A 1st introduction to Large Scale Computing

Concepts & Techniques

Chapter 2: LSC programming - Python

孔令波
mlinking@126.com
+86 15010255486

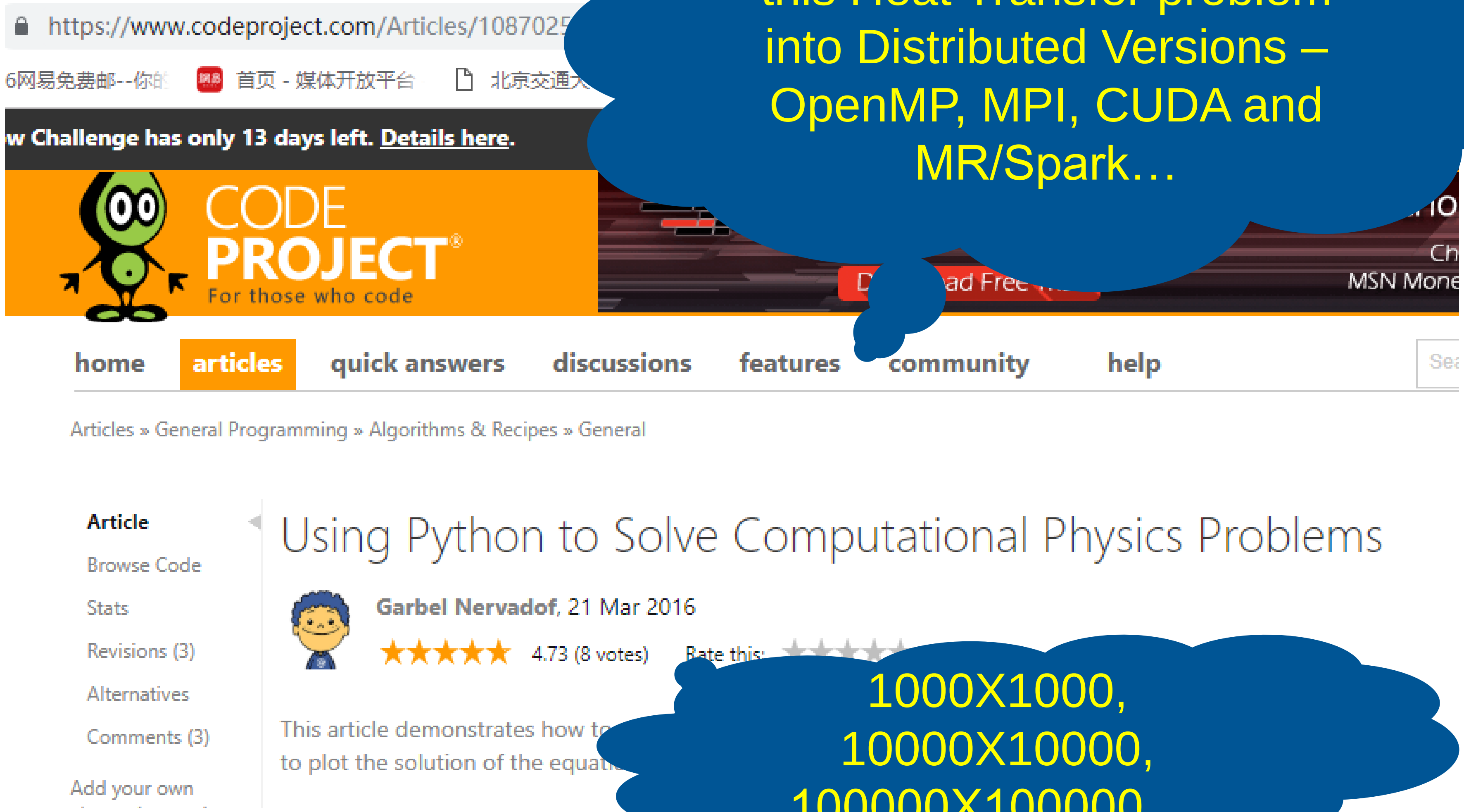
大纲

- **前言**
  - 为什么需要“大规模计算” [HPC, DL, Business platform system, Cloud已经合流]
    - 导入 – 科学计算(天气预报)，DL, 互联网平台(Google, Amazon, Alibaba, MeiTuan, …)
- **基础篇 – 概念和样子/例子**
  - 并发程序的样子 – Divide & Conquer, Model & Challenges, PCAM, Data/Task, …
    - 天气预报的计算
  - 运行环境
    - 硬件 – 自己梳理的3个方案 – Shared/Non-shared Memory, Hybrid
    - 系统软件 – 协议栈, Modern OS, Distributed Job Scheduler, GTM等
    - 大数据环境 – 开源运动的新贡献 – 分享是有威力的!
- **算法级篇 – 并行编程(Parallel Programming)的原理和实现**
  - OpenMP, MPI, CUDA (DL的实现), Big Data 中的MR/Spark等 (只涉及在Big Data SDK之上的编程；大数据本身的介绍放到后一部分)
    - Numba (OpenMP), MPI4PY, PyCUDA, PySpark, …
- **系统级篇 – 架构+集成 (Architecture + Integration Programming) 的威力**
  - “秒杀”的技术架构； 计算广告； 系统架构 (HTAP等)
    - Flink, ClickHouse, MaxCompute, ELK …

Chapter 2: LSC programming

- **Sequential implementation with Python**
  - "Using Python to Solve Computational Physics Problems"
- **Ideas to convert Sequential to Parallel**
  - DAOM, PCAM
- **Measure the performance**

Sequential Programming


https://www.codeproject.com/Articles/1087025
6网易免费邮--你的
首页 - 媒体开放平台
北京交通大
w Challenge has only 13 days left. Details here.
CODE PROJECT For those who code
home
articles
quick answers
discussions
features
community
help
Articles » General Programming » Algorithms & Recipes » General
Article
Browse Code
Stats
Revisions (3)
Alternatives
Comments (3)
Add your own

Using Python to Solve Computational Physics Problems

Garbel Nervadof, 21 Mar 2016

★★★★★ 4.73 (8 votes) Rate this:

This article demonstrates how to
to plot the solution of the equati

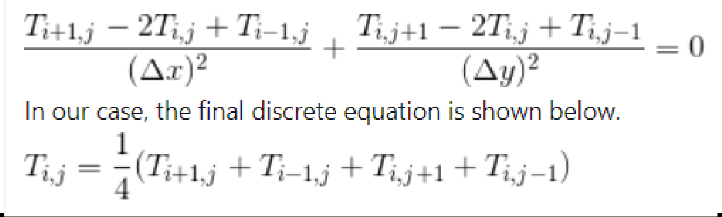
You're required to implement this Heat Transfer problem into Distributed Versions – OpenMP, MPI, CUDA and MR/Spark…

1000X1000, 10000X10000, 100000X100000,…

We have know Solution Technique

- **A grid is used to divide the region of interest.**
  - Since the PDE is satisfied at each point in the area, it must be satisfied at each point of the grid.
- **A finite difference approximation is obtained at each grid point.**

$$\frac{\partial^2 T(x,y)}{\partial x^2} \approx \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{(\Delta x)^2}, \quad \frac{\partial^2 T(x,y)}{\partial y^2} \approx \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{(\Delta y)^2}$$



$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{(\Delta x)^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{(\Delta y)^2} = 0$$

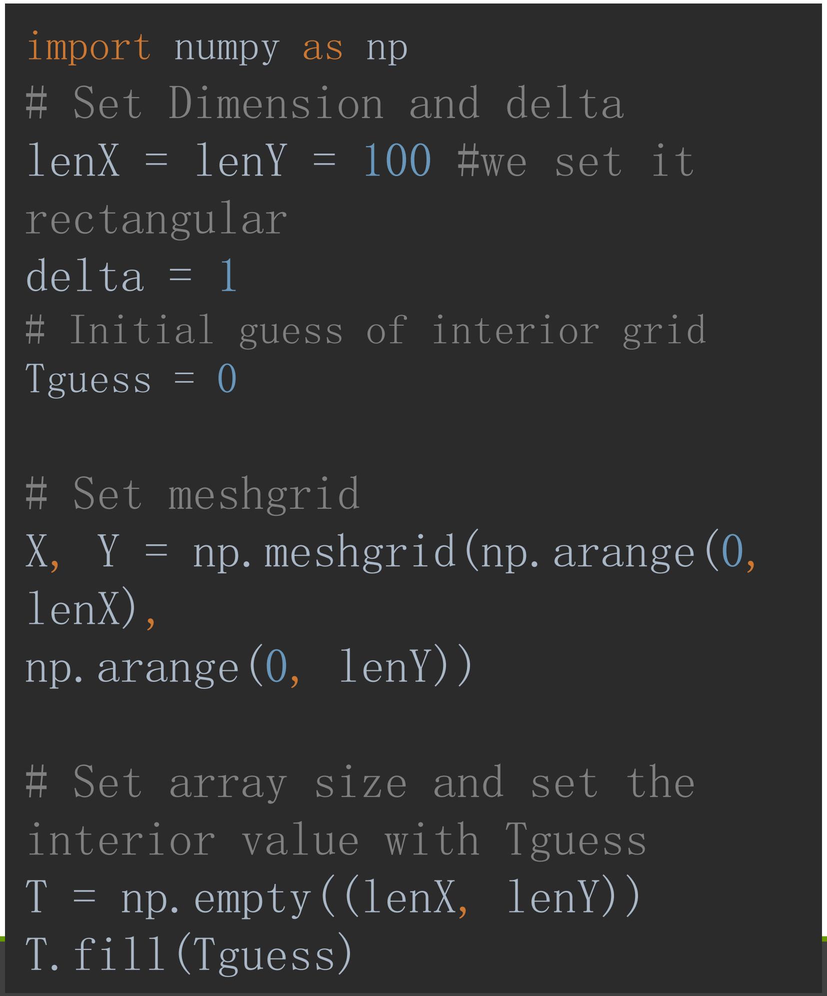
In our case, the final discrete equation is shown below.

$$T_{i,j} = \frac{1}{4}(T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1})$$

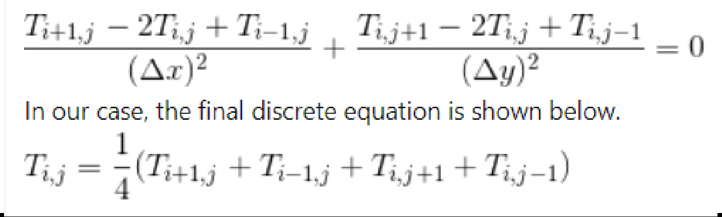
□ The code demonstration of "Using Python to Solve Computational Physics Problems"

1. Configure the parameters

- GRID
  - With Initial values [初始值]
  - Boundary conditions
    - ✓ [边界条件]
- Termination condition
  - Iteration number or Epsilon



```
import numpy as np
# Set Dimension and delta
lenX = lenY = 100 #we set it
rectangular
delta = 1
# Initial guess of interior grid
Tguess = 0

# Set meshgrid
X, Y = np.meshgrid(np.arange(0,
lenX),
np.arange(0, lenY))

# Set array size and set the
interior value with Tguess
T = np.empty((lenX, lenY))
T.fill(Tguess)
```



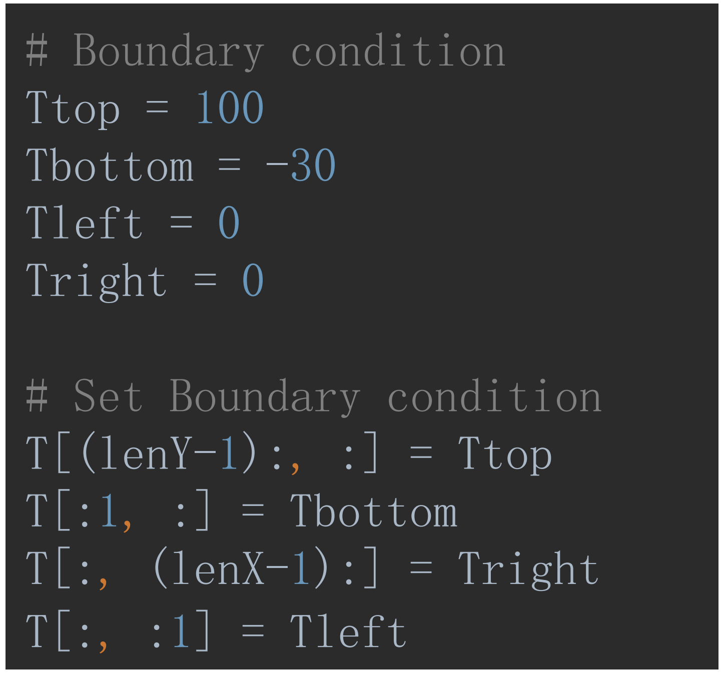
$$\frac{T_{i+1,j}-2T_{i,j}+T_{i-1,j}}{(\Delta x)^2}+\frac{T_{i,j+1}-2T_{i,j}+T_{i,j-1}}{(\Delta y)^2}=0$$

In our case, the final discrete equation is shown below.

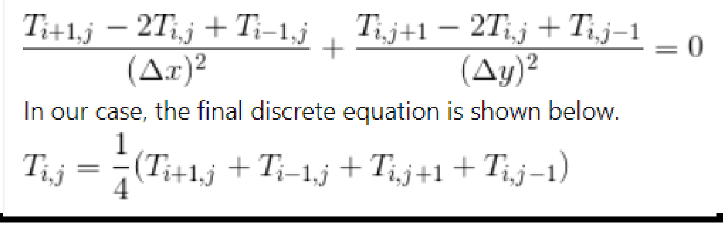
$$T_{i,j}=\frac{1}{4}(T_{i+1,j}+T_{i-1,j}+T_{i,j+1}+T_{i,j-1})$$

☐ The code demonstration of "Using Python to Solve Computational Physics Problems"

1. **Configure the parameters**
   - GRID
     - ➢ With Initial values [初始值]
     - ➢ With Boundary conditions
       - ✓ [边界条件]
   - Termination condition
     - ➢ Iteration number or Epsilon



```python
# Boundary condition
Ttop = 100
Tbottom = -30
Tleft = 0
Tright = 0

# Set Boundary condition
T[(lenY-1):, :] = Ttop
T[:1, :] = Tbottom
T[:, (lenX-1):] = Tright
T[:, :1] = Tleft
```



$$\frac{T_{i+1,j}-2T_{i,j}+T_{i-1,j}}{(\Delta x)^2}+\frac{T_{i,j+1}-2T_{i,j}+T_{i,j-1}}{(\Delta y)^2}=0$$

In our case, the final discrete equation is shown below.

$$T_{i,j}=\frac{1}{4}(T_{i+1,j}+T_{i-1,j}+T_{i,j+1}+T_{i,j-1})$$

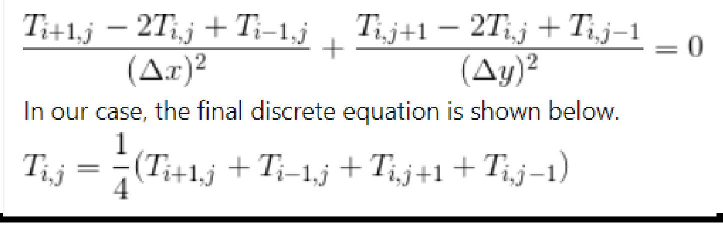
□ The code demonstration of "Using Python to Solve Computational Physics Problems"

1. **Configure the parameters**

- GRID
  - ➢With Initial values [初始值]
  - ➢With Boundary conditions
    - ✓[边界条件]

- Termination condition
  - ➢Iteration number or Epsilon

```
# Set maximum iteration
maxIter = 100
# Iteration (We assume that the
iteration is convergence in maxIter
= 500)
print("Please wait for a moment")


for iteration in range(0, maxIter):
```



$$\frac{T_{i+1,j}-2T_{i,j}+T_{i-1,j}}{(\Delta x)^2}+\frac{T_{i,j+1}-2T_{i,j}+T_{i,j-1}}{(\Delta y)^2}=0$$

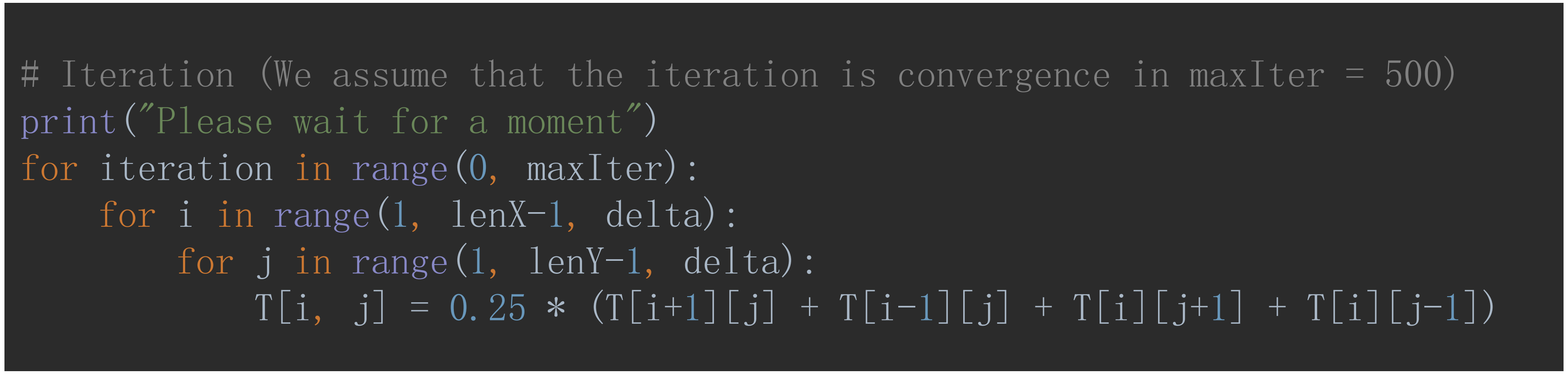
In our case, the final discrete equation is shown below.

$$T_{i,j}=\frac{1}{4}(T_{i+1,j}+T_{i-1,j}+T_{i,j+1}+T_{i,j-1})$$

◻ The code demonstration of "Using Python to Solve Computational Physics Problems"

2. Iterative updating

- Use "Termination condition" to control the updating of the internal vertices



```
# Iteration (We assume that the iteration is convergence in maxIter = 500)
print("Please wait for a moment")
for iteration in range(0, maxIter):
    for i in range(1, lenX-1, delta):
        for j in range(1, lenY-1, delta):
            T[i, j] = 0.25 * (T[i+1][j] + T[i-1][j] + T[i][j+1] + T[i][j-1])
```

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{(\Delta x)^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{(\Delta y)^2} = 0$$

In our case, the final discrete equation is shown below.

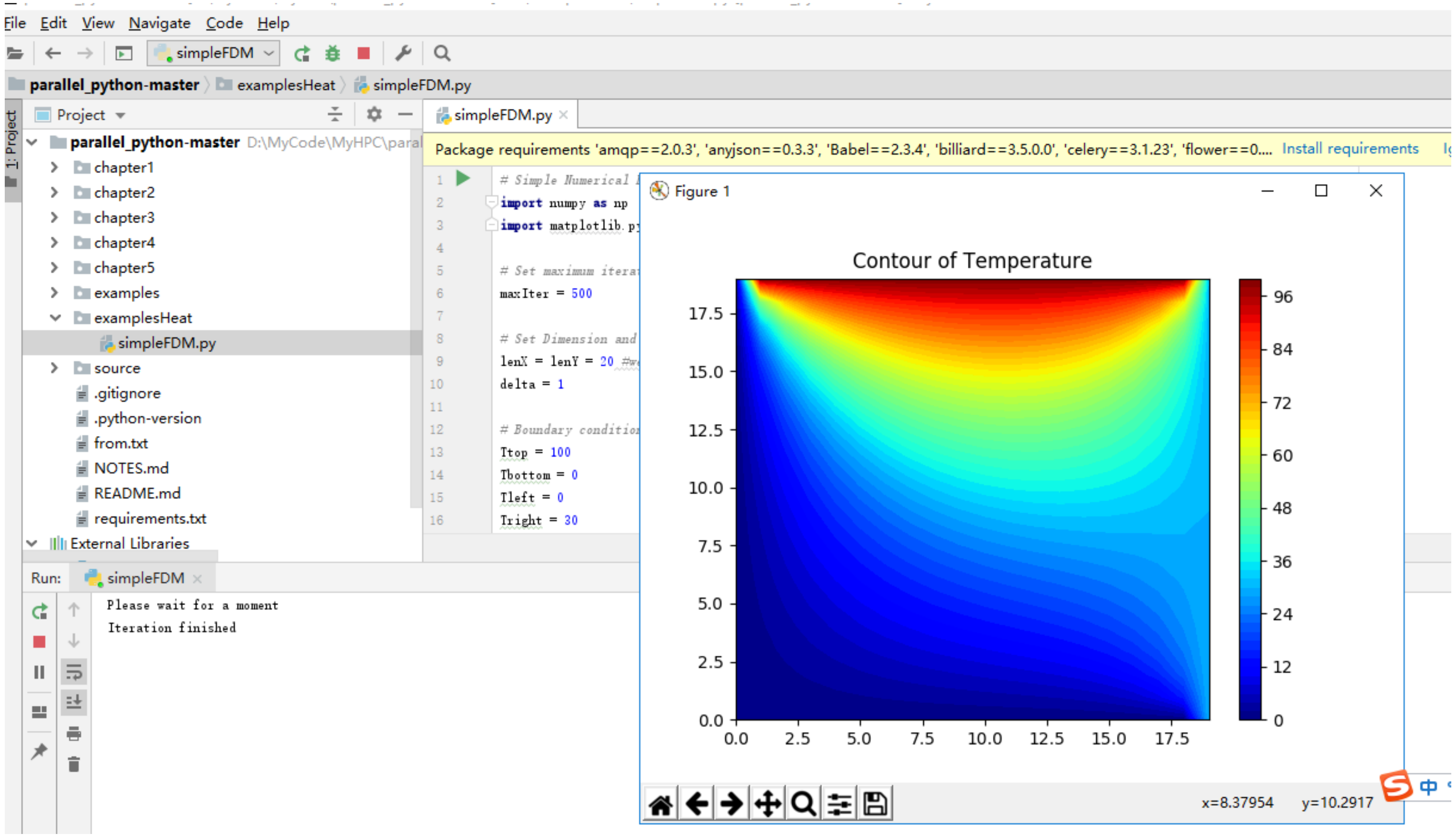
$$T_{i,j} = \frac{1}{4}(T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1})$$

☐ The code demonstration of "Using Python to Solve Computational Physics Problems"

3. Visualize the dynamics

```
# Set colour interpolation and colour map
colorinterpolation = 100
colourMap = plt.cm.jet #you can try: colourMap = plt.cm.coolwarm

<<Repeated updating>>

# Configure the contour
plt.title("Contour of Temperature")
plt.contourf(X, Y, T, colorinterpolation, cmap=colourMap)

# Set Colorbar
plt.colorbar()

# Show the result in the plot window
plt.show()
```

Copy the code into PyCharm project



Package requirements 'amqp==2.0.3', 'anyjson==0.3.3', 'Babel==2.3.4', 'billiard==3.5.0.0', 'celery==3.1.23', 'flower==0.... Install requirements

```
# Simple Numerical
import numpy as np
import matplotlib.py

# Set maximum itera

maxIter = 500

# Set Dimension and
lenX = lenY = 20 #w
delta = 1

# Boundary conditio
Ttop = 100
Tbottom = 0
Tleft = 0
Tright = 30
```

```
Please wait for a moment
Iteration finished
```

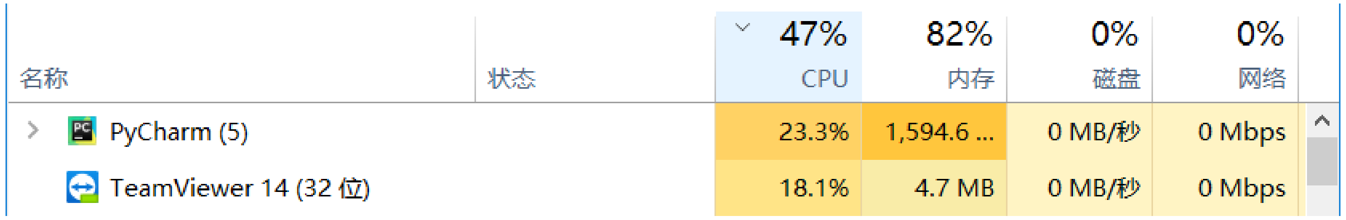
Small challenge

- **Define and use Epsilon to control the repetition?**

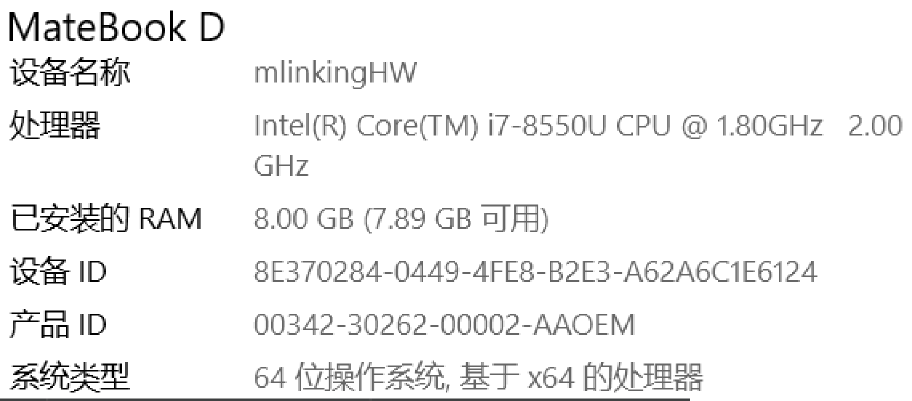
- **Hint:**
  - Use the **matrix norm**

Run the program with different scales

- When the “maxIter = 100000”, the program takes almost 40 minutes!
- When “lenX = lenY = 10000” + “maxIter = 1000”, it takes 64832 secs = **18 hours**!

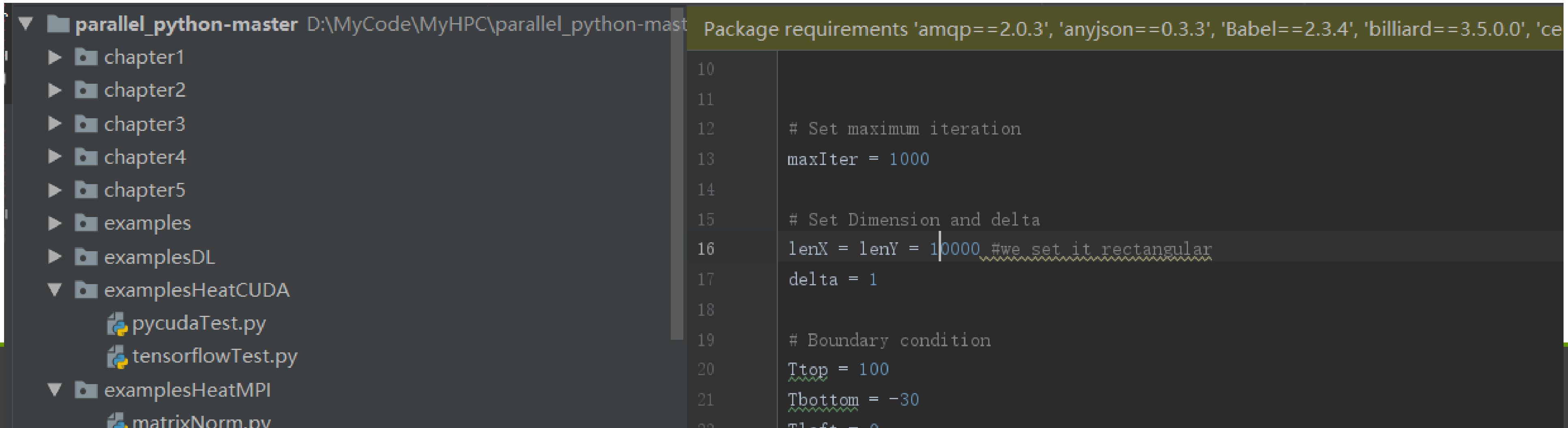


名称	状态	47% CPU	82% 内存	0% 磁盘	0% 网络
PyCharm (5)		23.3%	1,594.6 ...	0 MB/秒	0 Mbps
TeamViewer 14 (32 位)		18.1%	4.7 MB	0 MB/秒	0 Mbps



MateBook D

设备名称	mlinkingHW
处理器	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 2.00 GHz
已安装的 RAM	8.00 GB (7.89 GB 可用)
设备 ID	8E370284-0449-4FE8-B2E3-A62A6C1E6124
产品 ID	00342-30262-00002-AAOEM
系统类型	64 位操作系统, 基于 x64 的处理器

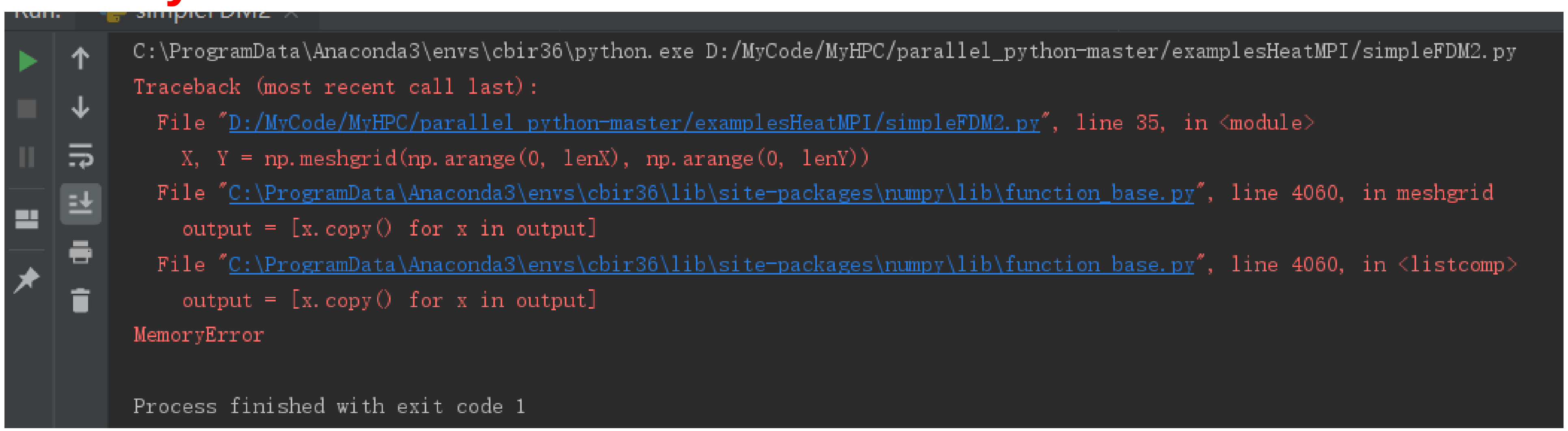


Package requirements 'amqp==2.0.3', 'anyjson==0.3.3', 'Babel==2.3.4', 'billiard==3.5.0.0', 'ce

```


# Set maximum iteration
maxIter = 1000

# Set Dimension and delta
lenX = lenY = 10000 #we set it rectangular
delta = 1

# Boundary condition
Ttop = 100
Tbottom = -30
```

You can try

- **Run the program with different scales**
  - When “lenX = lenY = 100000” + “maxIter = 1000”, the error of “**MemoryError**”!!



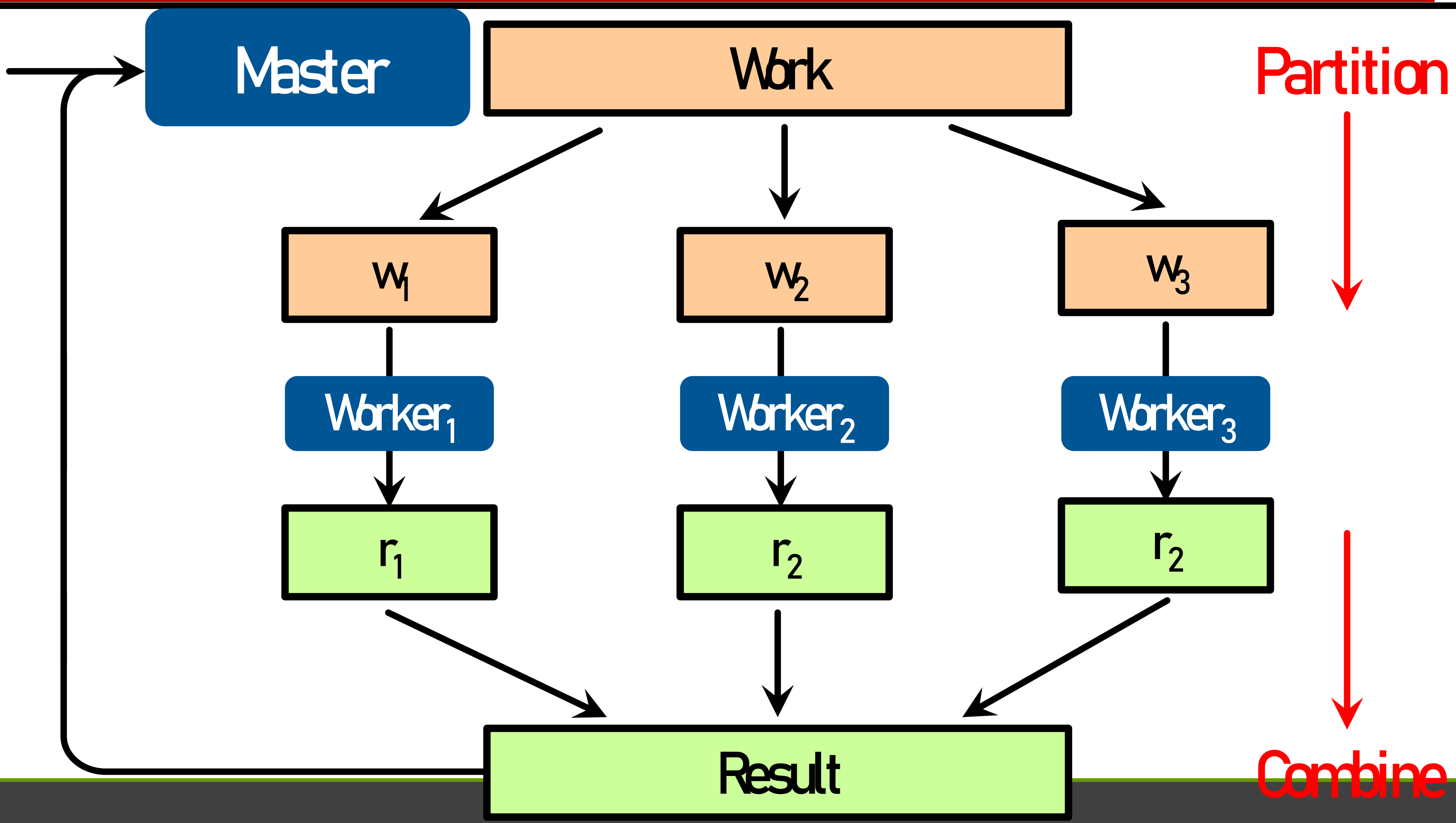
```
C:\ProgramData\Anaconda3\envs\cbir36\python.exe D:/MyCode/MyHPC/parallel_python-master/examplesHeatMPI/simpleFDM2.py
Traceback (most recent call last):
  File "D:/MyCode/MyHPC/parallel_python-master/examplesHeatMPI/simpleFDM2.py", line 35, in <module>
    X, Y = np.meshgrid(np.arange(0, lenX), np.arange(0, lenY))
  File "C:\ProgramData\Anaconda3\envs\cbir36\lib\site-packages\numpy\lib\function_base.py", line 4060, in meshgrid
    output = [x.copy() for x in output]
  File "C:\ProgramData\Anaconda3\envs\cbir36\lib\site-packages\numpy\lib\function_base.py", line 4060, in <listcomp>
    output = [x.copy() for x in output]
MemoryError

Process finished with exit code 1
```

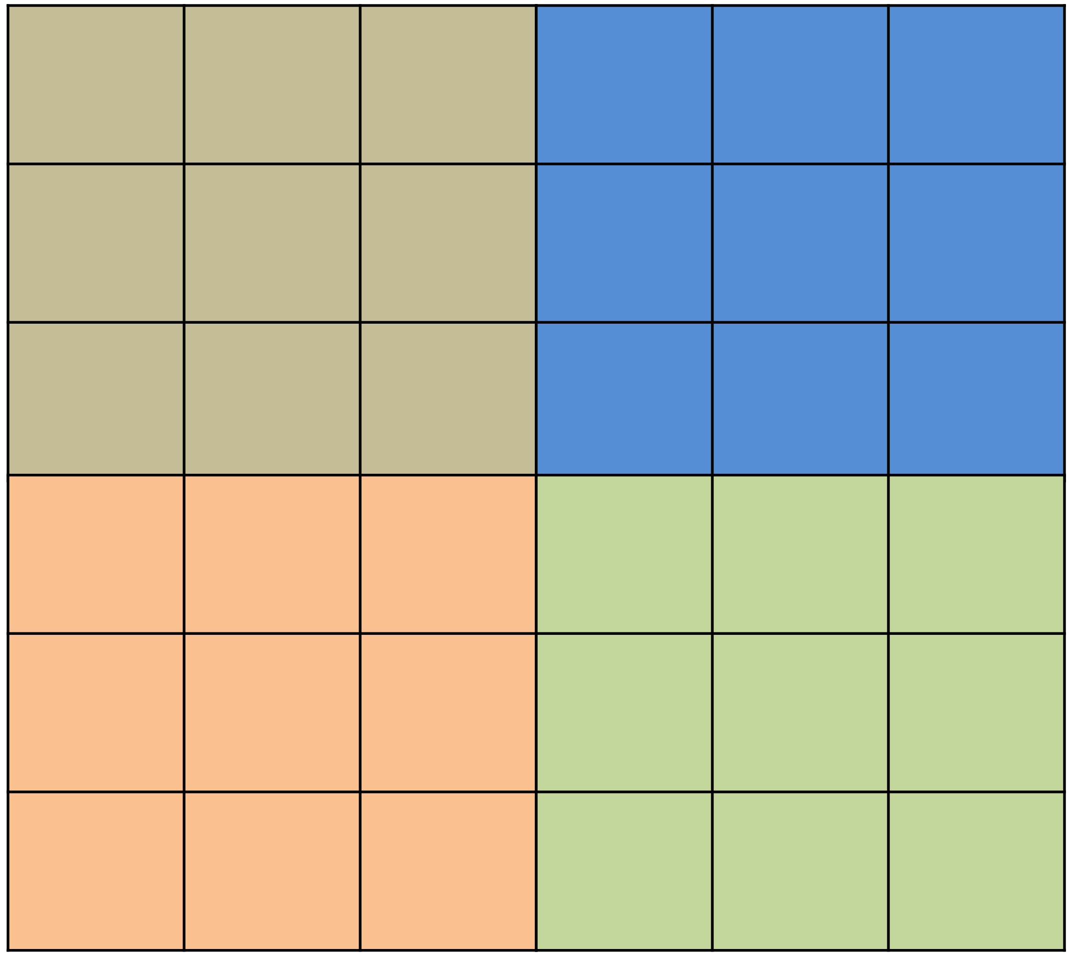
  - How to finish the computation of the weather-forecasting for BeiJing, China, Globe?

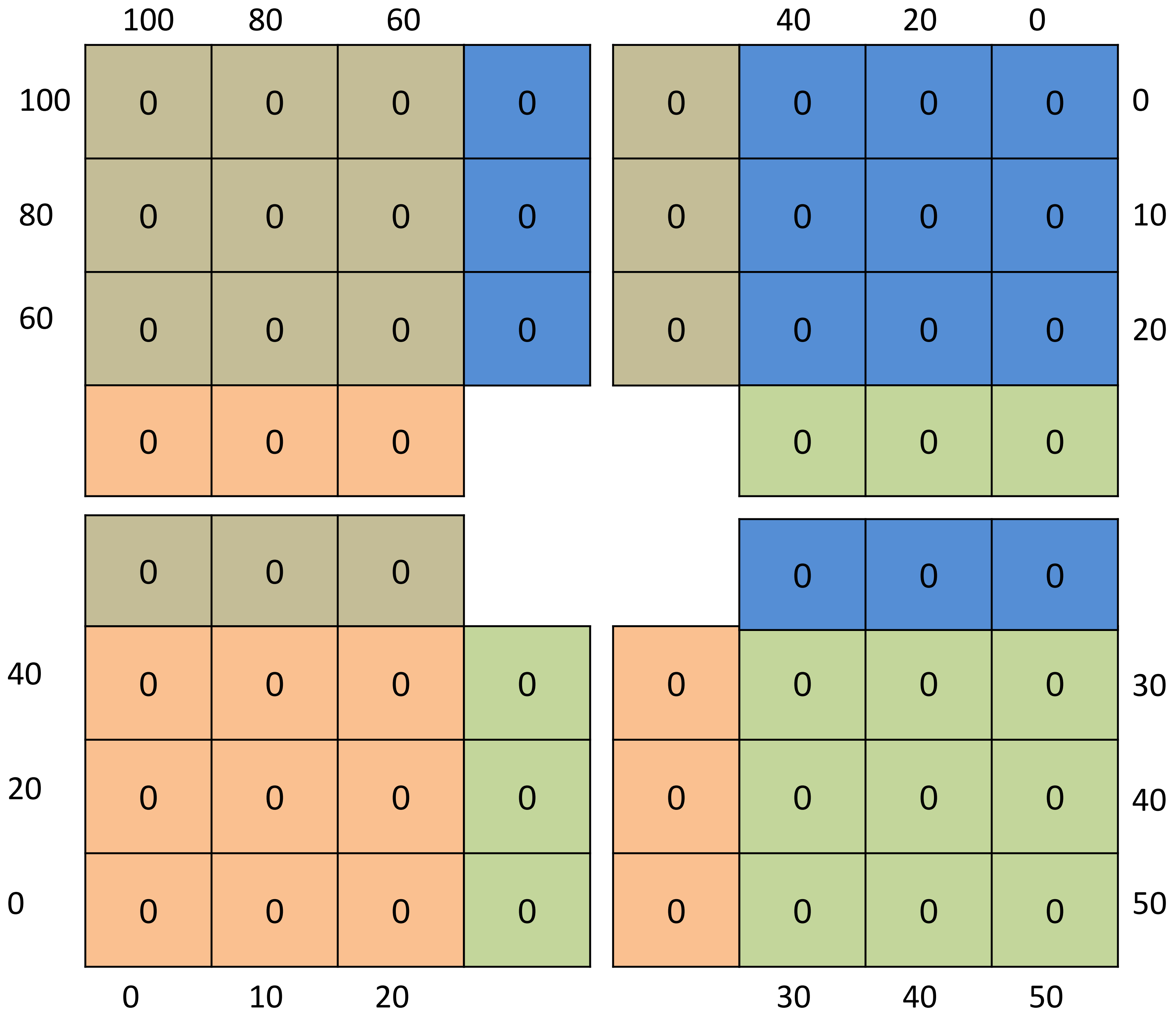
Chapter 2: LSC programming

- **Sequential implementation with Python**
  - "Using Python to Solve Computational Physics Problems"
- **Ideas to convert Sequential to Parallel**
  - DAOM, PCAM
- **Measure the performance**

LSC programming?
– Decomposition & Collaboration



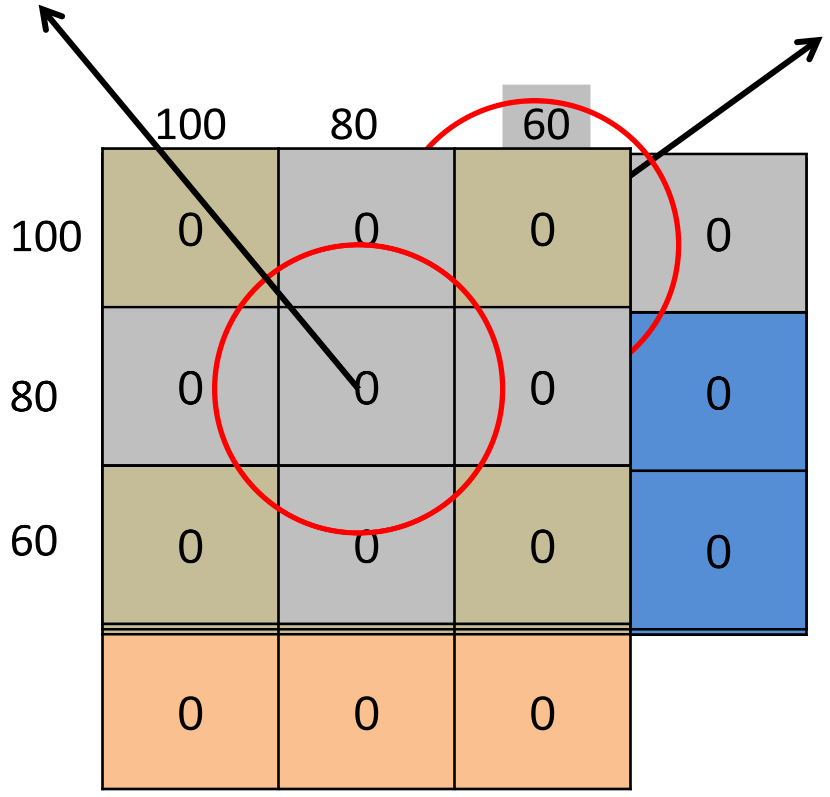



100
80
60
40
20
0
100
80
60
40
20
0
0
10
20
30
40
50
0
10
20
30
40
50

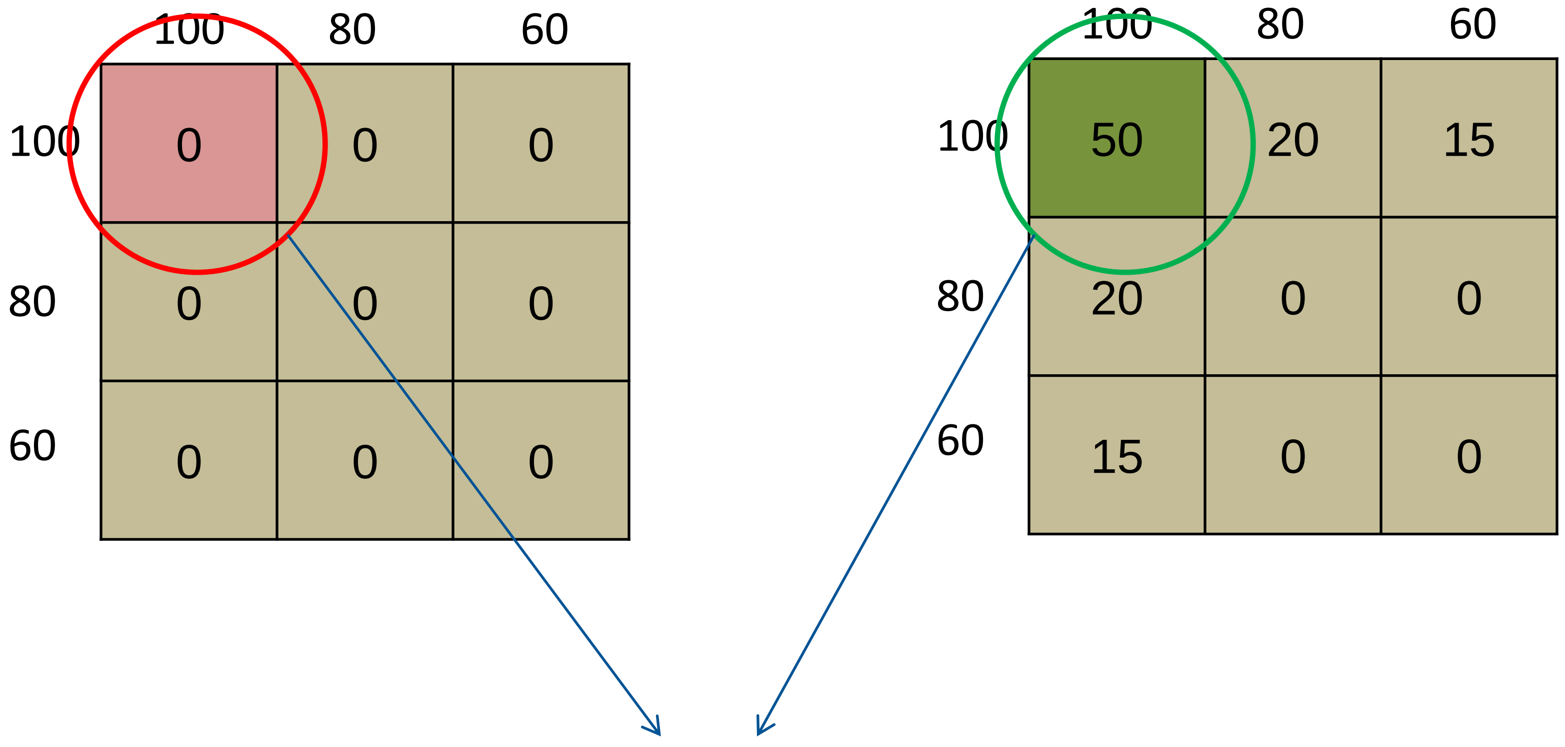
Calculate the value of each cell by averaging its 4 neighboring cells

$$\frac{0+0+0+0}{4} = \frac{0}{4} = 0$$

$$\frac{60+0+0+0}{4} = \frac{60}{4} = 15$$



Calculate the difference between the previous cell values and new cell values

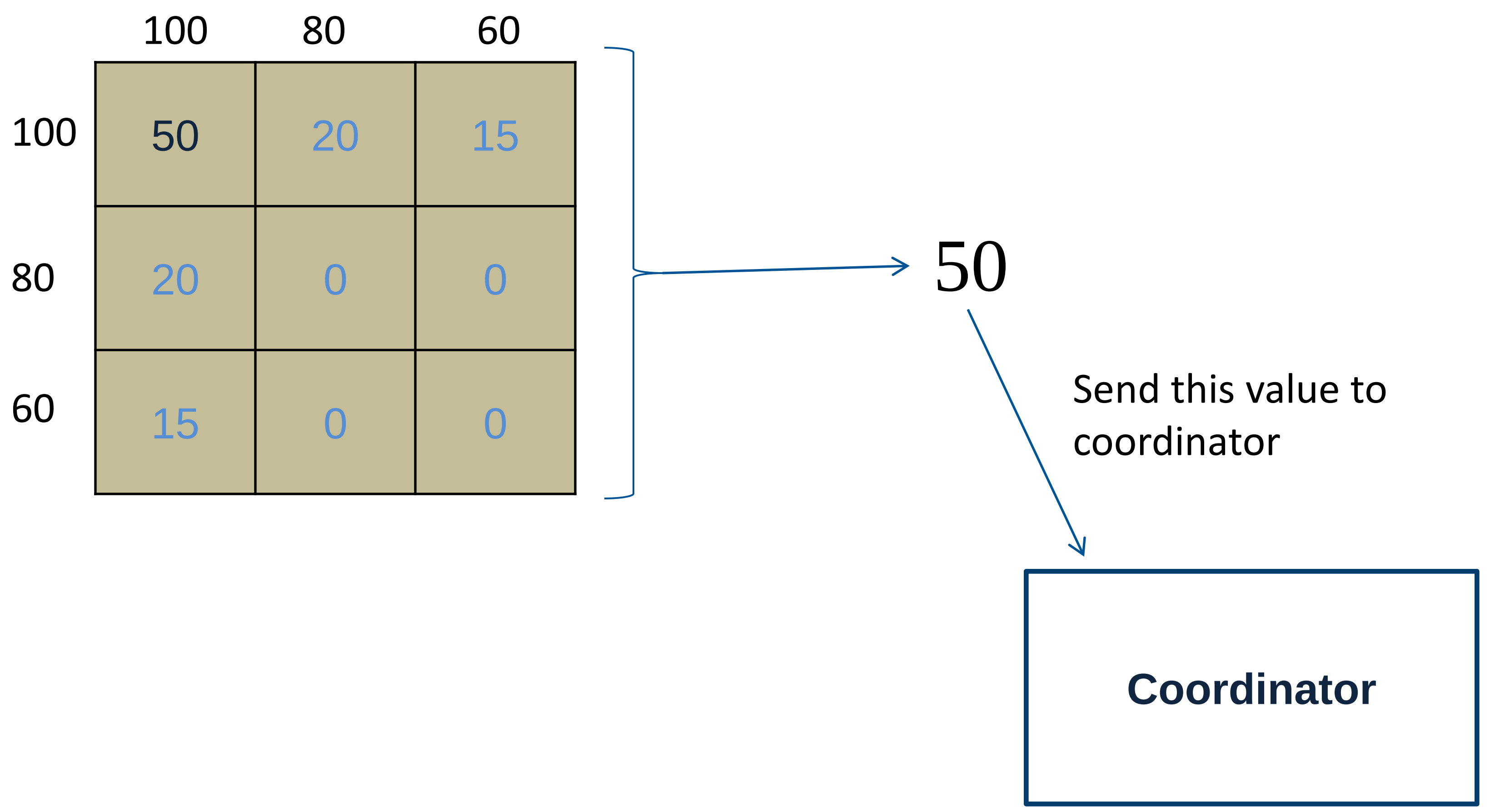


$$\begin{array}{c|ccc} & 100 & 80 & 60 \\ \hline 100 & 0 & 0 & 0 \\ 80 & 0 & 0 & 0 \\ 60 & 0 & 0 & 0 \end{array}$$

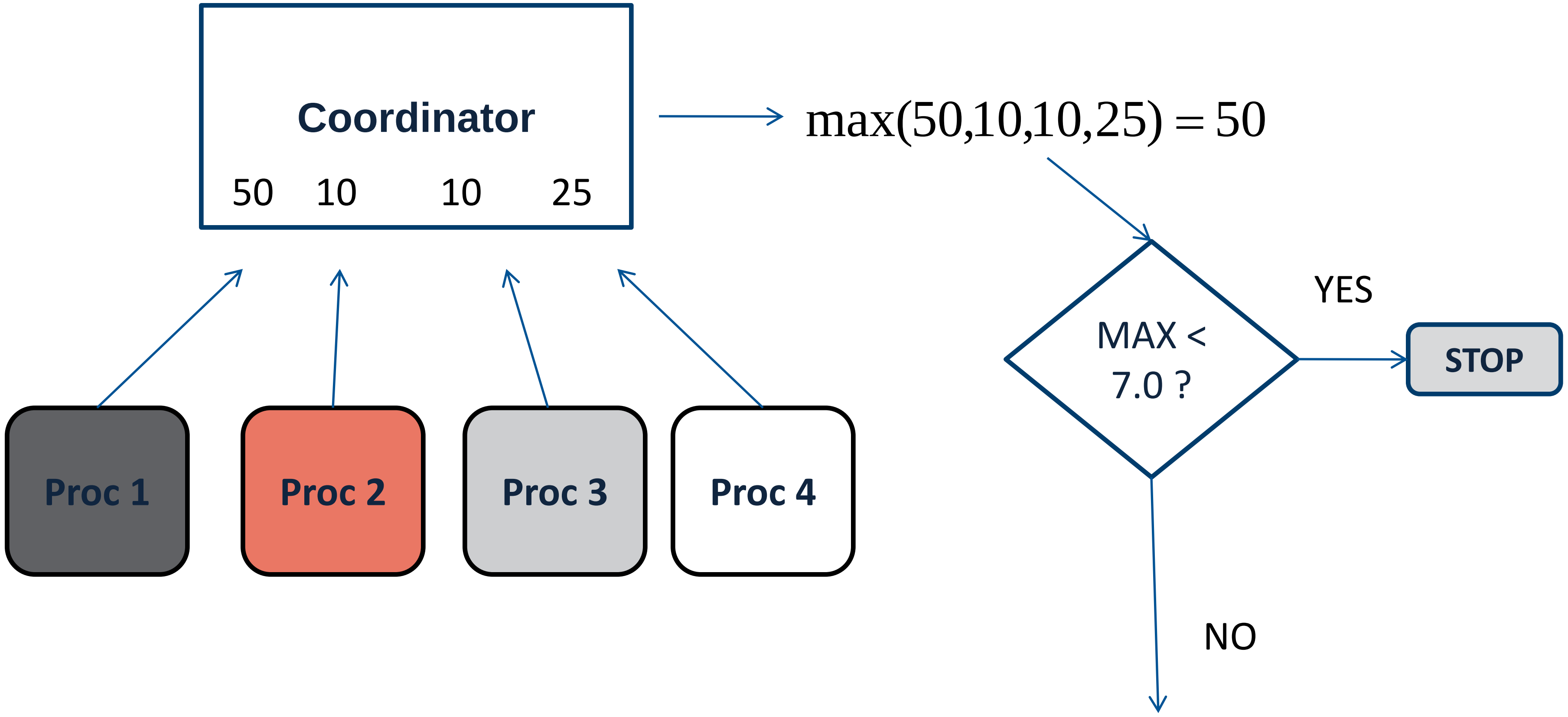
$$\begin{array}{c|ccc} & 100 & 80 & 60 \\ \hline 100 & 50 & 20 & 15 \\ 80 & 20 & 0 & 0 \\ 60 & 15 & 0 & 0 \end{array}$$

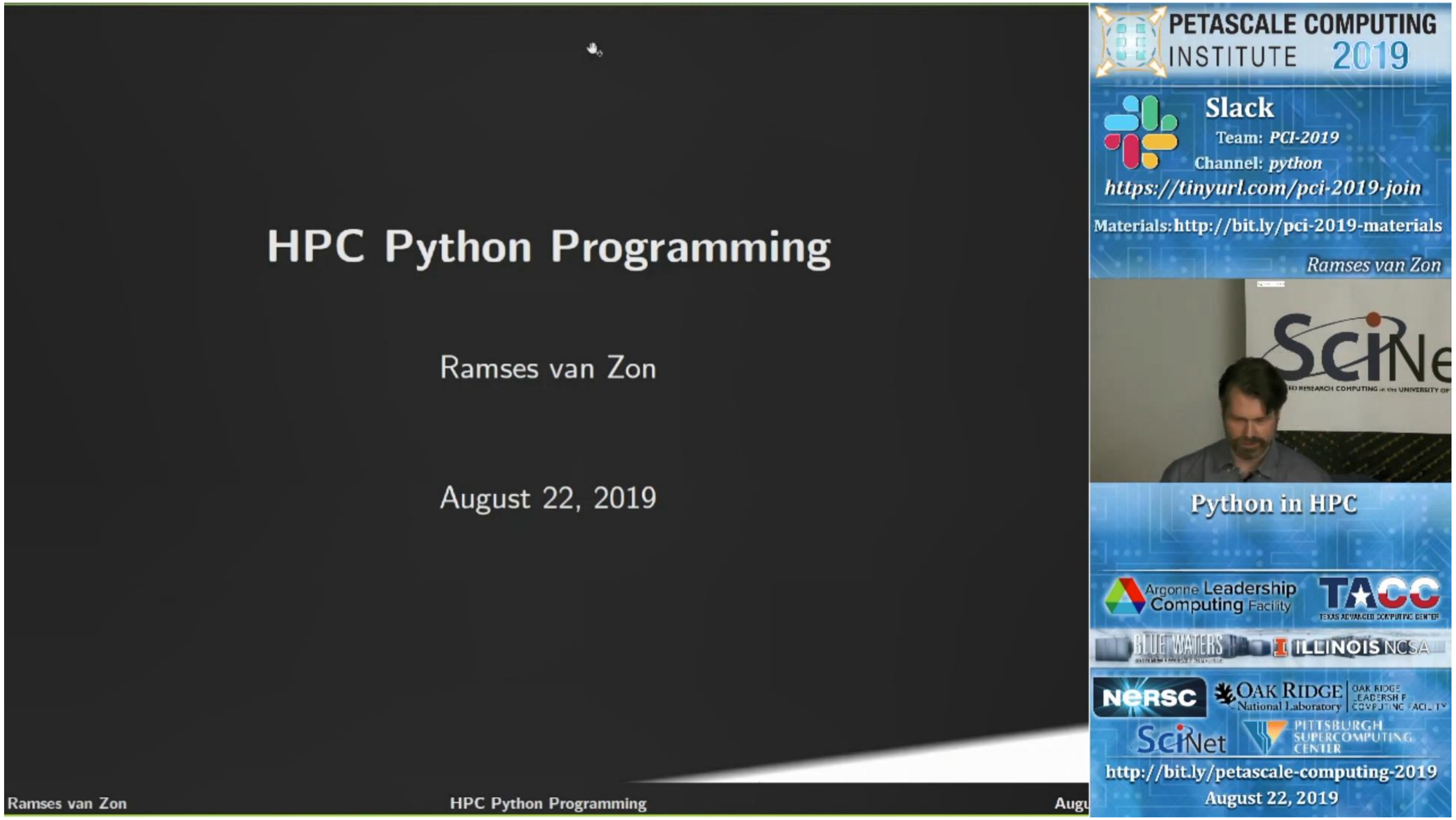
$$|50 - 0| = 50$$

After computing the difference for each cell, Determine the Maximum Temperature ACROSS your problem chunk



Coordinator Waits for all processing elements to send their values and determines the maximum of all the values it receives





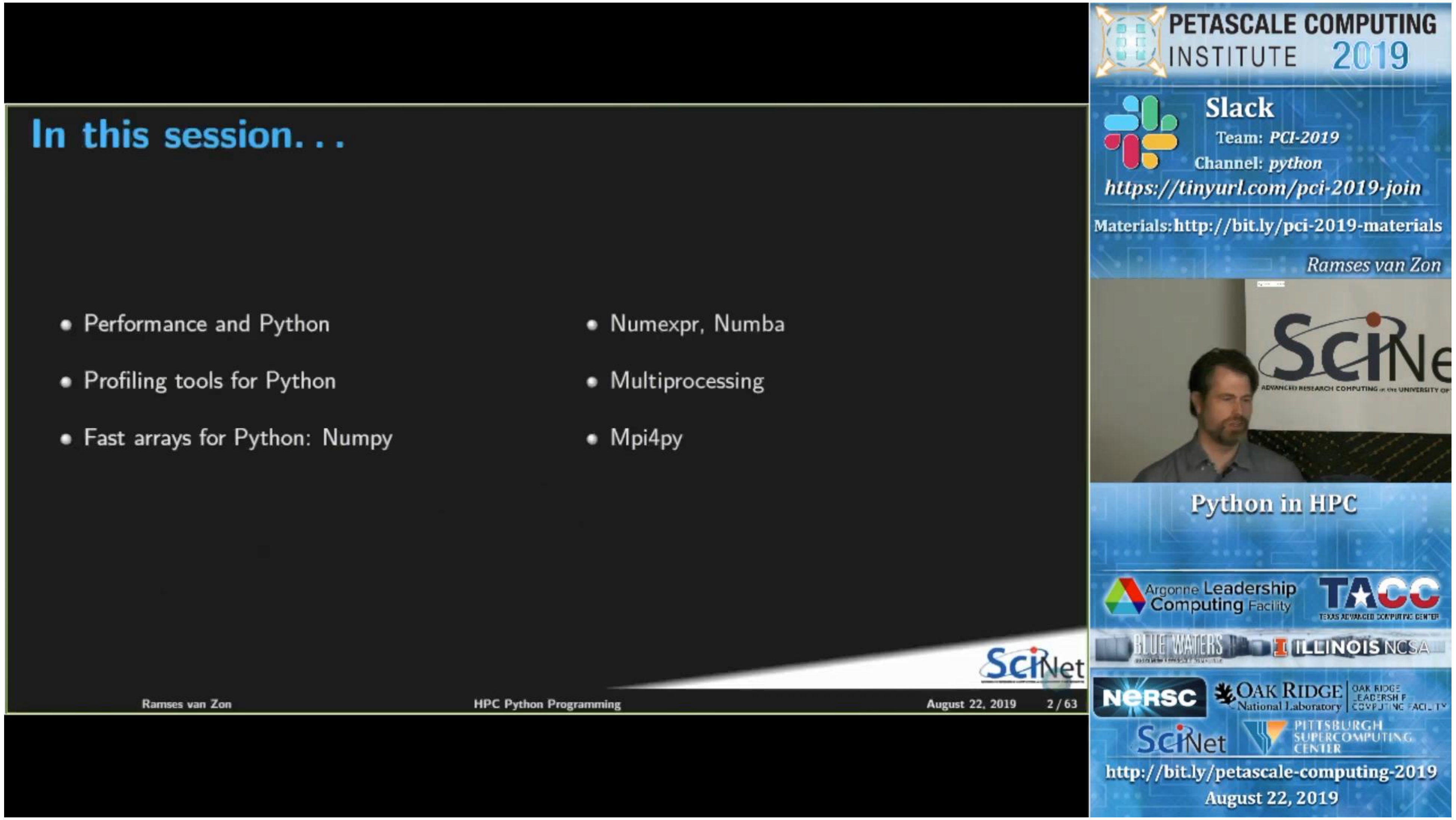
HPC Python Programming

Ramses van Zon

August 22, 2019

Ramses van Zon HPC Python Programming Augu

PETASCALE COMPUTING INSTITUTE 2019
Slack
Team: PCI-2019
Channel: python
https://tinyurl.com/pci-2019-join
Materials: http://bit.ly/pci-2019-materials
Ramses van Zon
Python in HPC
Argonne Leadership Computing Facility
TACC
BLUE WATERS
ILLINOIS NCSA
NERSC
OAK RIDGE National Laboratory
SciNet
PITTSBURGH SUPERCOMPUTING CENTER
http://bit.ly/petascale-computing-2019
August 22, 2019



In this session. . .

- Performance and Python
- Profiling tools for Python
- Fast arrays for Python: Numpy
- Numexpr, Numba
- Multiprocessing
- Mpi4py

Ramses van Zon HPC Python Programming August 22, 2019 2 / 63

PETASCALE COMPUTING INSTITUTE 2019

Slack
Team: *PCI-2019*
Channel: *python*
https://tinyurl.com/pci-2019-join

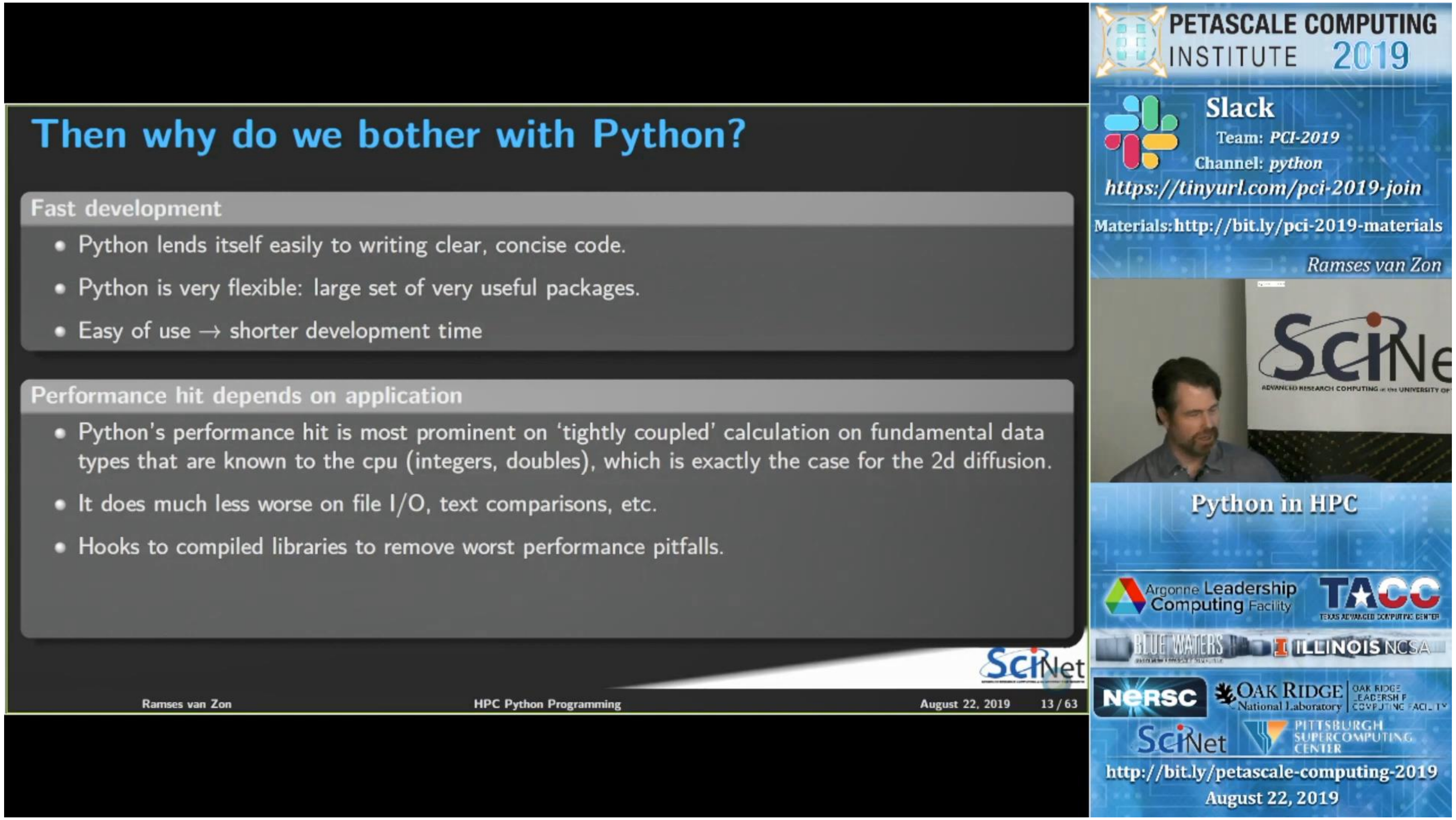
Materials: http://bit.ly/pci-2019-materials

Ramses van Zon

Python in HPC

http://bit.ly/petascale-computing-2019
August 22, 2019

SciNet
Argonne Leadership Computing Facility
TACC
Blue Waters
Illinois NCSA
NERSC
Oak Ridge National Laboratory
Pittsburgh Supercomputing Center



Then why do we bother with Python?

Fast development

- Python lends itself easily to writing clear, concise code.
- Python is very flexible: large set of very useful packages.
- Easy of use → shorter development time

Performance hit depends on application

- Python's performance hit is most prominent on 'tightly coupled' calculation on fundamental data types that are known to the cpu (integers, doubles), which is exactly the case for the 2d diffusion.
- It does much less worse on file I/O, text comparisons, etc.
- Hooks to compiled libraries to remove worst performance pitfalls.

Ramses van Zon HPC Python Programming August 22, 2019 13 / 63

PETASCALE COMPUTING INSTITUTE 2019
Slack
Team: PCI-2019
Channel: python
https://tinyurl.com/pci-2019-join
Materials: http://bit.ly/pci-2019-materials
Ramses van Zon
Python in HPC
http://bit.ly/petascale-computing-2019
August 22, 2019

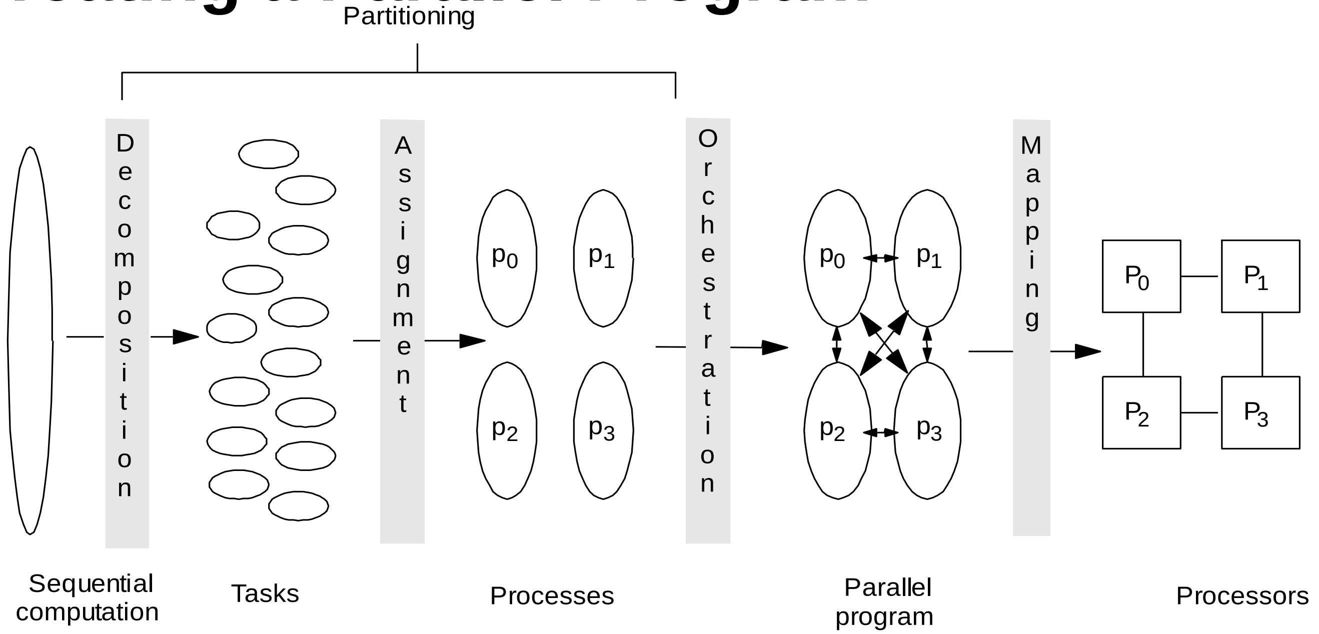
Chapter 2: LSC programming

- **Sequential implementation with Python**
  - "Using Python to Solve Computational Physics Problems"
- **Ideas to convert Sequential to Parallel**
  - DAOM, PCAM
- **Measure the performance**

DAOM (1991)

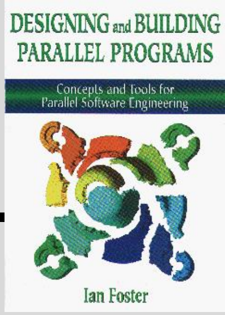
K. Mani Chandy, Stephen Taylor. An Introduction to Parallel Programming. Jones and Bartlett. Publishers, Inc., Burlington. 1991.

4 Steps in Creating a Parallel Program

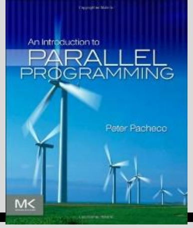


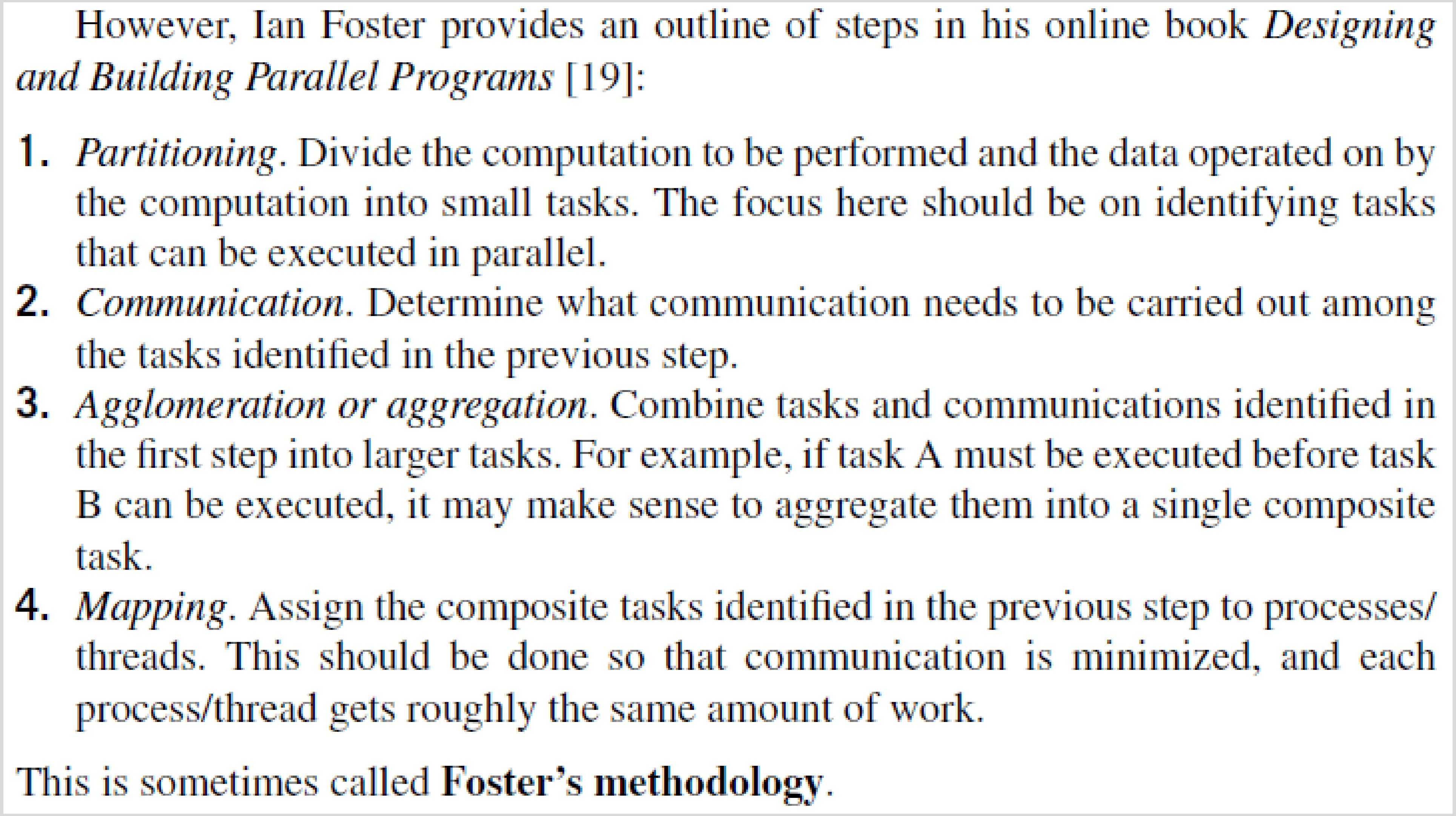
- **D**ecomposition of computation in tasks
- **A**ssignment of tasks to processes
- **O**rchestration of data access, comm, synch.
- **M**apping processes to processors

Foster's model – PCAM (1995)


DESIGNING and BUILDING PARALLEL PROGRAMS
Concepts and Tools for Parallel Software Engineering
Ian Foster

- **In "*Designing and Building Parallel Programs*" Ian Foster proposes a model with tasks that interact with each other by communicating through channels.**
  - A **task** is a program, its local memory, and its communication in-ports and out-ports.
  - A **channel** connects a task's in-port to another task's out-port.
  - Channels are buffered. Sending is **asynchronous** while receiving is **synchronous** (receiving task is blocked until expected message arrives).





However, Ian Foster provides an outline of steps in his online book *Designing and Building Parallel Programs* [19]:

1. *Partitioning*. Divide the computation to be performed and the data operated on by the computation into small tasks. The focus here should be on identifying tasks that can be executed in parallel.
2. *Communication*. Determine what communication needs to be carried out among the tasks identified in the previous step.
3. *Agglomeration or aggregation*. Combine tasks and communications identified in the first step into larger tasks. For example, if task A must be executed before task B can be executed, it may make sense to aggregate them into a single composite task.
4. *Mapping*. Assign the composite tasks identified in the previous step to processes/threads. This should be done so that communication is minimized, and each process/thread gets roughly the same amount of work.

This is sometimes called **Foster's methodology**.

PCAM设计方法学

任务划分(**Partitioning**)

- 将整个计算分解为一些小任务，其目的是尽量开拓并行执行的机会

通信(**Communication**)

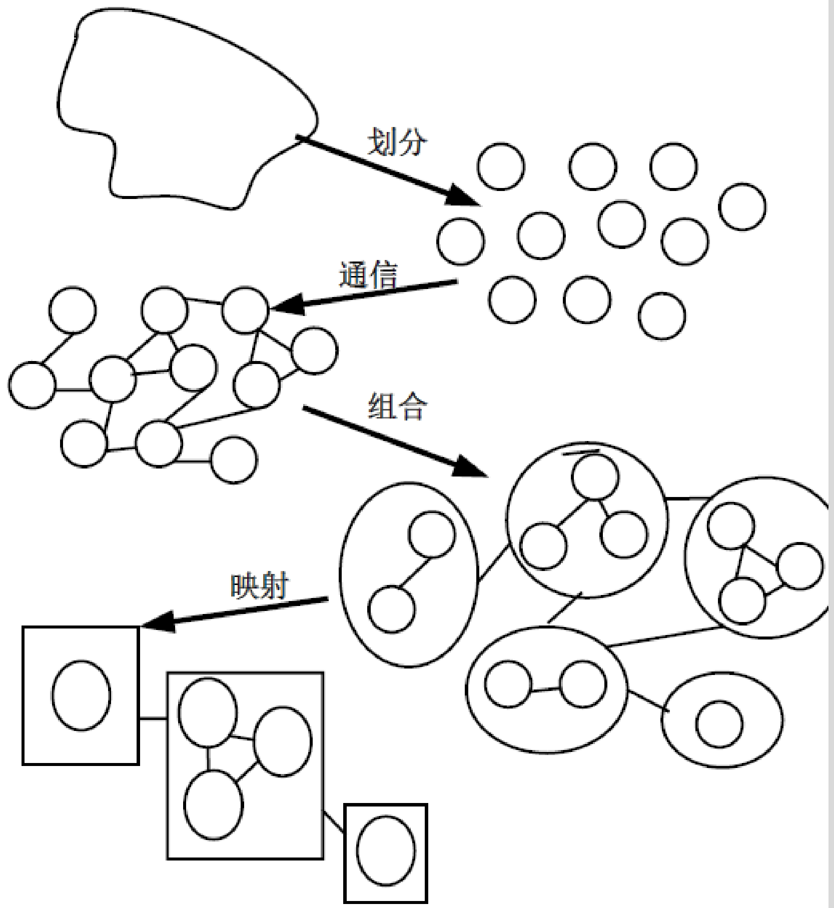
- 确定诸任务执行中所需要交换的数据和协调诸任务的执行，由此检测上述划分的合理性

任务组合(**Agglomeration**)

- 按性能要求和实现的代价来考察前两阶段的结果，必要时可将一些小任务组合成更大的任务以提高性能和减少通信开销

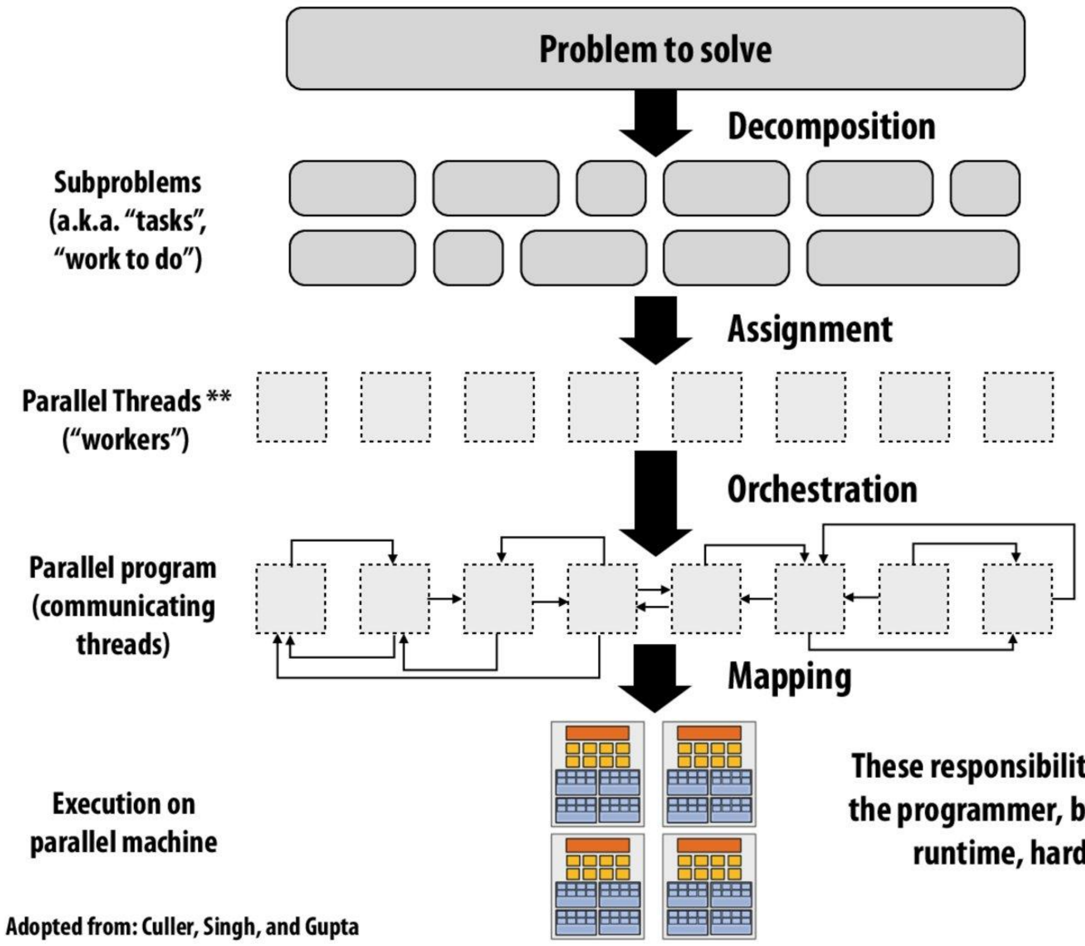
处理器映射(**Mapping**)

- 将每个任务分配到一个处理器上，其目的是最小化全局执行时间和通信成本以及最大化处理器的利用率



DAOM/PCAM

Creating a parallel program

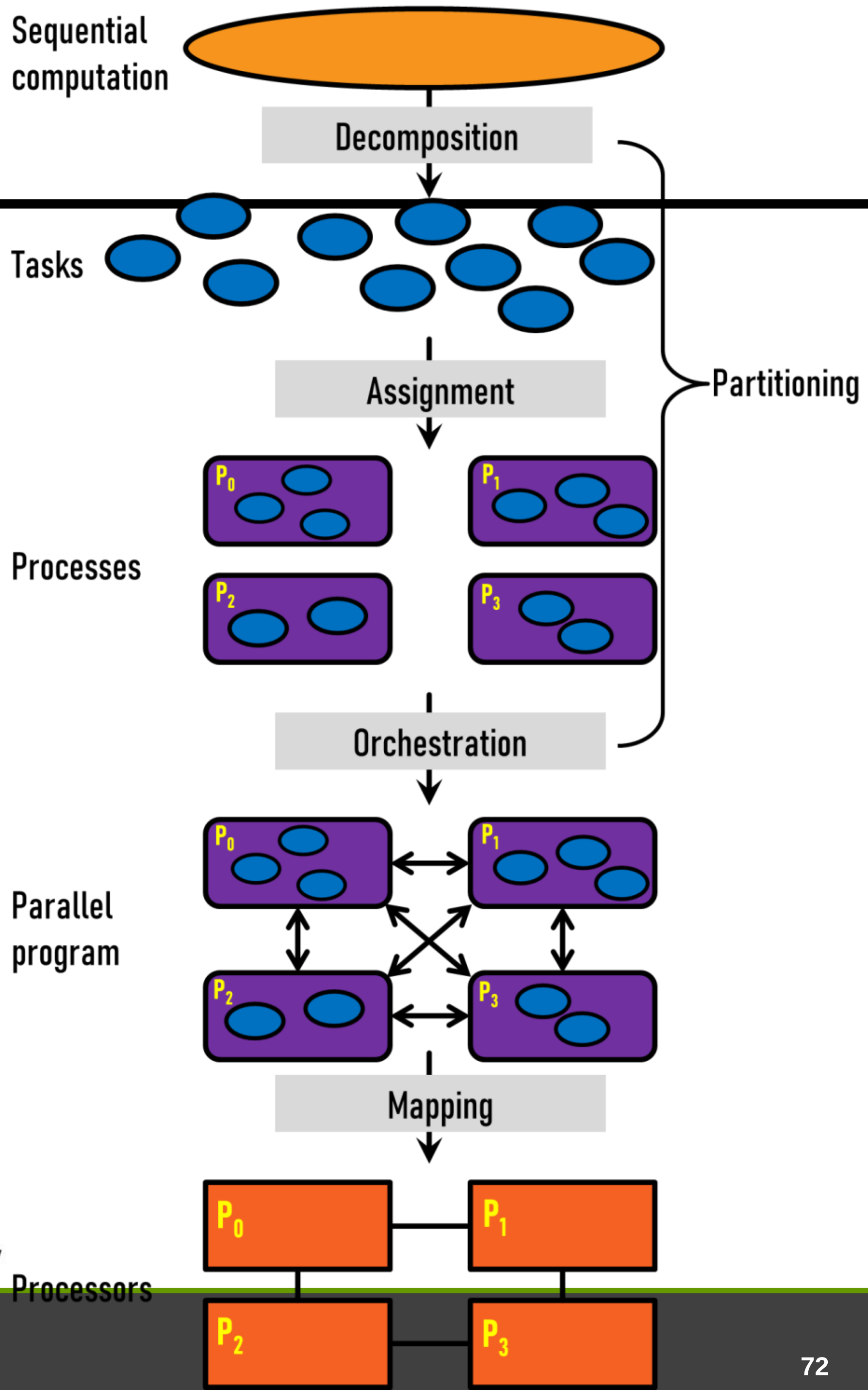

Problem to solve
Decomposition
Subproblems (a.k.a. "tasks", "work to do")
Assignment
Parallel Threads ** ("workers")
Orchestration
Parallel program (communicating threads)
Mapping
Execution on parallel machine

** I had to pick a term

These responsibilities may be assumed by the programmer, by the system (compiler, runtime, hardware), or by both!

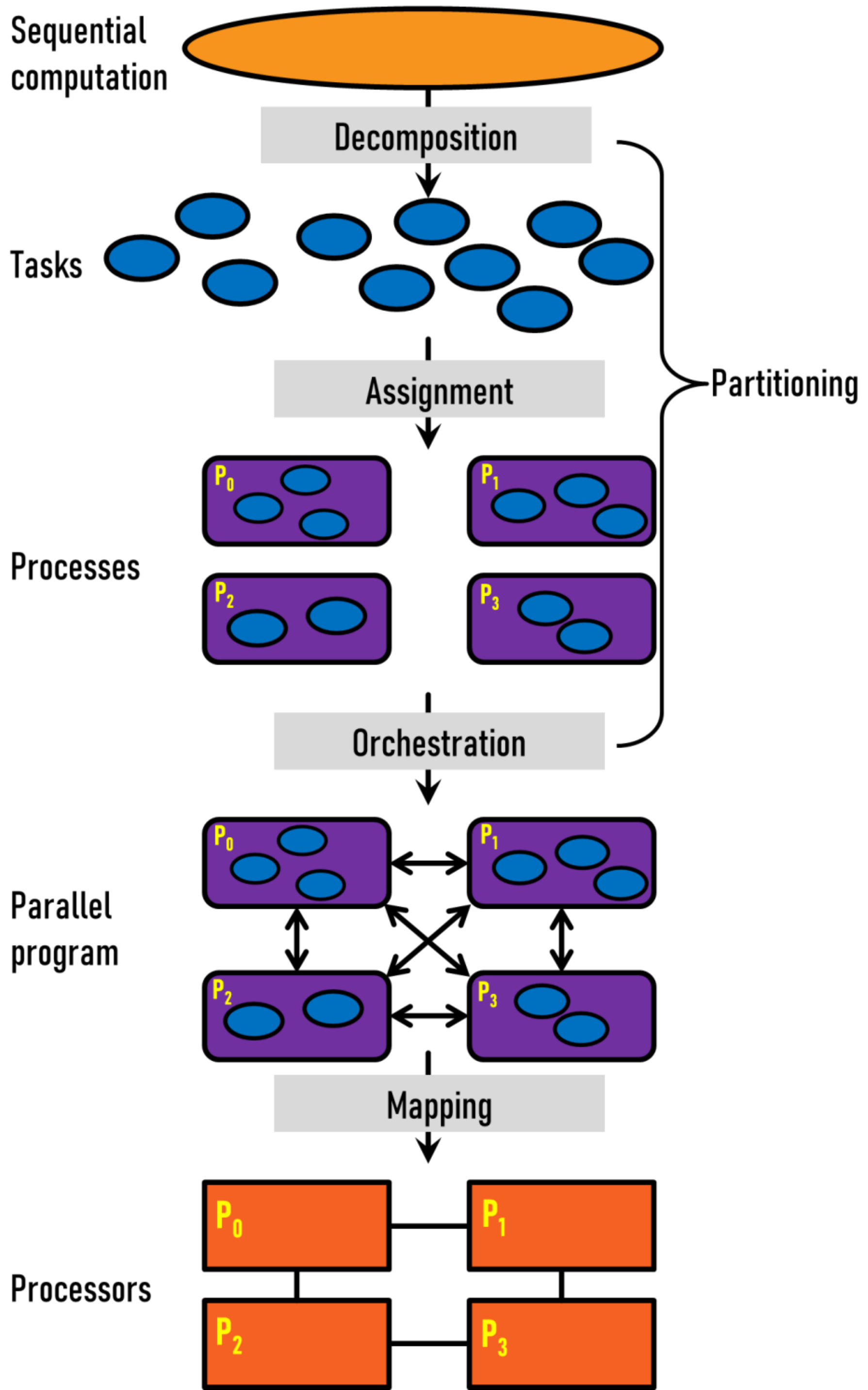
Adopted from: Culler, Singh, and Gupta

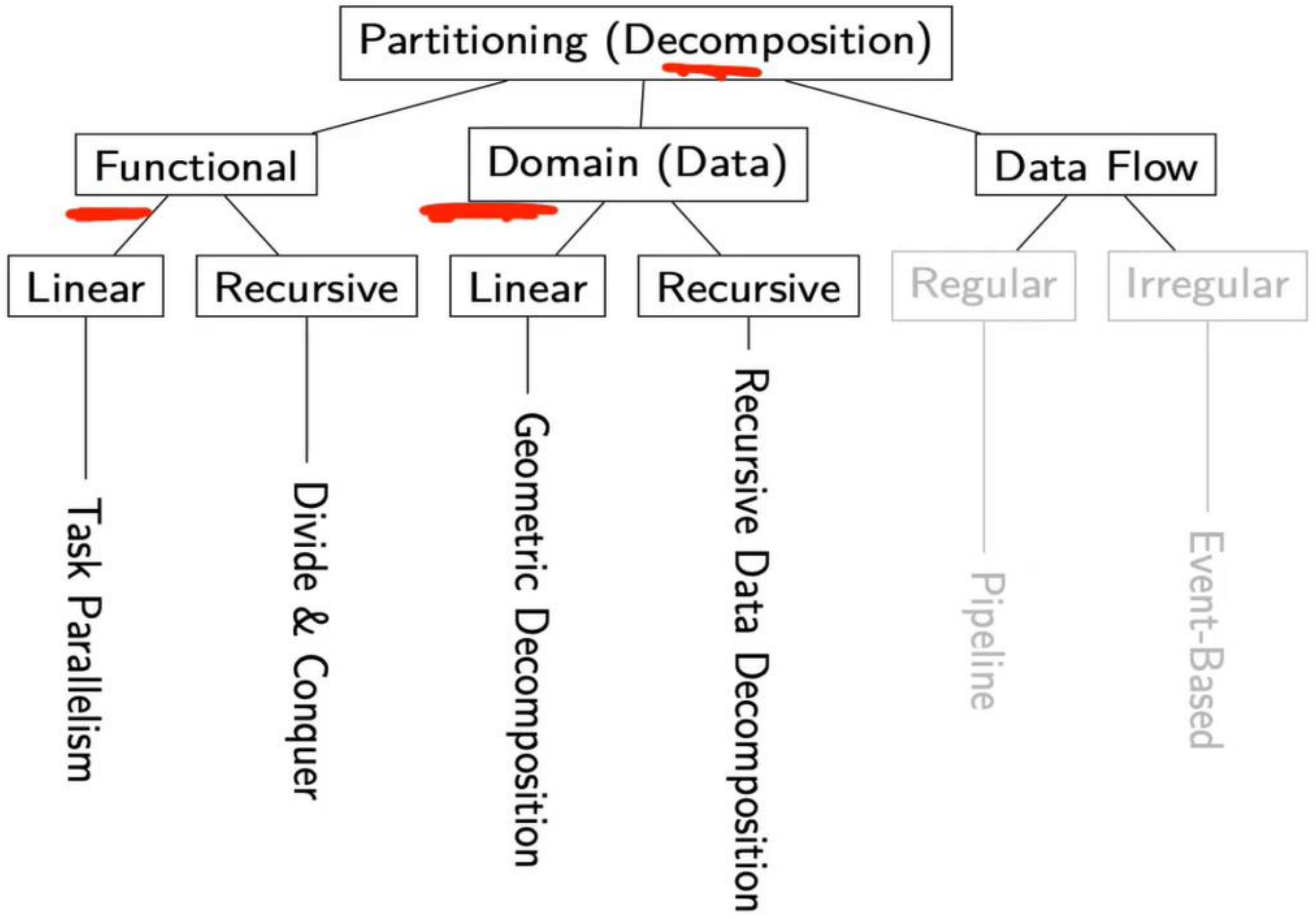
CMU / 清华大学, Summer 2017

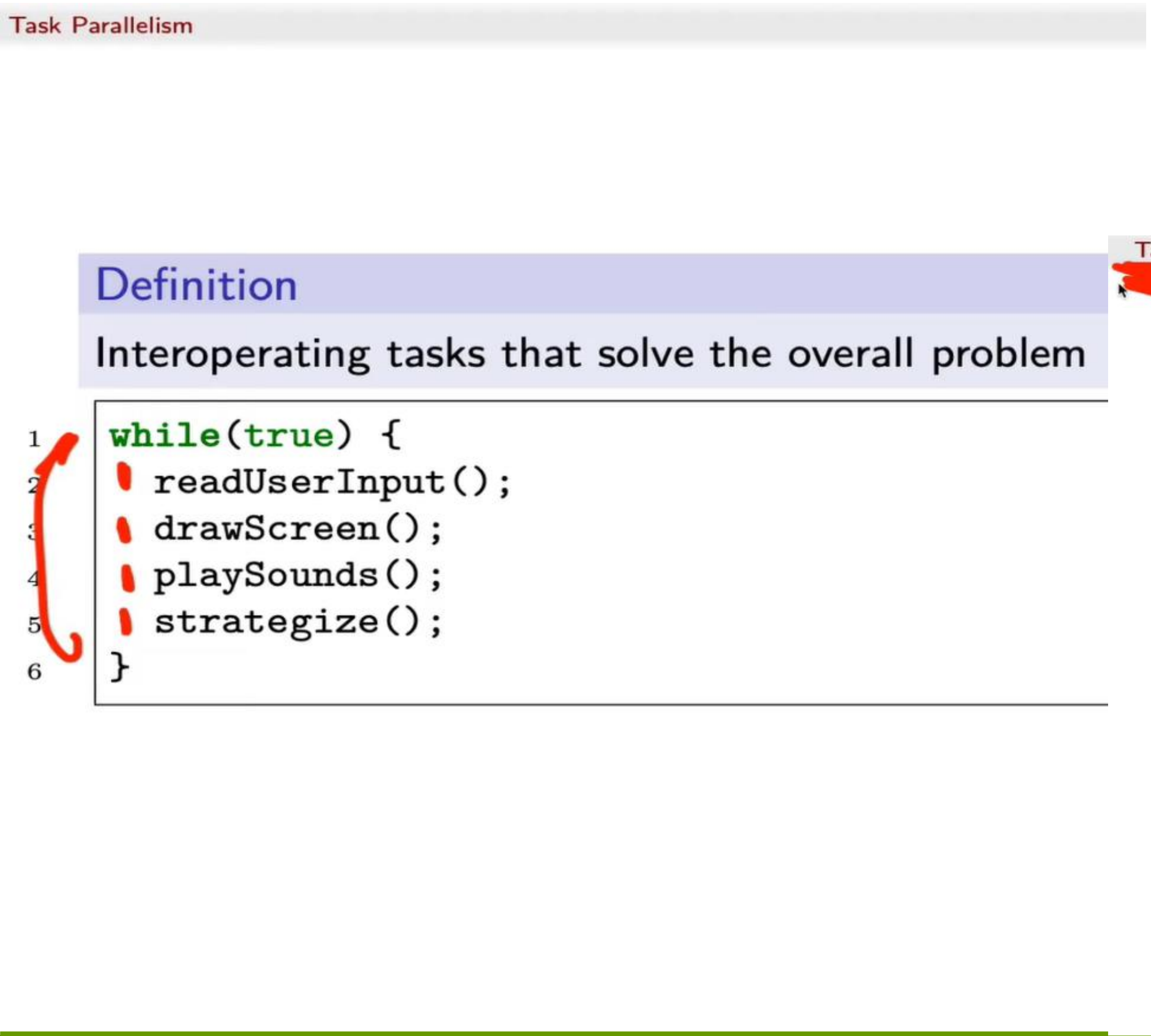


Partition/Decompose

- **划分 (Partition/Decompose)**
  - 充分开拓算法的并发性和可扩放性；
  - 先进行数据分解(称域分解)，再进行计算功能的分解(称功能分解)；
  - 使数据集和计算集互不相交；
  - 划分阶段忽略处理器数目和目标机器的体系结构；
  - 能分为两类划分：
    - 域分解(Domain Decomposition)
    - 功能分解(Functional Decomposition)



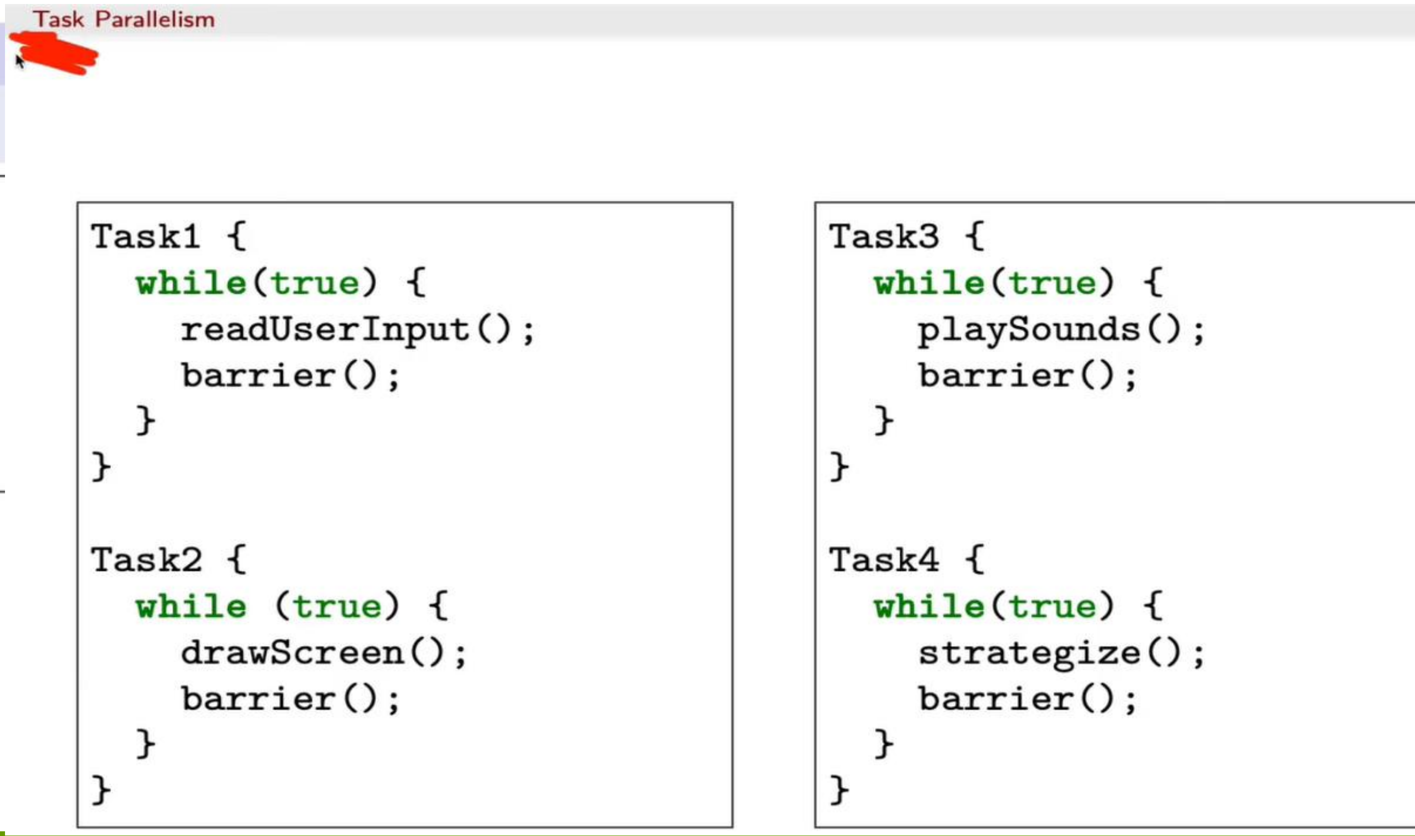

Partitioning (Decomposition)
Functional
Domain (Data)
Data Flow
Linear
Recursive
Linear
Recursive
Regular
Irregular
Task Parallelism
Divide & Conquer
Geometric Decomposition
Recursive Data Decomposition
Pipeline
Event-Based


Task Parallelism

Definition

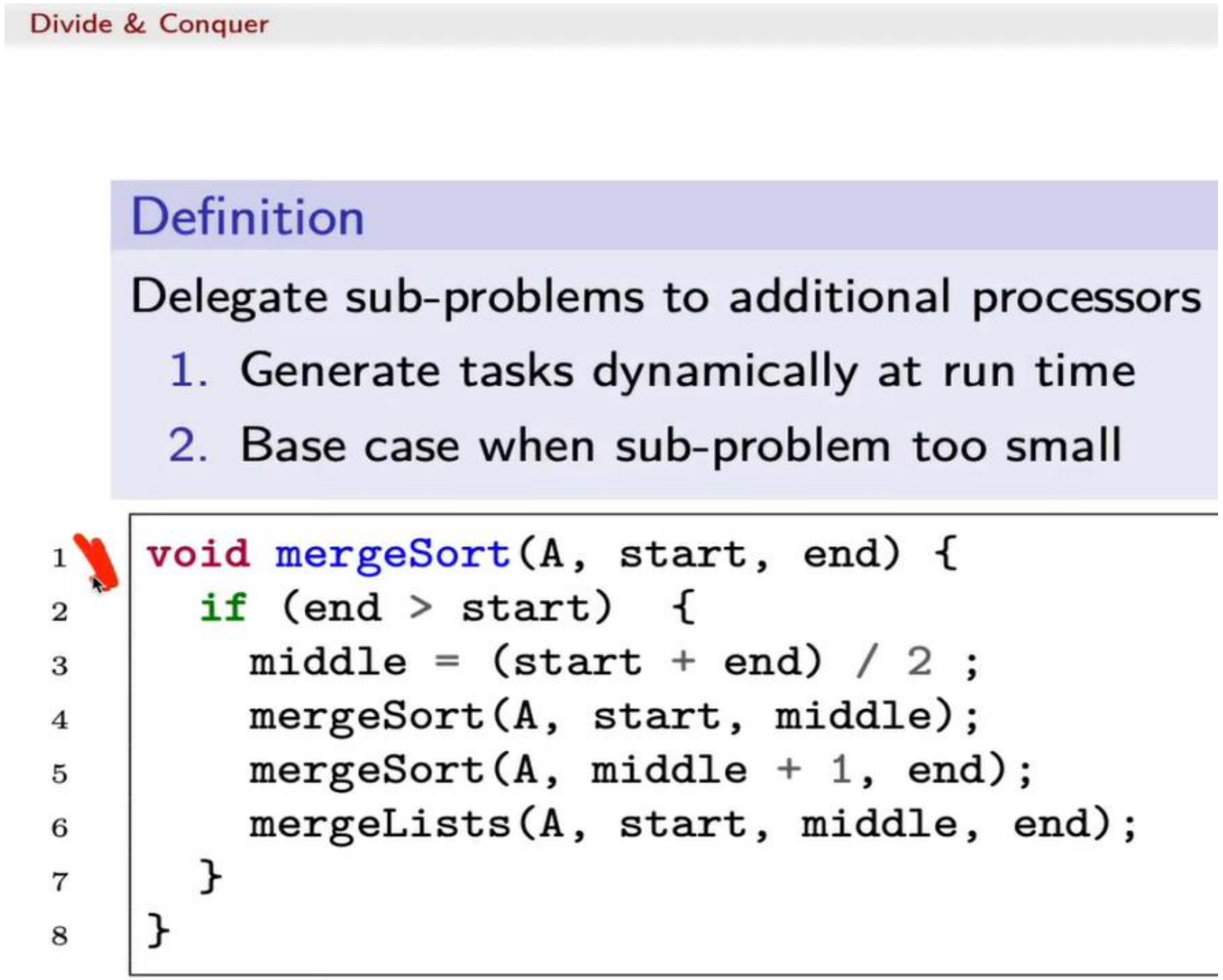
Interoperating tasks that solve the overall problem

```
while(true) {
  readUserInput();
  drawScreen();
  playSounds();
  strategize();
}
```


Task Parallelism

```
Task1 {
  while(true) {
    readUserInput();
    barrier();
  }
}

Task2 {
  while (true) {
    drawScreen();
    barrier();
  }
}
```

```
Task3 {
  while(true) {
    playSounds();
    barrier();
  }
}

Task4 {
  while(true) {
    strategize();
    barrier();
  }
}
```



Definition

Delegate sub-problems to additional processors

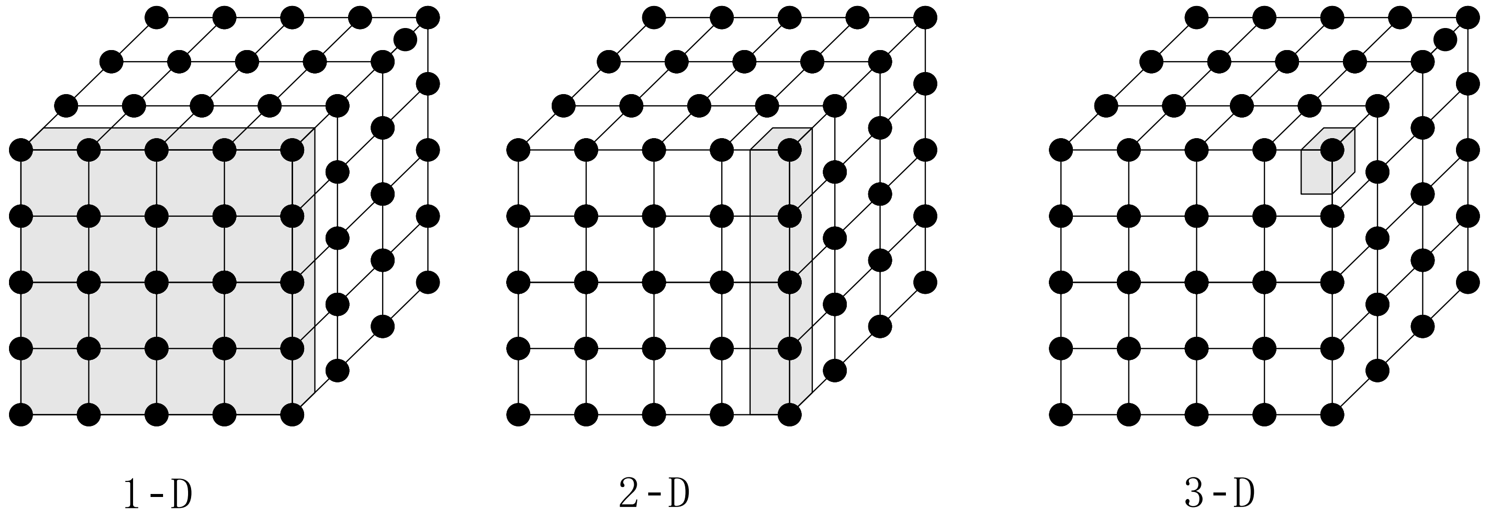
1. Generate tasks dynamically at run time
2. Base case when sub-problem too small

```
void mergeSort(A, start, end) {
  if (end > start)  {
    middle = (start + end) / 2 ;
    mergeSort(A, start, middle);
    mergeSort(A, middle + 1, end);
    mergeLists(A, start, middle, end);
  }
}
```

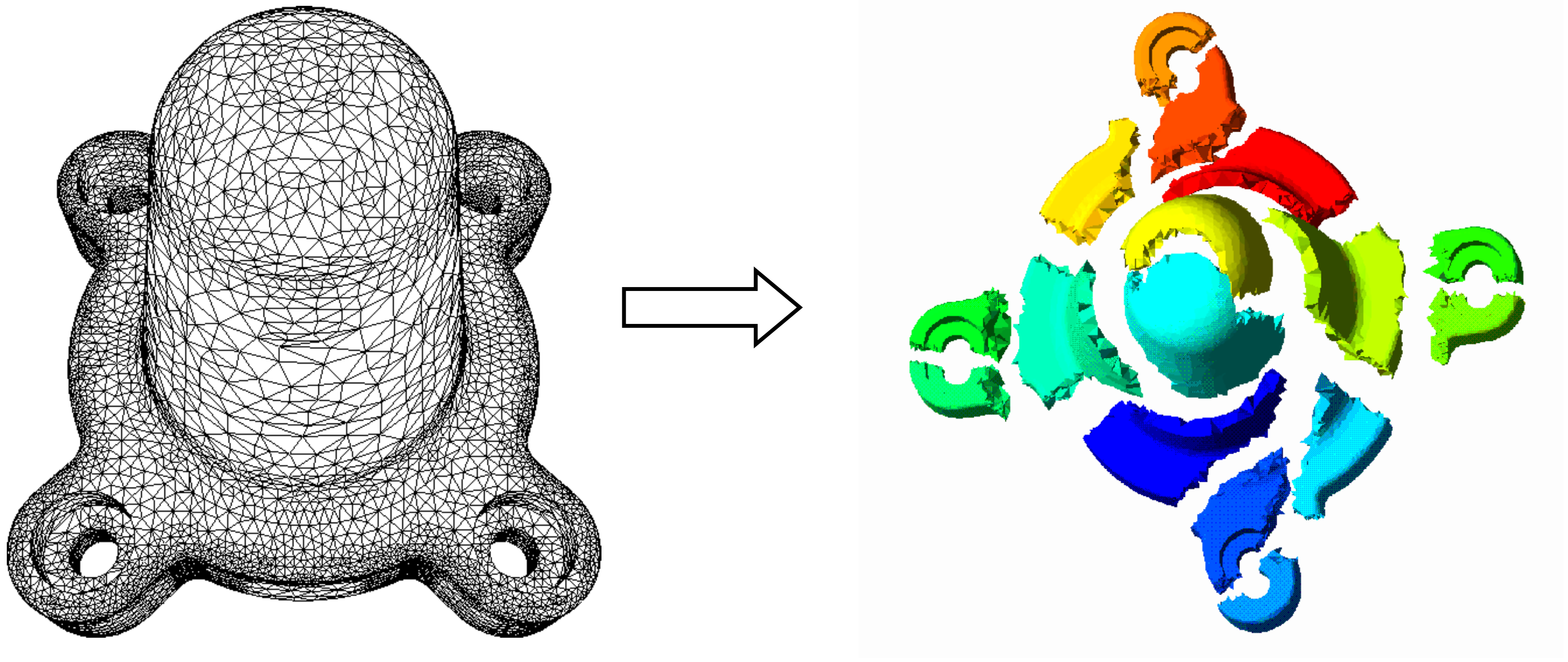
域分解

- 划分的对象是数据，可以是程序中的输入数据、中间处理数据和输出数据；
- 将数据分解成大致相等的小数据片；
- 划分时考虑数据上的相应操作；
- 如果一个任务需要别的任务中的数据，则会产生任务间的通信；

示例：三维网格的域分解，各格点上计算都是重复的。下图是三种分解方法



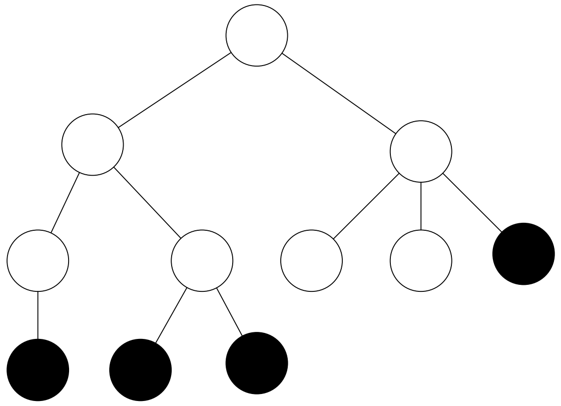
□ 不规则区域的分解示例



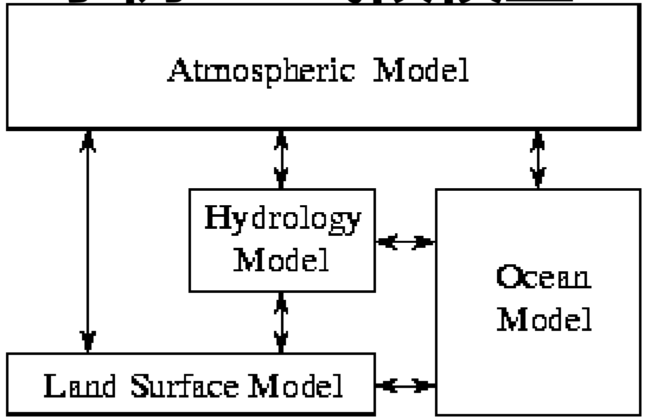
功能分解

- 划分的对象是计算（亦称为任务分解或计算划分），将计算划分为不同的任务，其出发点不同于域分解；
- 划分后，研究不同任务所需的数据。如果这些数据不相交的，则划分是成功的；如果数据有相当的重叠， 意味着存在大量的通信开销，要重新进行域分解和功能分解；
- 功能分解是一种更深层次的分解

示例1：搜索树

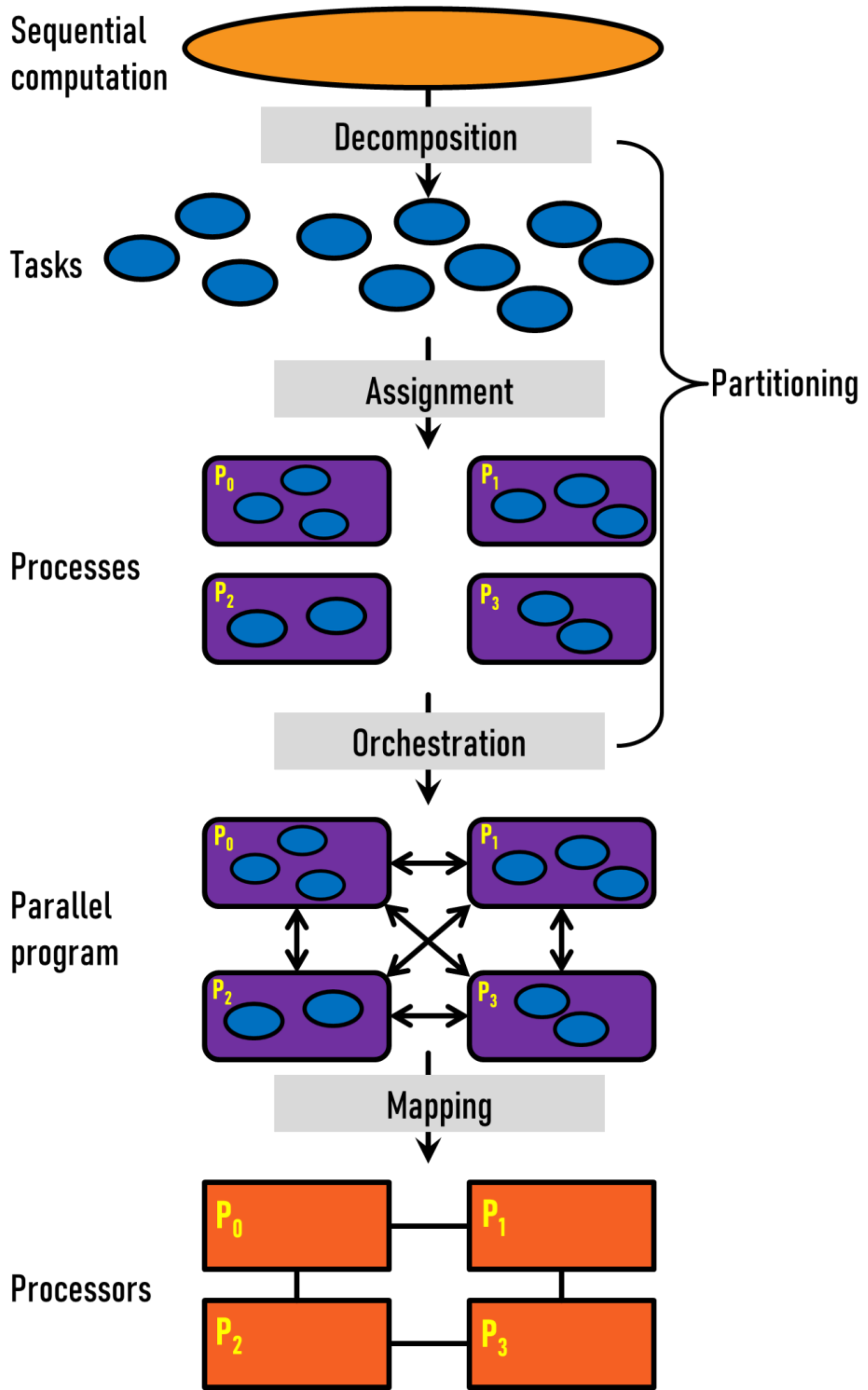


示例2：气候模型



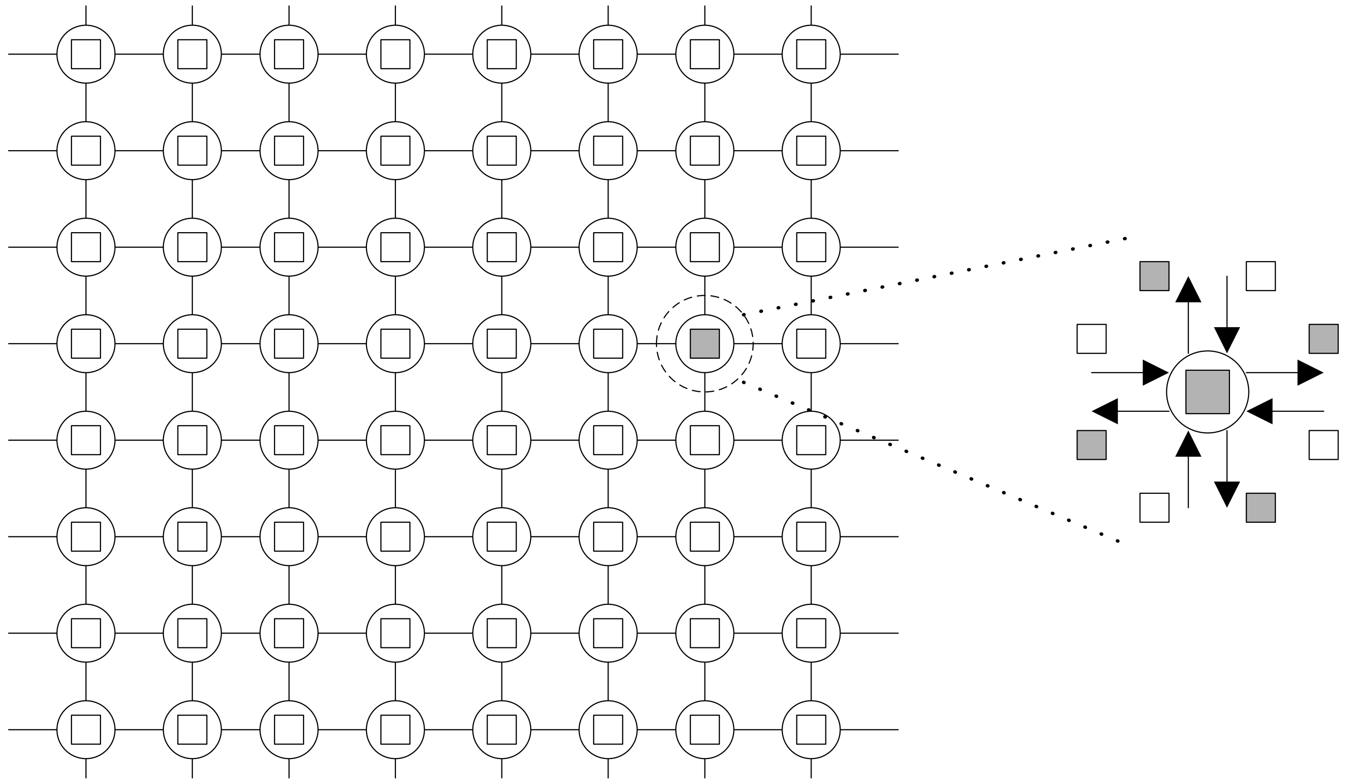
Assignment

- **任务组合**
  - 组合是由抽象到具体的过程，是将组合的任务能在一类并行机上有效的执行；
  - 合并小尺寸任务，减少任务数。如果任务数恰好等于处理器数，则也完成了映射过程；
  - 通过增加任务的粒度和重复计算，可以减少通信成本；
  - 保持映射和扩展的灵活性，降低软件工程成本
- **软件工程里的“高内聚，低耦合”**



表面-容积效应

- 通信量与任务子集的表面成正比，计算量与任务子集的体积成正比；
- 增加重复计算有可能减少通信量

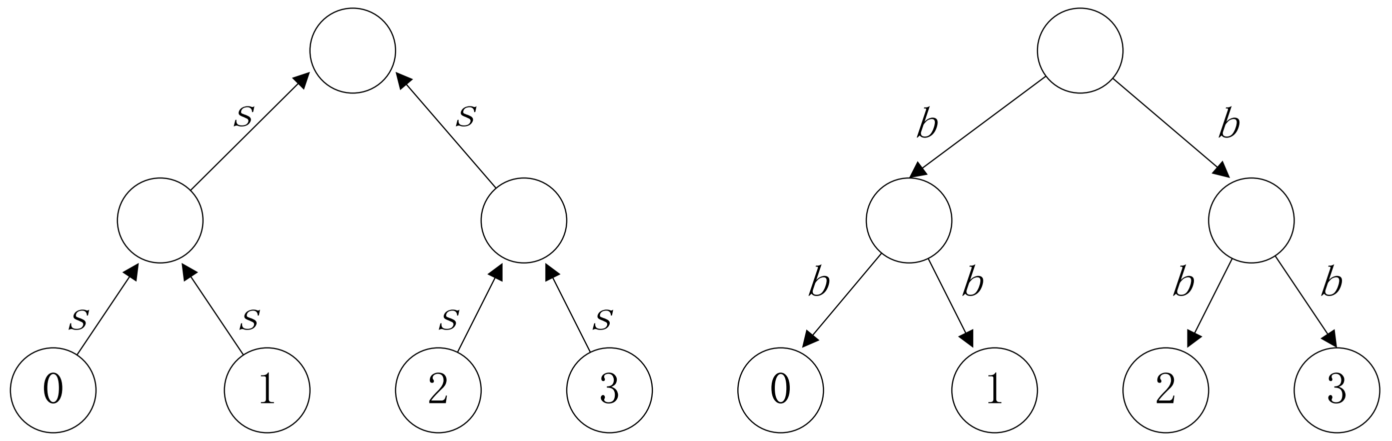


重复计算

- 重复计算减少通信量，但增加了计算量，应保持恰当的平衡；
- 重复计算的目标应减少算法的总运算时间

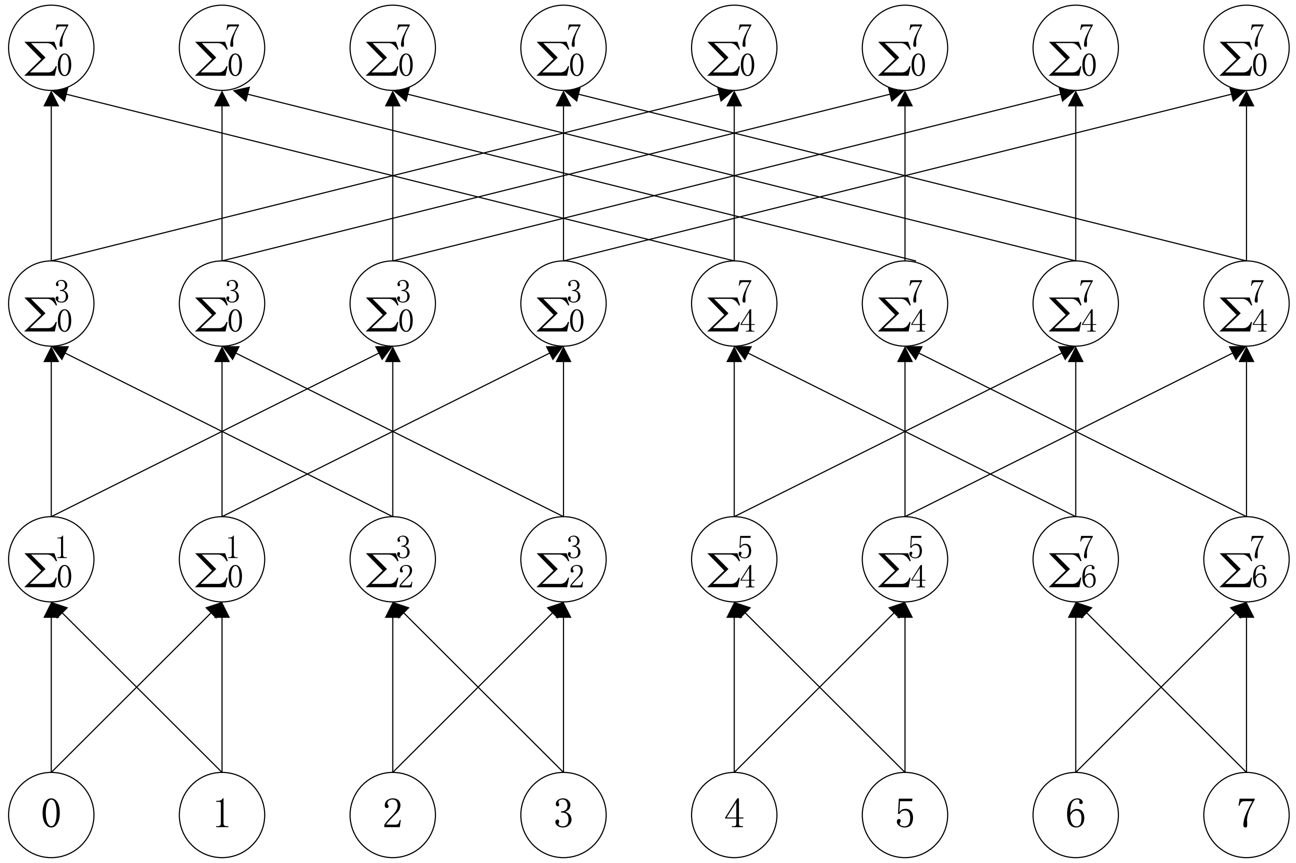
示例：二叉树上N个处理器求N个数的全和，要求每个处理器均保持全和

- 二叉树上求和，共需2logN步



示例：二叉树上N个处理器求N个数的全和，要求每个处理器均保持全和

- 蝶式结构求和，使用了重复计算，共需logN步

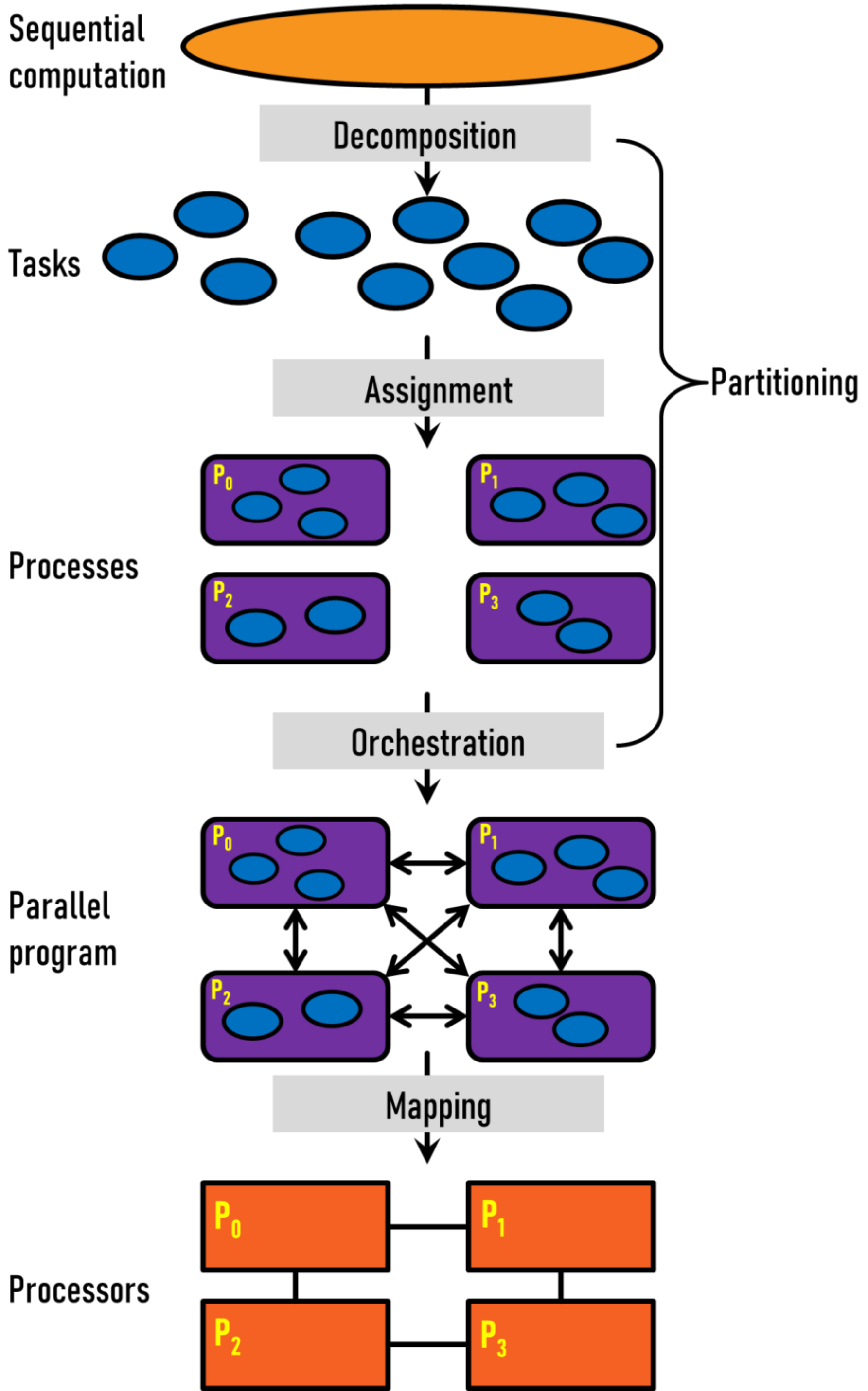

Σ_0^7 Σ_0^7 Σ_0^7 Σ_0^7 Σ_0^7 Σ_0^7 Σ_0^7 Σ_0^7
Σ_0^3 Σ_0^3 Σ_0^3 Σ_0^3 Σ_4^7 Σ_4^7 Σ_4^7 Σ_4^7
Σ_0^1 Σ_0^1 Σ_2^3 Σ_2^3 Σ_4^5 Σ_4^5 Σ_6^7 Σ_6^7
0 1 2 3 4 5 6 7

组合判据

- 增加粒度是否减少了通信成本?
- 重复计算是否已权衡了其得益?
- 是否保持了灵活性和可扩放性?
- 组合的任务数是否与问题尺寸成比例?
- 是否保持了类似的计算和通信?
- 有没有减少并行执行的机会?

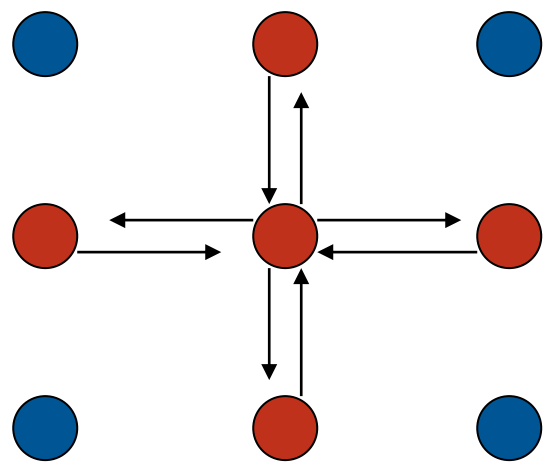
Communication/Orchestration

- **通信**
  - 通信是PCAM设计过程的重要阶段；
  - 划分产生的诸任务，一般不能完全独立执行，需要在任务间进行数据交流；从而产生了通信；
  - 功能分解确定了诸任务之间的数据流；
  - 诸任务是并发执行的，通信则限制了这种并发性
- **四种通信模式**
  - 局部/全局通信
  - 结构化/非结构化通信
  - 静态/动态通信
  - 同步/异步通信



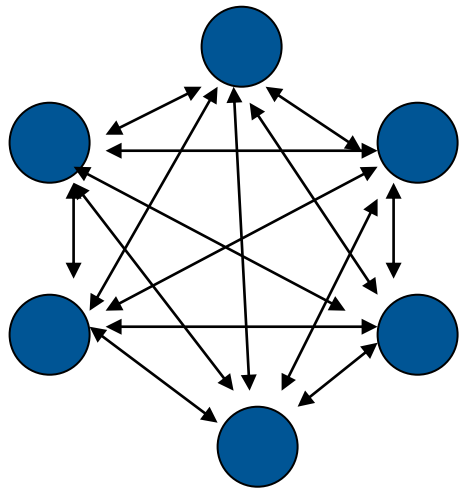
局部通信

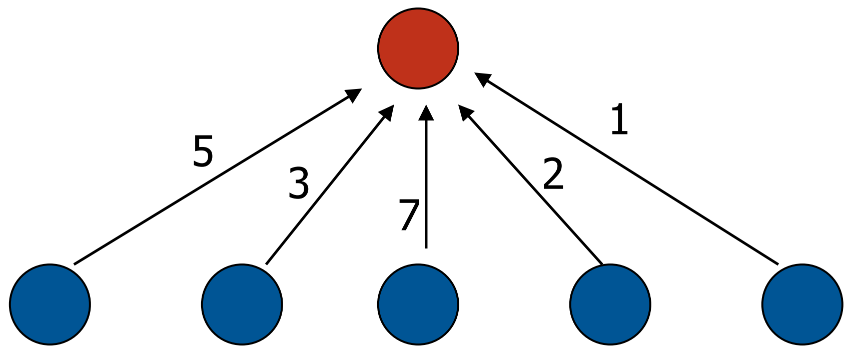
- 通信限制在一个邻域内



全局通信

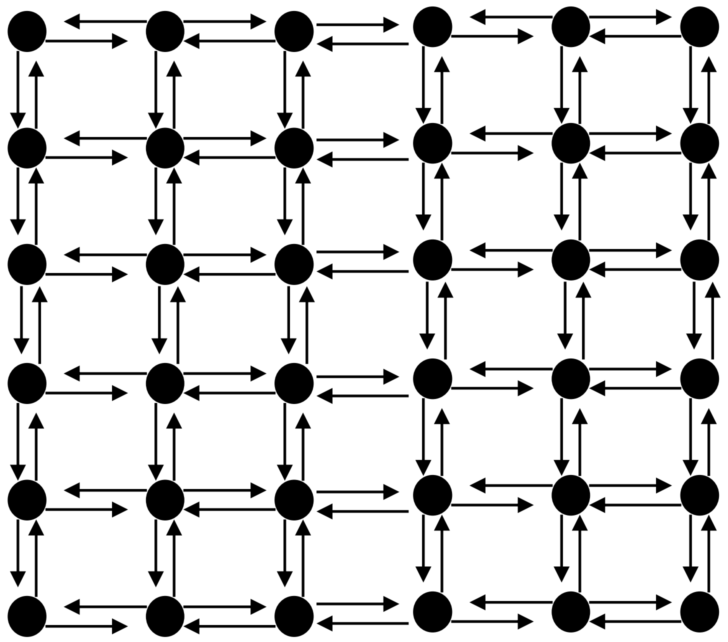
- 例如：
  - All to All
  - Master-Worker

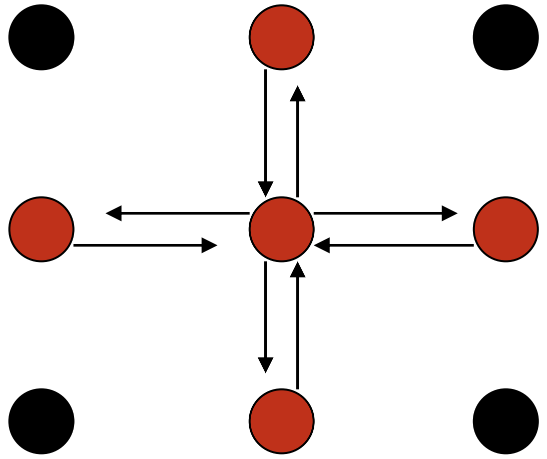




结构化通信

- 每个任务的通信模式是相同的；
- 下面是否存在一个相同通信模式？





同步/异步通信

- 同步通信时，接收方和发送方协同操作；异步通信中，接收方获取数据无需与发送方协同

通信判据

- 所有任务是否执行大致相当的通信?
- 是否尽可能的局部通信?
- 通信操作是否能并行执行?
- 同步任务的计算能否并行执行?

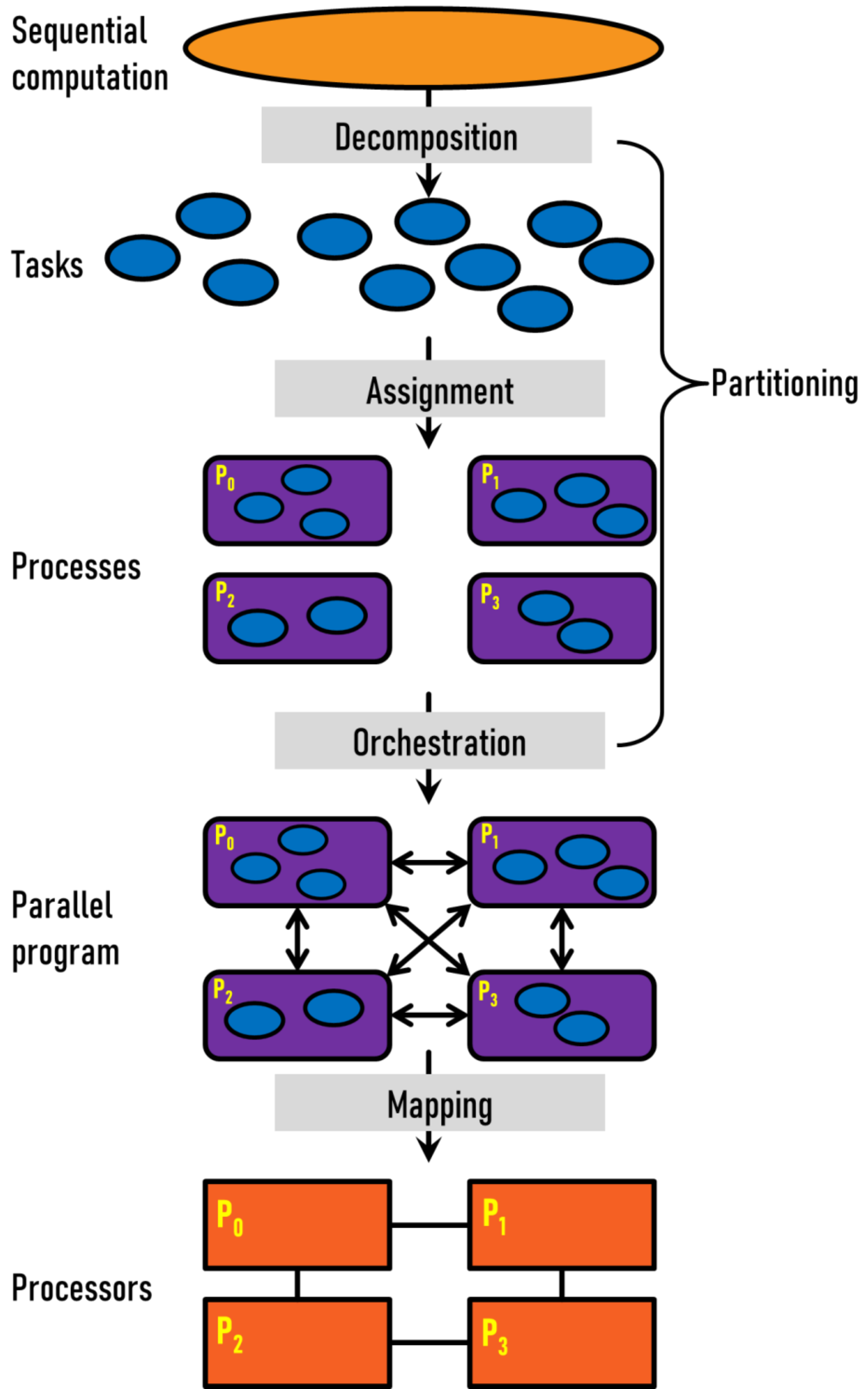
Mapping`

处理器映射

- 每个任务要映射到具体的处理器，定位到运行机器上；
- 任务数大于处理器数时，存在负载平衡和任务调度问题；
- 映射的目标：减少算法的执行时间
  - 并发的任务　　不同的处理器
  - 任务之间存在高通信的　　同一处理器
- 映射实际是一种权衡，属于**NP完全问题**

负载平衡

- 静态的：事先确定；
- 概率的：随机确定；
- 动态的：执行期间动态负载；
- 基于域分解的：递归对剖；局部算法；概率方法；循环映射



任务分配与调度

- 负载平衡与任务分配/调度密切相关，任务分配通常有静态的和动态的两种方法。
- 静态分配一般是任务到进程的算术映射。
  - 静态分配的优点是没有运行时任务管理的开销，但为了实现负载平衡，要求不同任务的工作量和处理器的性能是可以预测的并且拥有足够的可供分配的任务。
  - 静态调度（Static Scheduling）方案一般是静态地为每个处理器分配个连续的循环迭代，其中为迭代次数，是处理器数。也可以采用轮转（Round-robin）的方式来给处理器分配任务，即将第i个循环迭代分配给第i mod p个处理器
- 动态分配与调度相对灵活，可以运行时在不同处理器间动态地进行负载的调整
  - 各种动态调度(Dynamic Scheduling)技术是并行计算研究的热点，包括基本自调度SS(Self Scheduling)、块自调度BSS(Block Self Scheduling)、 指导自调度GSS(Guided Self Scheduling)、因子分解调度FS(Factoring Scheduling)、梯形自调度TSS(Trapezoid Self Scheduling)、 耦合调度AS(Affinity Scheduling)、安全自调度SSS(Safe Self Scheduling)和自适应耦合调度AAS(Adapt Affinity Scheduling)

Chapter 2: LSC programming

- **Sequential implementation with Python**
  - "Using Python to Solve Computational Physics Problems"
- **Ideas to convert Sequential to Parallel**
  - DAOM, PCAM
- **Measure the performance**

By Benchmark – Linpack (LINear algebra PACKage)

- Introduced by Jack Dongarra in 1979
- Based on LINPACK linear algebra package developed by J. Dongarra, J. Bunch, C. Moler and P. Stewart (now superseded by the LAPACK library)
- Solves a dense, regular system of linear equations, using matrices initialized with pseudo-random numbers
- Provides an estimate of system’s effective floating-point performance
- Does not reflect the overall performance of the machine!

Units of Measure

- **High Performance Computing (HPC) units are:**
  - **Flop: floating point operation, usually double precision unless noted**
  - **Flop/s: floating point operations per second**
  - **Bytes: size of data (a double precision floating point number is 8 bytes)**
- **Typical sizes are millions, billions, trillions…**

Mega	**Mflop/s = 10^6 flop/sec**	**Mbyte**
Giga	**Gflop/s = 10^9 flop/sec**	**G**
Tera	**Tflop/s = 10^{12} flop/sec**	**T**
Peta	**Pflop/s = 10^{15} flop/sec**	**P**
Exa	**Eflop/s = 10^{18} flop/se**	**byte = 2**
Zetta	**Zflop/s = 10^{21} flop/sec**	**Zbyte = 2^{70} ~ 10^{21} bytes**
Yotta	**Yflop/s = 10^{24} flop/sec**	**Ybyte = 2^{80} ~ 10^{24} bytes**

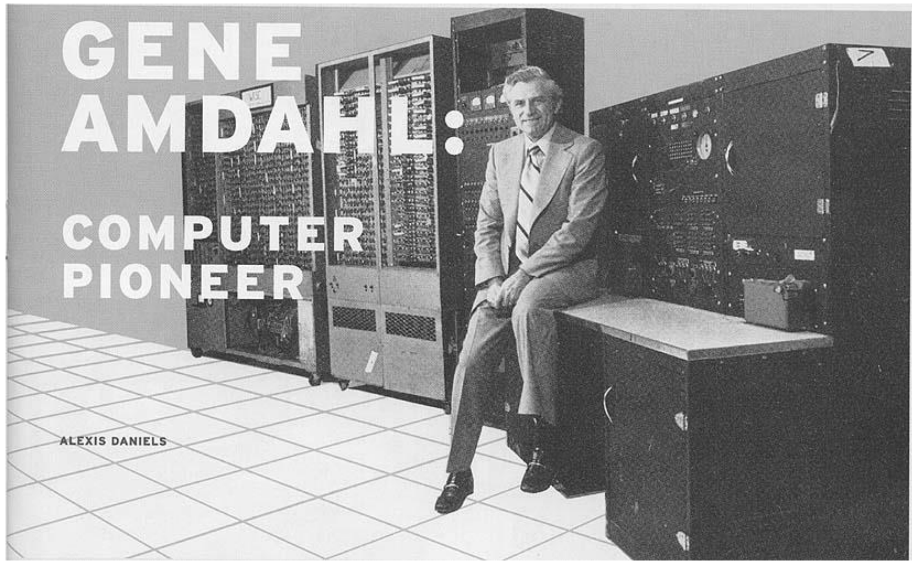
We are now striving to reach this scale computers – SuperComputer

- **Current fastest (public) machine ~ 55 Pflop/s, 3.1M cores**
  - **Up-to-date list at www.top500.org**

By theory – Is parallelization worth it ?

- **We parallelize our programs in order to run them faster**
- **How much faster will a parallel program run?**
  - Suppose that the sequential execution of a program takes T_1 time units and the parallel execution on p processors takes T_p time units
  - Suppose that out of the entire execution of the program, s fraction of it is not parallelizable while $1-s$ fraction is parallelizable
  - Then the speedup (Amdahl's formula):

$$\frac{T_1}{T_p} = \frac{T_1}{(T_1 \times s + T_1 \times \frac{1-s}{p})} = \frac{1}{s + \frac{1-s}{p}}$$



Amdahl's Law: An Example

- Suppose that 80% of you program can be parallelized and that you use 4 processors to run your parallel version of the program
- The speedup you can get according to Amdahl is:

$$\frac{1}{s+\frac{1-s}{p}} = \frac{1}{0.2+\frac{0.8}{4}} = 2.5 \text{ times}$$

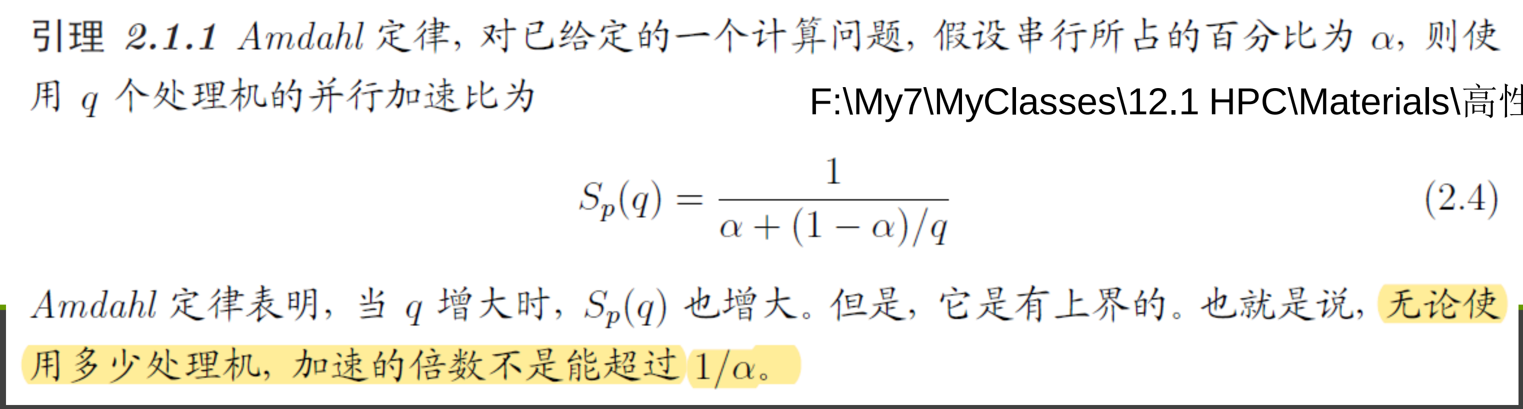
- Although you use 4 processors you cannot get a speedup more than 2.5 times (or 40% of the serial running time)

高性能并行计算

迟学斌 中国科学院计算机网络信息中心
chi@sccas.cn, chi@sc.cnic.cn

http://lssc.cc.ac.cn/
http://www.sccas.cn/
http://www.scgrid.cn/
http://www.cngrid.org/

2005 年 4 月 6 日



引理 *2.1.1* *Amdahl* 定律，对已给定的一个计算问题，假设串行所占的百分比为 α，则使用 q 个处理机的并行加速比为

$$S_p(q) = \frac{1}{\alpha+(1-\alpha)/q} \qquad (2.4)$$

Amdahl 定律表明，当 q 增大时，$S_p(q)$ 也增大。但是，它是有上界的。也就是说，无论使用多少处理机，加速的倍数不是能超过 $1/\alpha$。

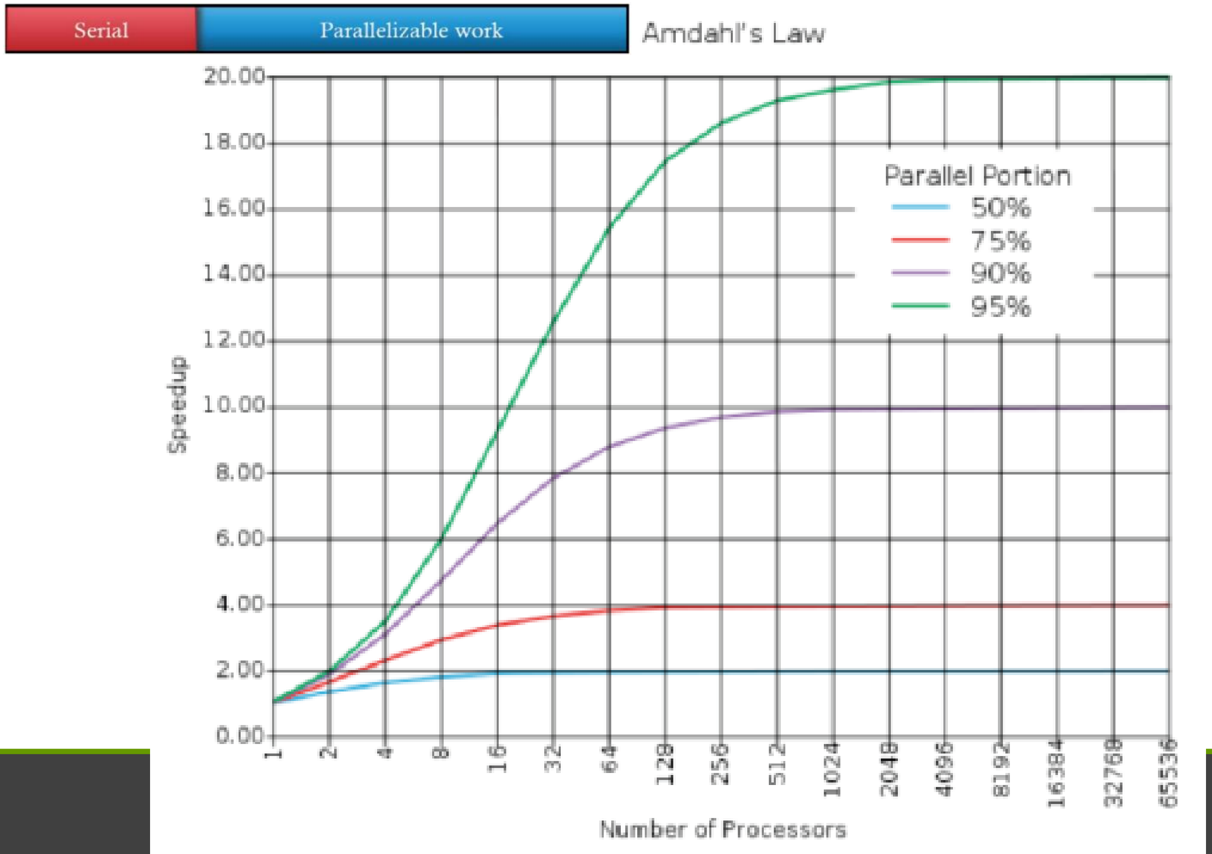
F:\My7\MyClasses\12.1 HPC\Materials\高性能并行计算(2005年4月6日).pdf

Amdahl's law: $R = \dfrac{1}{(1-p)+p/N}$

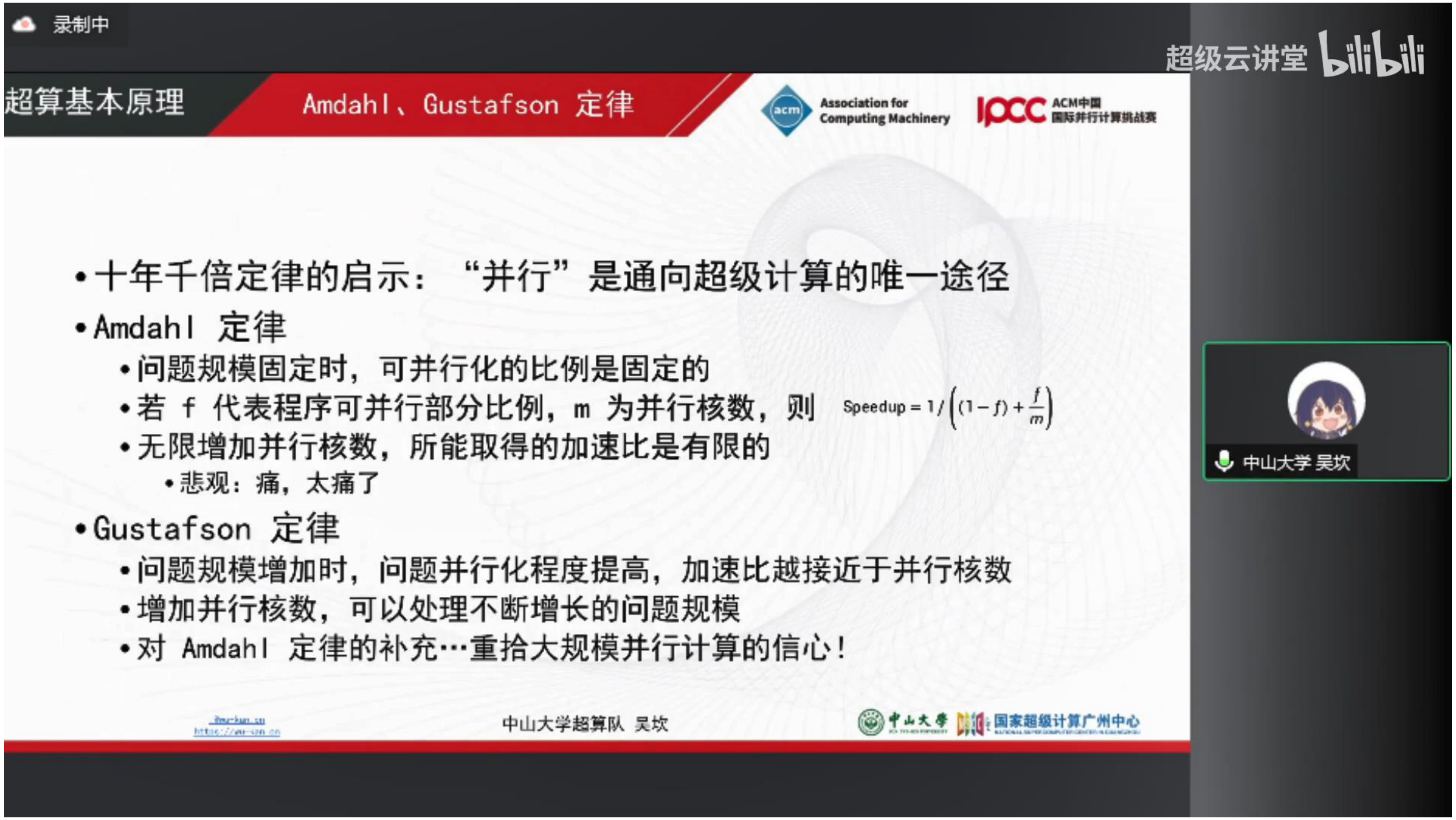
p: fraction of work that can be parallelized

$1 - p$: fraction of sequential work

R: prediction maximum speed-up using N parallel processors



Gustafson 定律


录制中
超级云讲堂 bilibili
中山大学 吴坎
Association for Computing Machinery
ACM中国 国际并行计算挑战赛

超算基本原理　Amdahl、Gustafson 定律

- 十年千倍定律的启示：“并行”是通向超级计算的唯一途径
- Amdahl 定律
  - 问题规模固定时，可并行化的比例是固定的
  - 若 f 代表程序可并行部分比例，m 为并行核数，则 $\text{Speedup} = 1/\left((1-f)+\frac{f}{m}\right)$
  - 无限增加并行核数，所能取得的加速比是有限的
    - 悲观：痛，太痛了
- Gustafson 定律
  - 问题规模增加时，问题并行化程度提高，加速比越接近于并行核数
  - 增加并行核数，可以处理不断增长的问题规模
  - 对 Amdahl 定律的补充…重拾大规模并行计算的信心！

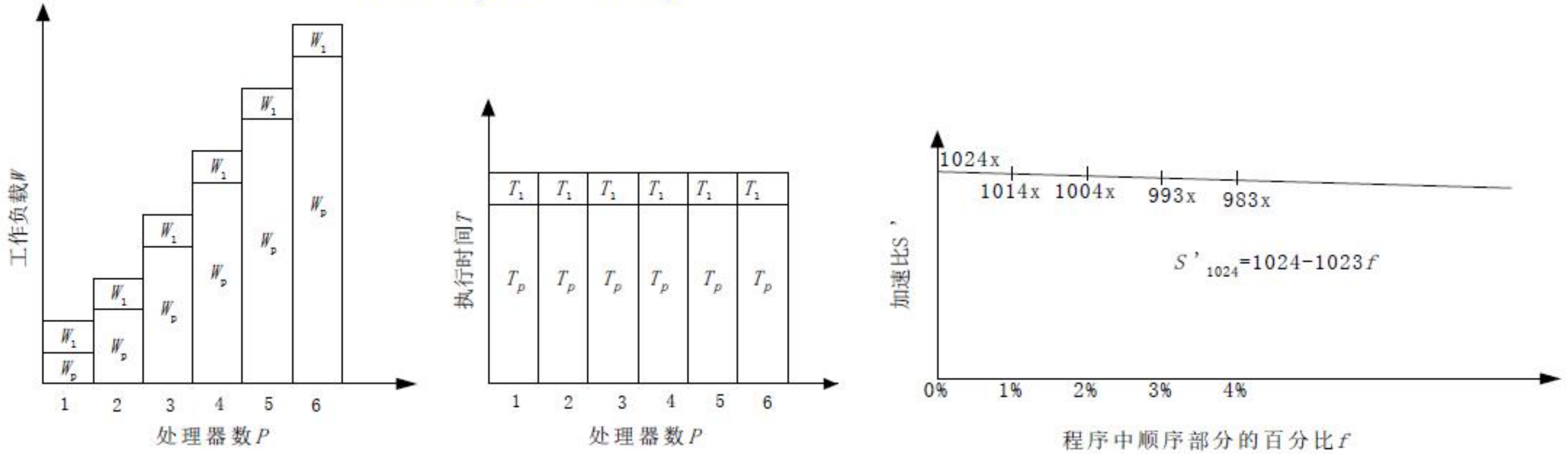
[illegible]
[illegible]
中山大学超算队　吴坎

并行程序性能评价

- Gustafson定律
  - Amdahl 定律有一个重要前提，就是处理的数据集大小是固定的，但是这在大数据计算的领域里，这个假设并不经常能达到，因为人们总是会为了在短时间内处理更多的数据
  - 很多大型计算，精度要求很高，即在此类应用中精度是一个关键因素，而计算时间是固定不变的。此时为了提高精度，必须加大计算量，相应的也必须增加处理器的数目来完成这部分计算，以保持计算时间不变
  - 研究在给定的时间内用不同数目的处理器能够完成多大的计算量是并行计算中一个很实际的问题

Gustafson定律

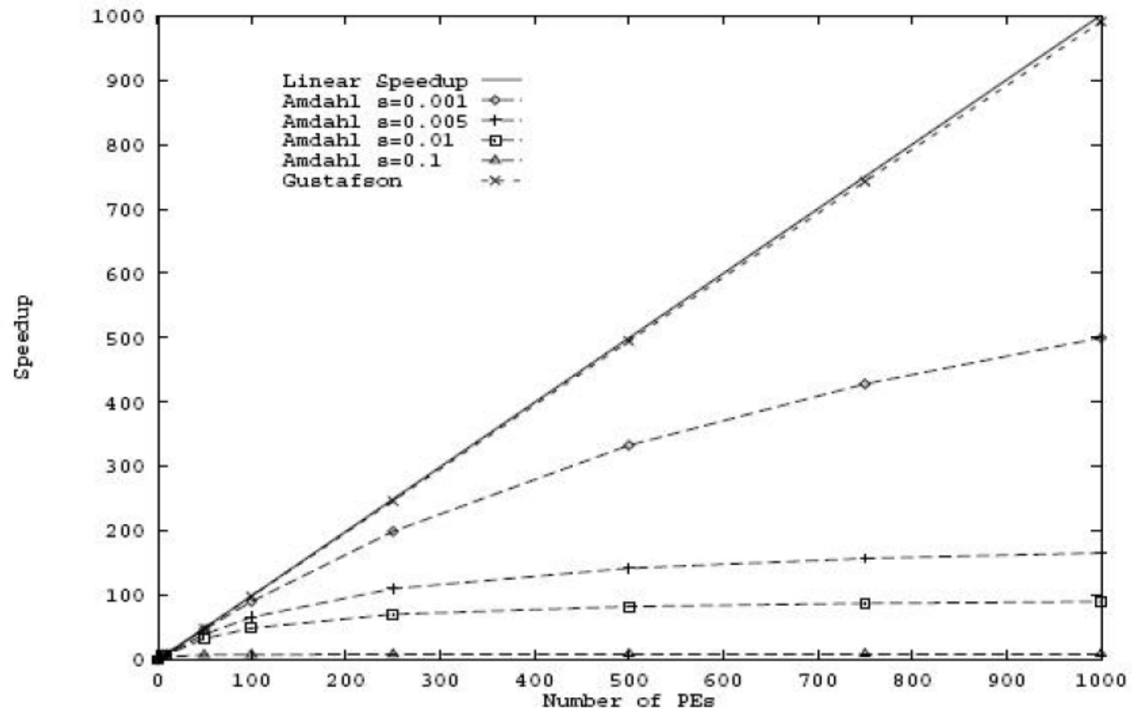
- t_s，t_p 表示串行部分和并行部分执行的时间
- f表示程序中不可以被并行化的部分的所占的比例
  - $f = \frac{t_s}{t_s + t_p}$
- $S(p) = \frac{t_s + p \times t_p}{t_s + p \times t_p / p} = \frac{t_s + p \times t_p}{t_s + t_p} = f + p \times (1 - f)$



(a) (b) (c)

并行程序性能评价

- Amdahl定律 vs Gustafson定律
  - 刻画的内容等价，可相互推导



Bell’s Law

https://gordonbell.azurewebsites.net/

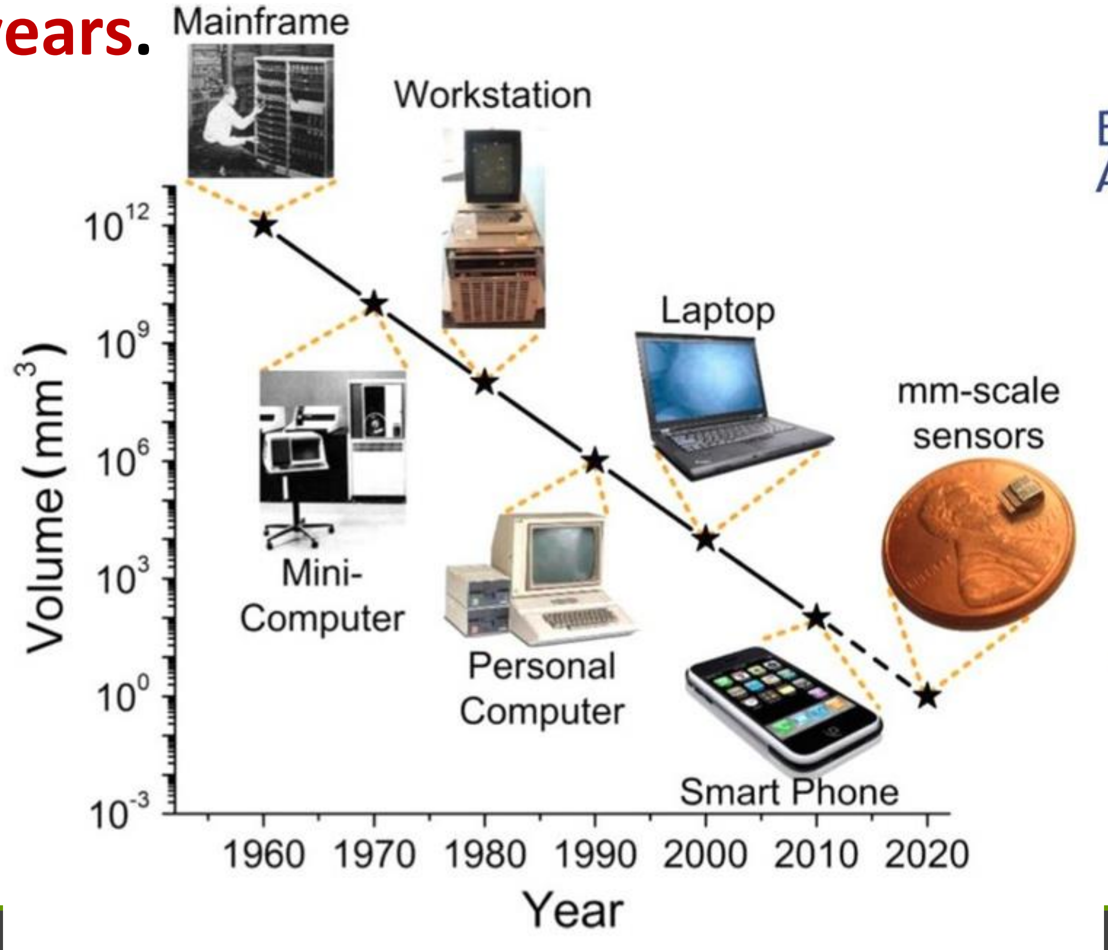
Bell's Law of Computer Class formation was discovered about 1972. It states that technology advances in semiconductors, storage, user interface and networking advance every decade enable a new, usually lower priced computing platform to form. Once formed, each class is maintained as a quite independent industry structure.

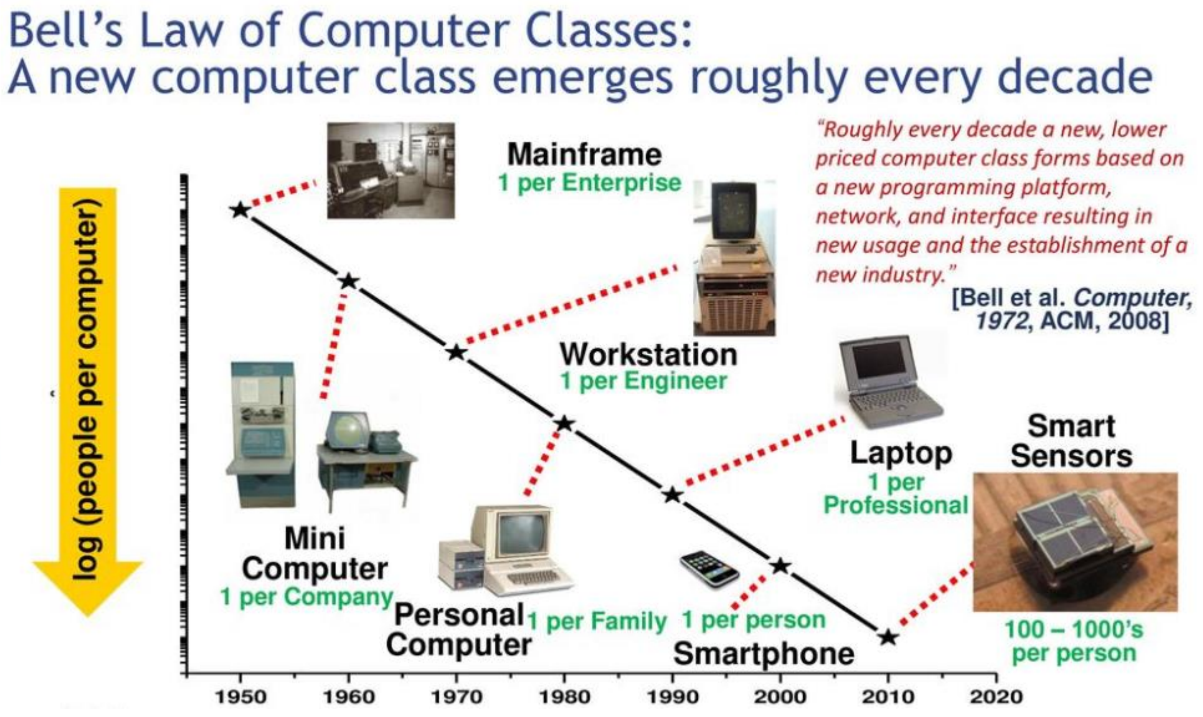
This explains mainframes, minicomputers, workstations and Personal computers, the web, emerging web services, palm and mobile devices, and ubiquitous interconnected networks. We can expect home and body area networks to follow this path.

From Gordon Bell (2007), http://research.microsoft.com/~GBell/Pubs.htm https://www.computerhistory.org/atchm/the-worlds-smallest-computer/

Bell's Law states, that:

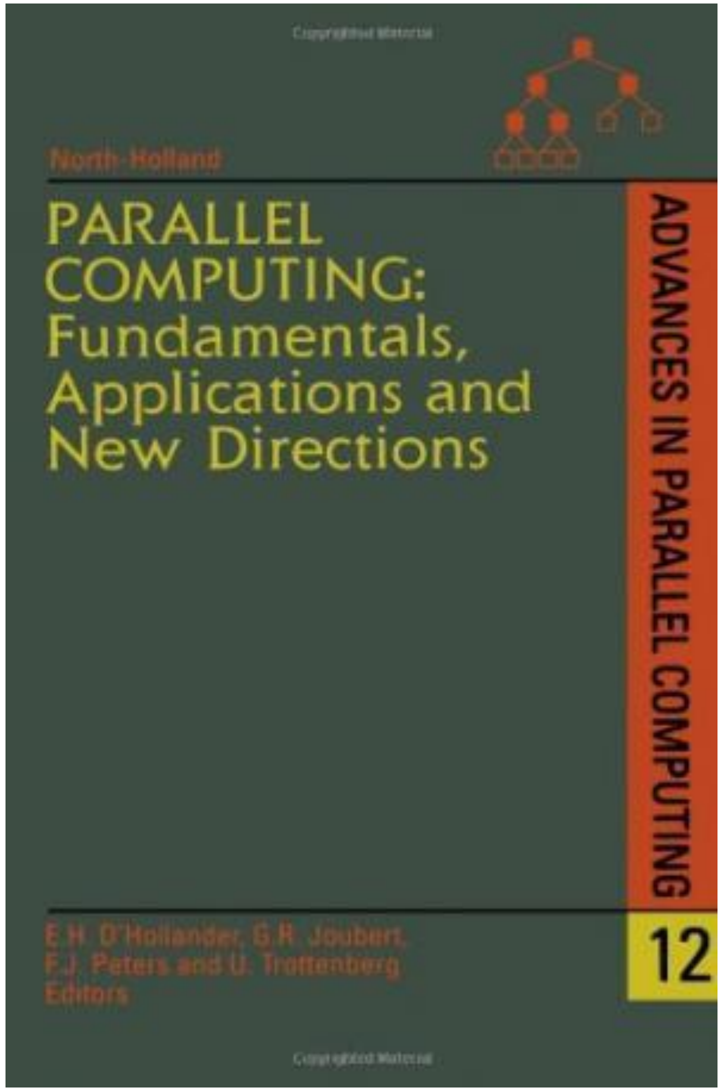
- **Important classes of computer architectures come in cycles of about 10 years.**





计 算 机 科 学 丛 书
高性能科学与工程计算
（德）Georg Hager Gerhard Wellein 著
Introduction to High Performance Computing for Scientists and Engineers
Introduction to High Performance Computing for Scientists and Engineers
Georg Hager
Gerhard Wellein
机械工业出版社
China Machine Press

- 高性能科学与工程计算
- [德] Georg Hager, [德] Gerhard Wellein
  - 《计算机科学丛书：高性能科学与工程计算》从工程实践的角度介绍了高性能计算的相关知识。主要内容包括现代处理器的体系结构、为读者理解当前体系结构和代码中的性能潜力和局限提供了坚实的理论基础。
  - 接下来讨论了高性能计算中的关键问题，包括串行优化、并行、OpenMP、MPI、混合程序设计技术
    。
  - 作者根据自身的研究也提出了一些前沿问题的解决方案，如编写有效的C++代码、GPU编程等。



North-Holland

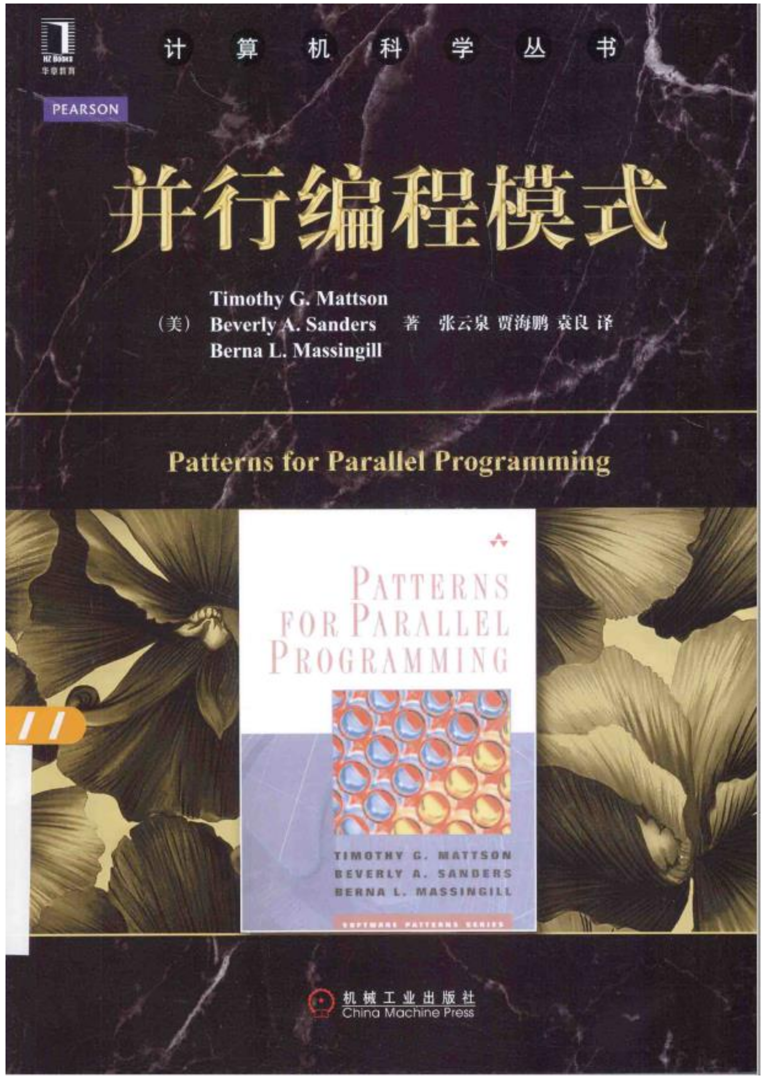
PARALLEL COMPUTING: Fundamentals, Applications and New Directions

ADVANCES IN PARALLEL COMPUTING

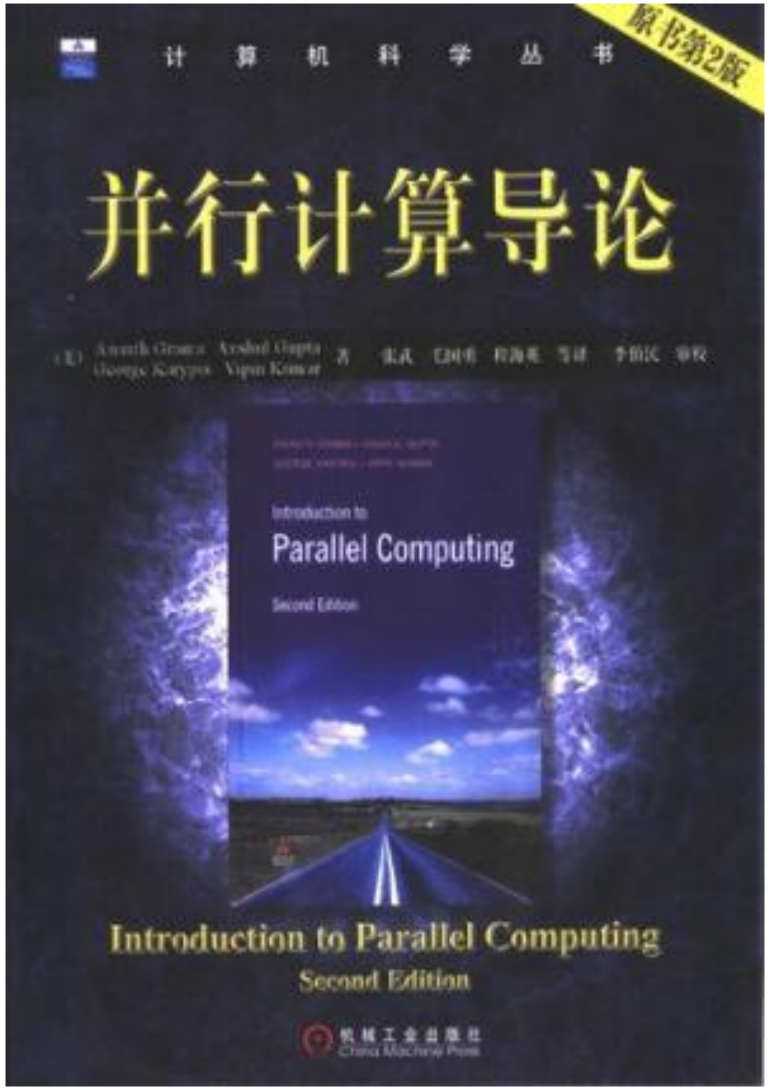
12

E.H. D'Hollander, G.R. Joubert, F.J. Peters and U. Trottenberg
Editors

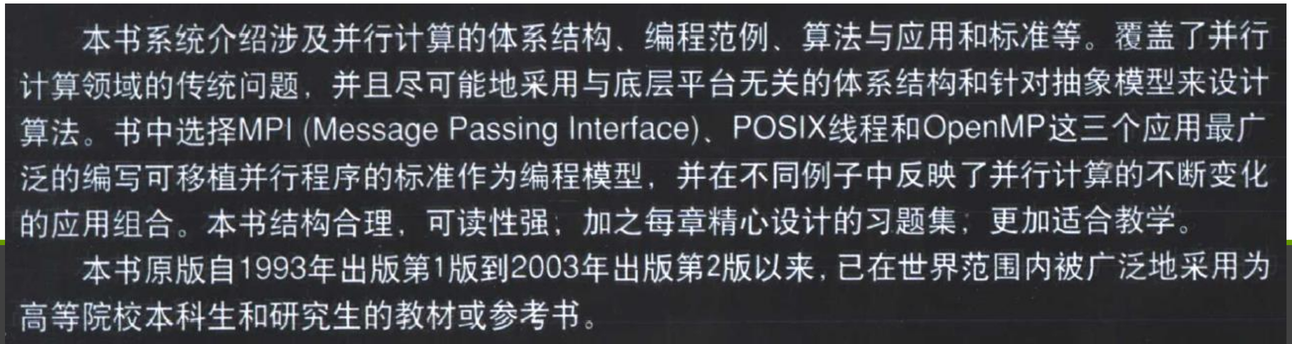
- Parallel Computing: Fundamentals, Applications and New Directions
- *E.H. D'Hollander*, *F.J. Peters*, *G.R. Joubert*, *U. Trottenberg and R. Völpel (Eds.)*
- North Holland
- 1998



- 并行编程模式
- *Timothy G. Mattson*, *Beverly A. Sanders*, *Berna L. Massingill*
- 本书介绍了并行编程模式的相关概念和技术，主要内容包括并行编程模式语言、并行计算的背景、软件开发中的并发性、并行算法结构设计、支持结构、设计的实现机制以及OpenMP、MPI等。本书可供软件专业的本科生或研究生使用，同时也可供从事软件开发工作的广大技术人员参考。
- 2015

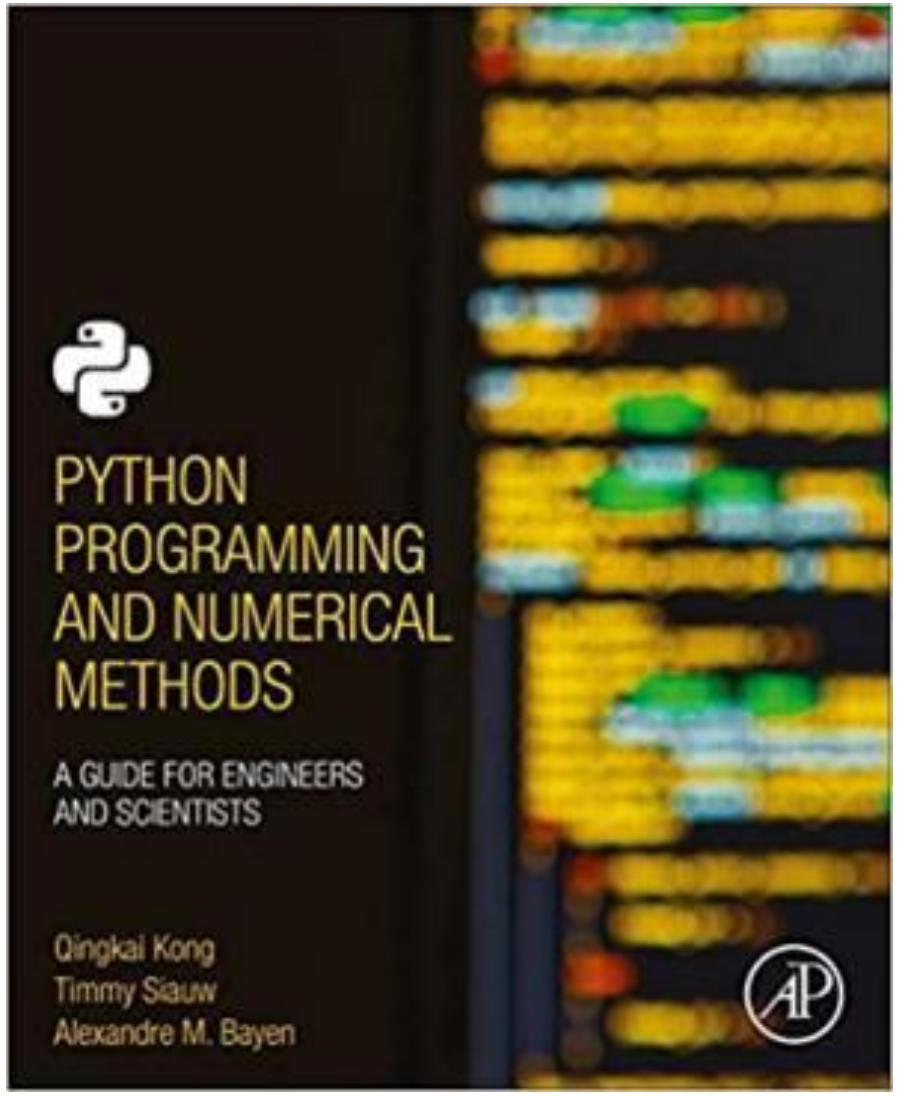


- 并行计算导论
- Ananth Grama, George Karypis, 张武, 毛国勇, Anshul Gupta, Vipin Kumar, 程海英
  - 《并行计算导论》(原书第2版)全面介绍并行计算的各个方面，包括体系结构、编程范例、算法与应用和标准等，涉及并行计算的新技术，也覆盖了较传统的算法，如排序、搜索、图和动态编程等。《并行计算导论》(原书第2版)尽可能采用与底层平台无关的体系结构并且针对抽象模型来设计处落地。书中选择MPI、POSIX线程和OpenMP作为编程模型，并在不同例子中反映了并行计算的不断变化的应用组合
- **2005**



本书系统介绍涉及并行计算的体系结构、编程范例、算法与应用和标准等。覆盖了并行计算领域的传统问题，并且尽可能地采用与底层平台无关的体系结构和针对抽象模型来设计算法。书中选择MPI (Message Passing Interface)、POSIX线程和OpenMP这三个应用最广泛的编写可移植并行程序的标准作为编程模型，并在不同例子中反映了并行计算的不断变化的应用组合。本书结构合理，可读性强，加之每章精心设计的习题集，更加适合教学。

本书原版自1993年出版第1版到2003年出版第2版以来，已在世界范围内被广泛地采用为高等院校本科生和研究生的教材或参考书。


PYTHON PROGRAMMING AND NUMERICAL METHODS
A GUIDE FOR ENGINEERS AND SCIENTISTS
Qingkai Kong
Timmy Siauw
Alexandre M. Bayen
AP

- **Python Programming and Numerical Methods: A Guide for Engineers and Scientist**
- **Qingkai Kong, Timmy Siauw, Alexandre Bayen**
- Academic Press
- **2020**

Powered by Jupyter Book