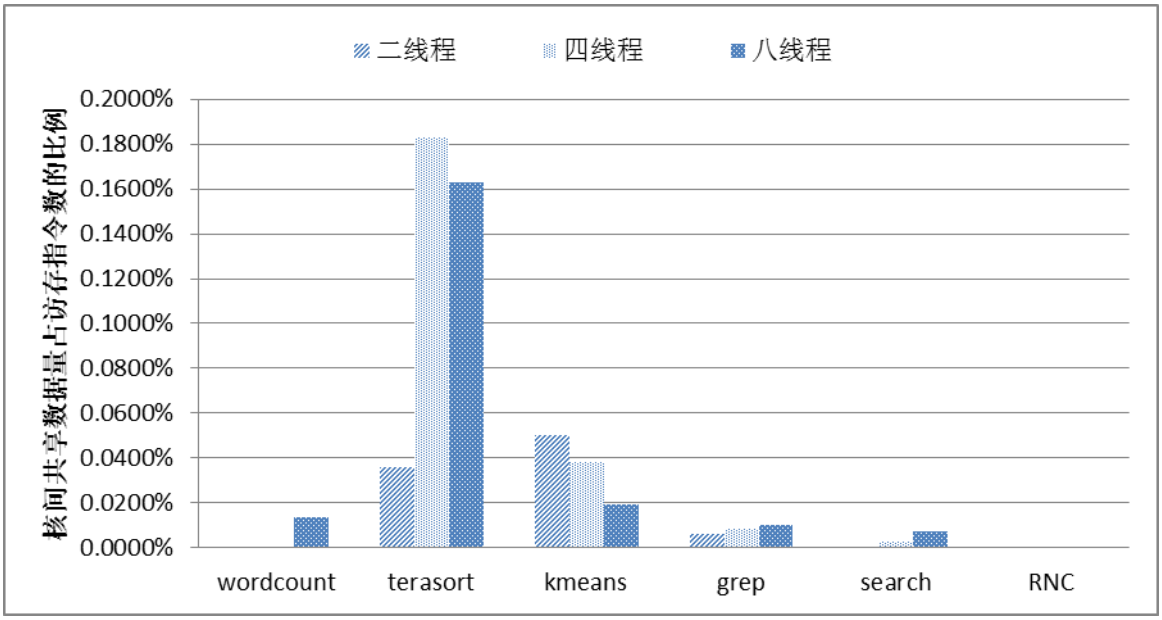



图 5.2 高通量 Benchmark 各测试程序核间共享数据情况

### 5.1.3 访存需求评估

高通量应用主要是面向数据处理和服务，而且数据量大，因而具有较大的访存需求，而计算量较小，与传统的高性能应用相比，具有相对较大的访存计算比。所设计的 Benchmark 需要满足这个特性。

本文对所提取和实现的高通量 Benchmark 的各个测试程序的访存指令所占比例进行了统计，所使用的处理器是 Xeon 处理器，实验结果如表 5.6所示。

表 5.6 高通量 Benchmark 各测试程序访存指令所占比例

| 测试程序 | 线程数 | 访存指令数所占比例（%） |
|---|---|---|
| Wordcount | 4 | 64.86 |
| | 8 | 73.07 |
| Terasort | 4 | 55.11 |
| | 8 | 57.34 |
| Kmeans | 4 | 46.13 |
| | 8 | 56.21 |

| | | |
|---|---|---|
| Grep | 4 | 34.05 |
| | 8 | 38.67 |
| Search | 4 | 51.79 |
| | 8 | 56.01 |
| RNC | 4 | 53.03 |
| | 8 | 52.46 |

从实验数据可以看出，高通量 Benchmark 的各个测试程序，访存需求所占比例是比较高的。

Linpack[31]是典型的高性能计算 Benchmark，在同样的硬件平台上，本文对 Linpack 进行测试访存请求比例的测试，4 路并行和 8 路并行，访存量所占比例分别为 35.39%和 35.03%。所以，与 Linpack 这个典型的高性能 Benchmark 相比较，本文所实现的高通量 Benchmark 具有较高的访存需求。

### 5.1.4 Cache 效率的评估

本文使用 Linux 的性能评估软件 Perf 进行实验测试，得到了 Benchmark 的 L1 Cache Hit Rate、L2 Cache Hit Rate 和 LLC Cache Hit Rate 三个指标，如表 5.7所示。

表 5.7 高通量 Benchmark 各测试程序 Cache 效率统计

| 测试程序 | 线程数 | L1 Cache Hit Rate（%） | L2 Cache Hit Rate（%） | LLC Cache Hit Rate（%） |
|---|---|---|---|---|
| Wordcount | 2 | 97.43 | 62.75 | 81.27 |
| | 4 | 97.43 | 59.60 | 77.86 |
| | 8 | 97.32 | 59.04 | 70.22 |
| Terasort | 2 | 90.28 | 40.08 | 39.92 |
| | 4 | 90.03 | 42.22 | 42.65 |
| | 8 | 90.19 | 43.81 | 41.44 |
| Kmeans | 2 | 99.68 | 81.04 | 30.56 |
| | 4 | 99.67 | 77.30 | 41.83 |
| | 8 | 99.64 | 72.46 | 56.21 |
| Grep | 2 | 99.26 | 84.18 | 45.73 |
| | 4 | 99.25 | 86.11 | 49.20 |
| | 8 | 99.24 | 84.42 | 43.20 |
| Search | 2 | 98.70 | 80.44 | 78.71 |
| | 4 | 98.49 | 80.57 | 77.83 |
| | 8 | 98.30 | 81.75 | 77.36 |

| | | | | |
|---|---|---|---|---|
| RNC | 2 | 99.01 | 43.45 | 50.49 |
| | 4 | 99.40 | 36.09 | 62.27 |
| | 8 | 99.60 | 34.58 | 60.27 |

Splash2[32]是传统并行应用领域中典型的 Benchmark，是斯坦福大学推出的共享存储并行应用 Benchmark，其中选取的算法和应用都是传统高性能并行应用，本文以此 Benchmark 为对比，选取了 Splash2 中的 Rautrace、Cholesky、Radix、Ocean、Radiosity 和 Barnes 六个测试程序进行相应指标测试，验证本文所设计的面向高通量应用的 Benchmark 与传统并行应用在 Cache 使用效率方面的差别，同时说明本文所实现的高通量 Benchmark 的有效性。实验结果如表 5.8所示。

表 5.8 Splash2 各测试程序 Cache 效率统计

| 测试程序 | 线程数 | L1 Cache Hit Rate（%） | L2 Cache Hit Rate（%） | LLC Cache Hit Rate（%） |
|---|---|---|---|---|
| Raytrace | 2 | 91.74 | 91.82 | 93.99 |
| | 4 | 91.64 | 93.19 | 83.00 |
| | 8 | 91.61 | 92.53 | 91.89 |
| Cholesky | 2 | 96.78 | 89.69 | 76.27 |
| | 4 | 97.43 | 89.58 | 71.88 |
| | 8 | 97.64 | 86.90 | 66.90 |
| Radix | 2 | 87.17 | 82.86 | 89.47 |
| | 4 | 87.29 | 82.70 | 82.84 |
| | 8 | 86.80 | 82.10 | 88.42 |
| Ocean | 2 | 90.63 | 87.29 | 92.97 |
| | 4 | 90.25 | 86.88 | 82.85 |
| | 8 | 90.36 | 88.54 | 81.93 |
| Radiosity | 2 | 99.72 | 87.12 | 88.32 |
| | 4 | 99.70 | 87.91 | 96.10 |
| | 8 | 99.70 | 87.50 | 96.03 |
| Barnes | 2 | 94.50 | 95.58 | 99.34 |
| | 4 | 94.33 | 96.27 | 93.59 |
| | 8 | 94.31 | 95.80 | 92.15 |

对 L1 Cache Hit Rate、L2 Cache Hit Rate 和 LLC Cache Hit Rate 分别做图表如图 5.3、图 5.4和图 5.5所示。

从实验结果可以看到，两种类型的应用的 Benchmark 的 L1 Cache Hit Rate 值都很高，都在 90%以上，没有明显的差别。

而对于 L2 Cache 和 L3 Cache，总体来说，高通量类应用的 Benchmark 的命中率比 Splash2 各个测试程序的命中率明显要小。

总体来说，与传统共享内存类应用，高通量应用 Benchmark 在 L2 和 L3 的 Cache 命中率方面表现出了较小的值。

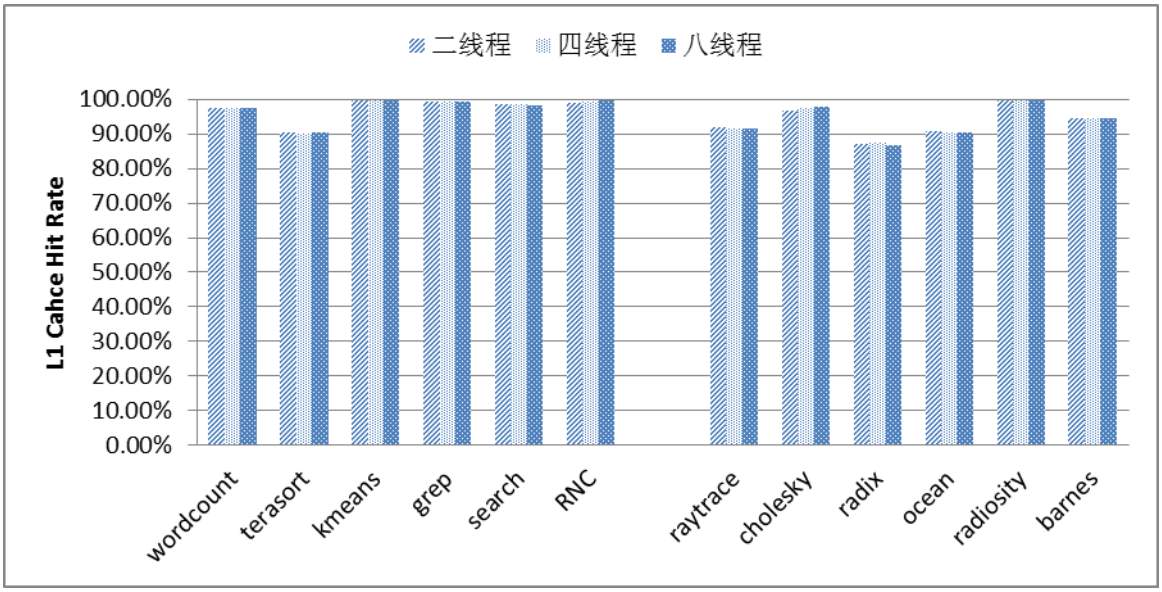



图 5.3 高通量 Benchmark 和 Splash2 各测试程序 L1 Cache Hit Rate

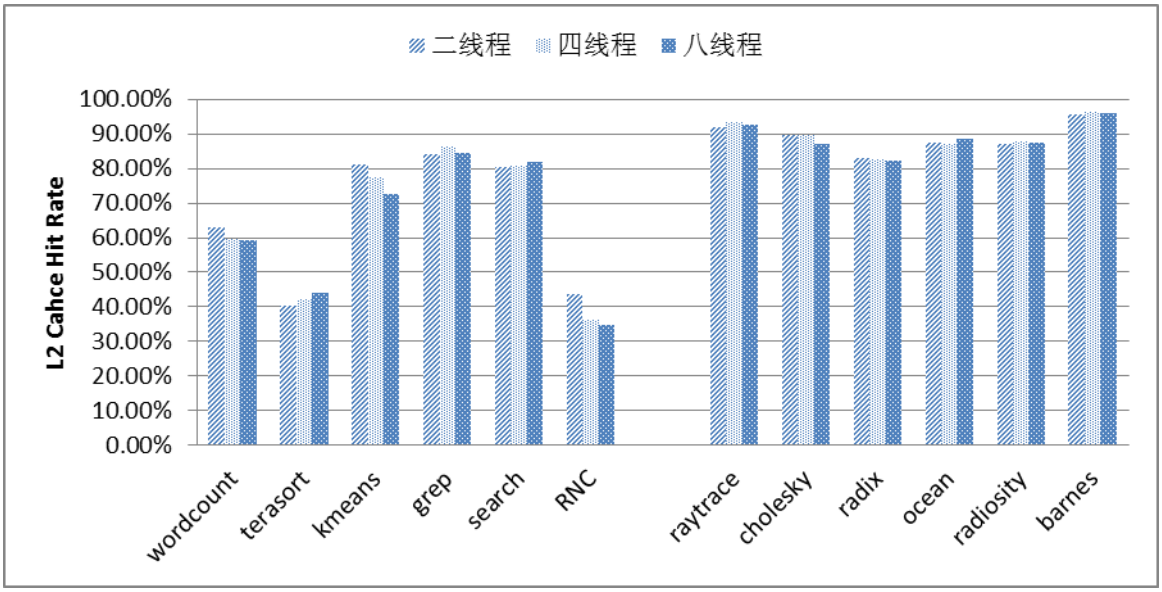



图 5.4 高通量 Benchmark 和 Splash2 各测试程序 L2 Cache Hit Rate

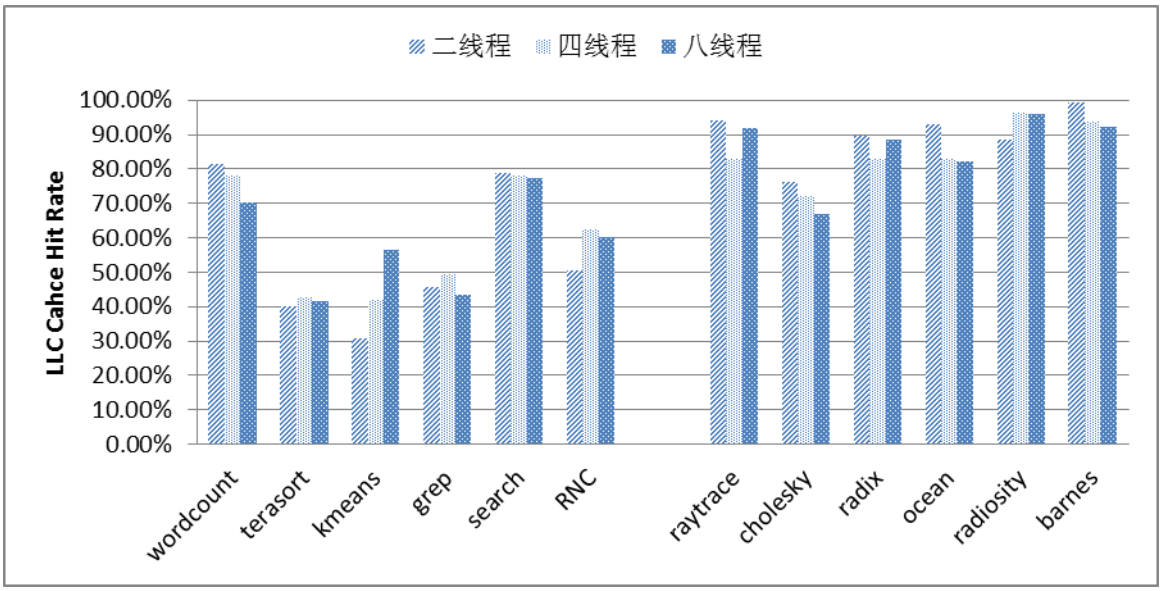



图 5.5 高通量 Benchmark 和 Splash2 各测试程序 LLC Cache Hit Rate

高通量应用由于所处理的数据具有数据量大、离散性强等特点，导致数据的空间局部性差，而数据具有流式的使用特点，导致数据的时间局部性差。因而，Cache 失效率较大。

### 5.1.5 访存宽度评估

传统领域的高性能应用多为数值计算，因而访存宽度多以 64bit 或 32bit 为主。

与传统的应用领域不同，高通量应用由于数据格式的为多样化和不规则性，如文本数据、图数据、K-V 格式数据、各类数据包等，导致访存宽度的多样化。而且，较小访存宽度的比例占到不可忽视的比例。

本文选取 Splash2[33]中 fft、lu、cholesky、ocean、radiosity 和 raytrace 六个算法和应用，来比较高通量应用与传统的共享存储并行应用在访存宽度方面的差别。

使用 Pin 工具抓取 Benchmark 中各个算法的访存情况，统计访存宽度的情况，数据如表 5.9和表 5.10所示：

表 5.9 高通量 Benchmark 各测试程序访存宽度统计

| 宽　度（Byte） | Wordcount | Terasort | Kmeans | Grep | Search | RNC |
|---|---|---|---|---|---|---|
| 1B | 348234806 | 353970602 | 85074808 | 1523572940 | 199962502 | 746825213 |
| 2B | 7534 | 10049810 | 7737 | 7404 | 19316131 | 272588874 |
| 4B | 63565003 | 815501558 | 11585485315 | 52131317 | 215589385 | 413784996 |
| 8B | 864005174 | 3614172029 | 1998436365 | 208425020 | 885078429 | 386060985 |
| 16B | 210 | 614 | 174 | 102 | 2099895 | 119560017 |
| 32B | 0 | 0 | 0 | 0 | 0 | 0 |
| 64B | 82 | 56 | 11444 | 68 | 0 | 0 |

表 5.10 Splash 各测试程序访存宽度统计

| 宽度（Byte） | fft | lu | cholesky | ocean | radiosity | raytrace |
|---|---|---|---|---|---|---|
| 1B | 1451 | 32 | 4961253 | 11682 | 0 | 395043 |
| 2B | 51 | 0 | 24182 | 0 | 0 | 14 |
| 4B | 8582073 | 240 | 1752385 | 60718 | 67972183 | 1518853 |
| 8B | 91365362 | 101442347 | 416924688 | 101435309 | 31421024 | 98626918 |
| 16B | 962936 | 0 | 131 | 288 | 2110161 | 1126127 |
| 32B | 0 | 0 | 0 | 0 | 0 | 0 |
| 64B | 0 | 0 | 0 | 0 | 0 | 0 |

计算各个应用中，各访存宽度占访存总数的百分比，得到结果如图 5.6所示：

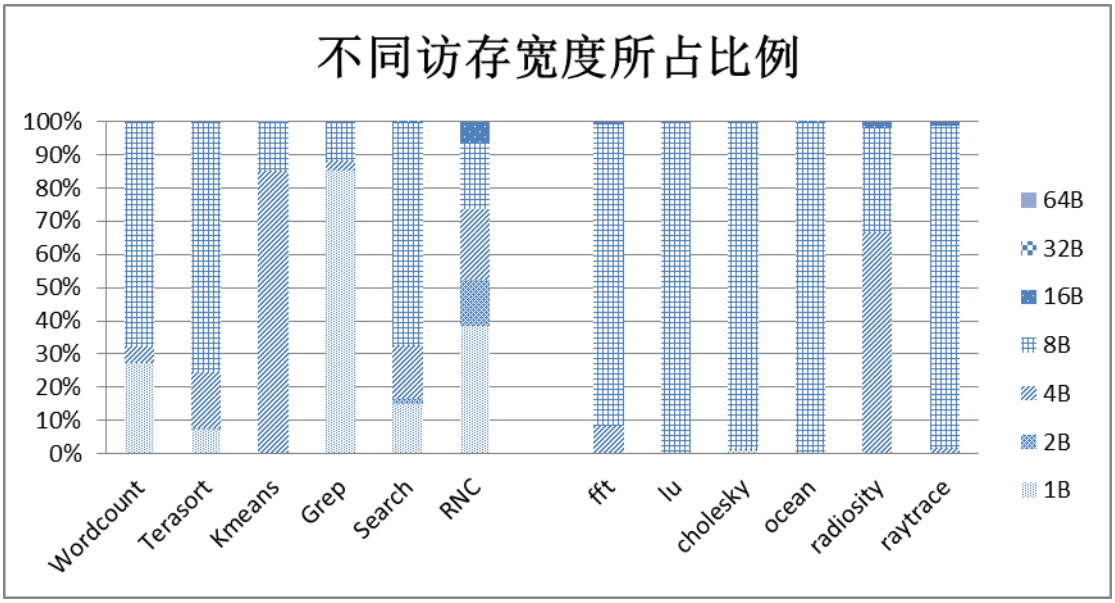



图 5.6 高通量 Benchmark 与 Splash2 各测试程序访存宽度比较

从图表可以看出：

1) 传统并行应用的访存宽度多集中于 8B 和 4B，而高通量应用的访存具有多样化和不规则性。

2) 高通量应用中 1B 和 2B 的访存占了较大的比例，这种较小的访存宽度用于传统的访存结构设计的系统中会导致带宽资源的浪费，对高通量处理器访存设计提出新的要求。

3) 本文所设计的高通量应用 Benchmark 良好的反映出了高通量应用在访存宽度方面的特征。

## 5.2 对 Tilera TILE-Gx 处理和 Intel Xeon 处理器进行评估

TILE-Gx[34]处理器是 Tilera 公司推出的一款众核处理器，TILE-Gx 的目的是将上百个处理器核集中到一颗处理器上。TILE-Gx 使用了一种新的 Mesh 网络，被称为 iMesh，iMesh 是一种五层网络结构，这一设计使得各个处理器核之间能够高效通信，有效的解

决了众核处理器中核间通信的问题。TILE-Gx 处理器达到了处理效率更高，并发处理能力更强，能耗更小和扩展性更强等方面的效果。而处理器中的每一个核处理能力有限，尤其对浮点计算的支持能力很弱。这是由于 TILE-Gx 更关注于网络、视频和云计算等具有大数据和高通量应用特点的应用领域，而不是针对高性能科学计算进行的设计。

本文实验所采用的是 TILE-Gx 系列中的 TILE-Gx8036 处理器，此处理器有 36 个核，L1 和 L2 两级 Cache 是独立的，L3 是共享的。使用的 TILE-Gx 和 Xeon 两种处理器的基本参数如表 5.11所示。

表 5.11 TILE-Gx 8036 与 Xeon X7550 参数对比图

| TILE-Gx 8036 | Xeon X7550 |
|---|---|
| 36 核<br>64bit<br>1.2GHz<br>26MB L3 cache<br>5 层独立 Mesh 互联网络<br>60 Tbps iMesh 带宽 | 8 核 16 线程<br>64bit<br>2.0GHz<br>18MB L3 cache |

本部分评估主要关注 TILE-gx 8036 与 Xeon X7550 在并行加速方面的性能。由于高通量应用都是处理大量的规模较小的耦合性较低的作业，所以，处理器的并行加速能力对作业处理能力具有决定性的影响。

数据处理类高通量 Benchmark，对不同线程数单位时间处理数据量以单线程为标准进行归一化，得到归一化单位时间处理数据量与线程数之间的关系。

Xeon X7550 的实验数据如表 5.12所示。

表 5.12 Xeon 处理器运行数据处理类各测试程序归一化吞吐率与线程数的关系

| 线程数 | Wordcount | Terasort | Kmeans | Grep |
|---|---|---|---|---|
| 1 | 1.000 | 1.000 | 1.000 | 1.000 |
| 2 | 1.781 | 1.651 | 1.891 | 1.891 |
| 4 | 3.112 | 2.642 | 3.732 | 3.700 |
| 8 | 4.621 | 3.755 | 6.466 | 6.510 |
| 16 | 7.138 | 4.365 | 9.377 | 11.288 |
| 32 | 8.302 | 3.350 | 10.747 | 14.405 |

得到图表如图 5.7所示。

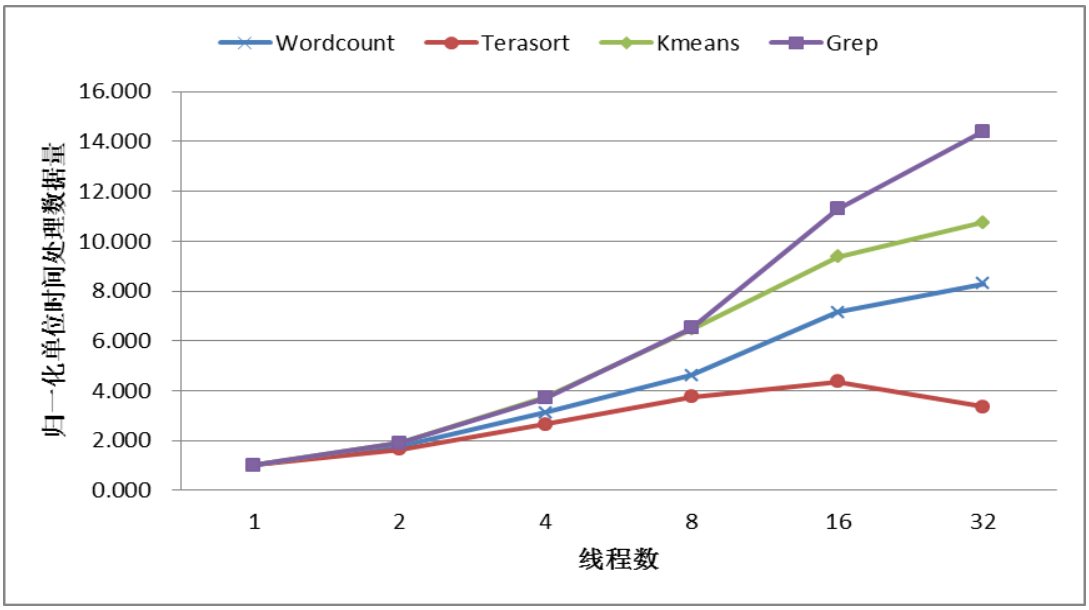



图 5.7 Xeon 处理器运行数据处理类各测试程序归一化吞吐率与线程数的关系

TILE-gx 8036 得到的实验数据如表 5.13所示。

表 5.13 TILE-Gx 处理器运行数据处理类各测试程序归一化吞吐率与线程数的关系

| 线程数 | Wordcount | Terasort | Kmeans | Grep |
|---|---|---|---|---|
| 1 | 1.000 | 1.000 | 1.000 | 1.000 |
| 2 | 2.268 | 1.905 | 1.936 | 2.042 |
| 4 | 4.112 | 3.231 | 3.221 | 3.933 |
| 8 | 6.889 | 4.377 | 6.942 | 7.468 |
| 16 | 10.341 | 5.578 | 13.203 | 13.654 |
| 32 | 13.003 | 5.773 | 22.683 | 22.366 |

得到图表如图 5.8所示。

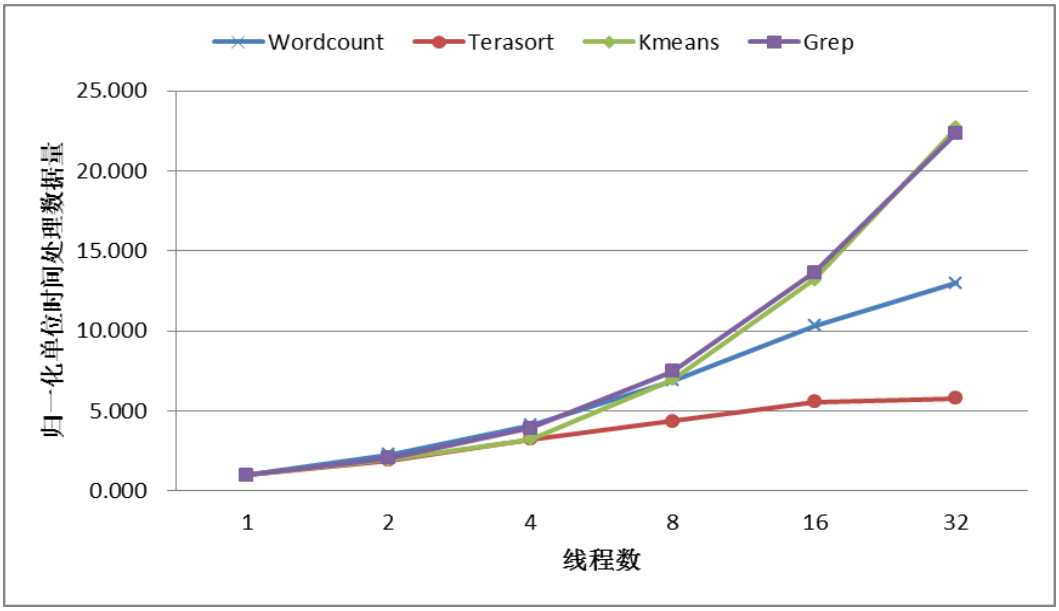



图 5.8 TILE-Gx 处理器运行数据处理类各测试程序归一化吞吐率与线程数的关系

比较实验结果可以看出，TILE-Gx 8036 处理器对数据处理类高通量应用有更好的并行加速比。这是由 TILE-Gx 系列处理器拥有更多的处理器核和更有效的核间互联网络以及更优的访存通路决定的。

对数据服务类高通量应用，实验测试单位时间内所能处理的请求数量，以单线程为基准做归一化，得到实验数据如表 5.14所示。

表 5.14 Search 在两种处理器平台并行加速比测试结果

| 线程数 | Xeon X7550 | TILE-gx 8036 |
|---|---|---|
| 1 | 1.000 | 1.000 |
| 2 | 1.914 | 2.008 |
| 4 | 3.590 | 4.013 |
| 8 | 6.373 | 7.915 |
| 16 | 9.706 | 15.755 |
| 20 | 12.125 | 19.511 |
| 24 | 12.693 | 22.204 |
| 32 | 13.468 | 22.370 |

得到图表如图 5.9所示。

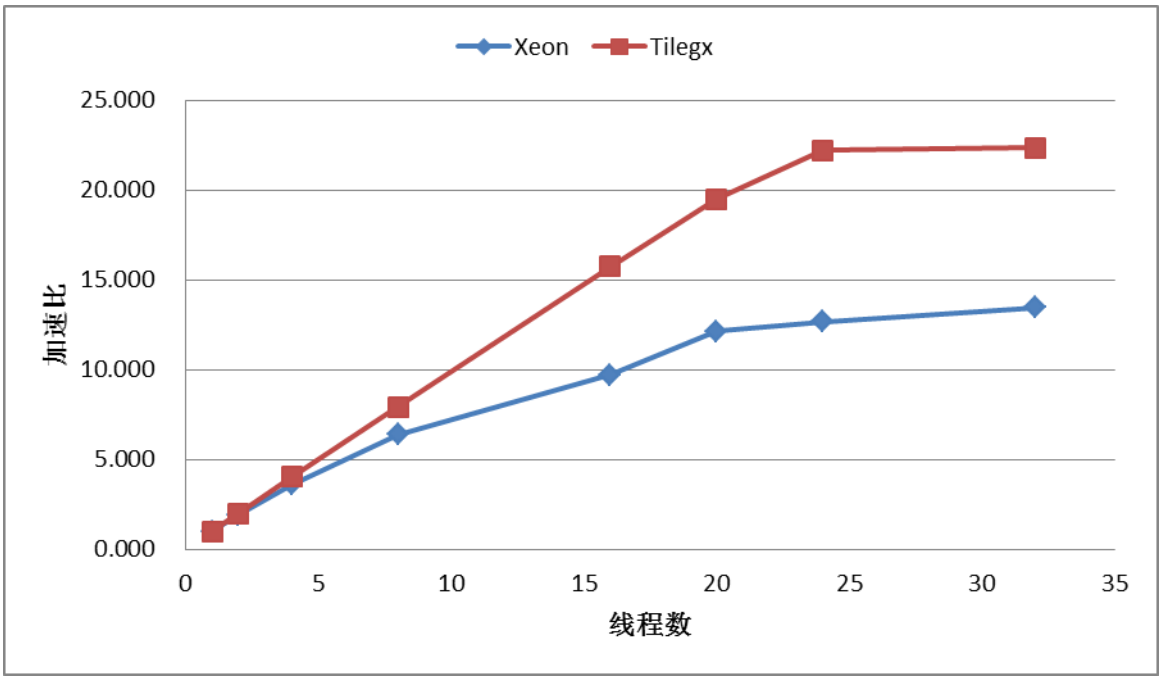



图 5.9 Search 在两种处理器平台并行加速比测试结果

通过实验数据，很明显可以看到，TILE-Gx 处理器有更高的并行加速能力。

实时交互类高通量应用所关注的高通量需求指标是，在保证对每个用户的服务质量的前提下，能够同时支持的在线用户数。

对于本文的实时交互类高通量应用的 Benchmark，RNC 用户面 Benchmark，保证服务质量是指用户数据包由于系统繁忙而导致的丢包率在一定的范围之内，本文取丢包率小于 5%为可接受的。

实验得到 RNC 用户面 Benchmark 在 Xeon 和 Tilegx 两种处理器上的实验结果如表 5.15所示，图表如图 5.10所示。

表 5.15 RNC 在两种处理器平台并行加速比测试结果

| 线程数 | Xeon 处理器 | Tilegx 处理器 |
|---|---|---|
| 1 | 5700 | 2600 |
| 2 | 7500 | 5100 |
| 4 | 9000 | 10120 |
| 8 | 10160 | 20080 |
| 12 | 13920 | 28800 |

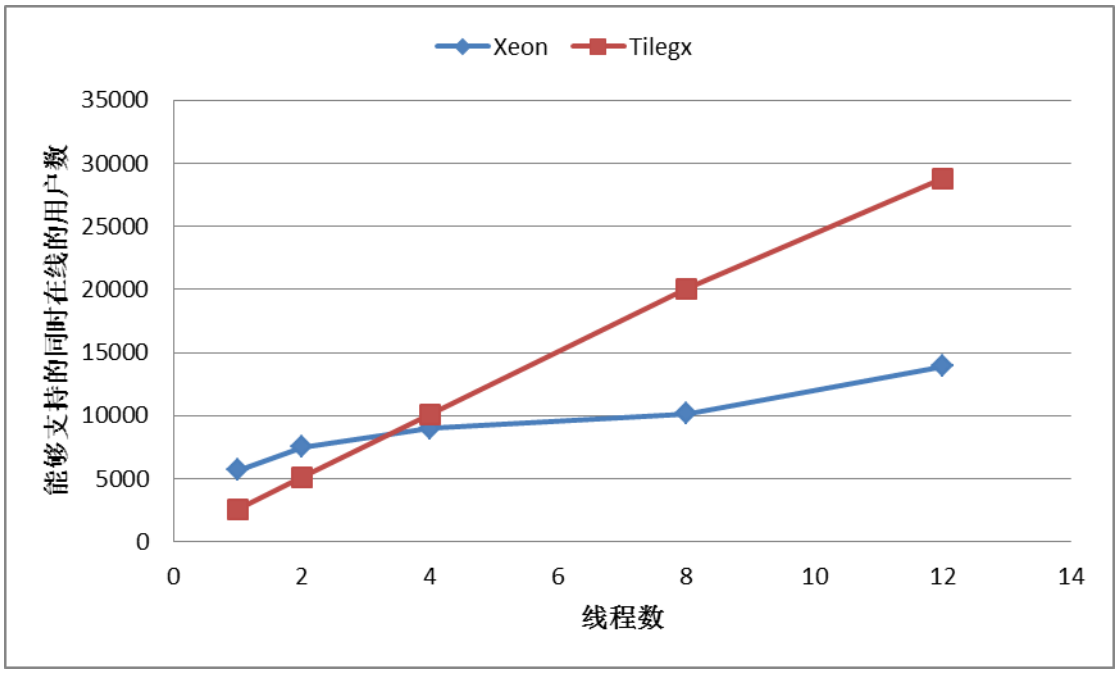



图 5.10 RNC 在两种处理器平台并行加速比测试结果

通过实验结果可以明显的看到，对于实时交互类高通量应用的 Benchmark，TILE-Gx 处理器同样具有更高的并行加速能力。

综上实验结果可以看出，对于高通量类应用的 Benchmark，Tilegx 处理器比 Xeon 处理器具有更好的并行加速比和并行处理能力。这与 Tilegx 的应用定位和具体结构设计是有关的。其结构设计更适应高通量类应用中大量作业、规模较小、耦合性较低的这种特点。

这也从另一个角度佐证了本文所抽取和实现的 Benchmark 是合理有效的。

## 5.3 本章小结

本章是实验部分。首先，设计实验从作业并发性、作业之间耦合性、访存需求、Cache 使用效率和访存宽度几个方面，对所实现的高通量 Benchmark 做了实验分析，从实验角度得到高通量 Benchmark 在这几方面的特性，与本文第三章中的分析做了相互印证，也

从实验的角度证明了本文所实现的 Benchmark 反应出了高通量应用的基本特征。

然后，本章使用本文实现的高通量 Benchmark 对 Tilera TILE-Gx 处理器和 Intel Xeon 处理器的并行加速能力进行了实验分析。Tilera TILE-Gx 作为一种具有特殊结构设计的众核处理器，设计目标就是应用于高通量类的应用，因而对本文实现的高通量 Benchmark 在并行加速方面应该有更好的支持。本部分通过实验验证了这一点，这也从另一个角度说明了本文所实现的高通量 Benchmark 在对高通量处理器测评方面的有效性。

# 第6章 总结与展望

随着众核/多核结构的高通量处理的研究的进一步发展，对面向高通量处理器的Benchmark的研究提出了新的挑战。现有的计算机领域的Benchmark不能适应于对高通量处理器的测评需求，这是由于各种Benchmark都有特定的目标系统和目标应用类型。本文提出了一套完整的面向高通量处理器的Benchmark研究的方法论，并实现了一个规模较小的面向高通量处理器的Benchmark集，达到了基本的测评需求。

## 6.1 本文工作总结

本文首先对数据中心高通量应用的发展现状和高通量处理器的研究现状做了简要介绍。高通量处理器是为了在处理器级别更好的支持当前数据中心高通量应用而发展起来的一个研究方向。而对高通量处理器的测评对面向高通量处理器的Benchmark的研究提出了很高的要求，这一类Benchmark的目标平台是高通量处理器，目标应用是高通量类应用。通过以上介绍，完成了对本文选题背景和意义的分析，提出了本文的主要贡献。

作为本文的相关研究，本文对当前已有的具有代表性的Benchmark做了充分的调研分析，包括传统高性能应用领域的Benchmark，如SPEC基准体系，PARSEC基准体系和HPCC基准体系等，大数据相关领域的Benchmark如HiBench、YCSB、LinkBench、BigDataBench和DCBench等，本文对这些有代表性的Benchmark各自的特点做了总结归纳，并从目标平台和目标应用两个角度分析了这些Benchmark不适用于高通量处理器的测评的原因。

本文通过调研分析，总结了数据中心的典型高通量应用，提出了一种基于高通量需求特点的分类模型，基于此分类模型对典型的高通量应用进行了分类。提取了每一种应用中的核心Workload，即关键算法和功能模块，通过分析这些Workload，得到了不同类型高通量应用所具有的程序特征。这些特征主要反映在作业的并发性、耦合性，访存需求，访存宽度，Cache使用效率等几方面。

针对高通量处理器这一目标测评平台，本文提出了一种基于线程的作业处理多节点并行模型思想。并针对三类不同的高通量应用提出了这种并行模型的具体实现方式。从三类不同的高通量应用中各选取了代表应用和Workload，作为实现Benchmark的应用和Workload。然后，介绍了Benchmark的详细实现方式。本文最终实现的Benchmark包含Big Data Analytics中的Wordcount、Terasort、Kmeans和Grep四个算法，Web Search中响应用户请求部分的全部Workload和Radio Network Controller中用户面部分的全部Workload。

实验部分通过实验分析了本文所实现的Benchmark反映出了高通量应用所具有的程序特点，说明了本文所实现的Benchmark的有效性。并使用本文的高通量Benchmark对TILE-Gx处理器和Xeon处理器进行了并行加速能力方面的测评，这也从另一个角度说明了本文所实现的Benchmark的有效性。

## 6.2 下一步研究方向

面向高通量处理器的 Benchmark 的研究方向还有较多的工作是需要做的，首先，本文实现的 Benchmark 选取了高通量应用中的几个代表性的应用和 Workload，这些能够很好的代表高通量应用的特征，而要非常完整的覆盖到高通量应用的需求，需要对尽量多的应用和 Workload 来实现 Benchmark。

其次，本文所实现的 Benchmark 主要是针对共享内存类高通量处理器的结构特点，而当前已有消息传递类的高通量处理器的相关研究，同样的应用和算法在针对这两种处理器的实现方面会有不同的需求。

最后，Benchmark 相关研究是一个不断发展的过程，因为随着互联网服务的发展，会不断的有新的应用产生，这些新产生的应用往往会有一些新的程序特点，对 Benchmark 的研究提出新的挑战。

# 参考文献

[1] 张引,陈敏,廖小飞.大数据应用的现状与展望[J].计算机研究与发展,2013,50:216-233.

[2] 曹维. 基于模拟的高性能体系结构评测技术研究. 国防科技大学工学硕士学位论文. 2008 年.

[3] http://prof.ict.ac.cn/DCBench/

[4] 吕超,戴晨,张为华.计算机体系结构基准测试程序集的研究[J].计算机应用与软件,2013,30(10):189-193.

[5] 徐洁,王华,吴晓华,王雁东. 浅析 SPEC 基准测试程序集及评价指标[J]. 实验科学与技术, 2010(6):21-24.

[6] 王晓英,李三立.HPCC:面向存储访问模型的基准测试[J].小型微型计算机系统,2006,27(5):950-955.

[7] PARSEC. http://parsec.cs.princeton.edu/.

[8] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang. The hibenchbenchmark suite: Characterization of the mapreducebaseddata analysis. In Data Engineering Workshops(ICDEW), 2010 IEEE 26th International Conference on,pages 41–51. IEEE, 2010.

[9] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, andR. Sears. Benchmarking cloud serving systems with ycsb.In Proceedings of the 1st ACM symposium on Cloud computing, SoCC '10, pages 143–154, 2010.

[10] T. G. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan.Linkbench: a database benchmark based onthe facebook social graph. 2013.

[11] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee,D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki,and B. Falsafi. Clearing the clouds: A study of emergingworkloads on modern hardware.Architectural Support for Programming Languages and Operating Systems, 2012.

[12] Bigdatabench. http://prof.ict.ac.cn/BigDataBench/.

[13] W. Gao, Y. Zhu, Z. Jia, C. Luo, L. Wang, J. Zhan, Y. He, S. Gong, X. Li, S. Zhang, and B. Qiu.Bigdatabench: a bigdata benchmark suite from web search engines. The Third Workshop on Architectures and Systems for Big Data (ASBD2013), in conjunction with ISCA 2013.

[14] JianfengZhan,LixinZhang,NinghuiSun,LeiWang,ZhenJia,andChunjie Luo. High Volume Throughput Computing: Identifying andCharacterizing Throughput Oriented Workloads in Data Centers.2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum.

[15] S. Chaudhry et al. High-performance throughput computing. Micro, IEEE, vol.25, no.3, pp. 32-45, May-June 2005.

[16] 熊文,喻之斌,须成忠.大数据基准测试程序包构建方法研究[J].集成技术,2014,3(4):1-9.
[17] 余长慧,潘和平. 商业智能及其核心技术[J]. 计算机应用研究,2002(9):14-16.
[18] Hadoop. http://hadoop.apache.org/.
[19] 赵天闻. 基于机器学习方法的人脸识别研究. 上海交通大学硕士学位论文. 2008.
[20] Alexa website. http://www.alexa.com/topsites/global.
[21] 3GPP TS 25.322 (v9.2.0, 2010). Radio Link Control (RLC) Protocol Specification [EB/OL]. http://www.3gpp.org. 2012.1.
[22] 李成华,张新访,金海,向文. MapRedcue:新型的分布式并行计算编程模型.
[23] C. Luo, J. Zhan, Z. Jia, L. Wang, G. Lu, L. Zhang, C. Xu and N. Sun. Cloudrank-d: benchmarking and ranking cloudcomputing systems for data processing applications. Frontiers of Computer Science, 6(4):347–362, 2012.
[24] 陈浩,王轶彤.基于阈值的社交网络影响力最大化算法[J].计算机研究与发展,2012,49(10):2181-2188.
[25] 徐泓,王京. WCDMA空中接口RLC层的AM传输方式关键技术的研究[J]. 高技术通讯, 2002, 12(05):22-24.
[26] 尹浩,林闯,文浩,陈洽佳,吴大鹏.大规模流媒体应用中关键技术研究[J].计算机学报,2008,31(5):755-774.
[27] 丁祥武,李清炳,乐嘉锦. 使用 MapReduce 构建列存储数据的索引[J]. 计算机应用与软件,2014(2):24-28.
[28] N. Karthikeyani Visalakshi, J. Suguna. K-Means Clustering using Max-min Distance Measure. North American Fuzzy Information Processing Society, 2009.
[29] Xapian. http://xapian.org/.
[30] S. Robertson, H. Zaragoza, M. Taylor. Simple BM25 extension to multiple weighted fields. ACM international conference on Information and knowledge management, 2004.
[31] J. Dongarra, P. Luszczek, A. Petitet. The LINPACK Benchmark: past, present and future[J]. Concurrency and computation: practice and experience, 2013, 15(9).
[32] C. Bienia, S. Kumar, K. Li. PARSEC vs. SPLASH-2: A Quantitative Comparison of Two Multithreaded Benchmark Suites on Chip-Multiprocessors. IEEE International Symposium on Workload Characterization, 2008.
[33] S. Zhu, Z. Xiao, H. Chen, R. Chen, W. Zhang, B. Zang. Evaluating SPLASH-2 Applications Using MapReduce. Advanced Parallel Processing Technologies, 2009.
[34] T. Fleig, O. Mattes, W. Karl. Evaluation of Adaptive Memory Management Techniques on the Tilera TILE-Gx Platform. Institute of Computer Science & Engineering, 2014.

# 致　谢

论文撰写即将结束，这也意味着我的研究生生活的结束。在计算所学习和生活的三年中，紧张而充实，非常有幸认识了很多优秀的人，也从他们身上学到了很多。在此对给予我帮助的老师、同学、朋友和家人表示衷心的感谢。

首先，非常感谢我的导师张志敏老师。感谢张老师给了我到计算所求学的机会，在计算所三年收获很多。张老师渊博的知识、 严谨的治学态度，认真负责孜孜不倦的工作精神以及丰富的工程经验都使我受益匪浅。张老师平易近人，有机会时会给我讲一些解决问题的方法以及做人的道理，让学到了很多方法和道理。在此衷心的感谢张老师。

感谢我的实验室主任范东睿老师。范老师为我们提供了良好的实验室环境，无论是在学习中还是工作中，范老师都是我们的良师益友。范老师温文尔雅，谈吐幽默，每次当我遇到问题的时候，范老师都会对我谆谆教诲，帮助我解决各种问题并以言传身教的方式告诉我人生的道路该怎么走。

感谢叶笑春老师、王达老师。叶老师是我们模拟器小组的组长，叶老师在模拟器研究领域有很深的造诣，在参与项目的过程中，叶老师给予我很多指导和帮助，让我能够顺利完成相关工作。王老师在我们研究生培养的过程中给予很大的帮助，尤其是在论文开题到撰写的过程中，给过我很多建设性的意见和建议，让我能顺利完成我的毕业论文。

感谢宋风龙老师。宋老师为人热情，无论工作还是生活中碰到了问题去问宋老师，宋老师都会及时给予帮助。在刚进实验室的一年之内，跟着宋老师参与项目，在宋老师的指导下，各方面的能力有了很大提高。

感谢唐士斌师兄、李文明师兄、申小伟师兄、尹力超师兄。师兄们都有很高的学术水平、广泛的知识面、很强解决问题的能力，在我求学的过程中，给予我各方面的关怀。

感谢乔雪笛师姐、常蕾师姐，她们两位为我留下了宝贵的求职的经验，在求职的过程中为我提供了极大的帮助，让我少走了很多弯路。

感谢张洋师兄和朱亚涛师兄。作为两位工作几年之后来读博的师兄，他们教给我很多走上社会之后工作和生活的经验，让我能够勇敢迎接接下来的生活。

感谢张梦雨同学、王国江同学、倪文显同学。作为同一届的同学，我们在课程学习、科研项目和平时生活中相互帮助相互支持共同成长，形成了深厚的友谊。

感谢实验室的师弟师妹们。感谢谭旭师弟多次为我们做答辩秘书，感谢马丽娜师妹积极的组织实验室的各种活动。

感谢我的父母和姐姐，亲人给我的爱是我面对一切困难最坚强的后盾，他们给了我不断向前的信心和挑战自我的勇气，同时，也是我激励自己，不断奋斗的源泉。

最后，感谢所有帮助过我的人，漫漫人生路，区区几十载，你们无私的帮助让我的人生倍感温暖。

# 作者简介

姓名：苗福涛　　性别：男　　出生日期：1989.07.26　　籍贯：山东烟台

2012.9 -- 2015.7　　中科院计算所计算机系统结构专业硕士生
2008.9 -- 2012.7　　东南大学电子科学与技术专业本科生

**【攻读硕士学位期间参加的科研项目】**

[1] 华为合作项目：处理器评估模型技术合作项目，2013 年 3 月~2014 年 6 月
[2] 国家“973”项目：面向高通量计算的可扩展、高效能并行微结构，2012 年 9 月~2013 年 3 月

**【攻读硕士学位期间的获奖情况】**

[1] 2014 年被评为中国科学院大学“三好学生”
[2] 2014 年被评为计算所体系结构国家重点实验室优秀学生
[3] 2013 年被评为海淀区“义工之星”