大规模并行计算
-- A walk through example

田荣
中国科学院 计算技术研究所

rongtian@ncic.ac.cn

2010年9月1日

- 参考资料
  - **http://www.cs.utk.edu/~dongarra/WEB-PAGES/SPRING-2006/Lect08.pdf**
  - **http://www2.hpcl.gwu.edu/pgas09/tutorials/upc_tut.pdf**
  - **http://upc.lbl.gov/publications/UPCReview-July2009/22Jul2009/upc4gpu.pdf**

课程内容

1. 为什么需要大规模计算？
2. Grand challenge
    1. 美国，日本，欧洲
    2. 中国：现状，挑战与机遇
3. 什么是大规模并行计算？
    1. 何谓大规模？
    2. 科学与工程中的大规模并行有限元计算
4. 并行计算，算法，与优化
    1. A work through example
5. 小结与讨论
    1. 众核加速之10年变迁预测

为什么需要大规模并行计算

- 简单地讲：内存限制(大问题)，计算时间(钱，经济效益)
- 哲学角度：计算科学
- 长远意义：国家战略需要

Grand challenge

-- 它的解决能极大地冲击科学、工程和经济领域的进步的大规模计算问题

- **美国DOE资助的P级计算项目(5年17M$ each，2008.9立项)**
  - **Predictive Modeling and Simulation of High-Energy Density Dynamic Response of Materials (Caltech) :穿甲，超高速(10km/s)碰撞**
  - **Prediction of Reliability, Integrity and Survivability of Microsystems (Purdue): 微机电系统全物理模拟**
  - **Predictive Simulations of Multi-Physics Flow Phenomena with Application to Integrated Hypersonic Systems (Stanford)：超音速喷气火箭的整体模拟**
  - **Radiative Shock Hydrodynamics (Michigan) ：辐射流体动力学**
  - **Predictive Engineering and Computational Sciences (U-Texas)：航天飞机返回大气层全系统模拟**
- **NCCS:美国计算科学中心INCITE项目：Innovative and Novel Computational Impact on Theories and Experiments**
  - **Jaguar (22万核)**

- **特点：“Predictive”，物理实验不允许或非常困难**

我国大规模并行计算现状、挑战、与机遇

1. 我国超级计算能力：2007年“曙光4000A” Top10；2008年“曙光5000A” Top10；2009年“天河一号” Top5；2010年“曙光星云”Top2. 目前，北京、深圳、天津、沈阳、武汉、广州、济南、成都、长沙……各大城市都在投巨资建立超级计算中心，从十万亿次、百万亿次到千万亿次，甚至万万亿次规模的大型超级计算中心。

2. **但是，我国高性能计算硬件高性能和应用低效率之间的矛盾十分突出。**目前我国大规模并行数值模拟软件（高端软件）严重不足。据上海超级计算中心2009年统计：我国60%的HPC应用为16处理器核以下的规模(受国外软件license(费用)的限制)，所用的开源大规模并行代码99%来自国外。而在美国橡树岭国家实验室，3万核以下的计算占到50%，4~9万核的计算占18%。

Recently Run Jobs at NCCS	
Processors	**Project**
40,992	Cellulosic Ethanol: Physical Basis of Recalcitrance to Hydrolysis of Lignocellulosic Biomass
34,764	FY2010 DOE/ASCR Joule Metric on Computational Effectiveness
28,800	FY2010 DOE/ASCR Joule Metric on Computational Effectiveness
19,200	Verification and Validation of Petascale Simulation of Turbulent Transport in Fusion Plasmas
7,200	An Integrated Approach to the Rational Design of Chemical Catalysts

美国Jaguar上计算任务列表

我国大规模并行计算现状、挑战、与机遇

3. **Everything goes parallel**。现有许多软件，特别是PC机时代诞生的数值模拟软件，其架构已不适应新型体系结构和海量并行计算的要求，需要重新设计。最典型的例子：大多的并行CAE软件都是通过调用并行求解器实现并行化，这种“串(前处理) -- 并(求解) -- 串(后处理)”的并行模式迟早会遇到扩展性的问题。因此 **“发挥海量并行性和大规模化”应该是我国新一代科学与工程计算软件实现跨越发展的契机**。

一些观点摘要

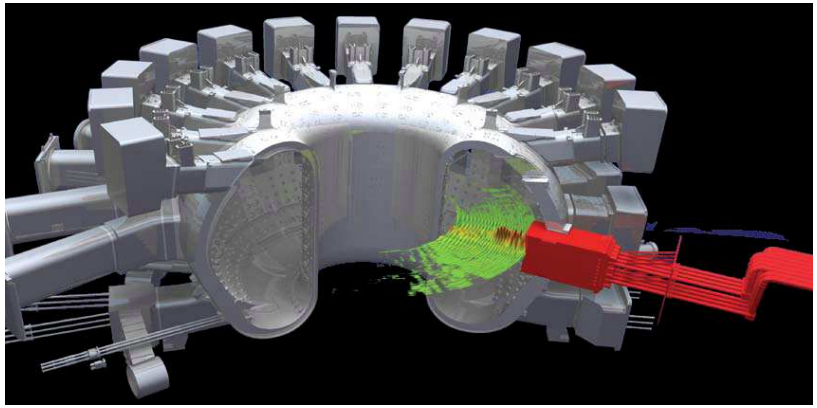
- **"Why the U.S. must lead in supercomputing" Jun 16th, 2010**
  - **"… challenges to U.S. leadership in supercomputing and chip design threaten our country's economic future."**
- **"China's Nebulae No. 2" Jun 1st, 2010**
  - **"Nebulae reports an impressive theoretical peak capability of almost 3 petaflop/s — the highest ever on the TOP500."**

- "有时候，国内的超算中心都很悲惨。" "因为我国超算领域的软件水平还很差，系统的应用规模上不去。"
- 美国材料分析领域的超前性比我国高10倍甚至百倍，其超算中心的作业规模也非常大，基本都是5～10万个 CPU同时运行，而我国每天最多只有几百或上千个CPU同时运行。"并非我们硬件比人家差几百倍，而是我们在高性能计算的软件方面非常落后，并行计算能力还很差。"
- "在这种情况下，我国的超算中心只能去购买发达国家的商业软件，有些专用软件的价格非常昂贵，甚至比建一个硬件平台还要贵，一般的超算中心根本承担不起。还有一些高科技产品，由于国外出口政策的限制，即使有钱也买不到。"

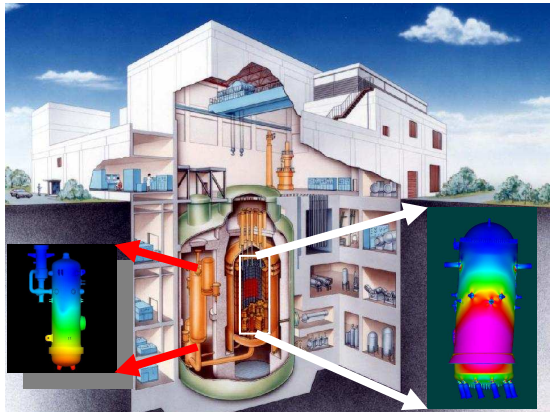
大规模并行计算应用软件发展将迎来好时机！

何谓大规模并行计算？

- 何谓大规模？
  - 在2002年日本制造出世界上最快的超级计算机“地球模拟器”的时候，日本人把百万自由度有限元求解作为当时追求的目标。8年后的今天，即使是千万单元级的计算规模，也对我国现有的高性能计算资源完全不构成挑战。目前世界上最大的有限元计算规模为10亿单元级(流体力学计算)。
  - 大规模并行计算一般是指具有一定挑战性的并行计算课题 (比如：16/32核以下的并行计算已经一般不能称之为大规模并行计算了)。



美国(2010-2015, 122M$)



“虚拟核电站”

日本(2005~)

何谓大规模并行计算？

- 目前世界上最快超级计算机：
  - **#1 Jaguar: 224,256cores (18688x16Istanbul, 2.6GHz, 16GB of DDR2-800 memory, and a SeaStar 2+ router (57.6GB/s)), 300T memory, peak: 2.3 Pflop/s, 安装于美国橡树岭国家实验室，由NCCS管理**
  - **#2 Nebulae(中国): Intel 6-core Xeon X5650 CPU, 2.66 GHz, 56,160 cores on 9,360 processors on 117 cabinets in 7 rows Each blade is equipped with a Nvidia Tesla C2050 GPU; 4680 blades interconnected by a two-level InfiniBand QDR 4x network; the sustained Linpack speed, Rmax , is 1.271 PFlop**

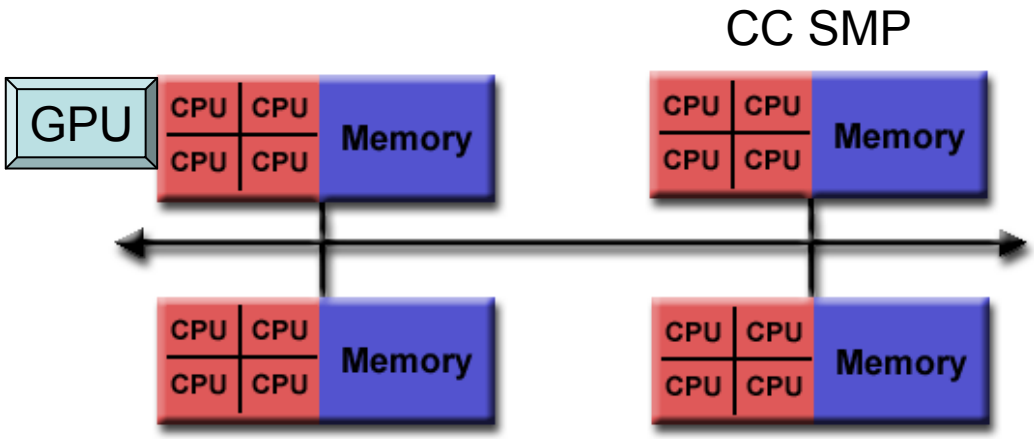
世界各国的“计算科学”中心：

- 美国：计算科学中心 www.nccs.gov (2004)
- 日本：計算科学研究機構(AICS) (2010年7月1日)
- 欧洲：HPC in Europe Taskforce (HET, 2006)
- 中国：国家超级计算中心深圳(筹) (2010)

猜猜这是什么？

并行计算机体系结构与编程模型

- 共享内存：OpenMP, 可提供有限的并行加速
- 分布内存，分布共享内存：
  - MPI
  - MPI-2单边通讯(put and get) √
  - PGAS (partitioned global-address-space √
  - GPU集群：MPI+OpenCL/CUDA √



Networking of multiple SMPs, certain SMPs may be equipped with GPGPUs

MPI-2单边通讯

- Send-receive: 双边通讯
- 单边通讯
  - 只需一方“主动”参与通讯
  - No matching, no ordering
  - 允许用户直接读写分布进程的内存
  - 理论上比双边通讯更有效，更方便
  - 许多网络支持单边通讯：InfinitBand, Myrinet等
  - MPI_Put(), MPI_Get(), MPI_Accumulate()
  - 同步操作: fence, post-start-complete, lock-unlock.
  - 早期实现基于send, receiv, 需一次握手的过程
  - 目前MVAPICH2等支持RDMA-based单边通讯(RDMA read/write)

MPI-2单边通讯: 例

```
If( 0 == rank ) {
        MPI_Isend( to_1 );
} else if( 1 == rank ) {
        MPI_Irecv( from_0 );
}
MPI_Wait()
```

MPI send/recv

```
If( 0 == rank )
    MPI_Win_create( NULL, 0, sizeof(int), MPI_INFO_NULL, comm, &win );
else if ( 1 == rank )
    MPI_Win_create( addr, length, sizeof(int), MPI_INFO_NULL, comm, &win );
MPI_Win_fence( 0, win );
If( 0 == rank )
    MPI_Put( to_remote_mem )
MPI_Win_fence( 0, win )
MPI_Win_free( &win )
```

MPI put/get

注释

- **int MPI_Win_create(void* base, MPI_Aint size, int disp_unit, MPI_Info info, MPI_Comm comm, MPI_Win* win)**
- **int MPI_Win_free(MPI_Win* win)**
  - **base: memory window起始地址**
  - **Disp_unit: 数据类型字长**
  - **info: 为优化MPI库预留 MPI_INFO_NULL**
  - **MPI_Win:** *Memory Window*, 连续内存区域, 用于put/get/accumulate操作, 为局部内存, 每一进程可指定不同的起始地址和大小

- **int MPI_Put/Get (void* origin_addr, int origin_count, MPI_Datatype origin_datatype int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target_datatype, MPI_Win win)**
  - **Nonblocking**

- **int MPI_Win_fence(int assert, MPI_Win win)**
  - **标志通讯开始和结束**
  - **在两个MPI_Win_fence之间不能同时对memory window通行进行局部和远程put/accumulate更新**

Fence同步机制

Process 0
MPI_Win_fence(win)
MPI_Put(1)
MPI_Get(1)
MPI_Win_fence(win)

Process 1
MPI_Win_fence(win)
MPI_Put(0)
MPI_Get(0)
MPI_Win_fence(win)

Collective on the communicator, 全局同步语义

Post-start-complete-wait同步机制

Process 0

MPI_Win_start(1)
MPI_Put(1)
MPI_Get(1)
MPI_Win_complete(1)

Process 1

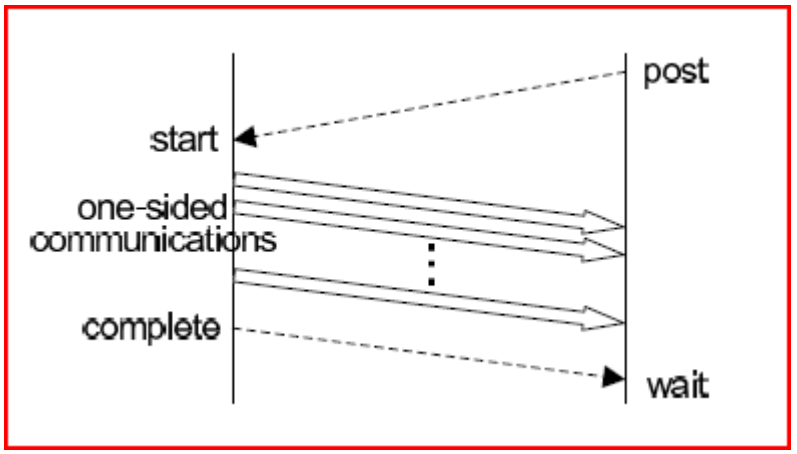
MPI_Win_post(0,2)

MPI_Win_wait(0,2)

Process 2

MPI_Win_start(1)
MPI_Put(1)
MPI_Get(1)
MPI_Win_complete(1)

只在相关联进程间同步
使用MPI groups
点到点同步语义



Lock-unlock同步机制

Process 0	*Process 1*	*Process 2*
MPI_Win_create(&win)	MPI_Win_create(&win)	MPI_Win_create(&win)
MPI_Win_lock(shared,1)		MPI_Win_lock(shared,1)
MPI_Put(1)		MPI_Put(1)
MPI_Get(1)		MPI_Get(1)
MPI_Win_unlock(1)		MPI_Win_unlock(1)
MPI_Win_free(&win)	MPI_Win_free(&win)	MPI_Win_free(&win)

完全的passive同步：*Process 1*只需准备好memory window
无需调用其他任何MPI函数

分区全局地址空间(PGAS)编程模型

- MPI不足：
  - 需要大量缓存(特别是计算规模大时)
  - 异构计算编程困难

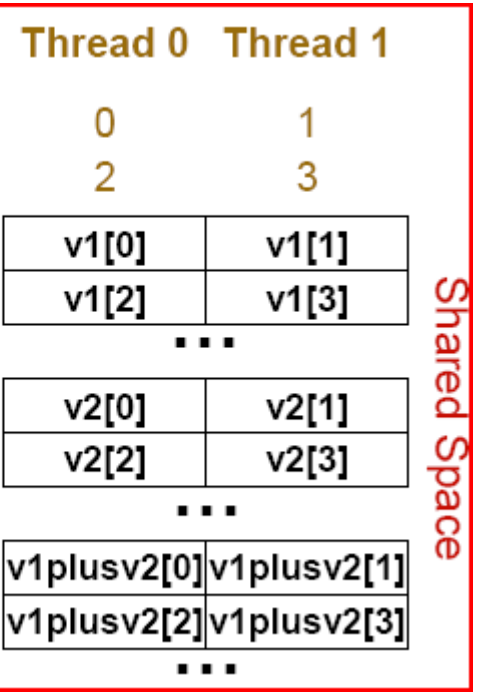
- 基于分区全局地址空间(PGAS)的新的编程模型
  - 为SPMD并行计算提供单一的地址空间
  - UPC: Unified Parallel C
    - 基于对C语言的显式并行扩展
    - 面向片内众核, 分布并行, 以及最近的异构环境
      - 对于片上众核，UPC比MPI更节省内存
      - 对于分布并行，UPC基于单边通讯(网络支持RDMA)
    - 较好的并行编程效率
    - 还是新鲜事物：V1.2: 2005年5月

UPC

- 目前UPC编译器有:
  - Hewlett-Packard
  - Cray
  - IBM
  - Berkeley
  - Intrepid (GCC UPC)
  - MTU
- UPC应用开发工具
  - Totalview
  - PPW (parallel performance wizard) from UF
  - Eclipse tools from IBM

UPC例1: 向量加

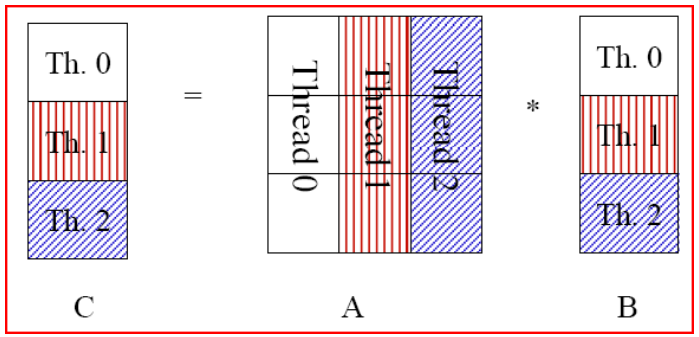
```
//vect_add.c

#include <upc_relaxed.h>
#define N 100*THREADS

shared int v1[N], v2[N], v1plusv2[N];
void main() {
    int i;
    for(i=MYTHREAD; i<N; i+=THREADS)
          v1plusv2[i]=v1[i]+v2[i];
}
```



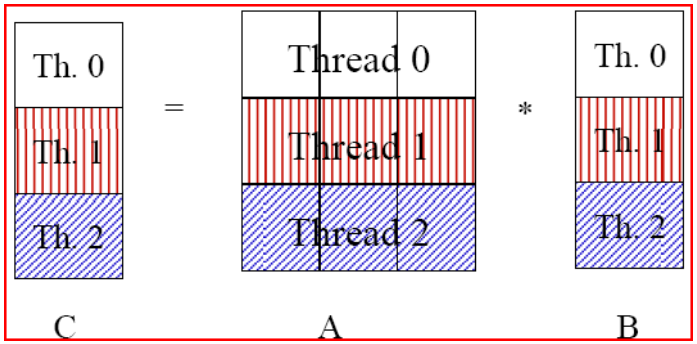
来源:http://www2.hpcl.gwu.edu/pgas09/tutorials/upc_tut.pdf

UPC例2：矩阵向量乘

```
// vect_mat_mult.c
#include <upc_relaxed.h>
shared int a[THREADS][THREADS] ;
shared [THREADS] int a[THREADS][THREADS];
shared int b[THREADS], c[THREADS] ;
void main (void)
{
   int i, j;
   upc_forall( i = 0 ; i < THREADS ; i++; i){
      c[i] = 0;
      for ( j= 0 ; j < THREADS ; j++)
         c[i] += a[i][j]*b[j];
   }
}
```



shared int a[THREADS][THREADS] ;

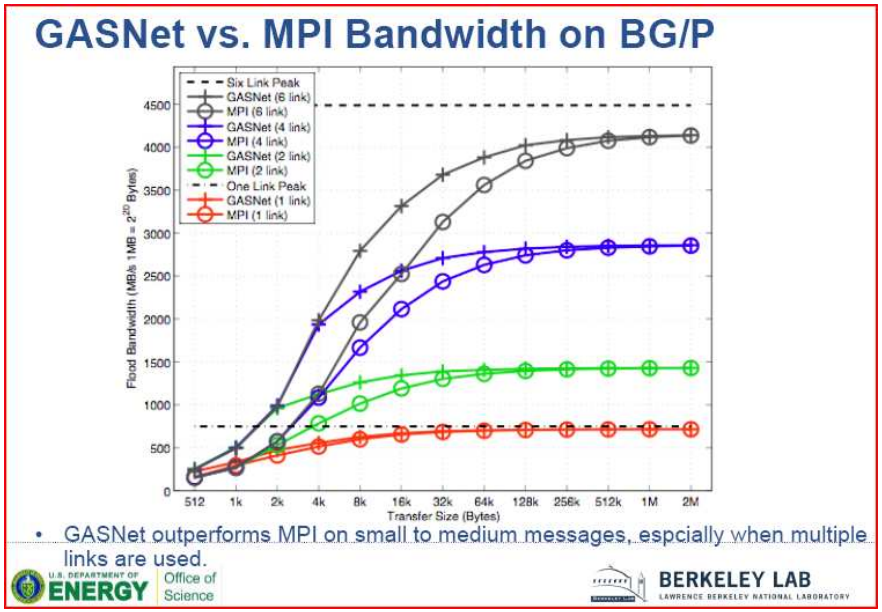


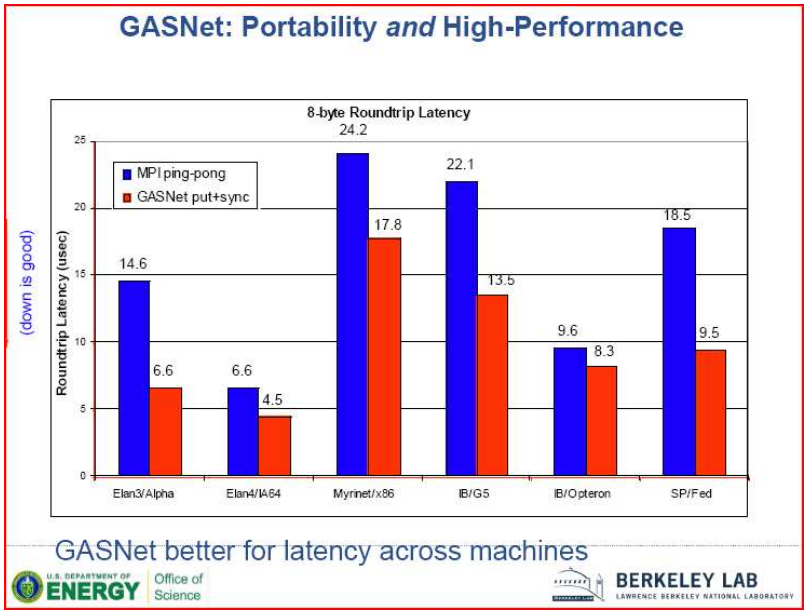
shared [THREADS] int a[THREADS][THREADS];

数据分割

来源:http://www2.hpcl.gwu.edu/pgas09/tutorials/upc_tut.pdf

UPC vs. MPI



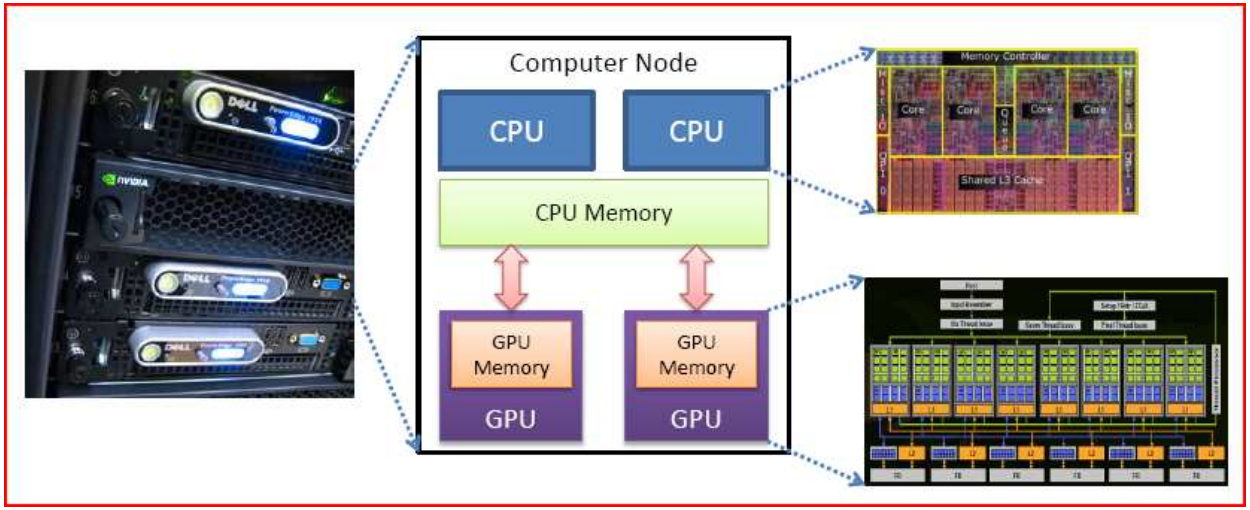


http://www.speedup.ch/workshops/w38_2009/pdf/Yelick-UPC-Lausanne09.pdf

如果你开始一个新的project，可以考虑UPC！

GPU集群：MPI+OpenCL/CUDA

- CUDA：nVidia
- OpenCL: GPGPU计算统一接口标准
- MPI+OpenCL √



http://upc.lbl.gov/publications/UPCReview-July2009/22Jul2009/upc4gpu.pdf

一个“Hello World” OpenCL程序

```
//
// Hello World for OpenCL
// Copyright 2009 David Black-Schaffer
//

#include "OpenCL/opencl.h"
#include <stdlib.h>
#include <stdio.h>

#define LENGTH 10240


// Define our kernel. It just calculates the sin of the input data.
char *source = {
  "kernel calcSin(global float *data) {\n"
  "  int id = get_global_id(0);\n"
  "  data[id] = sin(data[id]);\n"
  "}\n"
};

// Note: there is no error handling

int main (int argc, const char * arg

  // OpenCL Objects
  cl_device_id device;
  cl_context context;
  cl_command_queue queue;
  cl_program program;
  cl_kernel kernel;
  cl_mem buffer;


  // Create and initialize the input data
  cl_float *data;
  data = (cl_float*)malloc(sizeof(cl_float)*LENGTH);
  for (int i=0; i<LENGTH; i++)
    data[i] = i;
```

```
  // Create and initialize the input data
  cl_float *data;
  data = (cl_float*)malloc(sizeof(cl_float)*LENGTH);
  for (int i=0; i<LENGTH; i++)
    data[i] = i;


  // Setup OpenCL
  clGetDeviceIDs(NULL, CL_DEVICE_TYPE_DEFAULT, 1, &device, NULL);
  context = clCreateContext(NULL, 1, &device, NULL, NULL, NULL);
  queue = clCreateCommandQueue(context, device, (cl_command_queue_properties)0, NULL);


  // Setup the input
  buffer = clCreateBuffer(context, CL_MEM_COPY_HOST_PTR, sizeof(cl_float)*10240, data, NULL);


  // Build the kernel
  program = clCreateProgramWithSource(context, 1, (const char**)&source, NULL, NULL);
                                                                    ;

                                                                    ;
                                                  bal_dimensions, NULL, 0, NULL, NULL);


  // Read back the results
  clEnqueueReadBuffer(queue, buffer, CL_TRUE, 0, sizeof(cl_float)*LENGTH, data, 0, NULL, NULL);

  // Clean up
  clReleaseMemObject(buffer);
  clReleaseKernel(kernel);
  clReleaseProgram(program);
  clReleaseCommandQueue(queue);
  clReleaseContext(context);

  // Print out the results
  for (int i=0; i<LENGTH; i++)
    printf("sin(%d) = %f\n", i, data[i]);


  free(data);
}
```

UPC vs. MPI+X?

体系结构的巨大变化，是目前并行应用开发面临的主要困难。

其他相关课程：

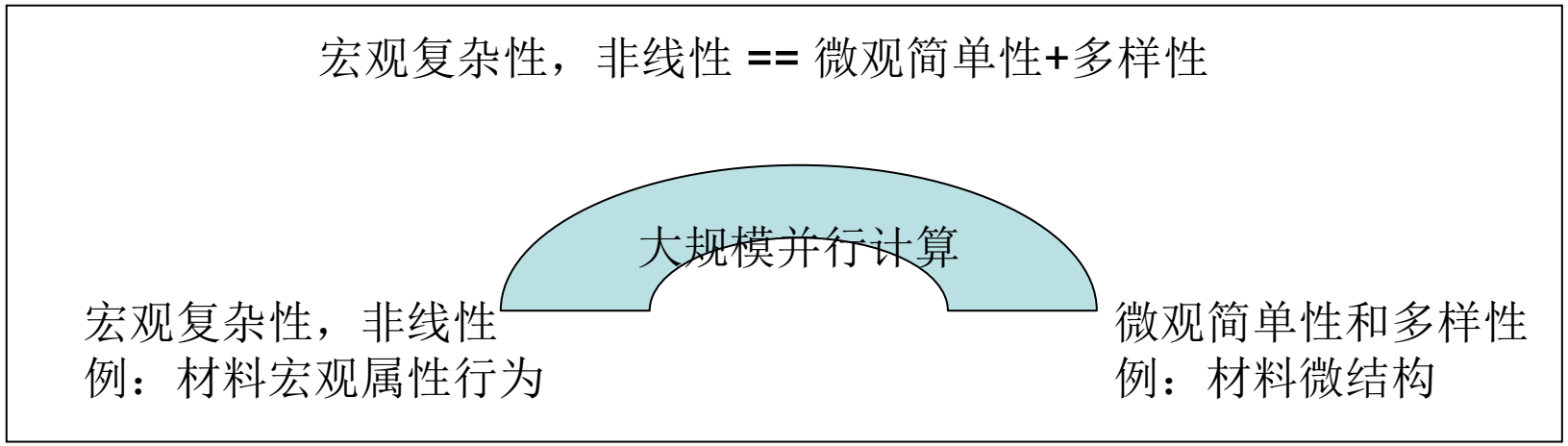
Fermi GPU编程与优化(谭光明)

如何进行大规模并行计算？

- 科学与工程中的大规模计算一般路线：
  - 科学问题的设立(*)
  - 偏微分方程
  - 数值方法选择(*)
  - 离散模型建立(*)
  - 区域分解(*)
  - 并行性能与优化(*)
  - 可扩展性测试(*)
  - 问题求解(*)

并行计算，算法，与优化
A walk through example

- 科学问题的建立
  - 例：材料微结构-宏观属性之间关系的建立
- 算法的设计：哲学思考



- 科学问题的求解策略(例)：
  - 采集微结构数据(nm-um级)
  - 建立大规模数值模型，使得cm(实验试样尺度)-um(微结构尺度)跨尺度分析成为可能，从而将微结构与宏观属性之间建立“直接”联系。

控制方程(偏，常微分方程)

- 描述自然现象的数学形式都可以是偏微分方程
  - 椭圆(平衡解, 静态问题, 与时间无关)
  - 双曲(流体，波动, 与时间相关)
  - 抛物线方程(热传导, 扩散, 与时间相关)

例：

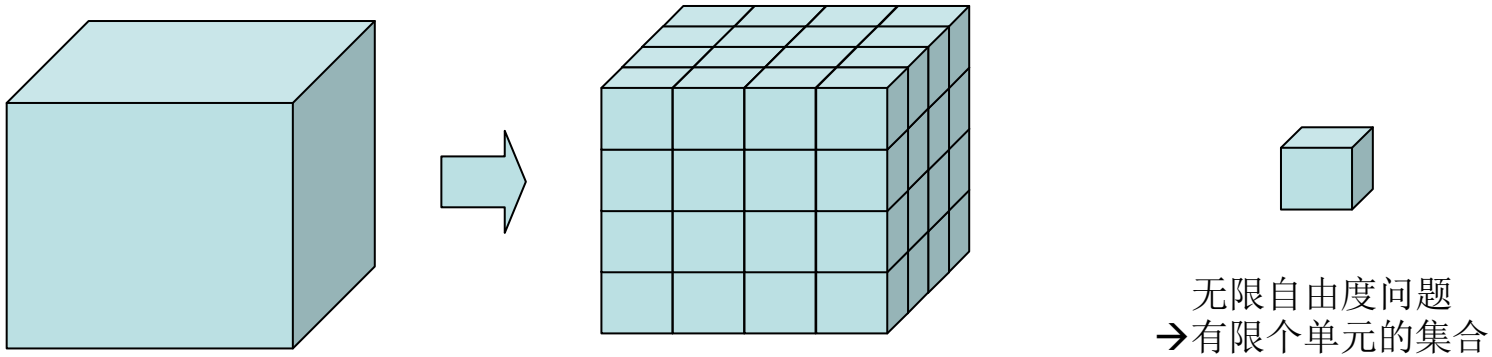
$$\frac{\partial^2}{\partial x^2}u(x)-f=0 \quad \text{on } x\in(x_0,x_1)$$

$$u=u_0 \quad \text{at } x=x_0$$

$$u=u_1 \quad \text{at } x=x_1$$

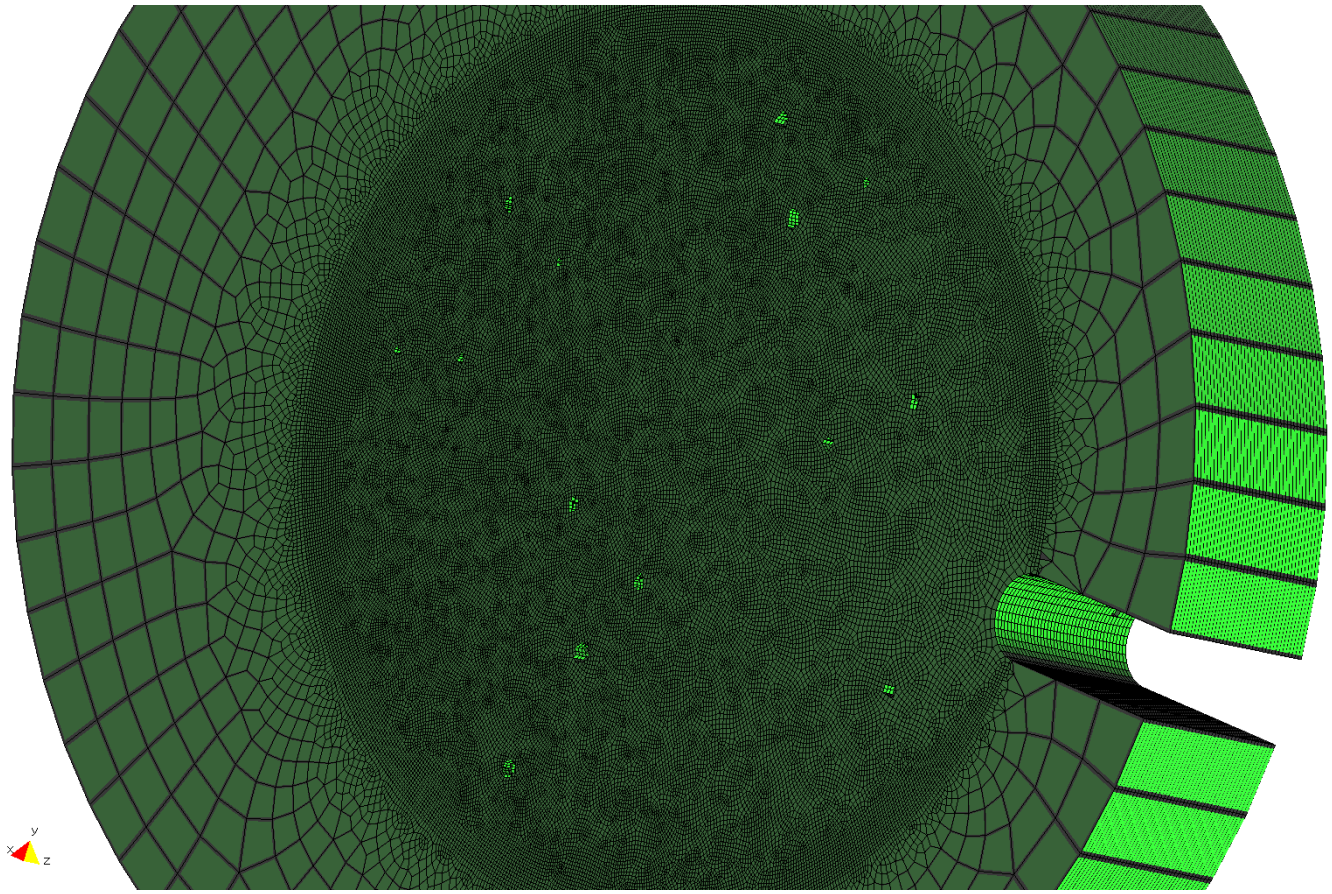
数值方法选择

- 有限元，有限差分，无网格方法，粒子法
  - 将原偏微分方程描述的无限自由度问题转化为有限个自由度问题(积分→离散), 使得在计算机上可解。



$$\int_{\Omega} f(x)\,\mathrm{d}\Omega = \sum_{e}\int_{\Omega_e} f(x)\,\mathrm{d}\Omega = \sum_{e}\left(\sum_{g} w_g f\left(x_g\right)\right)$$

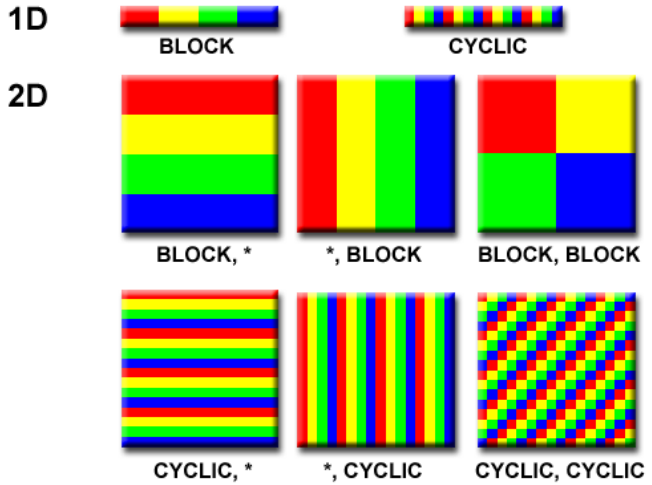
离散模型建立：网格



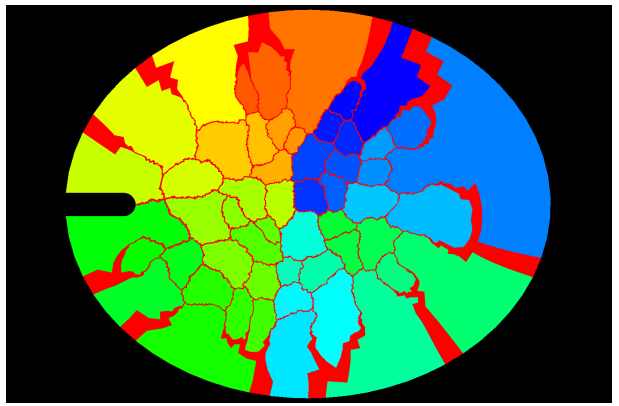
(a) Mesh

区域分解

- 是最常用的并行计算策略
  - 图分解
  - 负载平衡
  - METIS (Serial Graph Partitioning and Fill-reducing Matrix Ordering)为目前事实上的标准图分解算法。
  - 区域分解要使得通讯边界尽量的短



- 例：有限元计算
  - 节点-based分解
  - 单元-based分解
  - METIS对千万单元级网格的图分解仅需几秒



例：对FEM网格的节点-based 分解

并行程序设计

- SPMD vs. MPMD
  - MPMD：气候模拟，多物理场耦合计算，可理解为多个SPMD的耦合

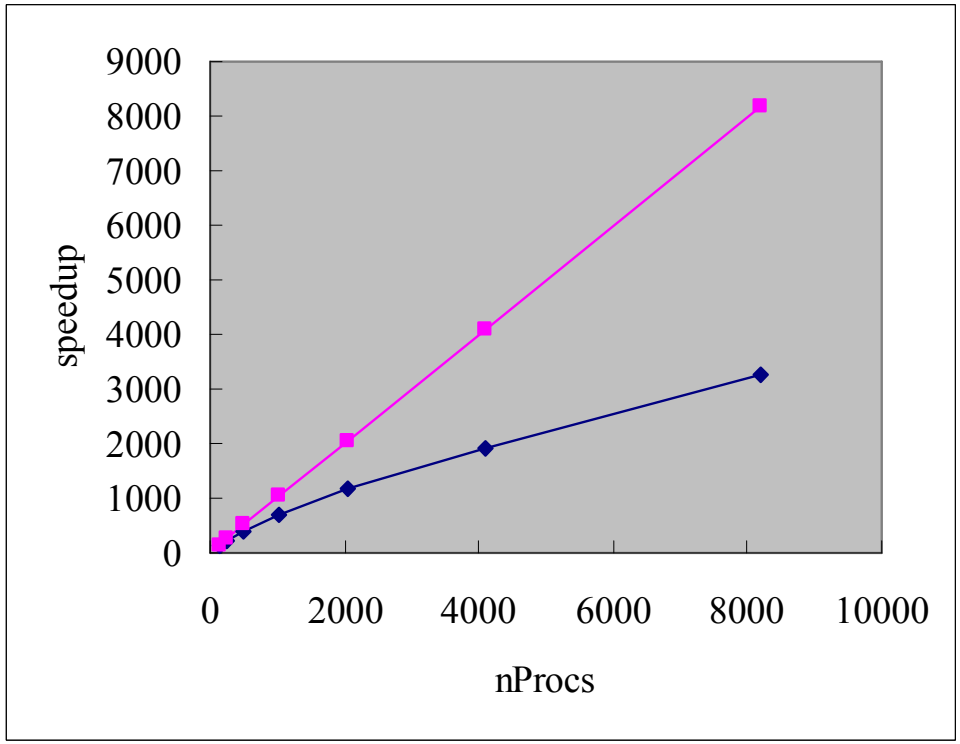
- 静态负载平衡
  - 对于大多数科学与工程应用，一旦“区域分解”方案确定，并行通讯模式也就固定。
  - 例：有限元并行计算中主要要建立一个节点通讯表
    - Export node/PE list + Import node/PE list ⟶
    - 如果网格固定不变，这张通讯表便在整个计算过程中不再改变; 负载平衡由区域分解的图分解算法保证

- 动态负载平衡
  - 对于可归结为求解线性方程组一类的并行计算(绝大多数的科学与工程计算)来说，负载平衡是目前算法设计的难点。例：自适应有限元, 无网格/粒子模拟
  - charm++是目前得到广泛认可的支持负载平衡的一个通用库(UIUC开发)

```
!
! 1.parallel information
1
3
0 4 5
4
0 2 4 5

!
! 2.mesh information (nodes and elements in partitic
! 2.1 node
22 12
14 1.000000e+01 1.000000e+01 4.000000e+01
15 1.000000e+01 0.000000e+00 4.000000e+01
16 1.000000e+01 -1.000000e+01 4.000000e+01
...
60 -1.000000e+01 1.000000e+01 5.000000e+01
61 -1.000000e+01 0.000000e+00 5.000000e+01
7 -1.000000e+01 0.000000e+00 4.000000e+01
8 -1.000000e+01 -1.000000e+01 4.000000e+01
! 2.2 element (connection)
5
11 5 6 9 8 13 14 2 1
12 6 7 10 9 14 15 3 2
13 8 9 16 17 1 2 18 4
31 19 20 21 12 5 6 9 8
32 20 11 22 21 6 7 10 9
!
!  3. import / export information
! 3.1 import
5  2  3
18 17 16 21 22 19 20 13 14 15
!
! 3.2 export
6  1  4  3
2 3 8 9 10 12 4 5 6 7 11 5 6 7
```

并行性能与优化

- 并行性能测试：
  - 建立测试模型，测试，优化
  - 例：在曙光星云上初步测试结果，8192核并行效率为39.7%

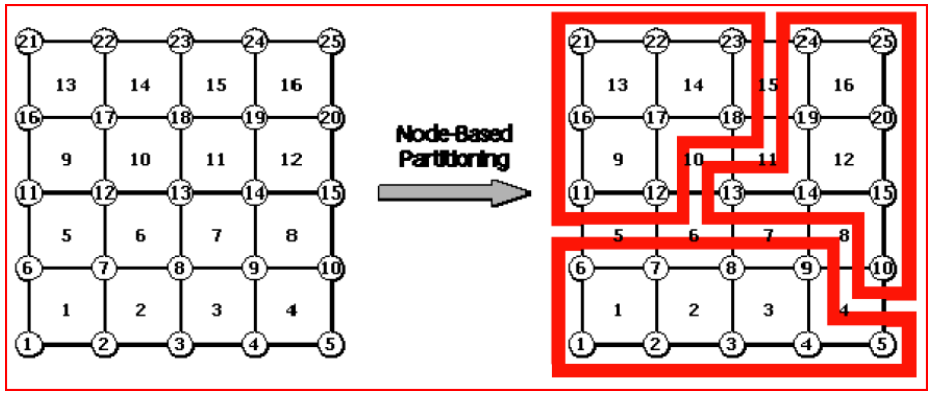


优化策略

- 虽然老生常谈，但具体不一而论：
  - 算法：改进区域分解算法，缩短通讯边界长度
  - 通讯：阻塞通讯改为非阻塞通讯，但是仅此可能还不够...
  - 计算与通讯重叠
  - 关于网络延迟与带宽
    - 随着网络带宽越来越大，频繁小消息发送带来网络延迟开销对性能影响最大。

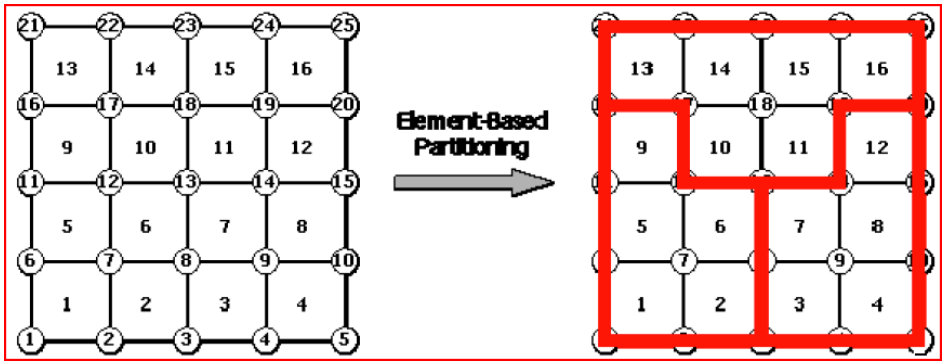
并行优化例1：算法改进

- 算法：新的区域分解，缩短通讯边界长度



节点-based 区域分割方法

目前算法



单元-based 区域分割方法　新算法

并行优化例2：非阻塞通讯的优化

- 通讯：关于MPI非阻塞通讯的使用
  - 在MPI中，根据消息大小有三种通讯协议：short(消息), eager(大消息)和rendezvous(特大消息)。对于小消息(1KB)，MPI是将消息直接拷入系统通讯缓冲区, 然后通过eager协议发出，这样MPI_Isend可以立即返回。而对于大消息(60KB),系统没法缓冲时, MPI不能对消息缓冲, 这时MPI启动rendezvous协议, 需要发收双方进行一次同步(握手). 此时的MPI_Isend在得到接收确认之前是不能立即返回的.
- 结论：对于大消息的非阻塞通讯，任然需要发送与接收进程间的握手过程，不加注意，大消息的非阻塞通讯优势完全不能发挥。
- 解决办法：
  - 办法1：增加缓冲区大小：由于消息累加效应，效果有限
  - 办法2：在MPI_Irecv()和MPI_Isend()后立即发出MPI_Test(), 将”接收确认”提前发出, 然后进行计算。 √
    - 例见下页

并行优化例2：非阻塞通讯的优化

```
for( i=0; i<nProc; i++){
  MPI_Irecv(recv_msg_id);
}

//TO DO: computing

for( i=0; i<nProc; i++){
  MPI_Isend(send_msg_id);
}

//TO DO: computing

MPI_Waitall(recv_msg_id*);
```

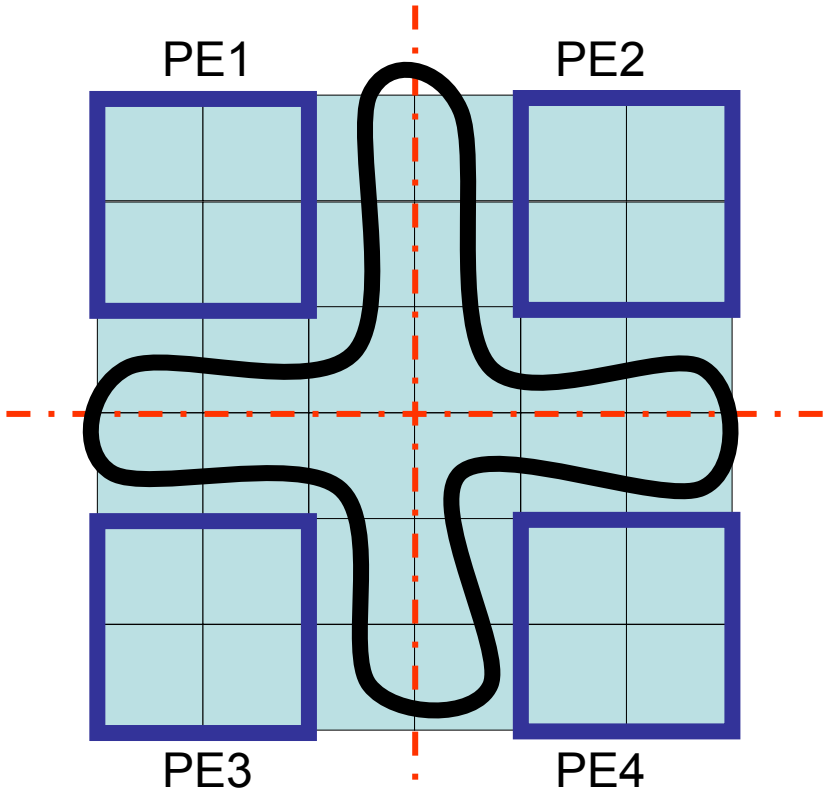
未优化

```
for( i=0; i<nProc; i++){
  MPI_Irecv(recv_msg_id);
}

//TO DO: computing

for( i=0; i<nProc; i++){
  MPI_Isend(send_msg_id);
}

MPI_Test(recv_msg_id*);

//TO DO: computing

MPI_Waitall(recv_msg_id*);
```

优化

并行优化例3：通讯与计算重叠

$$\Omega_i = \Omega_i^{\text{interface}} + \Omega_i^{\text{internal}},$$
$$i = 0,\ldots,\text{nProc-1}$$

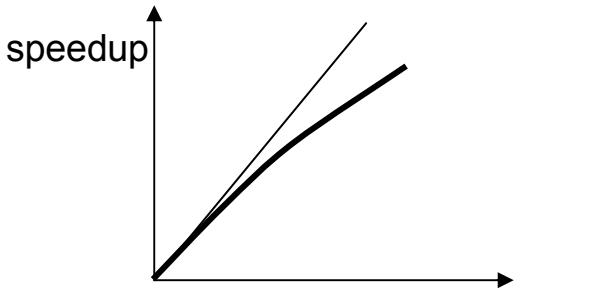



```
//TODO: 首先计算边界区域
Compute on Ω_i^interface
for( i=0; i<nProc; i++){
  MPI_Irecv(recv_msg_id);
  MPI_Isend(send_msg_id);
}
MPI_Test(recv_msg_id*);

//TO DO: 计算内部区域
Compute on Ω_i^internal
MPI_Waitall(recv_msg_id*);
```

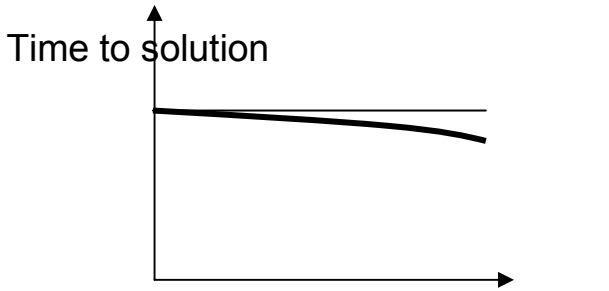
计算与通讯重叠

并行程序可扩展性测试

- **Strong scaling test**
  - 保持网格单元数不变，不断增加处理器个数,计算time-to-solution，换算成加速比。
- **Weak scaling test**
  - 在增加处理器个数时同时增加网格单元数目，使得每个处理器上分到的单元或节点个数不变，计算time-to-solution。
- 两种测试的工程意义：
  - 对于给定规模的一个问题，随着处理器个数的增加，任何一个有限元并行程序总会达到加速饱和，强可扩展测试可以评价评测最短time-to-solution的最优处理器个数
  - 此时，弱可扩展性的意义在于：同样的程序，可以在保持最优加速比的同时，求解更大规模的问题。



Strong scaling test

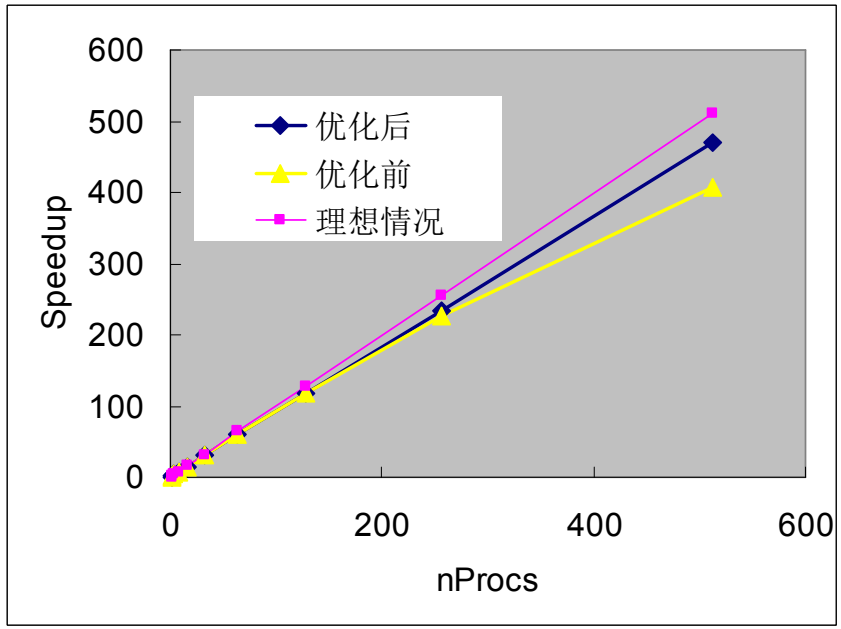


Weak scaling test

并行测试(例)：Strong scaling tests

优化后加速比

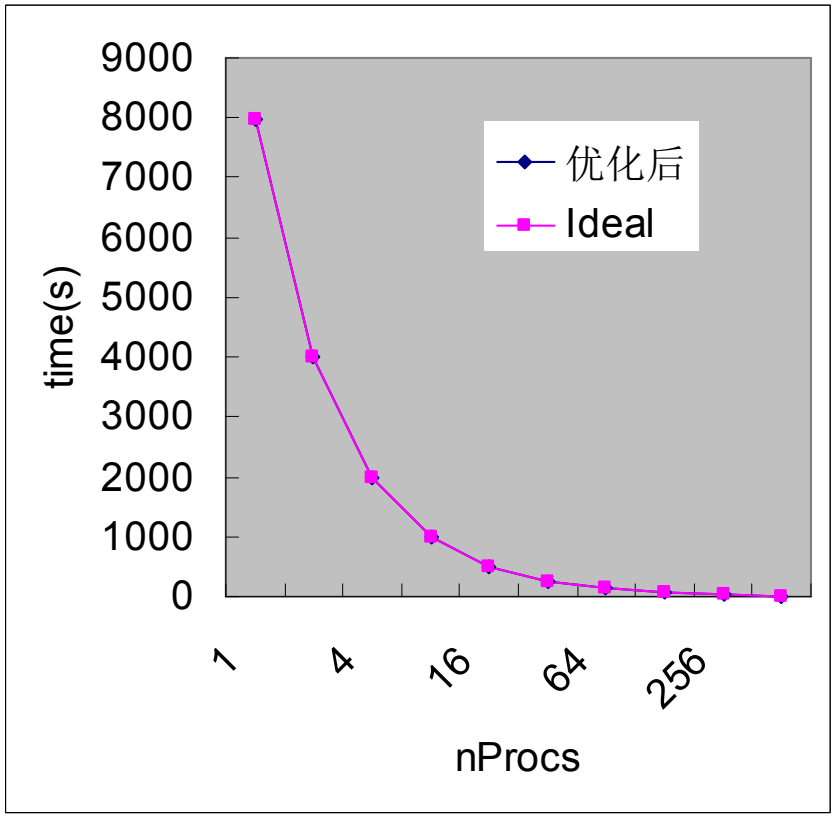
nProcs	Speedup	Ideal
1	1	1
2	1.999	2
4	3.986	4
8	8.016	8
16	15.608	16
32	30.676	32
64	61.353	64
128	117.294	128
256	234.588	256
512	469.176	512



并行测试(例)：Strong scaling tests

优化后time-to-solution

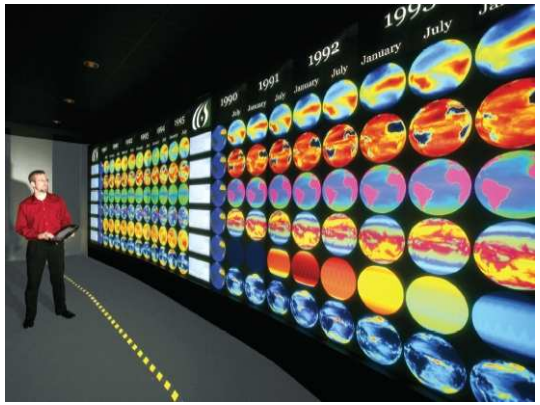
nProcs	Time to solution (s)	Ideal (s)
1	7976	7976
2	3989	3988
4	2001	1994
8	995	997
16	511	498.5
32	260	249.3
64	130	124.6
128	68	62.3
256	34	31.2
512	17	15.6

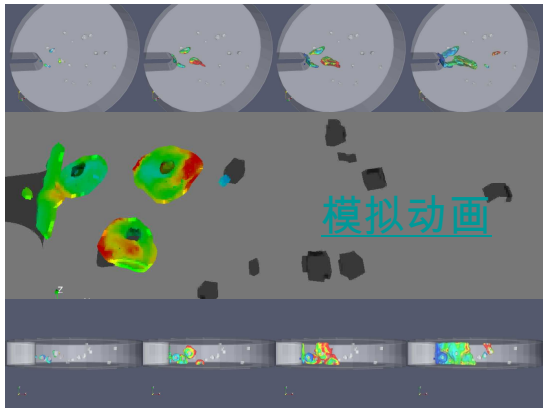


问题求解与其他问题

- 其他相关问题
  - I/O：embarrassing parallel
  - Checkpoint与容错: 在计算规模达到4096核以上，由于任务管理(8小时政策)与机器稳定性两方面的考虑，restart功能几乎是“必须的”！
  - 大规模可视化, 不可或缺, 其重要性不亚于计算本身
    - Paraview开源软件等

http://www.scidacreview.org/0704/html/hardware.html



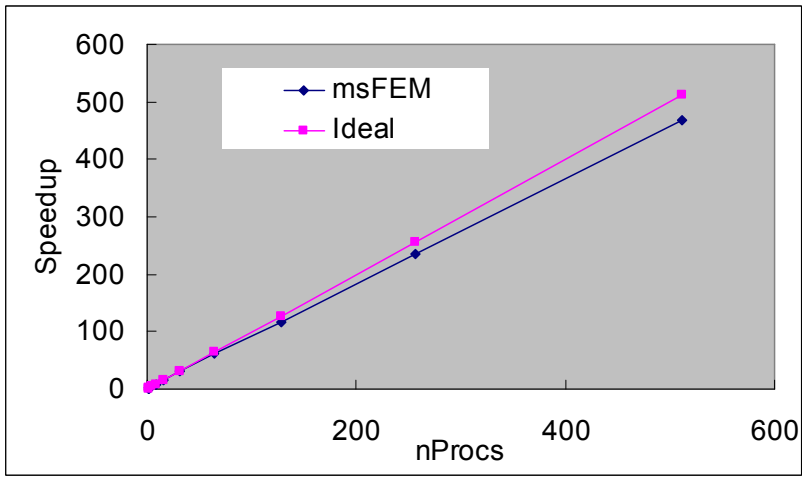


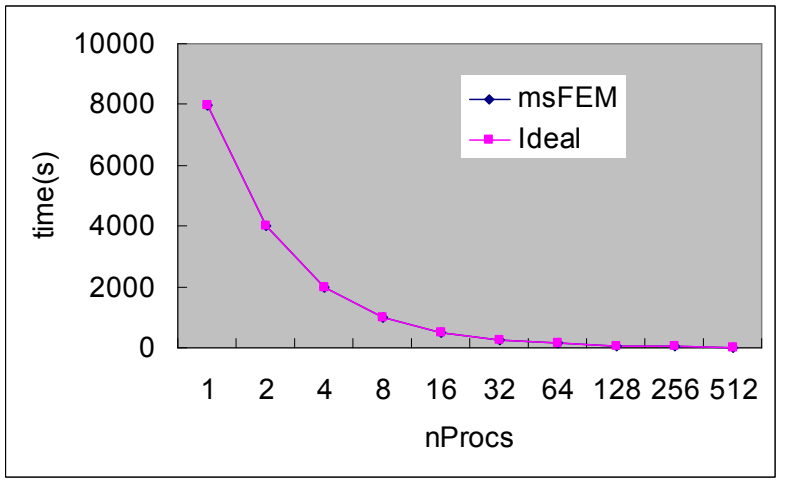
- 例：材料断裂过程的千万亿次级模拟(项目动画)[注]

注：目前该工作得到NCCS美国计算科学中心INCITE (Innovative and Novel Computational Impact on Theories and Experiments)项目的资助，将在Jaguar上进行高分辨率高精度计算(通过初审)。

思考与讨论

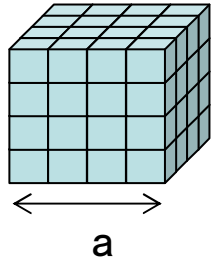
- 程序可扩展是大规模并行计算的基本要求！





- 上述512规模模型测试中：单核上边界单元/内部单元数之比为：103/138。这是通讯/计算之比。(注：这是非常大的一个比例。程序能任然维持好的加速，说明程序具有良好的加速特性)
- 但是，这个结果能说明这个算法的可扩展性一定是非常好的吗？

可扩展性的物理学涵义

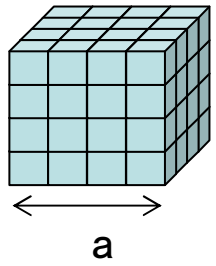


- 可扩展性的物理学涵义：
  - 如图所示,是一个典型的有限元计算区域分解示意图。假设n为各方向上的分割数, a为立方体的边长，那个在这个魔方结构中, 内表面积与体积之比为3(n-1)/a. 这个比值就是有限元计算中通讯与计算之间的比例关系.
  - 随着n(进程数为n^3)的增加, 这个比例越来越大, 算法将变得不再可扩展.

- 物理学涵义: 在越来越小的尺度下, 物体表面积所占体积之比越来越大——这便是物理学中的(纳米)尺度效应. 这一效应导致经典理论不可“扩展”. 在并行计算中，通讯边界随着分割数的增加越来越长，使得通讯占计算的比例越来越大，最终一定会达到加速饱和。

- 可扩展性是算法固有的问题，细枝末节的局部性优化不会到来根本性的改变。

可扩展性的物理学涵义



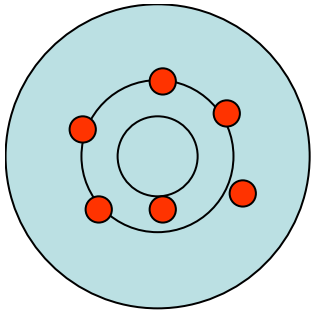
- 可扩展性解决办法
  - 物理学：表面效应修正，新的纳米力学的建立
  - 请大家思考一下：计算科学中可扩展问题的解决途径是什么？
    - 计算机专家：提高单节点计算能力，减少分割数(缩短通讯边界)→片上众核加速
    - 应用专家：算法级优化应优先考虑；其次是局部代码优化；最后是算法的变革
    - 加大计算规模，保持最优加速比(若可扩展性)
    - 计算机与领域应用间的co-design

容易并行的方法就是好方法(大规模并行计算白猫黑猫论)

关于众核加速: 混合精度计算

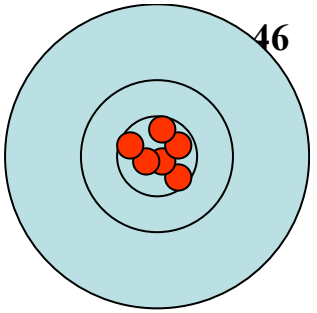
--面向体系结构的新算法

- 混合精度计算是多刃剑:
  - 应用角度: 变革科学与工程计算的思维方式
  - 体系结构: 发挥向量部件, 异构多核, GPU等突出的单精度性能
    - 龙芯3的2 × 256 bit 向量部件: 单精度可以实现16 flops/cycle浮点运算。在单核上混合精度科学计算可实现16倍的加速
  - 更高效能: 通讯带宽要求降低一半, 数据传输和通讯的效率提高一倍, cache利用率提高一倍, 硬件集成度增加2-4倍, 降低能耗…



Precision

对混合精度计算有疑虑？
Precision vs. Accuracy



Accuracy

An Instructive Example

Evaluating f(x,y) with powers as multiplications [S.M. Rump, 1988]

$$f(x,y) = (333.75 - x^2)\,y^6 + x^2(11x^2y^2 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

for $x_0 = 77617, \quad y_0 = 33096$ gives

float s23e8	1.1726
double s52e11	1.17260394005318
long double s63e15	1.1726039400531786 31

This is all wrong, even the sign is wrong!! The correct result is

-0.827396059946821368141165095479816291...

Lesson learnt: Computational Precision ≠ Accuracy of Result

Courtesy of R. Strzodka, D Goddeke, 2008

关于众核加速：混合精度计算

--面向体系结构的新算法

- 科学与工程计算 → Ax = b
- Ax = b的混合精度求解 [1]：

  - 用高精度计算残差方程 d = b – Ax.
  - 用低精度找残差方程的Ac = d近似解.
  - 用高精度更新 x = x + c，然后重复迭代.

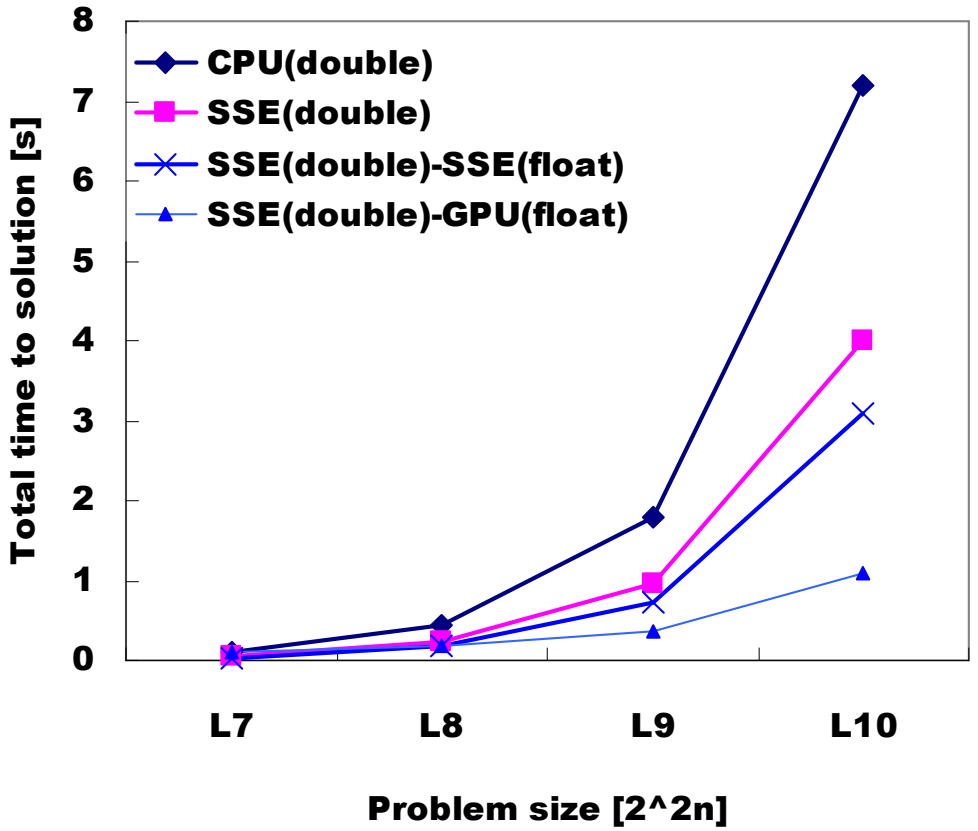
- 少数的高精度计算在CPU上完成，大量低精度的计算在向量部件(SSE等)或加速处理器(GPU以及异构多核(CELL))上完成 [2].

- 预期效果：
  - 90%+的科学与工程计算可望通过低精度完成而不影响解的精确性[1][2]。
  - 单精度计算可以节省一半的存储带宽和通讯带宽, 提高总体计算性能。
  - 降低能耗
    - 例如: 日本单精度千万亿机 MDGrape 单位Gflops能耗仅为0.1w，是通用CPU的1/100 [3]。

[1] J. H. Wilkinson (1963)
[2] D. Göddeke, R. Strzodka, S. Turek (2007)
[3] M. Taiji et al. (2003)

关于众核加速：混合精度计算

--面向体系结构的新算法



GPU混合精度可实现大规模线性方程组求解的有效加速！

[1] Danny van Dyk, Markus Geveler , Sven Mallach, Dirk Ribbrock (2009)

GPGPU10年变迁之预测

(Sept. 1, 2010-Aug. 31, 2020)

- 目前GPGPU如火如荼！但是，GPGPU终将不会成为主流科学计算平台！
  - GPU作为co-processor，注定他在科学计算中发挥的作用是短暂的。
  - 众核CPU是必然的归宿。
  - 编程语言：UPC vs. MPI+X

- 但是，GPGPU将改变10年后的大规模数值计算
  - 可视交互式的的大规模并行计算
    - 及时模拟，及时可视化，图形界面交互

总结

- 为什么需要大规模计算？
  - 内存/时间
- **Grand Challenge**
  - 带来对科学与工程领域的极大冲击的挑战性计算
  - 中国现状：应用软件严重不足
- 何谓大规模并行计算？
  - 具有一定挑战性的并行计算
- 并行计算，算法，与优化
  - 以材料多尺度模拟为例
- 思考与讨论
  - 可扩展性
  - 混合精度计算
  - 众核加速之10年变迁预测
  - UPC之未来

推荐

- https://computing.llnl.gov/tutorials/parallel_comp/ 一个图文并茂的并行计算导读
- http://www.scidacreview.org/ 了解目前世界上最大规模并行计算情况，打开一个绝对会令你激动的计算科学世界

田荣
rongtian@ncic.ac.cn
15910628238