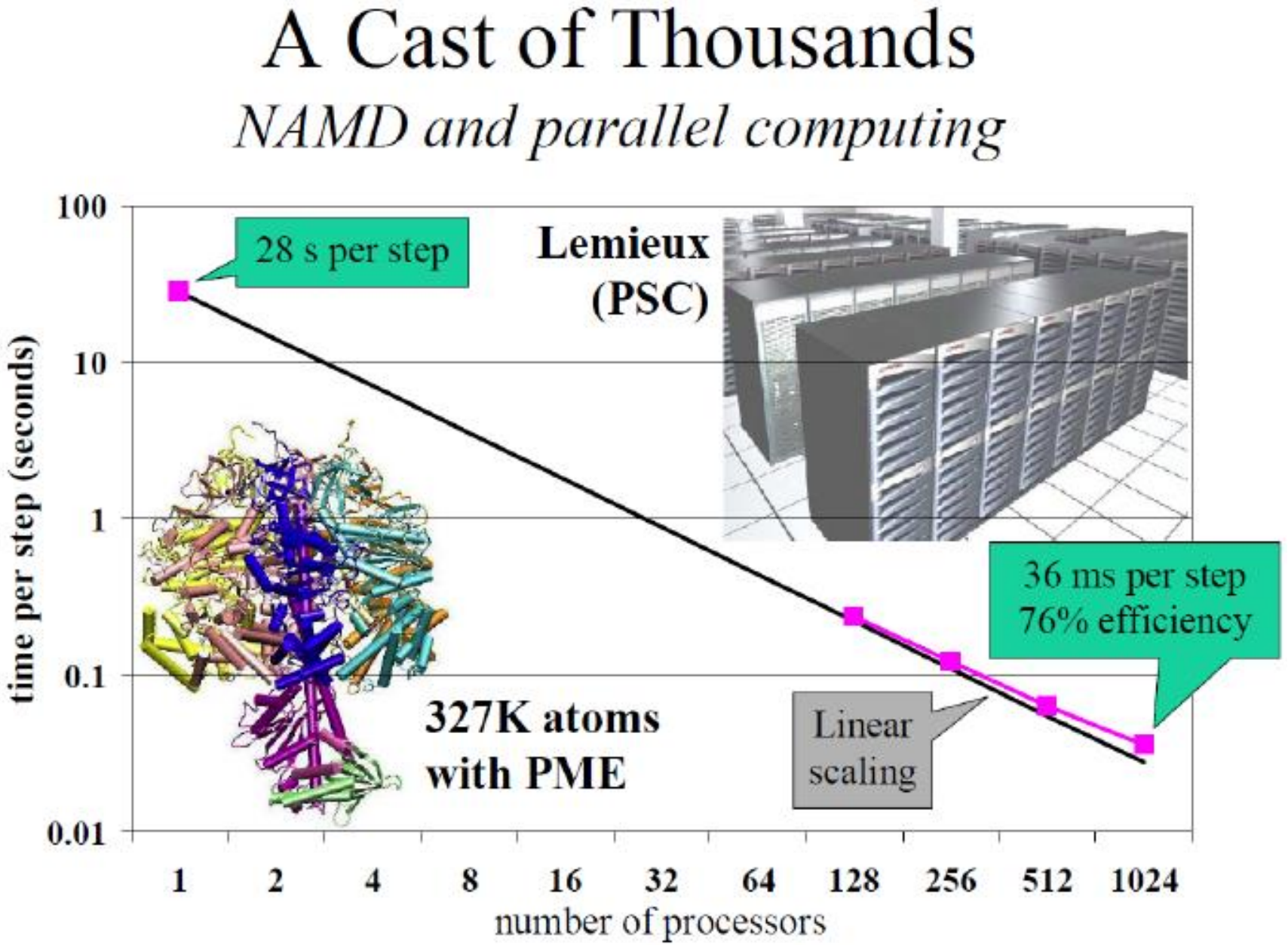
计算物理B

第十章 高性能计算和并行算法

主讲：张志勇
zzyzhang@ustc.edu.cn

并行分子动力学模拟


A Cast of Thousands
NAMD and parallel computing
28 s per step
Lemieux
(PSC)
36 ms per step
76% efficiency
Linear
scaling
327K atoms
with PME
time per step (seconds)
100
10
1
0.1
0.01
1
2
4
8
16
32
64
128
256
512
1024
number of processors

串行计算和并行计算
(Serial and Parallel Computing)

传统的**串行计算**，分为“指令”和“数据”两个部分，并在程序执行时“独立地申请和占有”内存空间，且所有计算均局限于该内存空间。

并行计算将进程相对独立地分配于不同的节点上，由各自独立的操作系统调度，享有独立的CPU和内存资源（内存可以共享）；进程间相互信息交换通过**消息传递**。

什么可以并行？
(Parallelizable and non-parallelizable Problem)

- 能否将顺序执行的程序转换成语义等价的、可并行执行的程序，主要取决于程序的结构形式，特别是其中的**数据相关性**。

- 创建一个并行程序的步骤，首先需要理解两个重要的概念：**任务**和**进程**。

任务（Task）

- 任务是程序要完成的一个工作，其内容和大小是随意的，它是并行程序所能处理的并发性最小的单元：即一个任务只能由一个处理器执行，处理器之间的并发性只能在任务之间开发。

进程（Process）

- 进程（也称为线程）是一个完成任务的实体。一个并行程序由许多合作的进程构成，每个完成程序中任务的一个子集。通过某种分配机制，任务被分配给进程。
- 进程完成其任务的方式是通过在机器的物理处理器上执行。

进程与处理器的区别（Process and Processor）

- 并行化的观点：处理器是物理资源，而进程是抽象或者虚拟化多处理器的一种方便的方式。我们通过进程，而不是处理器来写并行程序，将进程映射到处理器是下一步。在一次程序的执行中，进程数不一定要等于处理器数。如果进程多，一个处理器有可能要执行多个进程；如果进程少，则某些处理器要闲置。

串行程序并行化的步骤
(Steps of Parallelization)

- 从一个串行程序得到一个并行程序，由四个步骤构成：

1、将计算的问题分解成任务；

2、将任务分配给进程；

3、在进程间组织必要的数据访问、通信和同步；

4、将进程映射或绑定到处理器。

进程间通信
(Communications)

- 现代操作系统提供基本的系统调用函数，允许位于同一台处理机或不同处理机的多个进程之间相互交流信息，操作具体表现为三种形式：通信、同步和聚集。

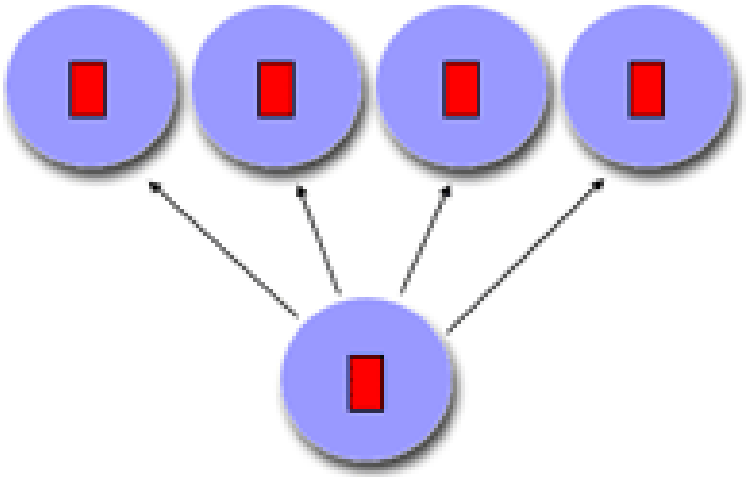
- 以上的三种形式统称为进程间通信，操作的具体数据对象为消息（message），具体的操作为消息传递（Message Passing）。

通信、同步和聚集
(Communications, Synchronization, Gathering)

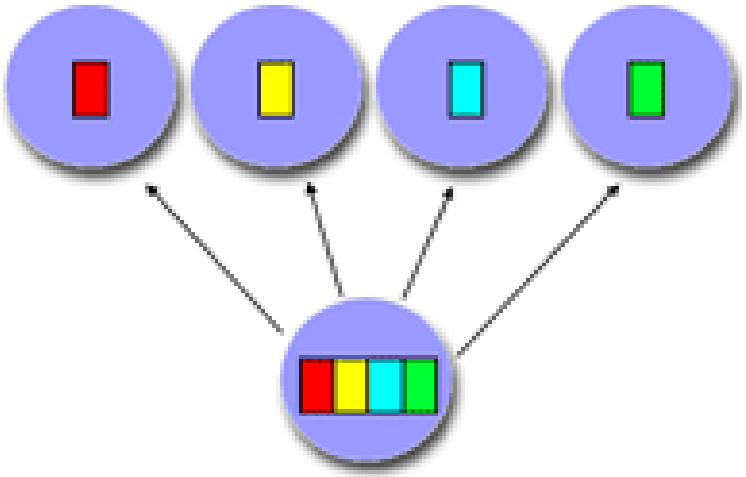
- 进程间的数据传递称为进程间通信。
- 在同一台处理机中，通信可以读/写操作系统提供的共享数据缓存区来实现。
- 不同处理机中，通信可以通过网络来实现。

- 同步（synchronization）是使位于相同或不同处理机中的多个进程之间的相互等待的操作，它要求进程的所有操作均必须等待到达某一控制状态之后才并行。

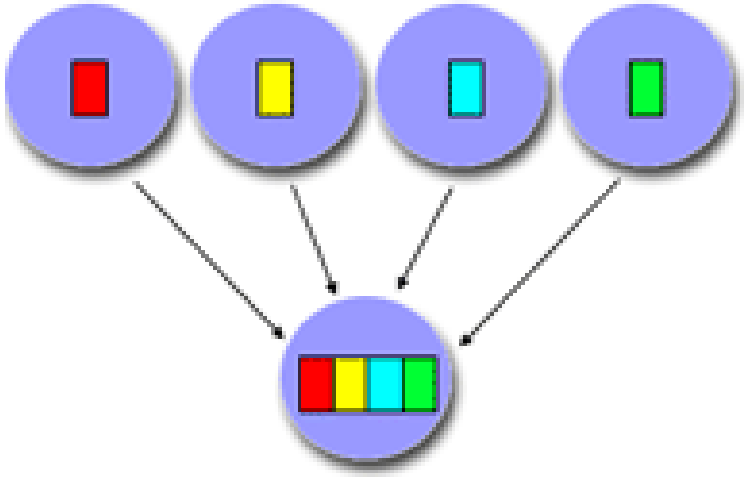
- 聚集（gathering）将位于不同处理机中的多个进程的局部结果综合起来，通过某种操作，例如最大值、最小值、累加和，产生一个新的结果，存储在某个指定的或者所有的进程变量中。



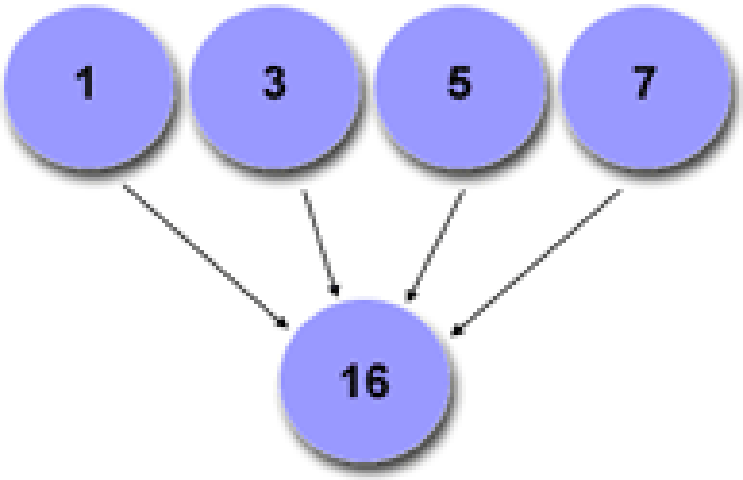
broadcast



scatter



gather

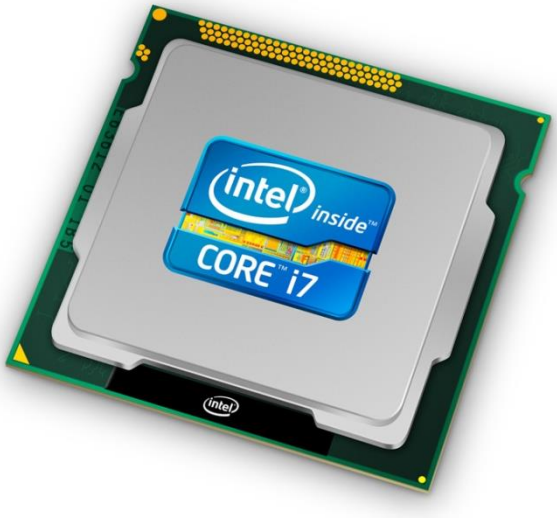


reduction

并行的效率
(Parallel Efficiency)

- 在以上几个步骤中，并没有考虑并行的效率问题。
- 考虑到消息传递的开销，如果在应用的一部分中，计算的时间是**分钟级**的而数据传输的时间是**秒级**的，那么这一部分可以并行执行。

MD并行的硬件基础（Hardware）：CPU



D. E. Shaw

MD并行的硬件基础（Hardware）：Infiniband

Infiniband技术

是一种开放标准的高带宽、高速网络互联技术
是一种支持多并发链接的“转换线缆”技术
支持三种连接（1x，4x，12x)，是基本传输速率的倍数，即支持速率是（QDR）10Gbps、40Gbps和120Gbps
InfiniBand技术不是用于一般网络连接的，它的主要设计目的是针对服务器端的连接问题。因此，InfiniBand技术主要应用于服务器与服务器（比如复制，分布式工作等），服务器和存储设备（比如SAN和直接存储附件）以及服务器和网络之间（比如LAN，WANs和Internet）的通信。

InfiniBand 传输速率规格

	SDR 模式	DDR 模式	QDR 模式
通道数 1X	2.5Gbps	5Gbps	10Gbps
通道数 4X	10Gbps	20bps	40Gbps
通道数 12X	30Gbps	60Gbps	120Gbps

万兆以太网（Ethernet） vs IB

万兆以太网	
说明	目前市场上最快的以太网技术，具有极低的延迟（不到500ns)的新适配器和交换机正在进入市场
优点	通常使用标准的以太网LAN设备、线缆和PC接口卡
缺点	延迟问题依然存在，因而可能把这种技术排斥在像集群和网格等对延迟极其敏感的应用之外，速度仍落后于其它的一些互联技术
Infiniband	
说明	用于互联服务器和交换机的半专有技术
优点	极低的延迟（不到100ns）和高吞吐量（高达120Gbps)，使它成为数据中心最强健的互联技术之一
缺点	在服务器硬件上需要昂贵的专有互联设备，并且在与数据中心或集群外部进行通信时，需要交换设备在以太网和Infiniband之间进行转换

在2010年英特尔信息技术峰会上，Intel等宣称已经可以将（10Gb）以太网的延迟做得很低，但那指的是在硬件层面，加上TCP/IP协议的开销之后仍然很难与Infiniband比肩

USTC邮件系统 | OPA技术_百度搜索 | Intel用来取代Infiniband的神 | 2017.10.24 澳大利亚格里菲斯大

www.360doc.com/content/16/0304/01/26186435_539243450.shtml

最常访问 火狐官方站点 新手上路 常用网址 京东商城 综合科研仪器共享中...

360doc 个人图书馆 千万人在用的知识管理与分享平台 我的图书馆 搜文章 找馆友 留言交流

2. Download on our website
3. Enjoy

Intel用来取代Infiniband的神秘技术原来是它！（转载）

Intel OPA

2016-03-04 victor1208 文章来源 阅 604 转 1

分享：微信 转藏到我的图书馆

victor1208 图书馆

关注 64 馆藏 1316

TA的最新馆藏

- 起底明天系：全能型混业金融巨头...
- 合伙创业中股权分配规则，很实用
- 股权激励 | 创业公司如何做股权激...
- 股权退出的七大方式（建议了解）
- 创业公司如何公平分配股权？
- 【原创】系列报告：上海市智慧城...

我要转藏

9:48 AM 10/24/2017

MD并行的软件基础（Software）：MPI

在当前所有的消息传递软件中，最重要最流行的是MPI (Message Passing Interface)，它能运行在所有的并行平台上。 程序设计语言支持C, Fortran等。

- MPI已经成为一种标准，它以与语言独立的形式来定义这个接口库，这个定义不包含任何专用于特别的制造商、操作系统或硬件的特性。由于这个原因，MPI在并行计算界被广泛地接受。

- 可以将MPI看成一个“库”，共有上百个库函数，在FORTRAN和C语言中可以直接对这些函数进行调用。多个进程通过调用这些函数（类似调用子程序），进行通信。

https://en.wikipedia.org/wiki/Message_Passing_Interface

Development of MPI

- History of MPI:
  - 草案: 1992;
  - MPI-1: 1994;
  - MPI-2:1997
- http://www.mpi-forum.org

- 典型的实现包括开源的MPICH、LAM MPI、OPENMPI以及不开源的INTEL MPI。

典型MPI实现

- MPI是一个标准，不属于任何厂商，不依赖于某个操作系统，也不是一种并行编程语言。各个厂商和组织遵循这个标准推出各自的实现。
- MPICH (2)是影响最大、用户最多的MPI实现。由美国的Argonne国家实验室开发的开放源码的MPI软件包。与MPI标准同步发展，简单易用。
- Intel MPI是由Intel公司推出的MPI实现。突出的特色在于提供了灵活的多架构支持，提供DAPL（Direct Access Programming Library）的中间层来支持多架构，Intel MPI在通信协议的选择上无需进行额外配置。

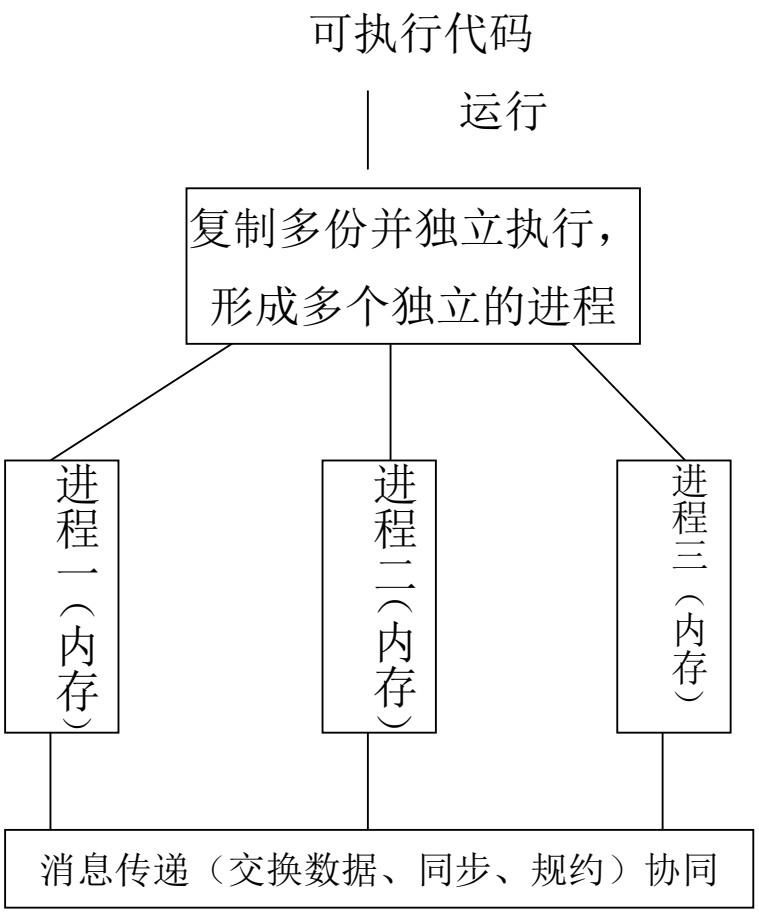
MPI程序特点

- MPI程序是基于消息传递的并行程序；
- 消息传递指的是并行执行的各个进程具有自己独立的堆栈和代码段；
- 各进程作为独立的程序独立执行；
- 进程间的信息交换完全是通过显示的调用通信函数来完成。

MPI并行编程模式

- 单程序多数据流模式（SPMD）
- 多程序多数据流模式（MPMD）

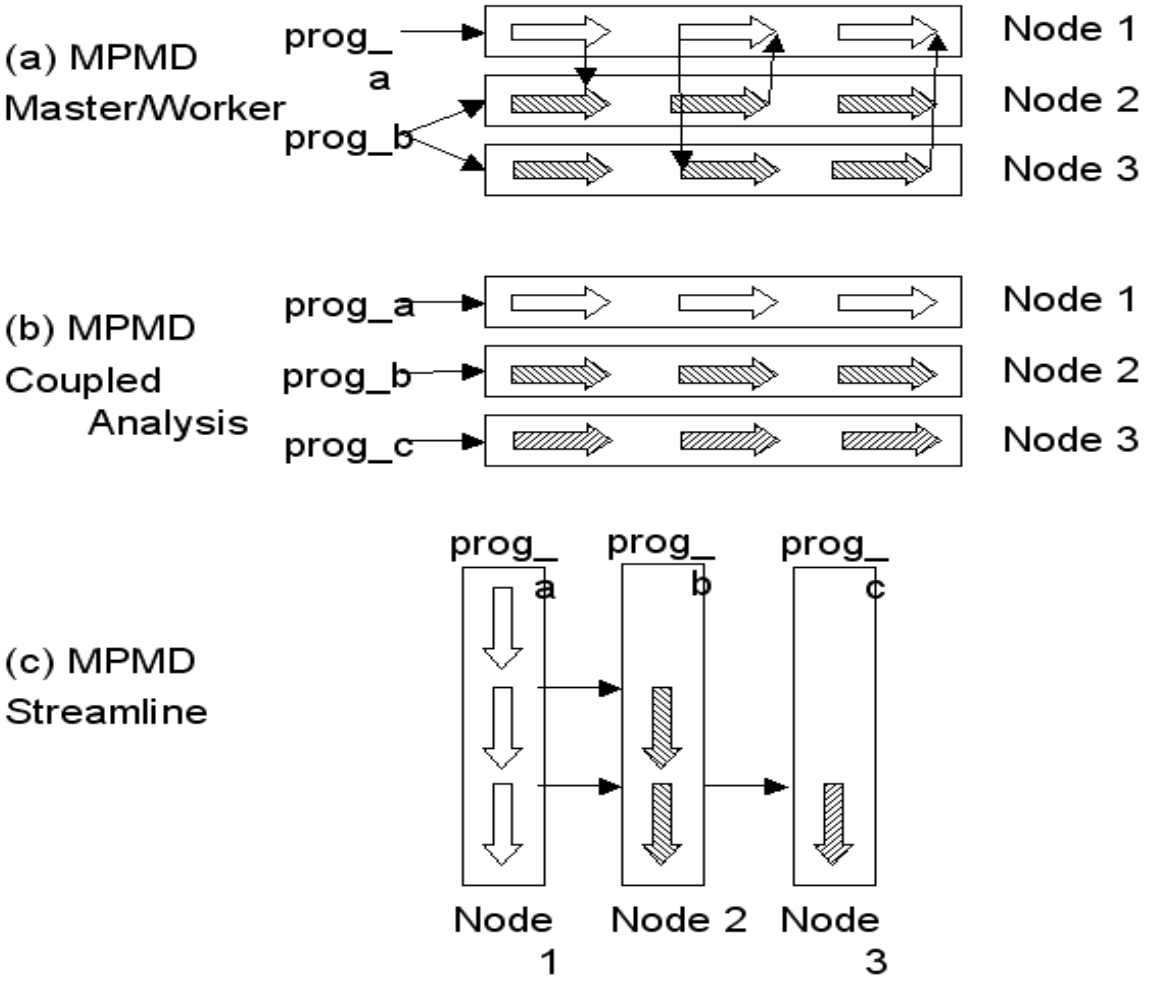
SPMD模式：单程序多数据流



- 一个程序同时启动多份，形成多个独立的进程，在不同的处理机上运行，拥有独立的内存空间，进程间通信通过调用MPI函数来实现。

多程序多数据

- MPMD使用不同的程序处理多个数据集，合作求解一个问题。



MPI并行编程模式

- 为了降低使用和维护并行应用软件的复杂度，一般采用**SPMD**模式

MPI functions

1、MPI初始化：MPI_Init函数

用法：int MPI_Init（&argc , &argv）

每一个MPI进程调用的第一个MPI函数都是MPI_Init。该函数指示系统完成所有的初始化工作，以备对后续MPI库的调用进行处理。

2、MPI结束：MPI_Finalize函数

用法：int MPI_Finalize（）

在一个进程执行完其全部MPI库函数调用后，将调用函数MPI_Finalize，从而让系统释放分配给MPI的资源。它是MPI程序的最后一条可执行语句，否则程序的运行结果是不可预知的。

MPI program

```
#include "mpi.h"
...

int main(int argc,char **argv)
{
   ...
   MPI_Init(&argc,&argv);

   MPI并行程序部分

   MPI_Finalize();
   ...
}
```

获取进程标志和机器名

3、确定进程的标识符

用法：int MPI_Comm_rank（MPI_COMM_WORLD，&id）

当MPI初始化后，每一个活动进程变成了一个叫做MPI_COMM_WORLD的通信域中的成员。通信域是一个不透明的对象，提供了在进程之间传递消息的环境。在一个通信域内的进程是有序的。在一个有p个进程的通信域中，每一个进程有一个唯一的序号（ID号），取值为0~p－1。进程可以通过调用函数MPI_Comm_rank来确定它在通信域中的序号。

4、确定进程数量

用法：int MPI_Comm_size（ MPI_COMM_WORLD ， &p）

进程通过调用函数MPI_Comm_size来确定一个通信域中的进程总数。

获取进程标志和机器名

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char **argv)
{
  int myid, numprocs;
  int namelen;
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Get_processor_name(processor_name,&namelen);
  printf("Hello World! Process %d of %d on %s\n"
            ,myid, numprocs, processor_name);
  MPI_Finalize();
}
```

```
Hello World! Process 0 of 4 on name1
Hello World! Process 1 of 4 on name2
Hello World! Process 2 of 4 on name3
Hello World! Process 3 of 4 on name4
```

MPI的点对点通信：发送

- 有消息传递功能的并行程序，消息传递是MPI编程的核心功能，掌握了MPI消息传递编程方法就掌握了MPI编程的核心。
- int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

  参数说明：

  buf:发送缓冲区的起始地址

  count:将要发送的数据的个数

  datatype:发送数据的数据类型

  dest:目的进程标识号

  tag:消息标志

  comm：通信域

  MPI_Send将发送缓冲区中的count个datatype数据类型的数据发送到目的进程，目的进程在通信域中的标识号是dest,本次发送的消息标志是tag,使用这一标志，就可以把本次发送的消息和本进程向同一目的进程发送的其他消息区别开。

MPI的点对点通信：接收

- int MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status status)

  参数说明：

  buf:接收缓冲区的起始地址

  count:将接收的数据的个数（最多个数）

  datatype: 接收数据的数据类型

  source:接收数据的来源进程标识号

  tag:消息标志，与发送标志相匹配

  comm：通信域

  status:返回类型(是由三个域组成的结构类型，这三个域分别是:MPI_SOURCE、MPI_TAG和MPI_ERROR)

  MPI_ Recv函数是MPI中基本的消息接收函数，MPI_ Recv从指定的进程source接收消息，并且该消息的数据类型和消息标识和本接收进程的datatype和tag相一致，接收到的消息所包含的数据元素的个数最多不能超过count。

MPI通信程序实例

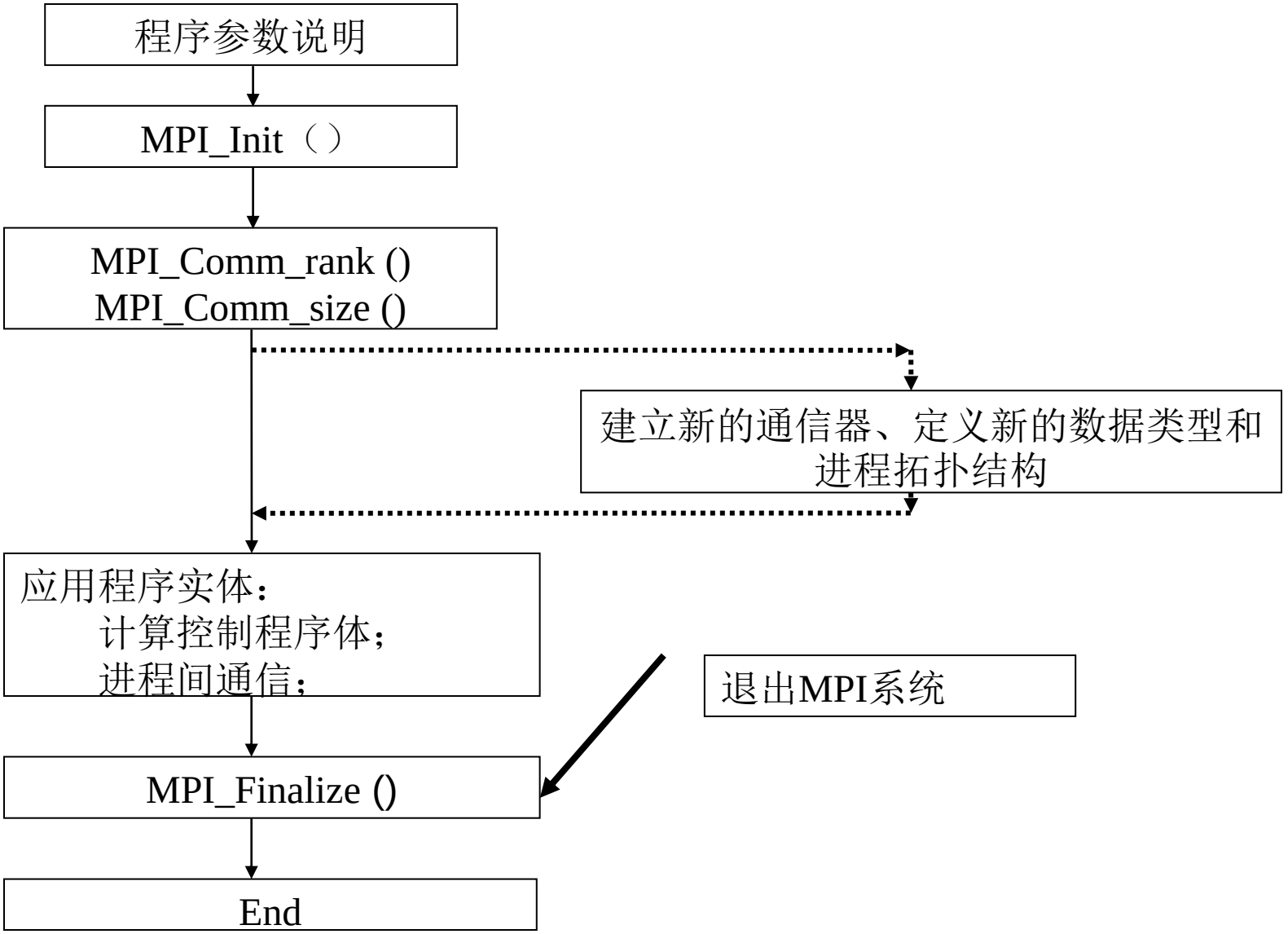
```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char** argv)
{
  int myid,numprocs, source;
  MPI_Status status;
  char message[100];
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  if (myid != 0)
  {
    strcpy(message, "Hello World!");
    MPI_Send(message,strlen(message)+1, MPI_CHAR,
             0,99,MPI_COMM_WORLD);
  }
```

MPI通信程序实例

```
  else
  {
    for (source = 1; source < numprocs; source++)
    {
      MPI_Recv(message, 100, MPI_CHAR, source, 99,
                 MPI_COMM_WORLD, &status);
      printf("I am process %d. I recv string '%s' from process %d.\n",
             myid,message,source);
    }
  }
  MPI_Finalize();
}
```

```
I am process 0. I recv string 'Hello World!' from process 1
I am process 0. I recv string 'Hello World!' from process 2
I am process 0. I recv string 'Hello World!' from process 3
```

Flow-chart of MPI



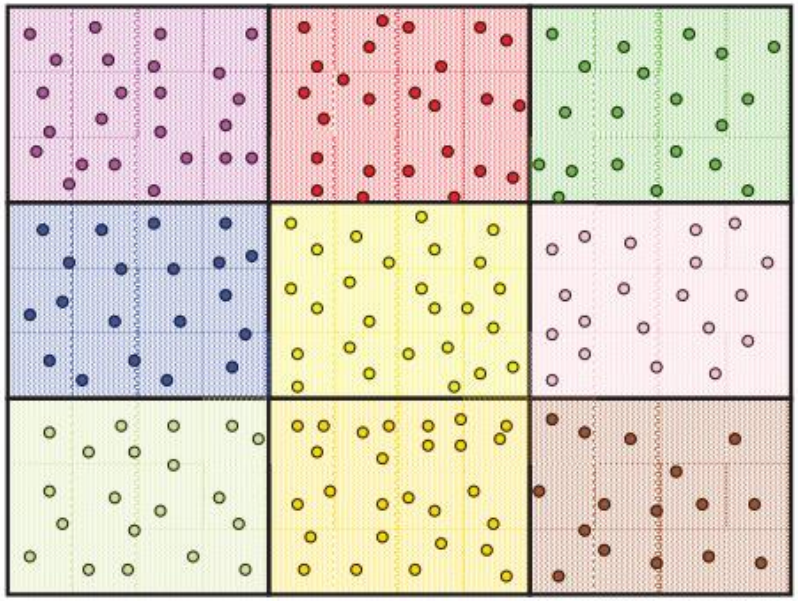
MD并行的算法基础
Particle Decomposition

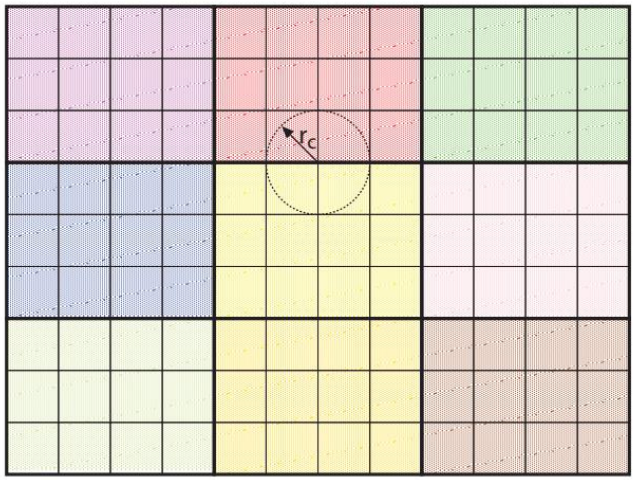
PD, also called force decomposition, is the simplest type.
At the start of the simulation, particles are assigned to processors, then forces between particles need to be assigned to processors such that the force load is evenly balanced.
This decomposition requires that each processor knows the coordinates of at least half of the particles in the system, thus for a high number of processors N, about $N \times N/2$ coordinates need to be communicated.
Because of this quadratic relation PD does not scale well.

PD算法比较容易实现，但是由于它需要各处理器不停交换粒子的位置信息，通信开销相当大，难以实现大规模MD模拟。同时它对硬件尤其是网络性能要求高，否则并行效率会大打折扣。

MD并行的算法基础
Domain Decomposition

DD算法的原理是：模拟盒子被分解为一些相同大小的区域小盒子，盒子的大小可以根据截断半径得到，一般取边长等于或者大于截断半径。每个处理器分担一个盒子内粒子的作用力、位置和速度等，这样粒子可以在小盒子之间自由移动。这样的好处在于一个处理器只要考虑其自身分担区域与相邻区域的粒子间作用力，从而大大减少处理器之间的通信量。





DD的优势与不足

DD通常可以得到很高的并行效率，适用于大量复杂粒子的模拟。但由于处理器存储的是特定范围内的粒子，在运行过程中难免会发生有些处理器存储粒子多，有些存储少的情况，这样会造成处理器负载不均。粒子数目少的处理器早早完成运算，然后等待负荷较重的处理器运算完才能交换信息，这会大大影响并行效率。所以DD算法中负载平衡是关键，其编程相当复杂。

MD并行的算法基础

Dynamic Load Balancing（动态负载平衡）

Load imbalance may occur due to three reasons: (1) inhomogeneous particle distribution; (2) inhomogeneous interaction cost distribution (charged/uncharged, water/non-water, etc); (3) statistical fluctuation (only with small particle numbers).

Therefore we need a dynamic load balancing algorithm where the volume of each DD cell can be adjusted independently. MD can automatically turn on the dynamic load balancing during a simulation when the total performance loss due to the force calculation imbalance is 5% or more.

GPU通用计算

10年前所有人都认为GPU只服务于制图、动画和游戏等电子娱乐领域，因为GPU发明的目的就是为了应对繁杂的3D图像处理。但是谁都没有想到目前GPU的内部架构和应用范围已经发生了翻天覆地的变化。

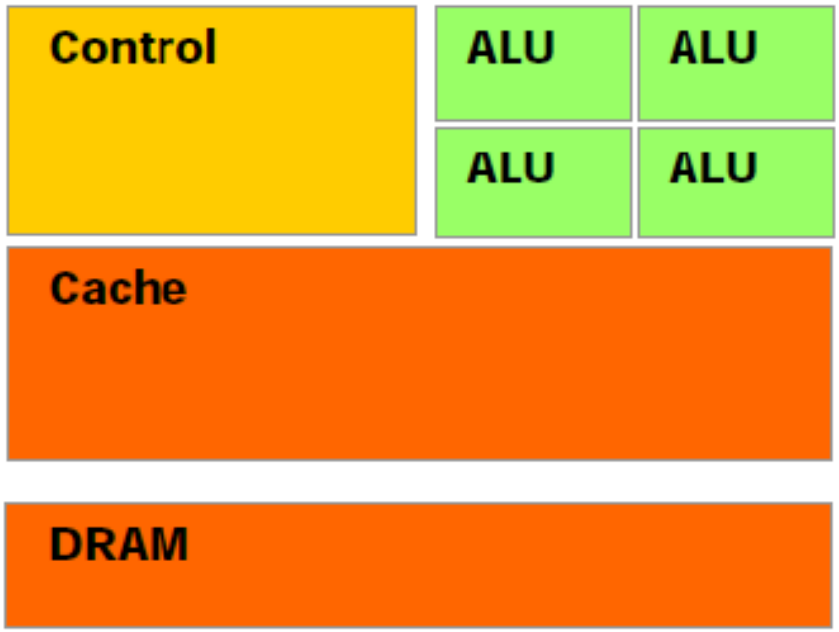
通用GPU将显卡的应用范围扩展到了图形之外。目前许多结果已经表明：将GPU用于解决数据并行计算问题可以明显地提高系统的运行速度。

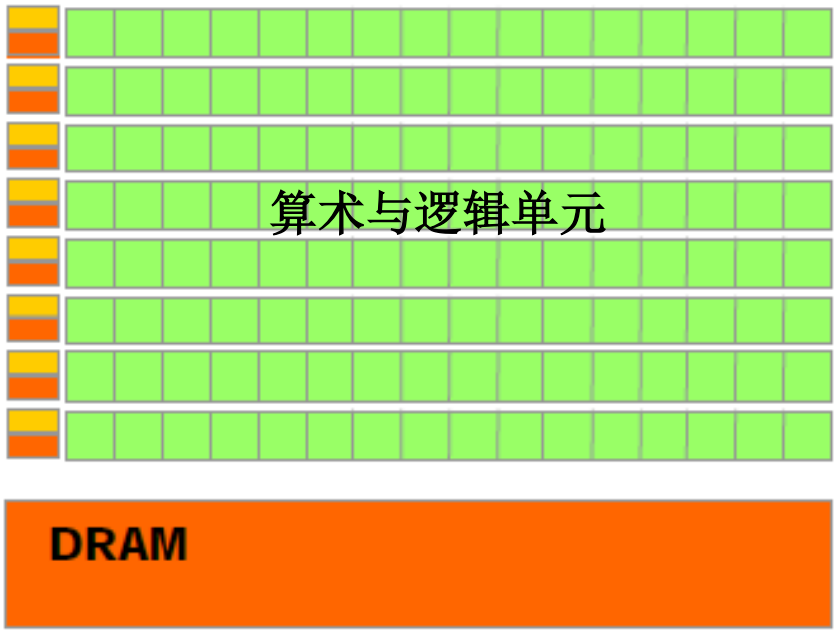
CPU与GPU

Comparison of CPU and GPU Hardware Architecture

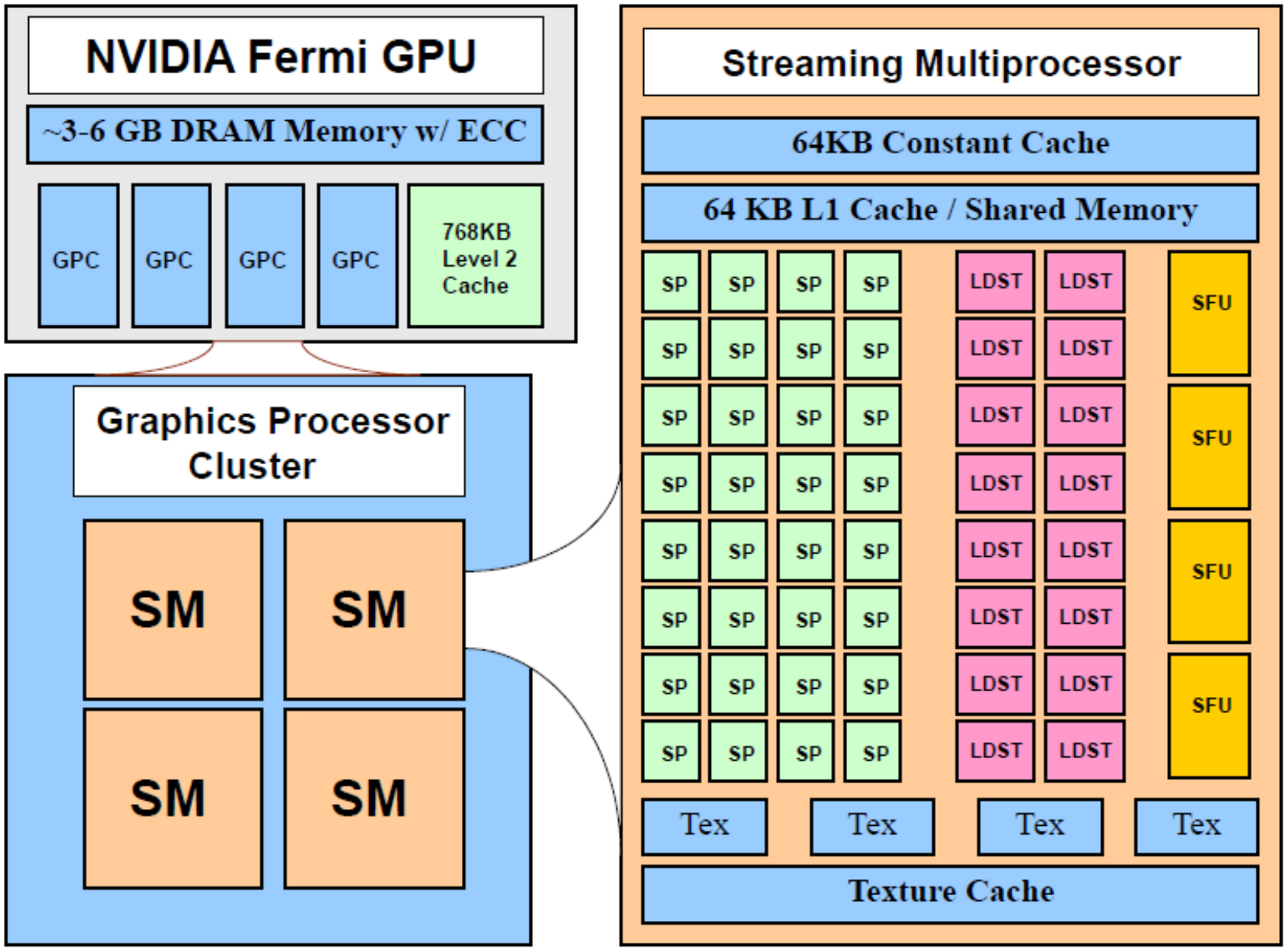
CPU: Cache heavy, focused on individual thread performance

GPU: ALU heavy, massively parallel, throughput oriented

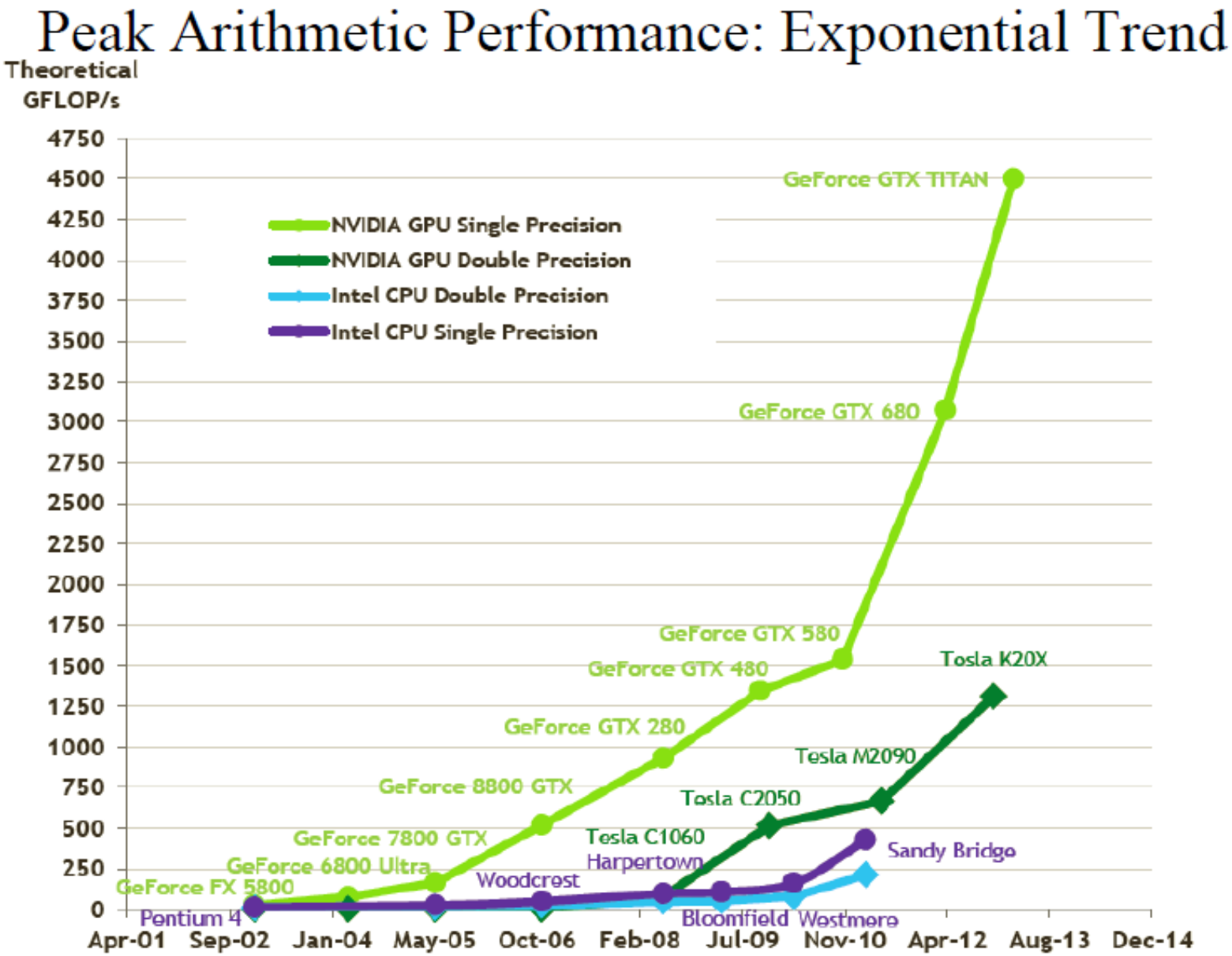




GPU架构

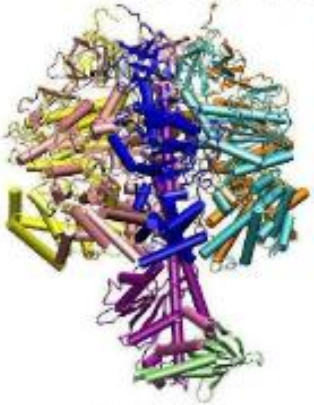

NVIDIA Fermi GPU
~3-6 GB DRAM Memory w/ ECC
GPC
GPC
GPC
GPC
768KB Level 2 Cache
Graphics Processor Cluster
SM
SM
SM
SM
Streaming Multiprocessor
64KB Constant Cache
64 KB L1 Cache / Shared Memory
SP
LDST
SFU
Tex
Texture Cache

Performance of GPU



NAMD: Parallel Molecular Dynamics

2002 Gordon Bell Award



ATP synthase

PSC Lemieux

34,000 Users, 1200 Citations

Computational Biophysics Summer School

Blue Waters Target Application

Illinois Petascale Computing Facility

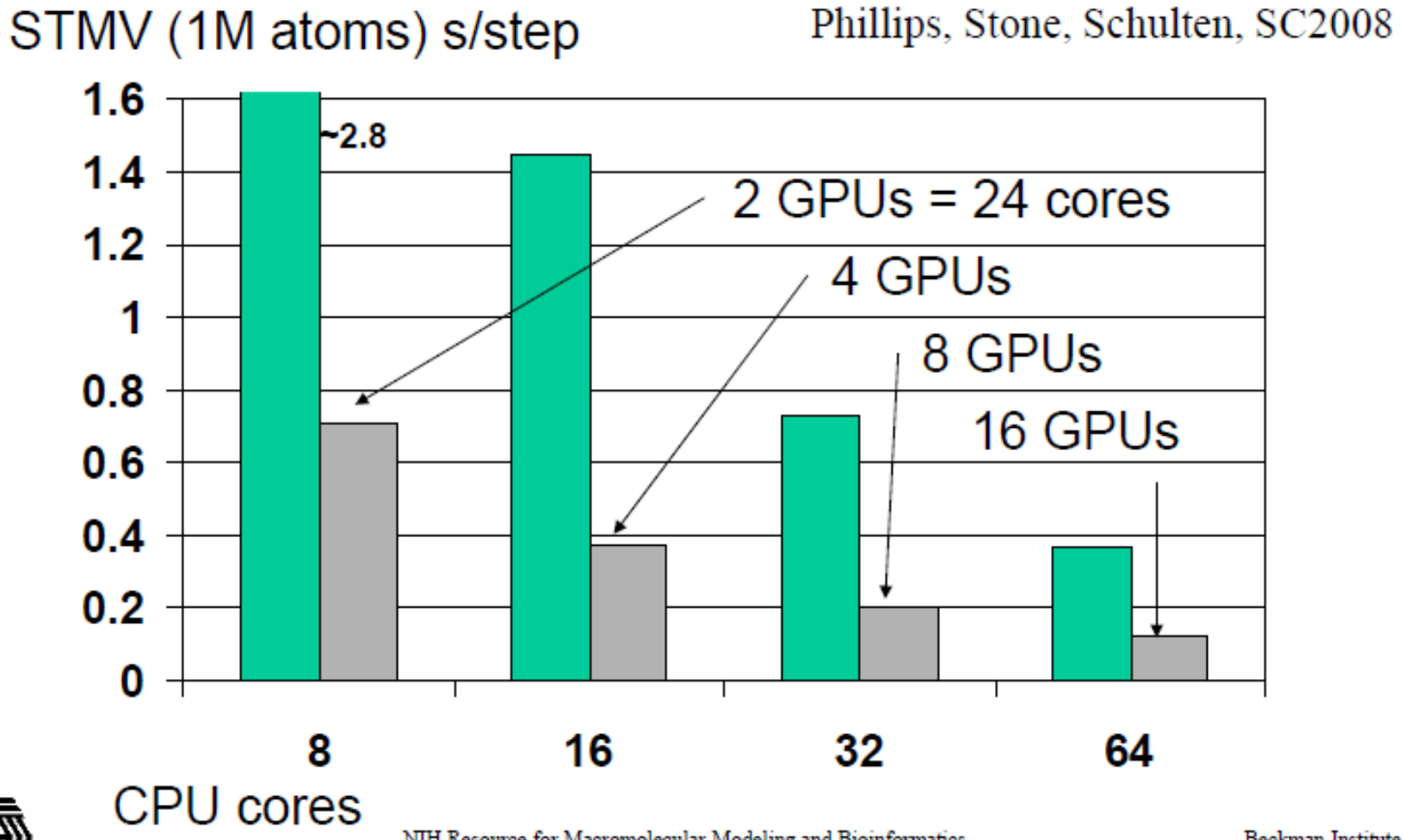
GPU Acceleration

NVIDIA Tesla

NCSA Lincoln

NCSA Lincoln Cluster Performance

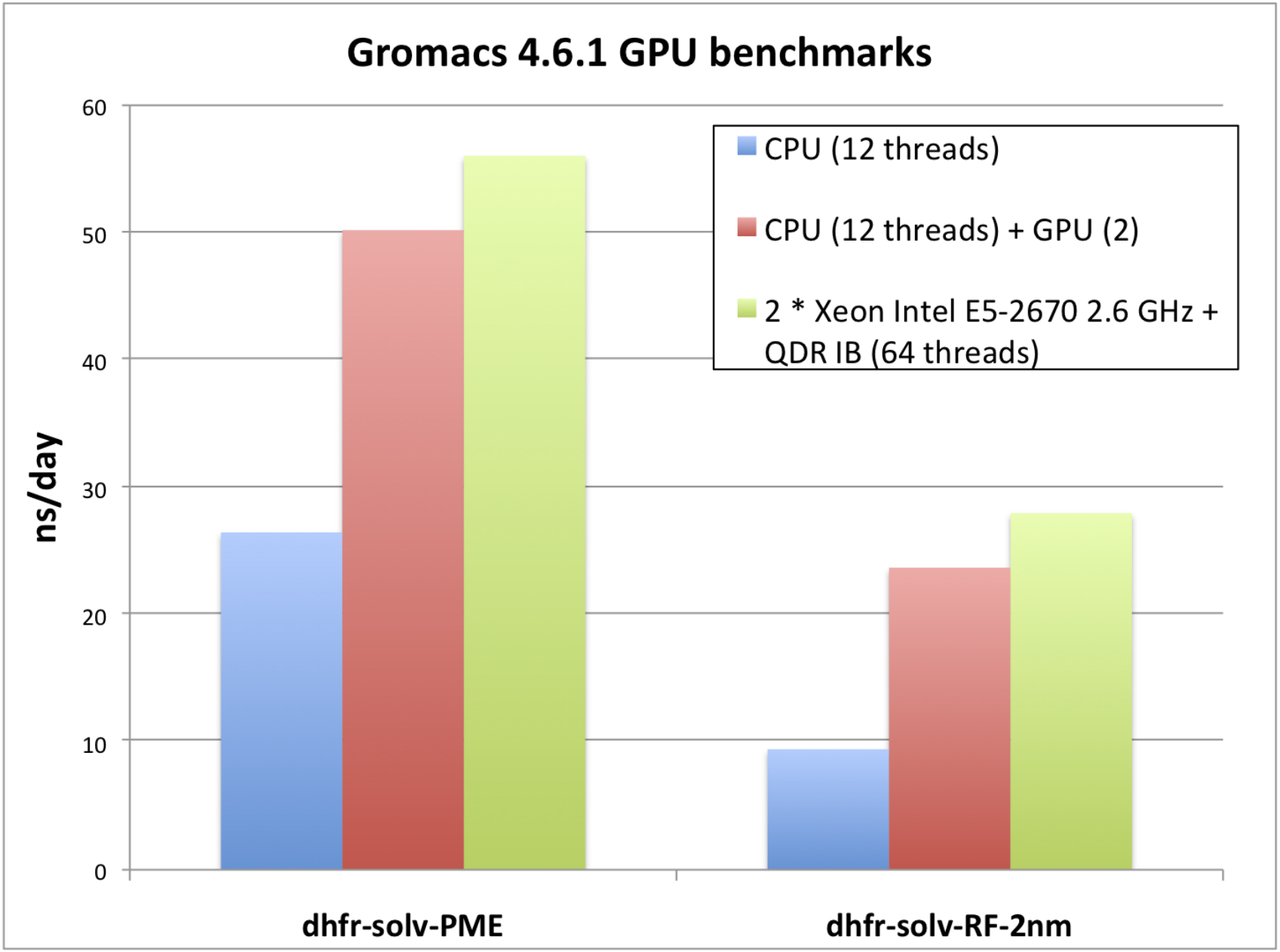
(8 Intel cores and 2 NVIDIA Telsa GPUs per node)

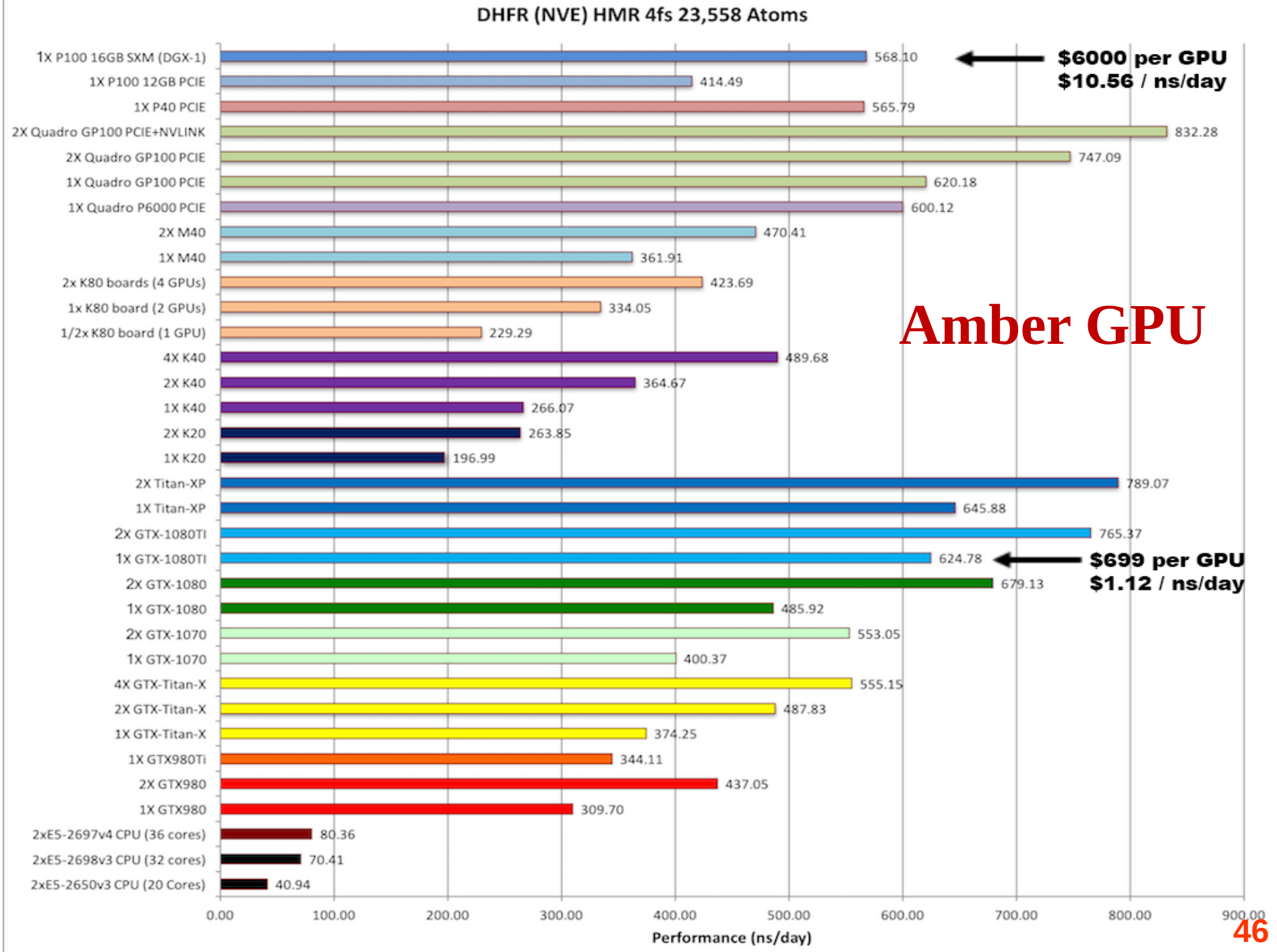


NIH Resource for Macromolecular Modeling and Bioinformatics
http://www.ks.uiuc.edu/

Beckman Institute, UIUC

GROMCS on GPU




DHFR (NVE) HMR 4fs 23,558 Atoms
1X P100 16GB SXM (DGX-1) 568.10
1X P100 12GB PCIE 414.49
1X P40 PCIE 565.79
2X Quadro GP100 PCIE+NVLINK 832.28
2X Quadro GP100 PCIE 747.09
1X Quadro GP100 PCIE 620.18
1X Quadro P6000 PCIE 600.12
2X M40 470.41
1X M40 361.91
2x K80 boards (4 GPUs) 423.69
1x K80 board (2 GPUs) 334.05
1/2x K80 board (1 GPU) 229.29
4X K40 489.68
2X K40 364.67
1X K40 266.07
2X K20 263.85
1X K20 196.99
2X Titan-XP 789.07
1X Titan-XP 645.88
2X GTX-1080TI 765.37
1X GTX-1080TI 624.78
2X GTX-1080 679.13
1X GTX-1080 485.92
2X GTX-1070 553.05
1X GTX-1070 400.37
4X GTX-Titan-X 555.15
2X GTX-Titan-X 487.83
1X GTX-Titan-X 374.25
1X GTX980Ti 344.11
2X GTX980 437.05
1X GTX980 309.70
2xE5-2697v4 CPU (36 cores) 80.36
2xE5-2698v3 CPU (32 cores) 70.41
2xE5-2650v3 CPU (20 Cores) 40.94
$6000 per GPU
$10.56 / ns/day
$699 per GPU
$1.12 / ns/day
Amber GPU
0.00
100.00
200.00
300.00
400.00
500.00
600.00
700.00
800.00
900.00
Performance (ns/day)