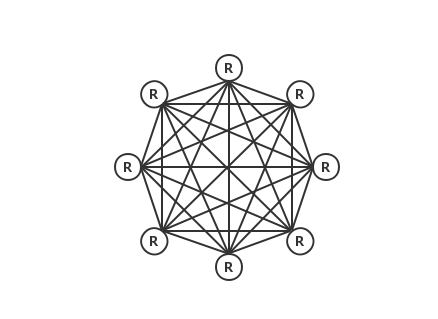
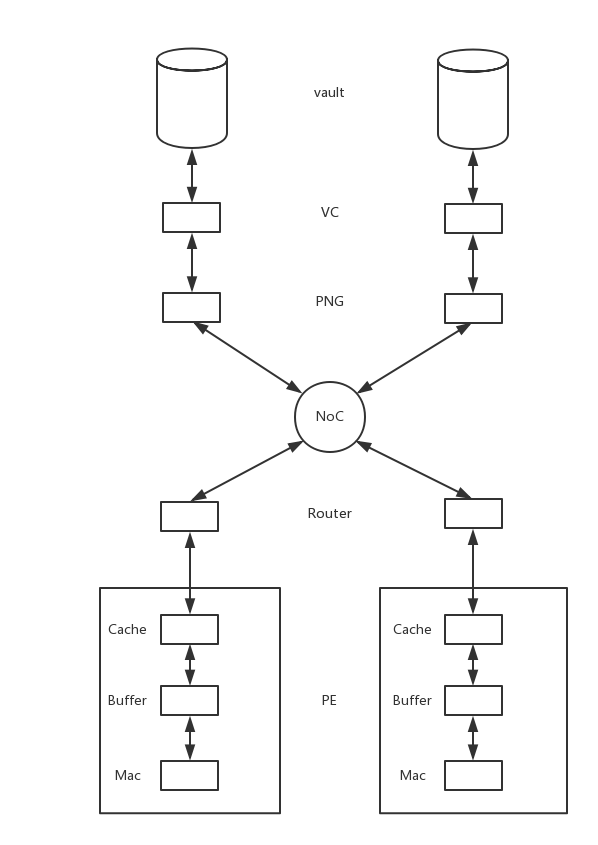
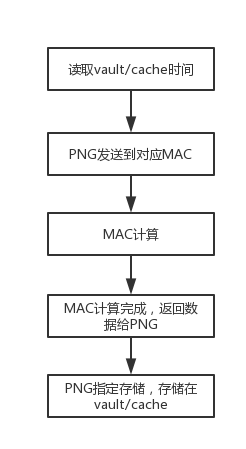
# 应用模型



通过输入的数据或是计算完的数据可以被存储vault（DRAM）和cache之中。众所周知，DRAM能够存储的数据远大于cache，但读取的速度缺慢了很多，所以**在存储数据的时候需要判断是存储在vault之中还是存储在cache之中**。由上一小节可知，PNG负责控制数据的流动，当计算开始时，PNG根据当前神经元所需要的数据去对应的存储位置取到数据，然后封装成对应的数据包，通过NoC网络传送给对应的PE进行计算。当PE中MAC计算完成后，将计算完成的结果传回给PNG，PNG判断后继需要这个数据的神经元，将其存储到对应PE的Cache之中，若Cache存储不下，则存储到对应的vault中。

时间消耗如下图所示。



# Algorithm

## 输入

定义一个有向无环图DAG，G=(V, E, P, R)，R表示图G的retiming，用来计算retiming。

定义K表示PE的总个数。

定义T表示G需要循环的次数。

## *Fast-Para-Conv*

*Fast-Para-Conv*分为如下几个步骤：

1. 单发射规则排列
   1. 周期排列
   2. 整体排列
2. 多发射排列
3. 恢复依赖

* 单发射规则排列

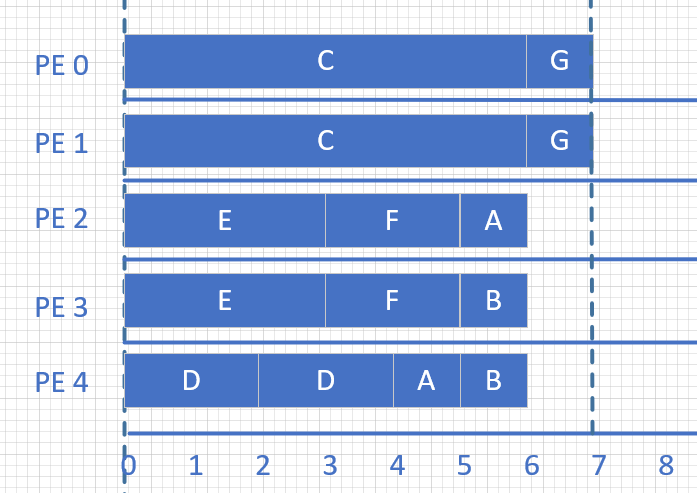
将所有的任务节点不考虑数据依赖的条件下按照一定的规则安排在各个PE的每个时间段，为了达到局部的最优解，每个PE的每个时间段前后的任务之间没有时间等待。直接对所有任务进行排列具有很大的不确定性，每次任务的时间消耗不同，对最终的排列有着很大的影响。*Fast-Para-Conv*中的规则排列可以分为两个部分：周期排列，整体排列，即先确定一个周期内的排列，再确定整体全部的排列。

* + 周期排列

考虑到一个周期只安排一轮图的话，可能会造成利用率较低，所以在一个周期内可以安排多轮，直至利用率达到预期或达到最大轮数后才终止安排。若安排了S轮，则可以认为这S轮为一个周期。为了保证完整性，利用率的计算应该在每一轮安排结束后再进行计算。

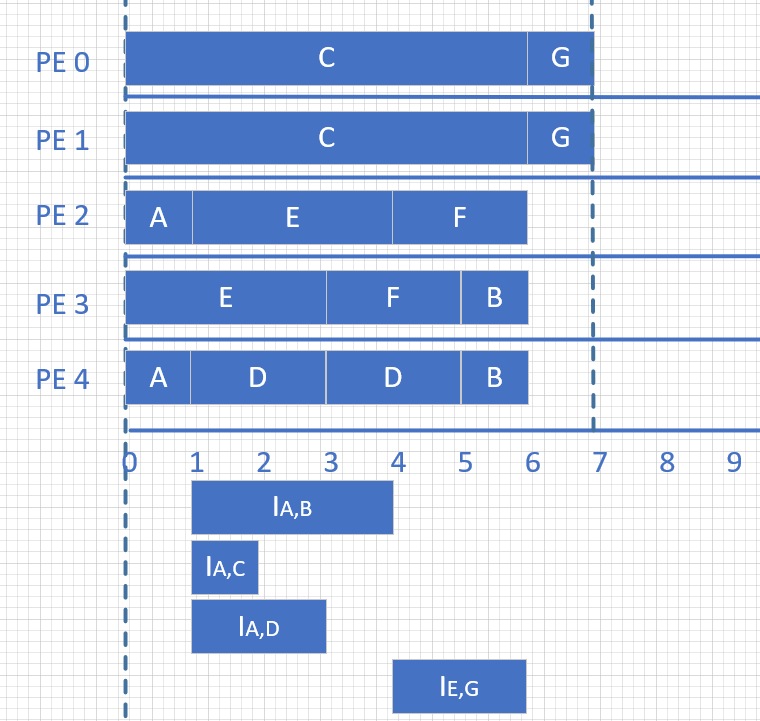
这个问题可以类比最小化最大值问题，可以用贪心的思想来解决。把PE看在箱子，需要将计算任务放入PE之中，让消耗时间最长的PE的时间尽可能短。

对任务数组以任务的时间长为第一优先级，拓扑序小为第二优先级进行排序。将这些任务排放在N个PE上，每次把任务放在当前结束时间最短的PE上。



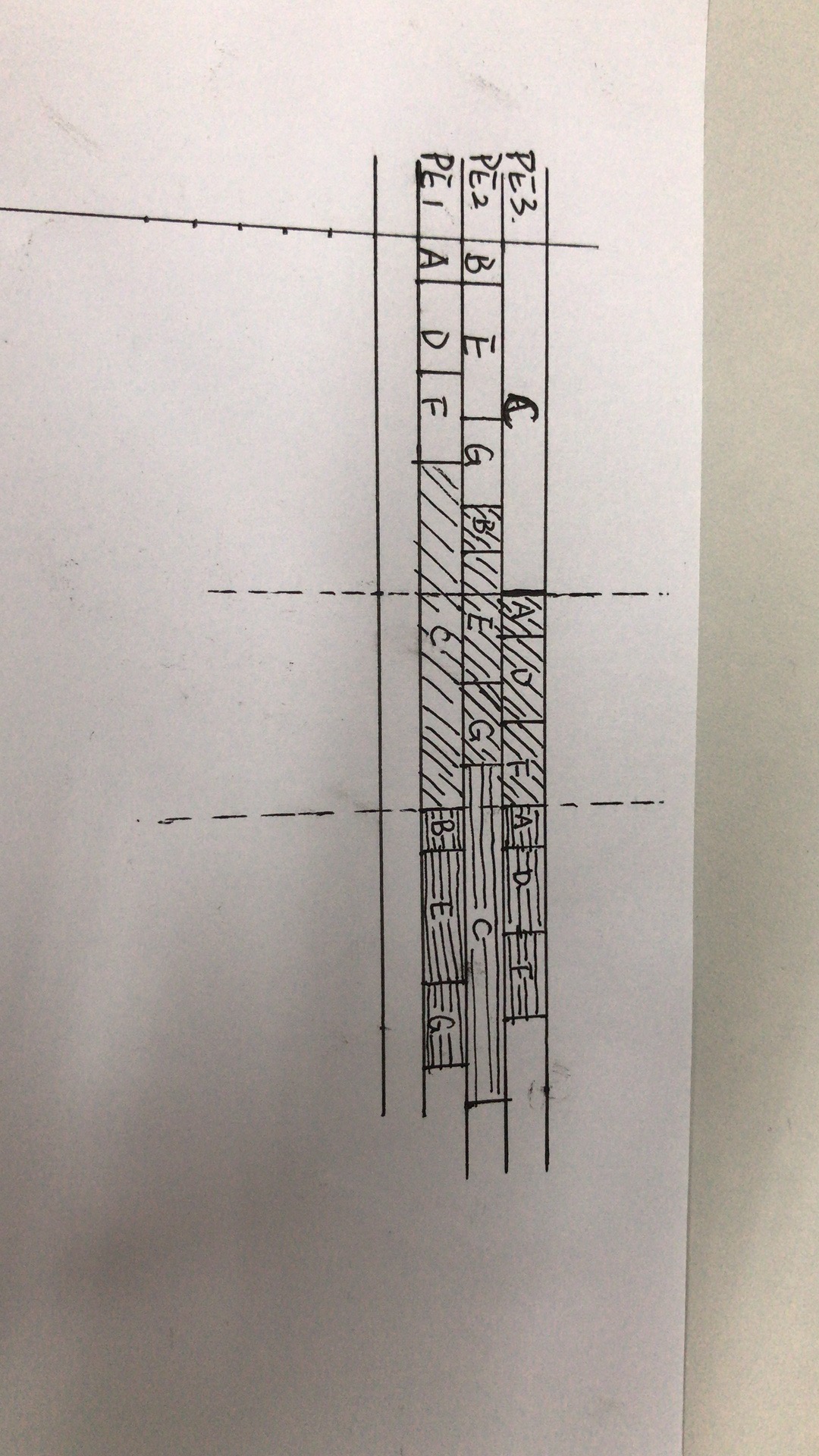
对每个PE上的任务进行重新排序。为了尽量减少retiming次数，尽量更有效的利用每个周期内的任务， 对每个PE上任务进行重新排序。可以发现在图中A有一条到B的边，说明B需要A计算完成后的数据来进行计算，所以在A计算完成后，B需要等待一个传输的时间才能进行计算。在PE4的排列中，若将A和B紧挨着排列的话，将导致A计算完成后不能直接利用当前周期内的B，而需要一次retiming，利用下一个周期的B。但考虑到两个D的计算任务，此时暂时用不到，若将他们摆放在A与B之间，则可以实现在一个周期内，A计算完成后，B可以在同一个周期进行计算，减少一次retiming。

对确定PE的计算任务重新排列的算法步骤是：按照图的拓扑序，把每个PE上拓扑序靠后的节点作为填充节点，依次判断每个节点和与它有边的后继节点，用填充节点填充数据传输的等待时间。若没有填充节点时，则直接紧挨着摆放，继续考察后续节点。



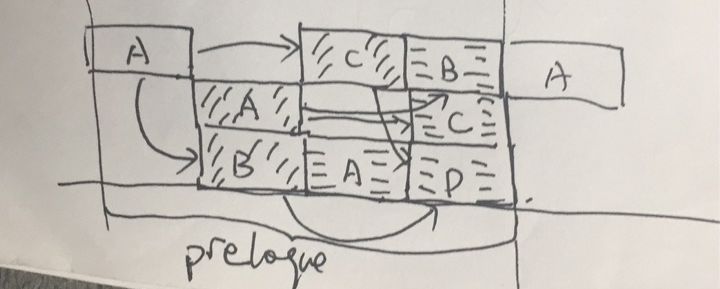
* + 整体排列

在得到每个周期内的排列后，若直接按照一个周期一个周期的排列下去，PE并不能得到充分的利用，因为每个周期内的利用率并不能达到100%。考虑到不同的PE都是相同的架构，所以这些PE具有无差性，那么当前一个周期排列完成后，可以将当前周期排列在每个PE的任务看作一个整体的任务，即当前周期共有N个任务，将它们按照时间长短排序，将耗时最长的任务摆放在当前结束时间最早的PE上，依次类推，排列完成当前周期。对下一个周期采取同样的策略摆放。



* 恢复依赖（考虑local cache）

因为第一步的时候忽略了节点之间的数据依赖关系，但最后在计算总时间的时候还是要考虑数据以来关系的，故每个节点需要依据依赖关系找到它后面满足依赖关系的后续节点。如下图所示，



当一个节点在同一个周期内找不到满足依赖关系的后续节点，需要跨几个周期才能找到后续节点时，此时就需要利用重定时（retiming）操作来确定。

*Fast-Para-Conv*利用上述的基础算法，结合多发射技术，重新对任务节点进行调度，并且在第一步的规则排列中，设计合理的规则以提高CPU的利用率。规则如下：对于任务安排， 尽可能让每个PE的总任务时间相差最小，同时每个PE里面安排的任务按照拓扑序列先后排序。换句话来说，就是让一次发射的m个PE中，安排后总时间最长的PE的运行时间尽可能少。

*Fast-Para-Conv*的实现可以很多的方法，故本文只介绍*Fast-Para-Conv*的大体方向，具体步骤如下：

1. 获取有向无环图G的拓扑序列。
2. while 不满足周期结束条件：
3. 依据规则在PE上安排G的各个节点。
4. 计算每次发射需要循环的轮数t。
5. for i = 0; i < t; i ++：
6. for Vj in {Topology of G}：
7. for e in Edge[Vj]：
8. Vk为Vj经过边e所到达的在安排的节点。
9. 获取Vk在规则排列图中的具体位置，即获取开始时间，结束时间和PEID等信息。
10. 更新Vk的开始时间为min(Vj.endtime + e.cost, Vk.starttime)。

虽然从伪代码上, *Fast-Para-Conv*的代码量并不大，但是如果考虑到代码运行效率前提下，还是需要一定的难度。

时间复杂度分析：从伪代码来看，每轮中，每条边都只循环了一次，所以时间复杂度为，其中E表示边的数量，t表示每次发射的循环次数。但在具体实现的过程中，尤其是伪代码的第9行可能会对最终的时间复杂度有所影响，一定程度上提高时间复杂度。