并行面向对象语言综述*

于 勇[1,2], 臧婉瑜[1,2], 谢 立[1,2], 过敏意[3]

[1](南京大学 计算机软件新技术国家重点实验室,江苏 南京 210093);

[2](南京大学 计算机科学与技术系,江苏 南京 210093);

[3](会津大学计算机理工学部 福导县会津若松市 日本)

E-mail: yumeng@nju.edu.cn; minyi@u-aizu.ac.jp

http://www.nju.edu.cn

摘要: 近年来,面向对象语言的并行化技术取得了许多重要进展.以 Mentat,CC++,pC++,HPC++和 ICC++等几个典型系统为代表,从并行执行模型、语言并行设施、优化技术和运行时支持等几个方面归纳、总结了近年来这些语言和编译器支持并行的新技术.同时,还给出了对这些技术的分析和评价,总结了需要进一步研究和解决的问题.

关键词: 并行计算;面向对象语言;对象分布

中图法分类号: TP311 **文献标识码**: A

并行技术和面向对象技术极大地促进了计算机应用和软件技术的发展.近些年来,在将这两种技术结合起来,即进行面向对象的编程并同时获得并行性的研究方面,进行了许多有益的探索,并取得了一些实质性的成果.

Mentat[1,2],pC++[3],CC++[4]和 HPC++[5]都是从 C++繁衍出来的显式并行面向对象语言.pC++侧重于数据和对象聚集(object aggregate)的并行,设计了与 HPF[6]类似的对象聚集的分布设施,提供了以 SPMD 模型为基础的执行模型.CC++侧重于描述语句间的并行关系,还设计了被许多其他系统所借鉴的全局指针.HPC++是高性能 C++协会在借鉴了 pC++,CC++,Javar[7]等许多语言和编译器的成功之处的基础上提出的一个规范.

在 Mentat 中特别定义了 Mentat 对象的语义,以对象为单位进行依赖关系的自动识别和并行性控制.Illinois 大学的 Concert 系统(ICC++)[8]也是一种显式面向对象语言,这一系统特别注重实现上的优化和粒度的控制,并且有些测试结果接近于手工进行底层编程的测试结果.

本文将在介绍这些并行面向对象语言各自特点的基础上对它们进行分析和比较.

1 并行模型

1.1 图模型

宏数据流(macro dataflow,简称 MDF)模型[9]是一种表示对象间作用的模型.在 MDF 中,顶

* 收稿日期: 2000-10-23; 修改日期: 2001-02-20

基金项目: 国家 863 高科技发展计划资助项目(863-306-ZT02-03-01);国家自然科学基金资助项目(69803005)

作者简介: 于勇(1972-),男,辽宁鞍山人,博士生,讲师,主要研究领域为分布式与并行计算,软件工程,系统安全;臧婉瑜(1972-),女,辽宁沈阳人,博士生,主要研究领域为计算机网络,分布式与并行计算,软件工程;谢立(1942-),男,江苏常熟人,教授,博士生导师,主要研究领域为分布式与并行计算;过敏意(1962-),男,江苏无锡人,博士,教授,主要研究领域为并行处理,并行化编译,软件工程.

点表示对象的方法调用,边表示调用间的依赖关系.但 MDF 没有反映出 OOL 的数据封装特性.

对象数据流图(object dataflow graph,简称 ODG)是我们在设计 JAPS-II 时提出的模型.ODG 是一种特殊的聚类任务图(clustered task graph),图中的顶点表示对象的方法调用,边表示系统中的数据流向或调用关系.与 MDF 的重要区别在于,ODG 是聚类任务图.ODG 的初始聚类是属于同一个对象的方法调用.这种聚类的依据在于面向对象设计技术的封装特性.这与高性能计算中数据操作的局部性要求具有良好的对应关系.我们认为,保证良好的封装是程序员的责任,而编译器负责将这种局部性映射到目标程序中,从而有利于实现较大粒度的并行性.

1.2 线程模型

pC++采用 SPMD 作为并行执行模型.pC++中绑定到处理器对象(请参阅本文第 2 节)的一组并行执行的线程用 TEClass 类说明.一个 TEClass 的成员函数由一个单独的线程顺序执行.pC++的线程机制使 pC++不能提供嵌套的并行性.

HPC++的线程模型有两种.一种是多线程共享存储器模型,即多个线程运行在相同的上下文环境中.线程集合与上下文可以绑定成为一个线程组,诸如归约和前缀等操作符可以应用于一个线程组进行同步协同操作.HPC++的另外一种线程执行模型是 SPMD 的,即 n 个程序拷贝运行于 n 个不同的上下文.与 pC++所不同的是,HPC++中每个独立的上下文还可以是多线程的,并且线程组的同步机制可以扩充到线程组集合.HPC++并不要求 SPMD 在所有计算结点是同构的:有些结点可以有多个上下文.HPC++的线程模型更易于实现多级并行性.

2 并行设施

2.1 并行程序块

在程序中,并行执行的程序块包括需要并行执行的多条语句和需要并行执行的循环迭代两种.并行程序块语句间的两种并行语义分别以 CC++和 ICC++为代表.

CC++中的并行程序块用 par{statements$_{opt}$}表示.并行块内的语句按照随机的公平交替的方式执行,在并行块中的全部语句执行完毕之后,控制流转入并行块的后继语句.

CC++中用 parfor(for-init-statement expression$_{opt}$;expression$_{opt}$)statement 来标识一个并行循环.循环的后继迭代可以在当前迭代执行完毕之后前开始运行,并行循环中的迭代是公平交替执行的.全部迭代执行完毕之后,parfor 的执行中止.CC++的并行语义要求程序员对循环控制变量和共享变量的使用要给予额外的考虑和关注.

ICC++用 conc 关键字来标识并行程序块和并行循环.与 CC++不同的是,ICC++中的并行块和并行循环中的迭代是由编译器来检查语句间的依赖关系,并根据依赖关系形成的偏序关系来构造运行时的并行程序结构.若两条语句不存在依赖关系,则编译器不保证两条语句的执行顺序.在 ICC++中,对于 conc 说明的循环结构,其语义规定为每个迭代都有独立的变量副本,给定变量的跨迭代的定义将被复制给后继迭代.这种语义规定了迭代之间的偏序关系,要求编译器生成的目标程序保留程序中的局部数据依赖关系.ICC++的这种语义规定要求编译器具有识别语句间依赖关系的能力,增加了编译器的设计难度.

HPC++采用与 Javar 相同的方式对循环进行并行化.程序员为需要并行化的循环标明 #pragma HPC_INDEPENDENT,由编译器为该循环生成多个并行执行的线程副本,每个线程完成部分循环计算.在循环中,程序员还可以用 #pragma HPC_REDUCE 指出归约变量,用

#pragma HPC INDEPENDENT,PRIVATE 指出可私有化的变量.HPC++的这种并行指示和目标并行程序结构具有良好的对应关系,与 ICC++相比,容易实现根据并行指示进行的并行化转换,并能获得充分的并行性.

2.2 Mentat 对象

Mentat 编程语言(Mentat program language,简称 MPL)对对象的语义进行了新的规定和扩充,以便支持对象的并行性.Mentat 中的对象分为 C++对象和 Mentat 对象两类.其中用 mentat 关键字定义的 Mentat 对象占据独立的、互不相交的存储空间.其成员函数调用都是传值调用,函数返回结果也是传值的.对象指针也可以作为参数和返回值传递,但实际传递的不是指针,而是复制的指针指向的对象.这种语义要求程序员为自己定义的可变大小的数据结构提供 sizeof()成员函数,以进行复制对象的传输.Mentat 对于每一个 Mentat 对象使用一个独立的线程控制,并在系统中使用一个惟一的名字标识.对于每个 Mentat 对象,每一时刻只提供一个成员函数的服务,这样由编译器为每个成员函数的调用生成一个监视器(monitor)进行并发控制.

Mentat 对象又分为 persistent mentat 和 regular mentat 对象.其中 persistent mentat 对象会保持对象调用间的状态,regular mentat 对象则不保持对象调用间的状态,其成员函数更像一个纯函数.这种语义规定使 regular mentat 对象的成员函数调用与已经有的调用序列无关,从而可以在需要时实例化,以获得系统所需要的并行性.

Mentat 对象的实例化也和 C++语义有所不同.在 C++中,当进入对象的作用范围之后,类说明后加上类变量的名字直接导致变量的实例化.在 Mentat 中,这种说明只导致在堆中分配一个未绑定(unbounded)的对象指针.Mentat 对象只有在使用时才真正实例化.这种语义为进一步开发并行性提供了可能.

尽管 Mentat 非常侧重于对象的自然的并行语义,尽量减少显式的描述并行性的语言设施,但是由于 Mentat 对象和大部分 OOL 的对象语义不同,一方面使编译器的设计实现复杂化,另一方面也增加了程序员的学习难度.

2.3 处理器对象和全局对象指针

CC++在类说明中用 global 来说明一个处理器对象(processor object)类.处理器对象的公共成员函数和成员数据的作用域为对象所在的处理器.处理器对象只能定义指针类型.

全局指针被 HPC++所借鉴.在 HPC++中,全局指针指向全局地址空间中的全局数据,这种全局引用相当于远程对象的一个代理.通过全局指针调用指针指向对象的成员函数需要事先的登记过程,并且只有定义了 pack(),unpack()例程的用户自定义对象才可以通过全局指针在不同的上下文之间进行拷贝.

pC++中的处理器对象的概念与上述不同,而与 HPF 中的处理器模板相似.pC++中用类似 Processor $P(m,n)$的语法说明处理器结构,定义多线程 SPMD 程序到各个处理器的绑定.另外,pC++还用下一节介绍的分布对象和对齐对象将对象聚积分布到处理器对象所定义的线程组上.pC++也提供了全局指针.

全局指针使全局对象与局部对象的存取语法和语义保持一致,对程序员屏蔽了物理存储位置的差别.从语言设计的角度来看,全局指针是一个必要的语言设施.另一方面,编译器最终会为非本地的引用生成通信代码,全局指针的效率还是很低的.

2.4 对象聚集

对象聚集(aggregate)是指需要并行执行相同操作的一组对象.我们认为,对象聚集是用于高性能计算的并行面向对象语言的一个必备的语言设施.

ICC++没有专门的关键字来进行对象聚集的处理,但却提供了进行对象聚集处理的编程范例.在该编程范例中,使用对象数组来表示对象聚集,用带有 conc 关键字的循环处理该对象数组,借此来完成较大规模的对象并行和并发操作.pC++中的 Collection 关键字采用和类相同的语法定义对象聚集,对象聚集中的每个对象叫做元素.Collection 元素成员函数的执行是以 SPMD 的方式由单独的线程来执行的.

Mentat 中没有提供对象聚集的支持,这是 Mentat 与其他并行面向对象语言比较时最严重的一个缺点,也成为用 Mentat 进行大规模数据处理的一个障碍.

2.5 对象分布

对象分布是指系统中的对象和可用处理器的映射关系.在 pC++中,用与 HPF 类似的方式支持对象聚集的规则分布.通过指定处理器集合、数据分布模板和分布模式,可以将一组对象聚积分布到不同的处理器上.

Mentat 中在对象实例化时选择对象的分布策略.若是系统自动实例化的,则由系统选择分布位置.否则,由程序员来指定对象的分配位置.这种对象分布方式只能一个对象一个对象地进行.

我们认为,pC++的对象聚集的分布方式是最实用也最为方便的一种对象分布方式,其他语言的逐个对象分布的方式使编程复杂化,也不利于数量比较多的对象的分布.

2.6 同步和互斥

· 单赋值同步变量.CC++中的 sync 说明符声明一个单赋值或同步对象.所有 sync 对象的建立状态为非初始化状态.一个 sync 对象一旦初始化之后,sync 对象不再可被作为左值被赋值.对未初始化的 sync 对象的引用将被阻塞到该对象被初始化之后才能完成.HPC++在提供了简单的单赋值同步变量的同时,还提供了同步双队列对象.这种对象将读对象的线程和写给对象的值分别排入两个不同的队列,第 i 个读线程将接收第 i 个写入同步对象的值,在对象赋值前的读操作将使读线程进入等待队列.

· 计数信号量.HPC++中的计数信号量提供了同步一组线程中止的方法.一个计数信号量被设置有一个上限并被初始化为 0 值.waitAndReset()将挂起当前的线程,直到计数信号量达到上限值再次重置为 0 值.而 incr()用来增加计数信号量的值.

· 协同操作和通信.HPC++线程组的协同操作包括以下 3 种形式:

(1) 基本同步;

(2) 归约计算;

(3) 广播和收集数据.

在 pC++中,TEClass 线程组也提供了线程组返回值的归约操作.用户可以对线程组的所有返回值选择进行求最小值、求最大值、求和、求积的归约运算.在 pC++中,进程间的通信除了提供和 HPC++相同的广播机制以外,还提供了 send 和 receive 两个函数.

· 对象级同步.ICC++中由编译器自动维护对象一级的一致性.尽管对象间的一致性不能保证,但组合的方法调用可以用来构造多对象同步结构.Mentat 中每个 Mentat 对象由单独的线程运行,每个函数调用也有编译器生成的监视器进行并发管理,所以,Mentat 中的数据一致性的粒度也

控制在以对象为基本单位.

• 函数级同步. CC++中的 atomic 和 Java 中的 synchronized 具有相同的语义. 同一个对象的不同 atomic 函数的执行是非交错的.

• 语句块的同步. HPC++用不同的互斥锁标识程序中的每个临界区. 在互斥锁为 l 的临界区的入口和出口分别用 l.lock()和 l.unlock()来保证同一时刻只有一个线程进入临界区.

一种语言应该提供足够多样化的同步和互斥结构,由程序员根据需要作出选择.

2.7 隐式粒度控制

在显式并行面向对象语言中,程序员一般要根据编程语言的语义,通过控制同步函数和程序块的大小,用同步机构来控制并行粒度. ICC++中的粒度是由编译器来动态分析控制的. ICC++编译器在编译时刻生成多个可执行的并行线程版本,由于线程的开销也不相同(在 ICC++中,堆线程有 1 种,栈线程有 3 种),编译器生成的目标代码在运行时刻根据当时的动态绑定以及对象分布情况选择具有最小执行开销的线程版本运行. 在 ICC++的编译器 Concert 中称为隐式粒度控制. 这种粒度控制使目标程序的规模迅速增大. 另外,动态测试和绑定也会损失一些执行效率.

2.8 函数调用

Mentat 中的对象成员函数调用都是非阻塞的. Mentat 规定函数值在引用时阻塞,返回值传送到需要的处理器. 这是因为经过对象分布之后,使用函数值的处理器可能不是调用函数的处理器,这种规定和实现减少了通信开销,也提高了函数调用间的并行性. 这种实现要求编译器在使用函数值之前插入接收数据的原语,动态地构造了调用之间的依赖关系,这也是 Mentat 运行系统的一个关键技术.

在 HPC++中,不同上下文之间的函数调用需要上下文标识和全局函数. 需要远程引用的函数要事先进行登记,使用者首先获取登记标识,再使用远程上下文标识和函数标识进行调用. 调用者随即获得控制权继续执行,返回值用单赋值同步变量保存,仅当调用者读该变量并且返回结果尚未返回时调用者才会阻塞.

2.9 全局名字空间

全局对象名字空间是指在系统中的对象都有惟一的标识,以便用该标识进行对象间的操作. 全局对象名字空间允许数据一致的存取,并将数据和任务的位置分配和函数说明分离开. 全局对象名字空间的管理技术不是很复杂. 例如,对于 Unix 机器,可以采用主机 IP 和 UDP 端口号等结合在一起作为标识. 在考虑对象迁移和容错时,实现起来就复杂一些. 例如,可以采用标识名和对象地址映射表来定位系统中的对象,修改系统中的映射表,这样就可以实现对象的迁移,该映射表的存储可以是集中的,也可以是分布的. 目前,还没有文献报道支持对象迁移的系统,在文献[2]中指出这是可能要做的工作.

pC++中的全局名字空间分配到 main()所在的处理器,其他线程不能修改全局名字空间的对象值.

3 优化技术

这一节主要介绍与并行化密切相关的面向对象系统的优化技术. 对于内嵌等非并行的面向对象语言也普遍采用的技术,则侧重讨论其在并行系统中的应用以及和并行相关的方面.

3.1 内 嵌

内嵌(inline)技术[8]主要应用于分布在相同处理器的对象之间.内嵌可以是对象内嵌或方法调用内嵌.对于以对象为单位用监视器维护对象一致性的语言来说,对象和方法调用的内嵌同时减少了调用开销和一致性维护开销.内嵌对象和调用的一致性可以通过被嵌入的对象的监视器来进行管理.对象内嵌对于使用全局对象名字空间和对象cache的系统则减少了存储管理和cache维护的开销.方法调用的内嵌也是为了解决面向对象语言中大量的小函数调用开销的主要方法.内嵌优化是所有面向对象语言都采用的优化技术,在分布式环境下,要求内嵌的对象或内嵌方法的所属对象分配在相同的处理器.

3.2 访问区域扩张

访问区域扩张(access region expansion)优化是ICC++中采用的优化技术.在分布式计算环境下,由于对象是否分配在相同的处理器上是由运行时刻确定的,并且由于面向对象语言的动态绑定特性,使得有些对象的方法调用只能在运行时刻确定,这种特性使完全在编译时内嵌变得不可能.为了减小函数调用开销,编译器在编译时刻生成内嵌和非内嵌两个版本,运行时再进行访问区域的测试,确定是否符合内嵌条件之后,再选择相应的版本运行.访问区域是指访问目标对象的程序段.这个访问区域测试作为两个不同版本的卫兵(guard),具有比较大的开销.为了减小运行时的测试开销,如果访问区域比较小,或访问区域测试位于循环中,则进行访问区域扩张,将小的访问区域合并,或将测试卫兵外提作为循环的前置结点而将原来的循环改造称为两个不同的循环版本.这种优化技术就是访问区域扩张优化.访问区域扩张优化是结合面向对象语言内嵌优化,同时考虑面向对象语言动态绑定特性和对象分布的一种优化技术.访问区域扩张增大了目标程序的规模,动态测试也会损失部分性能.

3.3 局部性优化

改善访问的局部性,可以减少远程访问和通信开销.这类优化主要包括下面两种:

动态指针对齐(dynamic pointer alignment)是循环剥离(strip-mining)和迭代平铺(tiling)技术的推广.通过在编译时编译器构造循环体和函数调用的线程形式的逻辑迭代,在运行时程序的并发结构使这些逻辑迭代可以被动态地记录,再根据运行时的数据访问信息,利用指针对齐来增加数据重用和减少通信延迟.动态指针对齐主要应用于基于指针的计算的循环通信优化.

视图缓冲(view caching)利用应用知识和数据访问语义来支持高效的运行时对象缓冲,从而减少维护全局对象一致性的通信和同步开销.视图缓冲设计了专门的低延迟的一致性维护协议.编译器根据应用知识推断系统中对象及其副本的全局状态信息,以减少在运行时获得这些信息的需求.视图缓冲将一致性操作分解为access_grant,access_revoke,data_transfer这3个部分,为这3个部分提供了不同的实现.根据应用信息确定的预定义集合,按照针对特定的访问模式优化结果,编译器选择这些不同实现的最佳组合构成最终的一致性协议代码.

4 运行时支持

4.1 MDF虚拟机

Mentat中的Mentat对象语义由MDF虚拟机支持.虚拟机负责运行时的数据依赖关系测试、MDF的构造、MDF的运行、调度、通信和同步.编译器需要与虚拟机配合,生成相应的通信代码和虚拟机交互,以便正确地执行程序.

虚拟机由逻辑网络联接在一起的主机构成,逻辑网络可以是总线(bus)、网格(mesh),也可以是超立方(hypercube)结构.虚拟机由实例化管理器(instantiation,*i_m*)和标记匹配部件(token matching unit,简称 TMU)构成.实例化管理器负责 Mentat 对象的调度,即决定在哪个处理器上实例化对象,并负责具体的实例化对象.TMU 负责 regular 对象的标记匹配和在需要时通过实例化管理器来实例化对象.

由于采用了虚拟机技术,Mentat 可以运行在多种操作系统平台上,并且可以很容易地移植到新的操作系统平台上.

4.2 自动存储管理

自动存储管理是指系统中无用单元的自动回收.尽管这已经是一些程序设计语言,如 Java 中的特性,但在分布式系统中的自动存储管理还要复杂一些.自动存储管理将程序员从存储管理的细节中解放出来,这使得并行处理复杂分布数据结构成为可能.目前,ICC++的编译器 Concert 支持这项特性.

5 总 结

关于对几种典型的并行面向对象语言和编译器实现面向对象并行化的特点的比较情况请见表 1.

Table 1 Comparison of some parallel object-oriented languages

表 1 几种并行面向对象语言的比较

Parallel support[1]	Mentat	pC++	CC++	HPC++	ICC++
Goal of design[2]	Object parallelism[3]	Data parallelism[4]	Task parallelism[5]	Heterogeneous parallelism[6]	Implement optimization[7]
Parallel model[8]	MDF	SPMD	MPMD	SPMD	MPMD
Parallel block[9]	—	TEClass	par,parfor	#pragma	Conc
Object aggregation[10]		Collection			
Object distribution[11]	Automatic or directive[12]	Same as HPF[13]	Automatic[14]	—	Automatic
Global pointer[15]	—	Support[16]	Support	—	
Synchronize facilities[17]	Single assignment variable, object level synchronization[18]	Thread cooperation[19]	Single assignment variable, function level synchronization[20]	Single assignment variable, count semaphore, thread cooperation, critical section[21]	Object level synchronization[22]
Remote invocatino[23]	MDF virtual machine[24]	Not named[25]	Not named	Nexus RMI or CORBA IDL[26]	Not named

①并行支持,②设计目标,③对象并行,④数据并行,⑤任务并行,⑥异构并行,⑦实现优化,⑧并行模型,⑨并行程序块,⑩对象聚集,⑪对象分布,⑫自动或指定,⑬同 HPF,⑭自动,⑮全局指针,⑯支持,⑰同步机构,⑱单赋值变量,对象级同步,⑲线程协同,⑳单赋值变量,函数级同步,㉑单赋值变量,计数信号量,线程协同,临界区,㉒对象级同步,㉓远程调用,㉔MDF 虚拟机,㉕未命名,㉖NexusRMI 或 CORBA IDL.

面向对象语言要在并行计算中广泛应用还有许多问题需要解决.我们认为这些问题主要集中在以下几个方面:

• 对于显式并行面向对象语言来说,什么样的并行设施是必需的,编译器和程序员所分担的并行性的描述和实现应进行怎样的折衷?

·针对分布式对象计算环境的目标程序的优化技术的开发.

·用于高性能计算的通用和专用并行类库的构造.

·同时提供对粗粒度和细粒度并行的支持.

·同时提供对 UMA 和 NUMA 的支持.

由于并行技术和面向对象技术给计算机技术带来了革命性的进展,我们相信,采用面向对象技术的并行计算是高性能计算发展的必然趋势之一.

References:

[1] Grimshaw, A. S. Easy-to-Use object-oriented parallel processing with Mentat. IEEE Computer, 1993,26(5):39~51.

[2] Grimshaw, A. S., Weissman, J. B., Strayer, W. T. Portable run-time support for dynamic object oriented parallel processing. ACM Transactions on Computer Systems, 1996,14(2):139~170.

[3] Gannon, D., Yang, S. X., Beckman, P. User guide for a portable parallel C++ programming system: pC++. Technical Report, Indiana University, 1994.

[4] Carlin, P., Chundy, M., Kesselman, C. The CC++ language definition. http://globus.isi.edu/cepp/.

[5] Diwan, S., Johnson, E., Gannon, D. HPC++ and the Europa call reification model. ACM Applied Computing Review, 1996,4(1):251~269.

[6] Koelbel, C., Loveman, D., Schreiber, R., *et al*. High Performance Fortran Handbook. Cambridge, MA: MIT Press, 1994.

[7] Bik, A. J. C., Gannon, D. B. Automatic exploiting implicit parallelism in Java. Concurrency, Practice and Experience, 1997,9(6):576~619.

[8] Chien, A., Dolby, J., Ganguly, B., *et al*. Supporting high level programming with high performance: the illinois concert system. In: Gottlieb, A., ed. Proceedings of the 2nd International Workshop on High Level Parallel Programming Models and Supportive Environments (Workshop at IPPS'97). 1997. 147~162.

[9] Grimshaw, A. S. The Mentat computation model-data-driven support for dynamic object-oriented parallel processing. Technical Report, University of Virginia, Charlottesville, 1993.

A Survey of Parallel Object-Oriented Language*

YU Meng[1,2], ZANG Wan-yu[1,2], XIE Li[1,2], GUO Min-yi[3]

[1](*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210093, *China*);

[2](*Department of Computer Science and Technology, Nanjing University, Nanjing* 210093, *China*);

[3](*School of Computer Science and Engineering, Aizu-Wakamatsu City, Fukushima, Japan*)

E-mail: yumeng@nju.edu.cn; minyi@u-aizu.ac.jp

http://www.nju.edu.cn

Abstract: Techniques of parallelizing object-oriented language have had great progress in recent years. The authors introduce these new techniques in the aspects of parallel execution model, parallel facilities, optimization and runtime support mainly based on Mentat, CC++, pC++, HPC++, ICC++. Analysis and evaluation of these techniques are described and the problems that should be resolved in the future are pointed out.

Key words: parallel computing; object-oriented languages; object distribution

* Received October 23, 2000; accepted February 20, 2001

Supported by the National High Technology Development Program of China under Grant No. 863-306-ZT02-03-01; the National Natural Science Foundation of China under Grant No. 69803005