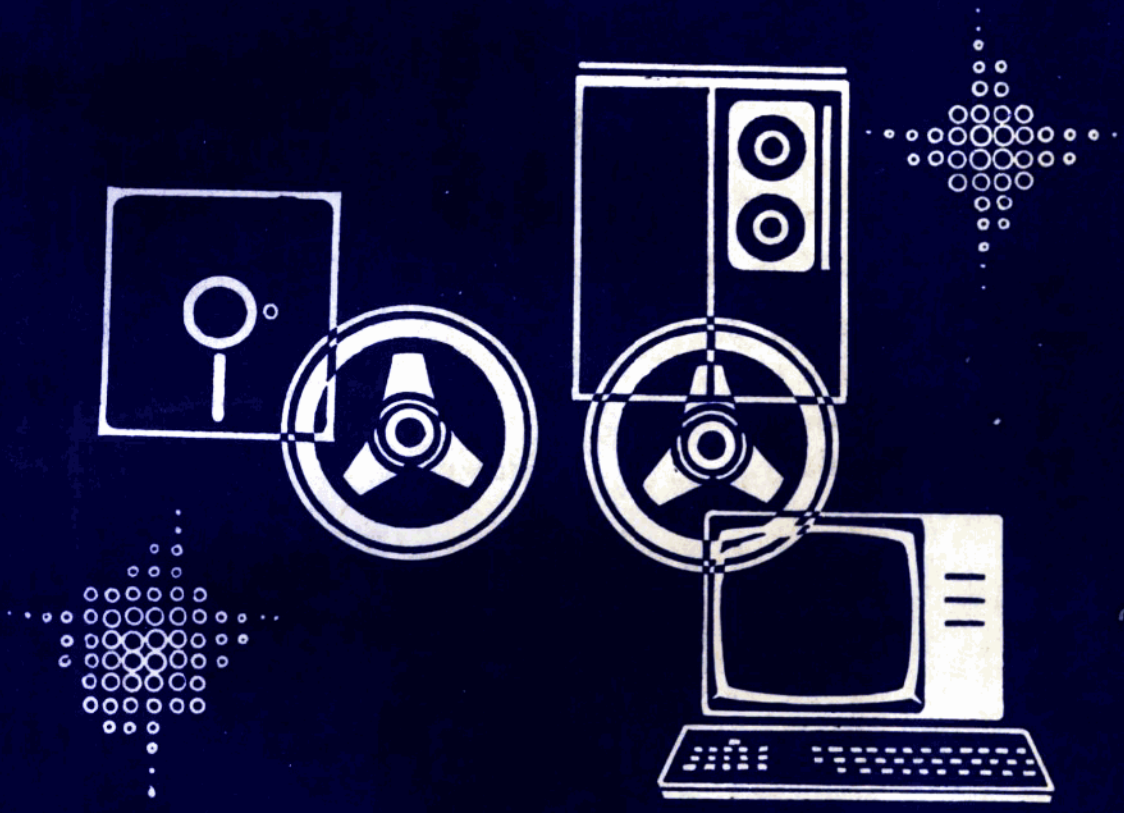
The First International Conference on

COMPUTERS AND APPLICATIONS

第一届国际计算机及其应用会议

论文集



本书承蒙下列单位资助
特致衷心感谢

四川电子科学技术开发公司

（地址：成都市桂王桥西街四川省电子学会）

北京市京海计算机技术开发公司

（地址：北京市海淀区）

华南计算机公司

（地址：广州市人民中路362号）

四川省蒙特现代技术开发公司

（地址：成都市人民南路）

广东省电子技术研究所

（地址：广州市）

上海计算机房技术开发公司

（地址：上海市）

中南电脑厂

（地址：广东南海新城区）

前　言

来自全世界的代表参加的第一次国际计算机及其应用会议将作为一个论坛讨论计算机时代新的、激动人心的重要进展。在一个具有如此长久而古老历史的城市—北京—召开我们这个如此现代题目的第一次大会看来是最合适不过的了。我们希望这次少有的将文化和技术联系在一起的机会将证明对每个参加者都是有价值的。

正如标题一样明显，这次会议的范围很广泛。今年程序委员会想将以下领域中前沿的研究人员聚集在一起以突出某些重要的题目。

- 应用
- 计算机制图和图象处理
- 计算机设计及子系统
- 局部网络
- 办公室信息系统
- 并行处理
- 软件及方法学/性能估价

关于这些题目的文章将安排在三个会场同时宣读，将尽可能使这些安排少发生冲突。我们希望这种安排有可能使每个参加者都能出席他所感兴趣的每个会场。

最后，程序委员会和审稿人安排了我们认为是高质量的技术程序，我们谨致公开谢意。我们还特别要感谢Wu Jikang先生和Feng Tse—yun教授答应满足这样一个国际会议所出现的所有要求。

E·A·Parrish, Jr.

Jiang Shifei

目　　录

Ada任务设施在多处理机体系结构中的效率问题*)

P.Laface（意大利吐灵工业大学自动化与信息系）

S.Rivoira（意大利吐灵大学信息系）

译 校 饶生忠

摘要：本文考察Ada任务设施在单处理机和多处理机系统中实现有关的效率问题。讨论并比较了Ada的基本通讯机制——会合概念的三种不同实现。在三种情况下的实现途径都是相同的：它在于将Ada并行构造变换成对核心原语的过程调用。描述了相应于最复杂实现的核心原语。

引 言

程序设计语言Ada[1]含有用于并发和实时程序设计的强有力的任务设施，这种设施可以使程序员集中精力于并行系统的设计，而可以不管内部任务的同步和通信细节。

会合（RV）机制的优点是，它统一了不同的多处理构造的语义，为并行系统的研制提供了一个很一般的精致的表示手段。但是，这些高级设施的实现由于其通用性，却给Ada编译程序和运行时支撑的设计者引起了大问题[2，3]。主要问题之一是如何避免许多同步情形中的过量调度的交互，这些同步情形在实时应用中常常发生的[4]。

大量研究与实验工作花在，提出RV的实现策略[5，6，7，8，9]，比较Ada任务设施和其它语言的设施[10，11]，探索在约束条件下RV的不同实现在给定机器上的性能[12]。

遗憾的是，要比较不同作者的解法和经验往往是困难的，甚至是不可能的，因为它们基于不同的假设或者是对不同体系结构得到的。

本工作的最终目标是试验Ada并行设施在单处理机和多处理机体系结构上实现的效率，并且将其性能和不同的通信和同步机制的性能加以比较。本文我们描述这一研究的基本想法并介绍它的现状。用作基准的模块体系结构是由意大利国家科学研究会研究的并提出作为连续的和不连续的工业过程实时控制的国家标准[13]。这种体系结构被组织成一个基于16位微处理机的紧耦合的多处理机系统，它目前支撑一个分布操作系统内

*）本工作得到意大利国家科学研究委员会支助并在吐灵数字信号接收中心进行的。

核（MODOSK），后者扩充程序语言Pascal以进程间的同步、消息传送和短时调度原语[14]。

考虑这种体系结构的Ada任务设施的三种不同实现策略。实现是从Ada原理[5]和Ada多任务构造的一个形式模型推得的[15]，这个形式模型是作为该语言整个形式描述的部分提出的。

Ada内核被设计成一组在每个处理机专用存贮器中完全重复的原语。原语决定着运行于处理机池上的任务的并发执行，并且它们允许任务之间交互而与其物理分配无关。

RV的不同实现用顺序Pascal语言编码，例外的只是一些最低级的功能，是用汇编语言编码的。Ada任务设施由一组核心原语支撑，这些原语可以作为过程由Ada任务调用也可以作为Pascal过程来实现。

使用高级语言可以减少研制时间和生产更加可读和更可移植的代码。利用Pascal在这个情况下还有这样的目的，就是比较Ada并发构造的效率和同样环境下MODOSK提供的同步机制的效率。为此，MODOSK和Ada核心原语利用类似的数据结构和就绪排队、时间调度排队和任务描述符管理的类似实现。

随后几节将描述RV机制不同实现和核心结构的主要特色。最后将更细致的描述"到达次序"解法的语义并将提出和讨论实现它的核心原语。

RV机制的各种实现

因为效率是实时过程控制应用主要关心的问题，这种应用要求多处理机系统的计算能力，所以，Ada的并行控制特色必须细心加以实现，以便使系统开销和任务停止时间达到最小。

由Ada原理[5]提出的并在本文中称之为"服务者"RV的这种RV实现中，已调用任务执行接受体（accept body）之前，正在调用的任务一直保持挂起。值得注意的是，为了完成一次RV，在项目调用先于接受语句的执行的情况下，调用调度程序（也可能产生上下文转换）二次，而在其它情况下则调用三次。如果交互任务运行于不同的处理机上，那么各自需要一个或两个处理机间的中断信号和两个或四个调度操作。这种情况示于图1a，其中调度点和处理机间的中断分别用黑圆点和双箭头表示。接受体有一个付本就够了并且它可以存贮在运行接受任务的处理机专用存贮器内。参数传送可以通过共享存贮器来进行。

第二条途径"过程调用"RV[7，8]叙述，接受体总是由调用任务来执行。这种情形的任务交互示于图1b，这种实现的优点是，参数传送不需特殊机制，因为调用程序在其控制线下执行接受体。缺点是接受体必须能访问调用任务。获得可访问性有两条途径：或者重复在运行调用任务的每个处理机专用存贮器上的接受体代码，或者将代码存贮在共享存贮器中。

接受体有可能涉及较大环境中说明的全局变量，因此运行于不同处理机上的不同任务必须能引用这些变量，从而这些变量也必须存贮在共享存贮器中。

单总线体系结构严重影响效率，而在多总线或多口存贮器体系结构中却能较好地保

持效率。

在接受体要求的某些资源只是在一个具体处理机上可用时，上述解法变得低效或不可能。

减少一次RV期间调度点个数的解法在于，将接受体作为连系RV的最新任务控制线部分来执行。这个途径可以称之为“到达次序”实现，它是几个作者提出的[6，7，9]。不论对单处理机体系结构（其中调度点减到二个）还是对紧耦合的多处理机体系结构（其中只需要处理机间的一个中断信号和两个调度操作来完成一次RV），它都是有效的（图1c）。这种解法和过程调用实现法由于以下因素都会引起复杂情况：全局变量引用，嵌套RV，故障和终止语句以及一次RV期间正在调用的任务和已调用的任务所引起的异常传播。

这时有一个未解决的问题，就是由于很少的上下文转换带来的性能增益能否补偿由于这种复杂情况带来的开销。

我们工作的目标之一就是想要回答这个问题。

Ada核心的结构

如前所述，实验分析是在基于16位微处理机（Zilog，Z8001）的紧耦合的多处理机体系结构上进行的。

系统有一个多总线互连型式，它有三级总线：专用总线、共享总线和全局总线[13]。连到专用总线上的物理资源只能由占有该总线的处理机访问；连到全局总线上的资源能由系统各个处理机访问；连到共享总线上的资源，有一个处理机可以直接访问（没有竞争），而所有其它处理机只能间接访问（通过全局总线）。

要分配全局总线的任意机制，而总线竞争是根据单一事务来解决的。

要锁住对公共变量的操作，要求有不可分的改读指令，这可以通过禁止一个或多个事务访问全局总线来获得。

每个处理机可以送中断信号到其它任一处理机；更多的要求可以同时送到同一个处理机。

下面我们设想，由一个处理机可执行的指令码驻留在该处理机专用存贮器中而某些数据结构放置在共享存贮器中，分配在不同处理机上的任务可能涉及这些数据。任务描述符分为两部分：局部描述符和全局描述符。

局部描述符（LTD）驻留在一个处理机的专用存贮器中并且它包含该任务与驻留同一处理机其它任务并发执行所需要的全部信息。全局描述符（GTD）驻留在直接连到占有该任务的处理机的共享存贮器中，并且它包含分配在不同处理机上的任务之间交互的信息。核心就是由一组在每个处理机专用存储器中完全重复的原语组成。

当一个任务调用核心与另一任务交互时，所调用的原语（由占有调用任务的处理机执行）检查已调用任务的全局描述符中的处理机标识符。如果已调用任务和正在调用的任务驻留在同一处理机，那么关于它的状态的全部信息在局部描述符中可以得到并且可以进行所要求的操作。如果已调用任务驻留在不同处理机上，那么核心原语把中断要

求，连同所要求操作的代码和已调用任务与正调用任务的引用一起送到那个处理机。驻留在中断处理机上的中断服务过程于是将调用相应于所要求操作的核心原语，以便完成任务交互。

值得注意的是，逻辑资源巧妙分配在适当的总线级别上可以大大改进系统的性能。对RV的过程调用和到达次序的实现尤其如此，因为它们要求，一个环境中说明的某些变量可以由运行于不同处理机上的任务访问。

局部任务描述符（LTD）有以下内容：

——涉及任务软硬方面的状态工作S；

——可以设想有下列值的状态字段：

RUNNING：虚拟处理机（VP）有处理机控制；

READY：VP根据其优先权已有使用处理机的资格；

BLOCKED：VP不竞争处理机的使用，因为要么调度于较后的时间，要么在等待一个同伴任务引起的事件；

——任务优先权；

——调度时间，也就是由延迟语句所置的时间值；

——与另一LTD的连接；

——与同一任务的GTD连接。

全局任务描述符（GTD）包含以下信息：

——处理机号；

——锁存变量；

——状态字段，当相应的LTD状态是RUNNING或READY时它取ACTIVE值，而当LTD状态是BLOCKED时，它取下列之一值：

ENTRY CALLING：由于项目调用而停止任务并且等待另一任务执行相应的接受语句；

ACCEPTING / SELECTING：由于接受（选择）语句的执行而停止任务并且正等待一个相应的项目调用；

ENGAGED：由于同伴任务竞争RV（亦即在执行接受体）而停止任务。

TERMINATED：终止任务而且它释放其所有资源。

——包含一个停止项目调用之后的实参地址的记录的指针（PA）；

——对应于未解决方案的项目的一组地址（OA）；

——一组排队（QUEUES），任务中说明的每个项目都有一排队，它包含执行相应的接受语句之前完成项目调用的任务标识符；

——包含一次RV涉及的任务和由LTD描述的任务的标识符的堆栈（PARTNERS），它可以处理嵌套的接受体；

——用于计时的项目调用或选择等待语句的延迟部分的地址（DPR）；

——与同一任务的LTD连接。

“到达次序”的实现

RV机制的服务者、过程调用和到达次序实现法都有将Ada构造变成核心原语的过程调用（前面的文章[16]描述的）的变换也有任务描述符的结构。

它们只是在由某些核心原语完成的一些操作上是有区别的。由于到达次序实现比其它实现复杂，而且在一定程度上它包括了其它实现，所以下面更详细的描述它。为了更好理解嵌套的RV涉及的任务的状态转换，讨论下面的例子。

设有三个任务T_1，T_2，T_3，结构如图2所画。不同状态转换由一次RV涉及的任务完成，决定于其到达次序。

这种考虑使我们可以规定两种不同的情形：

a）完成相应的项目调用（E）前到达接受语句（A）；

b）完成相应的项目调用后到达接受语句。

于是在出现两个嵌套的接受体时产生四个不同的情形，如下面的树所示

情形1表示接受语句和项目调用的所有可能的时间序列，其中A_1先于E_1和E_2先于A_2。

从树可以类似的推导其它情形。

对于项目调用和接受序列的每种相关情形，图3示出了图2描绘的任务状态转换。

在此图中B_1和B_2分别表示嵌套接受语句的外体和内体。

时间图表明，每个体作为连接RV的最新任务控制线部分运行的，并且在各种情况下执行内体的任务决不停止。

下面描述由实现到达次序的RV的核心原语完成的主要操作（见代码级附录的细节）。为了简单起见，任务和它对任务终止的影响之间的祖先关系以及异常处理均未作考虑。

三个变量（CALLER，CALLED，RUNID）将用来访问一次RV涉及的任务：

——CALLER　包含完成项目调用任务的标识符；

——CALLED　包含占有项目的任务的标识符；

——RUNID　包含运行任务的标识符。

ACCEPT原语（它实现接受语句的语义）首先检查CALLED的QUEUES字段中的任务。在相应于项目的排队是空的情形，CALLED和RUNID状态都被置到ACCEPTING并将项目名插入CALLED的OA字段中；下一个要执行的任务此时就从就绪列表中去选取。如果相应于CALLED的QUEUES字段中的项目的排队不空，就抽出第一个任务（CALLER）并将其状态置到ENGAGED；然后调用核心过程

BEGIN-RENDEZVOUS，并把CALLER和RUNID作为实参。

BEGIN-RENDEZVOUS过程涉及形参CALLER和CALLED：它首先保存CALLED的PARTNERS字段中的CALLER标识符，然后它调用实现接受体的过程。

接受体执行后，控制返回到完成RV的BEGIN-RENDEZVOUS。RV完成的部分使用CALLED-PARTNERS堆栈中所含的信息，以验明RUNID的必须恢复到READY状态的同伴。抽取堆栈顶的任务标识符并赋给LCALLER变量。因为接受体刚好由RUNID执行，所以如果LCALLER任务与RUNID不同，那么它就是RUNID要恢复的同伴。否则若检索的LCALLER任务是RUNID，那么出现三种情况：

1) CALLED.PARTNERS堆栈是空的，亦即出现一个非嵌套的接受语句。因为LCALLER执行接受体，所以必须恢复CALLED。

2) CALLED.PARTNERS堆栈不空，亦即恰由RUNID执行的接受体嵌套在另一接受体内；前面的RV由识别为PREVIOUS-CALLER的任务所激发。

2a) 如果PREVIOUS-CALLER状态是ENGAGED，则在它完成项目调用后CALLED达到它的接受语句。在此情况下PREVIOUS-CALLER将由CALLED恢复，而CALLED必须由RUNID恢复。

2b) 如果PREVIOUS-CALLER状态是ACCEPTING或SELECTING，则在它完成项目调用前CALLED达到它的接受或选择语句。从而CALLED将由PREVIOUS.CALLER恢复，而RUNID必须恢复PREVIOUS-CALLER。

原语EC实现项目调用语句。如果CALLED状态是ACCEPTING或SELECTTING而且项目名是在CALLED的OA字段中，则CALLED状态被置到ENGAGED而且调用过程BEGIN-RENDEZVOUS，其中RUNID和CALLED作为实参，否则RUNID状态被置到ENTRY CALLING，项目实参地址的指针被保持在RUNID的PA字段中，RUNID标识符被插入到与CALLED的QUEUES字段中的项目各有关的排队中，而下一个要执行的任务就从就绪列表中选取。

SELECT原语（它实现条件项目调用的语义）和EC的差别仅仅在RV不是立即可能发生时，在这一情况下调用过程ELSE-PART。

计时项目调用由过程SELECTD实现：完成RV当且仅当在固定时间间隔T内执行相应的接受语句，否则就执行延迟部分。计时项目调用的语义是由过程SCHEDULE和系统时间调度机制的组合效果获得的，后一机制是由过程PACTIVATE实现的。

过程SCHEDULE将RUNID的局部描述符插入调度表SL中，SL是按上升调度时间排序的。

每当实时时钟走动，就调用过程PACTIVATE：它控制SL表中第一个LTD的调度时间。如果它是零，任务描述符就从SL表中去掉而且不执行延迟部分。如果时间间隔T已经过去，也就是该任务描述符是SL表中的第一个描述符，那么任务就从ENTRY CALLING移到READY状态而且它的起始地址被置到延迟部分的地址。

选择等待是最复杂的构造而且它由过程SELECTW实现，如果没有能立即选择的

方案，那么过程SELECTW的行为就决定于参数SEL-MODE．在SEL-MODE是else（否则）方式的情形，就执行ELSE-PART．如果SEL-MODE是accept（接受）方式，亦即没有ELSE-PART，那么在能选择一个未解决的方案之前任务必须一直等待。和ACCEPT过程类似，RUNID和CALLED状态都被置到SELECTING．如果SEL-MODE是delay（延迟）方式，则SELECTW就象过程SELECTD那样起作用：执行DELAY PART之前必然过去的时间间隔被置到未解决的延迟方案的最小说明间隔．最后，如果SEL-MODE是tern，那么在RUNID和CALLED状态都置到SELECTING之后可以试验任务的终止，因此若不发生终止，就可以完成RV．

结　　语

本文提出了在多处理机体系结构上Ada任务构造的变换的一般和灵活的方案．主要从效率的观点讨论了RV概念的三种不同实现，较详细地描述了“到达次序”解法，由于允许接受体作为连接RV的最新任务控制线部分来运行，这种解法减少了上下文转换的次数。

在最近的将来，Ada任务通信的三种解法的时间性能将根据实验和不同的通信和同步机制（管程、邮箱、信号量）的性能加以比较．某些连系着并发进程的经典通信和调度问题通过利用不同机制将得以解决，并且将在我们的多处理机系统上测量和比较它们的时间性能。

附　　录

本附录以类似PASCAL的形式提出了实现到达次序RV的主要核心原语．

为了使表示简单，一些简单操作的实现，连同机器有关的功能以及其它一些细节，如使用LOCK和UNLOCK原语访问互斥公共变量，都加以省略。

```
procedure BECIN_RENDEZVOUS (CALLER,CALLED: task_id;
                                    EN: entry_n; PA: address);
  var  LCALLER, PREVIOUS_CALLER:  task_id;
    begin
      PUSHT (CALLER,CALLED↑.PARTNERS);
            (* CALLER in the *)
            (* CALLED↑.PARTNERS stack *)
      BODY (EN, PA); (* execute the accept body *)
      LCALLER: =POPT (CALLED↑.PARTNERS);
                        (* Retrieve the *)
```

```
                        (* CALLER identifier *)
    if RUNID=LCALLER
    then
      begin
        PREVIOUS_CALLER; =TOPT(CALLED.PARTNERS);
        if PREVIOUS_CALLER=NIL
        then RESUME (CALLED)
        else case PREVIOUS_CALLER .STATE of
                  SELECTING,ACCEPTING;
                   RESUME(PREVIOUS_CALLER);
                  ENGAGED; RESUME (CALLED)
             end
      end
    else RESUME (LCALLER)
  end;

procedure ACCEPT (CALLED; task_id; EN; entry_n);
  var CALLER; task_id;
  begin
    GET (EN, CALLER, CALLED↑.QUEUES);
    if CALLER=NIL
    then
      begin     (* accept precedes entry call *)
        CALLED↑.State; =ACCEPTING;
        RUNID↑.State; =ACCEPTING;
        CALLED↑.OA; =CALLED↑.OAU{EN};
        SELECT_NEXT_TASK
      end
    else begin (* entry call precedes accept *)
          CALLER↑.State; =ENGAGED;
          CALLER↑.Start; = 0;
          BEGIN_RENDEZVOUS (CALLER,RUNID,EN,
                                CALLER↑.PA)
          end
  end;

procedure EC (CALLED;  task_id; EN; entry_n; PA; address);
  begin
    if CALLED↑.State=TERMINATED then Exception_handler;
```

```
    with CALLED↑do
      begin
        if (((State=ACCEPTING) or
            (State=SELECTING)) and (EN∈OA))
        then
          begin
            (* accept precedes entry calling *)
            State; =ENGAGED;
            START; =0;
            BEGIN_RENDEZVOUS(RUNID,CALLED,EN,PA)
          end
        else
          begin   (* entry call precedes accept *)
            RUNID↑.State; =ENTRY CALLING;
            RUNID↑.PA ; =PA;
            PUT (EN, RUNID, CALLED↑.QUEUES);
            SELECT_NEXT_TASK
          end
      end
  end;
procedure SELECTC (CALLED; task_id; EN; entry_n;
                   PA; address; EP; procedure;
                   ELSE_PART; procedure) ;
  begin
    if CALLED↑.State=TERMINATED then
                      Exception_handler;
    with CALLED↑do
      begin
        if  (((State=ACCEPTING) or
             (State=SELECTING)) and (EN∈OA))
        then
          begin
            State; =ENGAGED;
            START; =0;
            BEGIN_RENDEZVOUS (RUNID,CALLED,EN,PA);
            BODY (EP, NIL)
          end
```