# 任务间的次并行性*

杜建成 徐 融 陈道蓄 谢 立

(南京大学计算机科学与技术系 南京 210093)
(南京大学计算机软件新技术国家重点实验室 南京 210093)

**摘要** 首先给出了任务间次并行性存在的条件,讨论了两个任务之间的通讯、通讯等待开销的计算和任务间次并行性发掘的一般过程.此外,还就代码移动和任务合并对增强并行性、消减不必要的通讯等待开销的影响作了说明.

**关键词** 数据并行性,任务并行性,数据依赖,控制依赖,层次任务图.
**中图法分类号** TP316

在程序自动并行化研究中,数据并行化已被广泛和深入地研究,而对于跨越循环和过程的任务级并行化的工作,相对来说研究得还不够.我们在开发JAVA程序自动并行化工具的过程中发现,很多存在依赖关系的任务依然具有并行执行的价值,因此提出次并行性的概念,并通过分析并行和通讯的关系,给出了次并行性的识别和强化方法.

## 1 任务间的次并行性

### 1.1 基本概念

$u,v$ 并行执行是指 $u,v$ 独占不同的处理节点,在同一时刻启动.$u,v$ 并行执行时间由 $u,v$ 中最后一个结束执行的任务确定.$u,v$ 串行执行是指 $v$ 在 $u$ 结束执行之后开始执行.$u,v$ 串行执行时间是指串行执行时完成任务的总时间,不包括 $u,v$ 之间可能的通讯开销.两个任务 $u,v$ 间若无依赖关系(如不特别指明,本文所指依赖均指数据依赖),则 $u,v$ 可以并行执行,我们称 $u,v$ 之间存在主并行性.若 $u,v$ 间存在依赖关系,通过插入通讯原语,让 $u,v$ 并行执行,若 $u,v$ 并行执行时间小于 $u,v$ 串行执行时间,则 $u,v$ 有并行执行意义,我们称 $u,v$ 之间存在次并行性,或者说 $u,v$ 可以次并行化;否则,$u,v$ 只能串行执行.

如图1所示.在 $o1.m1$ 和 $o2.m2$ 之间存在依赖关系,为了使 $o1.m1$ 和 $o2.m2$ 并行执行时保持语义不变,需要在 $o1.m1$ 中 $R(x)$ 之后插入发送原语 send($o2.m2,x$),在 $o2.m2$ 中 $W(x)$ 之前插入接收原语 receive($o1.m1,x$).send 原语和 receive 原语均包含两个参数:第1个参数表明接收者(发送者)的地址;第2个参数表明发送(接收)的内容.send 原语是非阻塞通讯原语,receive 原语是阻塞通讯原语.send 原语将信息发送给接收者所在的处理节点,receive 原语从本处理节点接收发送者传来的消息,若该消息不存在,则等待.在分布存储系统中,send 和 receive 原语完成数据的传送和同步功能;而在共享存储系统中,通讯原语只需完成同步功能.

```
main()                C1:m1
{C1 o1;               { ...
 C2 o2;                 R(x);//对x进行读操作
 ...                    ...}
 o1.m1;               C2:m2
 o2.m2;               { ...
 ...}                   W(x);//对x进行写操作
                        ...}
```

(a) 插入通讯原语前

```
C1:m1                 C2:m2
{ ...                 { ...
  R(x);                 receive(o1.m1,x);
  send(o2.m2,x);        W(x);
  ...}                  ...}
```

(b) 插入通讯原语后

图1 通讯原语的插入

---

* 本文研究得到国家863高科技项目和国家攀登计划基金资助.作者杜建成,1971年生,博士生,主要研究领域为并行编译,并行处理.徐融,1973年,硕士生,主要研究领域为并行编译.陈道蓄,1949年生,副教授,主要研究领域为并行/分布系统.谢立,1942年生,教授,博导.主要研究领域为并行/分布系统.

本文通讯联系人:杜建成,南京 210093,南京大学计算机科学与技术系

本文1997-07-21收到原稿,1997-11-04收到修改稿

如果任务 $u$,$v$ 由于存在对变量 $x$ 的访问冲突而发生了依赖,称 $u$ 中对 $x$ 的访问点为依赖源,$v$ 中对 $x$ 的访问点为依赖目的. 通讯开销是指处于不同处理节点上的两个任务 $u$,$v$ 通讯时,从 $u$ 发出消息至消息到达 $v$ 所在处理节点的时间. 通讯等待开销是指某个依赖目的从需要对应依赖源的消息到获得该依赖源的消息的时间,其对应于该处的receive原语的执行时间,简称等待开销. 依赖源距离是指从任务起始点执行到依赖源的时间,不包括依赖源本身的执行时间. 依赖目的距离是指从任务起始点执行到依赖目的处的时间,包括依赖目的本身的执行时间. 任务的独立执行时间是指在依赖关系满足条件下,任务本身完成执行所需时间. 通讯斜率指依赖源的距离和依赖目的距离之差.

程序经过转换,为了保证语义不变,需保持原有的依赖关系. 我们在任一对依赖源和依赖目的之间插入通讯原语,以保证程序的正确转换. 这会带来一些冗余通讯,可以用冗余通讯化简技术来消除不必要的通讯开销.

### 1.2 简单任务间的一次通讯

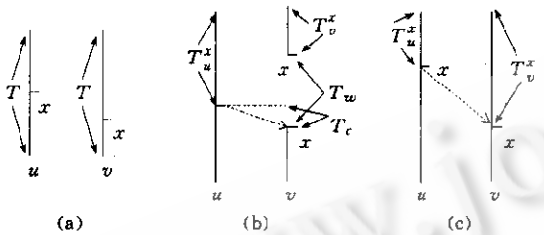


图2 一次通讯

我们先考察两个简单任务间的一次通讯. 在图2中,任务 $u$ 和 $v$ 由于存在对 $x$ 的访问冲突,在并行执行时需要插入通讯原语进行信息交流和同步. 我们用 $T_u^x$ 和 $T_v^x$ 分别表示依赖源距离和依赖目的距离,$T_c$ 表示通讯时间,假设它为一常数,$T_w$ 表示 $u$,$v$ 之间的通讯等待开销. 图2(b)和(c)分别对应于 $T_u^x+T_c\geqslant T_v^x$ 和 $T_u^x+T_c<T_v^x$,$T_u$,$T_v$ 表示 $u$,$v$ 的独立执行时间. 图2(b)中,$v$ 与 $u$ 并行执行有等待开销 $T_w$,在图2(c)中,不存在等待开销,甚至连通讯开销也被 $v$ 的计算重叠掉了.

$u$,$v$ 并行执行时间 $P_{u,v}=\max(T_u,T_v+T_w)$,$u$,$v$ 串行执行时间 $S_{u,v}=T_u+T_v$,其中

$$T_w=\max(T_u^x-T_v^x+T_c,0). \tag{1}$$

下面讨论任务间的通讯和可次并行化的关系.

**定理 1.** 若 $S_{u,v}>P_{u,v}$,则 $u$,$v$ 可次并行化,这等价于结论:若 $T_u>T_w$,则 $u$ 和 $v$ 可次并行化.

证明:若 $T_u>T_w$,有两种情况:

(1) $T_v+T_w\geqslant T_u$,有 $T_u+T_v>T_v+T_w=\max(T_u,T_v+T_w)$;

(2) $T_v+T_w<T_u$,也有 $T_u+T_v>T_u=\max(T_u,T_v+T_w)$. 故充分性得证.

若 $T_u+T_v>\max(T_u,T_v+T_w)$,也有两种情况:

(1) $T_u\geqslant T_c+T_w>T_w$;

(2) $T_u<T_c+T_w$,则 $T_u+T_v>\max(T_u,T_v+T_w)=T_v+T_w$,故 $T_u>T_w$. 故必要性得证. □

### 1.3 简单任务间的多次通讯

如果 $u$,$v$ 之间存在多次通讯,通讯平行或者交叉,如图3(a)和(b)所示. 用 $T_u^{x_i}(T_v^{x_i})$ 表示依赖源 $x_i$ 的距离和依赖目的 $x_i$ 的距离($1\leqslant i\leqslant n$),其中 $x_i$ 按其在 $v$ 中被访问的先后次序排序. $T_w^i$ 表示 $v$ 中 $x_i$ 处的通讯等待开销. 由式(1)易得

$$T_w^i=\max(T_u^{x_i}-T_v^{x_i}+T_c-\sum_{j=1}^{i-1}T_w^j,0)\qquad i\geqslant 1,T_w^0=0. \tag{2}$$

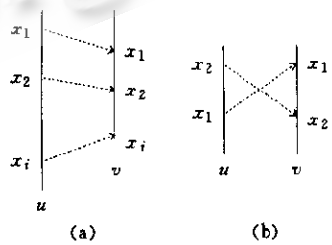


图3 多次通讯

**定理 2.** 若 $u$,$v$ 之间存在多对通讯,则 $v$ 的总通讯等待开销为

$$T_w=\sum_{i=1}^{n}T_w^i=\max_{i=1,n}(T_u^{x_i}-T_v^{x_i})+T_c.$$

证明:(1) 当 $n=1$ 时,结论成立.

(2) 假设 $n=k$ 时,结论成立,即 $T_w=\sum_{i=1}^{k}T_w^i=\max_{i=1,k}(T_u^{x_i}-T_v^{x_i})+T_c$

当 $n=k+1$ 时,由式(2)可得

$$\begin{aligned}T_w^{k+1}&=\max(T_u^{x_{k+1}}-T_v^{x_{k+1}}+T_c-\sum_{j=1}^{i-1}T_w^j,0)\\&=\max(T_u^{x_{k+1}}-T_v^{x_{k+1}}+T_c-(\max_{i=1,k}(T_u^{x_i}-T_v^{x_i})+T_c),0)\\&=\max((T_u^{x_{k+1}}-T_v^{x_{k+1}})-\max_{i=1,k}(T_u^{x_i}-T_v^{x_i}),0)\end{aligned}$$

(a) 若$(T_u^{k+1}-T_v^{k+1})>\max_{i=1,k}(T_u^i-T_v^i)$,即

$$T_u^{k+1}-T_v^{k+1}-\max_{i=1,k+1}(T_u^i-T_v^i)>0$$

$$T_w^{k+1}=T_u^{k+1}-T_v^{k+1}-\max_{i=1,k}(T_u^i-T_v^i)$$

故 $T_w=\sum_{i=1}^{k+1}T_w^i=\sum_{i=1}^{k}T_w^i+T_w^{k+1}=T_u^{k+1}-T_v^{k+1}+T_c=\max_{i=1,k+1}(T_u^i-T_v^i)$.

(b) 若 $T_u^{k+1}-T_v^{k+1}\leqslant\max_{i=1,k}(T_u^i-T_v^i)$,则 $T_w^{k+1}=0$,

故 $T_w=\sum_{i=1}^{k+1}T_w^i=\sum_{i=1}^{k}T_w^i+T_w^{k+1}=\max_{i=1,k}(T_u^i-T_v^i)+T_c=\max_{i=1,k+1}(T_u^i-T_v^i)+T_c$.

综上所述,命题得证. □

从通讯图上可以看出,两个任务 $u,v$ 并行执行总的等待开销由斜率最大的通讯决定,其他通讯所引起的等待开销或者与其他的通讯等待开销重叠,或者与 $v$ 的计算重叠.

如图 4 所示.$u,v$ 间包括两次通讯 $c_1,c_2$. 由定理 2 可知,图 4(a)和(b)中的通讯等待开销由斜率最大的通讯 $c_1$ 决定. 在图 4(a)中,由 $c_1$ 通讯所传递的消息会随着 $v$ 的执行到达 $c_2$ 通讯的接收点,故 $c_2$ 通讯就成为冗余,可以被删除. 在图 4(b)中,由 $c_2$ 通讯传递给 $v$ 的信息最终会被 $c_1$ 通讯传递给 $v$ 的信息所覆盖,故 $c_2$ 通讯也成为冗余,可以被删除. 一般而言,若干个依赖源与同一个依赖目的或者一个依赖源与若干个依赖目的之间的通讯,只有斜率最大的那次通讯是必需的,这称为冗余通讯的化简.

### 1.4　复合任务间的通讯

上面所讨论的 $u,v$ 的控制结构都很简单,都是顺序结构,实际上,$u,v$ 中常常包括分支和循环. 如图 5 所示,$u,v$ 中都包含了分支,图中每个节点代表 $u$ 或 $v$ 的一个子任务,每个节点的执行时间和 $u,v$ 之间的依赖关系已知. 在 $u(v)$ 中存在着不同的路径 $u_1,u_2,\ldots,u_n(v_1,v_2,\ldots,v_m)$,分别计算发生在 $u_i$ 和 $v_j$ 之间的通讯等待开销 $T_w^{ij}(1\leqslant i\leqslant n,1\leqslant j\leqslant m)$,然后选择最大值作为 $u,v$ 之间通讯的等待开销,即 $T_w=\max(T_w^{ij})$.

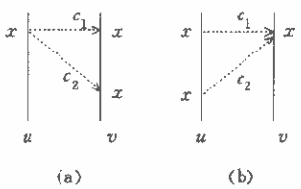


图4　冗余通讯

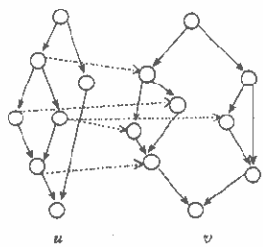


图5　复合任务

以上讨论都假定 $u,v$ 本身按串行方式执行,实际上,$u,v$ 本身可以在保持语义不变的前提下,以任意方式执行,以上结论依然适用.

## 2　任务间次并行性的识别

为典型起见,我们以两个复合任务 $u,v$ 为例,来说明任务间次并行性识别的一般过程.

(1) 根据 $u,v$ 的 SI 信息,找到每一对依赖源和依赖目的. 这要求 $u,v$ 的 SI 中要记录变量引用的地址,然后对 $u,v$ 的 SI 集进行交运算. 相交变量 $u,v$ 中的地址便是相应的依赖源和依赖目的.

(2) 根据 $u,v$ 的任务流图,计算 $u$ 中每个节点的最短距离和最长距离,计算 $v$ 中每个节点的最短距离. 任务流图中每个节点都被赋予一个时间权,表明该节点的执行时间. 节点的最短距离是指从 $u$(或 $v$)的起始节点执行到该节点处(不包括该节点)的不同路径中最短者. 节点的最长距离是指从 $u$(或 $v$)的起始节点执行到该节点处(包括该节点)的不同路径中最长者. 路径的长度定义为该路径上所有节点的时间权之和.

(3) 计算 $u,v$ 的通讯等待开销. $u$ 中某个依赖源的距离等于该节点的最短距离,$v$ 中某个依赖目的距离等于该节点的最长距离.

(4) 计算 $u$ 的独立执行时间. 该时间为 $u$ 的终止节点的最短时间加上该节点的执行时间.

(5) 判断是否存在次任务并行性. 若存在,则插入通讯原语.

以上几步中,第 2 步比较耗时,其实质是寻找最短路径问题,可以采用 Dijkstra 算法来解决这个问题. 在最坏情况下,其时间复杂度为 $O(|V|^2)$,$|V|$ 为任务流图中的结点数.

## 3 任务间次并行性的强化

### 3.1 任务合并

在分析过程中发现，两个任务之间既不存在主并行性，又不存在次并行性. 如果两个只能串行执行的任务运行于分布存储系统中不同的处理节点上，在 $u$ 完成执行之后，依然要同 $v$ 通讯，传送必要的信息. 如果将 $u$，$v$ 合并成一个任务，就可以消除这种通讯开销. 我们考虑两种一般情况.

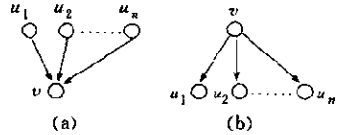


图6 任务的合并

(1) 任务 $v$ 同时依赖于 $u_1,u_2,\ldots,u_n$(如图 6(a)所示). $v$ 与 $u_i$ 不存在次并行性($1\leqslant i\leqslant n$). $v$ 的执行时间为 $T$，$u_i$ 的执行时间为 $T_i$，$v$ 与 $u_i$ 并行执行的等待开销为 $W_i$，通讯开销为 $T_c$，假设为一常数，显然有 $T_c\leqslant W_i\leqslant T_i+T_c$，将 $v$ 与 $u_i$ 合并成新的任务 $u'_i$，此时整个任务组的完成时间 $T'_k$，取决于 $v$ 完成的时刻，$T'_k=\max_{j=1,n,j\neq i}(W_j+T,T_i+T)=\max_{j=1,n,j\neq i}(W_j+T)$，为了使整个任务组的执行时间最短，需要选择最佳合并项. 若 $T'_k=\min_{i=1,n}T'_k=\min_{i=1,n}(\max_{j=1,n,j\neq i}(W_j+T))$，则 $u_k$ 为最佳合并项. 易证，等待开销最大的任务即为最佳合并项.

(2) 任务 $u_1,u_2,\ldots,u_n$ 同时依赖于 $v$(如图 6(b)所示). $u_i$ 与 $v$ 不存在次并行性($1\leqslant i\leqslant n$). 将 $v$ 与 $u_i$ 合并成新的任务 $u'_i$，此时整个任务组的完成时间 $T'_k$ 由最后一个完成执行的任务 $u_i$ 决定，即 $T'_k=\max_{j=1,n,j\neq i}(W_j+T_j,T+T_i)=\max_{j=1,n,j\neq i}(W_j+T_j)$. 若 $u_k$ 为最佳合并项，则 $T'_k=\min_{i=1,n}T'_k=\min_{i=1,n}(\max_{j=1,n,j\neq i}(W_j+T_j))$，也易证等待开销和执行时间之和最大的任务为最佳合并项.

### 3.2 代码的移动

如果 $u$，$v$ 之间存在依赖关系($u\delta_d v$)，由式(1)可知，依赖源在词法上越靠前，依赖目的在词法上越靠后，则通讯等待开销越小. 在这里，我们只考虑依赖源和依赖目的位于一段顺序结构代码上，其中不包括分支和循环结构，或者分支和循环结构可被当作一个基本单位的情况. 将依赖源(依赖目的)从 $A$ 点移到 $B$ 点的充分条件是：$AB$ 之间不存在对依赖源(依赖目的)变址的访问. 例如图 7(a)和(b)是两段代码，它们有依赖关系，依赖源是 $p$，依赖目的为 $q$，这两段代码均满足代码移动条件，故可将图 7(a)中的 $p$ 从 $A$ 移到 $B$，如图 7(c)；将图 7(b)中的 $q$ 从 $C$ 移到 $D$，如图 7(d). 这样操作的结果是缩短了通讯等待开销，增强了并行性.

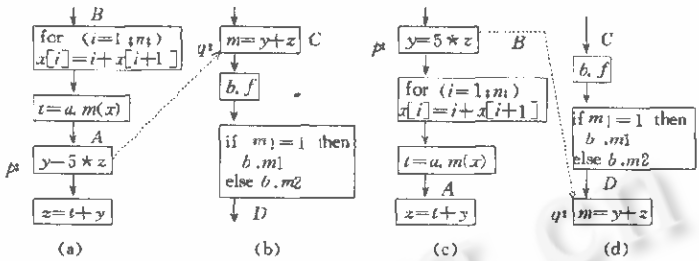


图7 代码的移动

## 4 任务执行时间的静态分析

如果能在编译时刻知道程序中各模块的相对执行时间，对开发任务次并行性以及任务的调度都有重要意义. 其一般方法为：确定不同类型指令的相对执行时间 $t_1,t_2,\ldots,t_k$，然后统计程序中不同类型的指令数 $n_1,n_2,\ldots,n_k$，则总的程序执行时间 $T=\sum_{i=1}^{k}n_i*t_i$. 但是，很多情况下，一些重要的参数在编译时刻难以确定，如循环中迭代的次数，条件变址的值等等. 而这些参数对 $T$ 的计算都有重要影响. 我们观察到，在某些情况下，编译时刻的不确定性对一些相关模块具有同等的影响，此时，我们仍可得到模块执行时间的相对值. 在另外一些情况下，只有借助于用户在编译时刻的提示或推迟到运行时刻，才能得到结果.

由于静态的通讯等待开销和程序执行时间的分析存在误差，我们可以采用 $T_u>kT_v$ 来判定次并行性的存在条件，$k$ 为大于 1 的经验值.

## 5 相关工作

为了开发程序中的任务级并行性，Girker 用 HTG 作为中间表达形式对程序进行抽象.[1]在 HTG 图中，任意两个

节点 $A$,$B$ 之间若存在依赖关系(控制依赖或数据依赖),即在 DDG 或 CDG 中存在由 $A$ 指向 $B$ 的边,则 $A$,$B$ 不能并行执行. 如果 $A$ 是复合节点,这种处理过于简单,不利于发掘潜在的并行性,尤其当 $B$ 也是复合节点时. 系统 Mentat[2] 在有依赖的任务间插入通讯原语,强行并行化,不考虑通讯和并行的平衡,有时导致相反的结果. 文献[3]在考察两个任务间的多对通讯时,没有考虑通讯与计算和通讯与通讯之间的重叠,因此也会导致并行性的损失. 在国内,国防科技大学开发的 KD-PASTE 和复旦大学开发的 FAT 都侧重于开发数据并行性,对任务并行性的开发支持不够,也没有考虑存在依赖关系情况下的任务并行执行的可能性.

## 6　实例研究

图 8(a)、(b)分别表示 task1,task2 的代码. 我们在由 6 台 RS6000 通过 ATM 构成的 NOW 上做了几组实验. 表 1 给出了 task1,task2 串行执行时间 $s1$,$s2$、串行执行总时间 $s$、并行执行时间 $p1$,$p2$、通讯等待开销 $c$ 和并行执行总时间 $p$. task1,task2 串行执行时,忽略图 8 中的通讯操作;并行执行时,通讯操作通过调用 PVM 有关函数实现. 由表中可见,当 $K=10$ 时,$s1<c$,$s<p$,即 task1,task2 无次并行性;当 $K=100$,200 时,$s1>c$,$s>p$,task1,task2 存在次并行性.

```
for(n=0;n<500;n++)                    for(n=0;n<500;n++)
for(i=0;i<K;i++)                      for(i=0;i<K;i++)
  for(j=0;j<K;j++)                      for(j=0;j<K;j++)
  A[i,j]=A[i,j]+A[i-1,j-1];             A[i,j]=A[i,j]+A[i-1,j-1];
send(task2,A);                        receive(task1,A);
for(n=0;n<500;n++)                    for(n=0;n<500;n++)
for(i=0;i<K;i--)                      for(i=0;i<K;i++)
  for(j=0;j<K;j--)                      for(j=0;j<K;j++)
  B[i,j]=B[i,j]+A[i,j];                 D[i,j]=C[i,j]+A[i,j];
```

(a) task1　　(b) task2

图 8　次并行性的存在条件

表 1

| K | s1 | s2 | s | p1 | p2 | c | p |
|---|---|---|---|---|---|---|---|
| 10 | 0.022 8 | 0.023 1 | 0.045 9 | 0.032 1 | 0.048 1 | 0.025 1 | 0.048 1 |
| 100 | 2.531 1 | 2.534 0 | 5.065 5 | 2.540 1 | 2.544 3 | 0.029 8 | 2.544 3 |
| 200 | 10.280 1 | 10.310 2 | 20.590 5 | 10.300 1 | 10.340 3 | 0.030 2 | 10.340 3 |

## 7　结束语

我们在开发 JAVA 程序自动并行化工具的过程中提出次并行性概念. 判定两个任务之间是否存在次并行性,关键是等待开销的大小. 本文还讨论了任务合并、代码移动、通讯插入等问题,利用这些措施挖掘和强化任务间的次并行性.

## 参考文献


1　Girker M B. Functional parallelism, theoretical foundations and implementation. CSRD Report No. 1182, University of Illinois at Urbana-Champaign. Dec. 1991

2　Grimshow A S. Easy-to-use object-oriented parallel processing with mentat. Computer, 1993,26(5):39～51

3　Polychropodlos C D D. Constantine: Parallel Programming and Compilers. Boston-Kluver Academic Publishers, 1988. 179～194


## Inter-task Secondary Parallelism


DU Jian-cheng　XU Rong　CHEN Dao-xu　XIE Li

(*Department of Computer Science and Technology　Nanjing University　Nanjing*　210093)

(*State Key Laboratory for Novel Software Technology　Nanjing University　Nanjing*　210093)



**Abstract**　In this paper, the definition of inter-task secondary parallelism and its existence condition are presented. Communications between two tasks and the calculation of communication waiting overheads are discussed along with the exploitation procedure of task secondary parallelism. And the effects of code motion and tasks fusion for enhancing task parallelism and reducing communication overheads are also illustrated.



**Key words**　Data parallelism, task parallelism, data dependence, control dependence, hierarchical task graph.