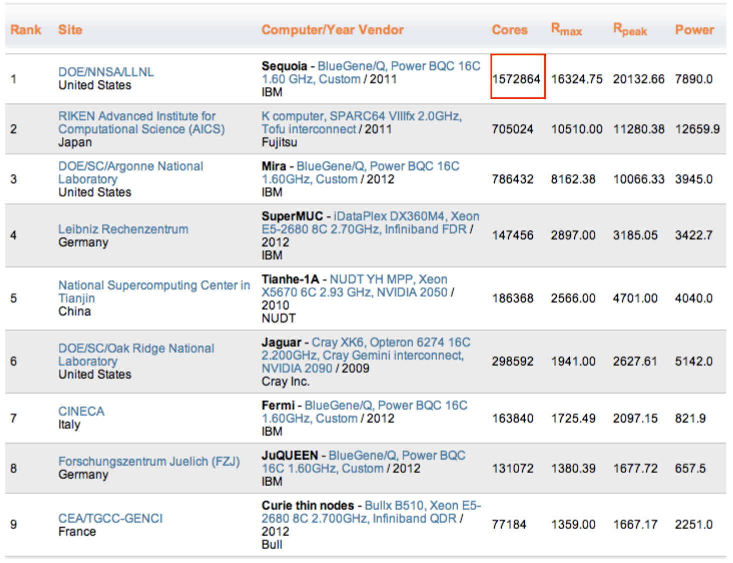
2012 Basic Numerical Analysis

Parallel computing using MPI

Massive parallel systems

- Current main architecture (top500.org)
- 100s to >10000 cores
- Consists of many “PCs”（CPU+ memory + network slot）. Some recent machines have GPUs.

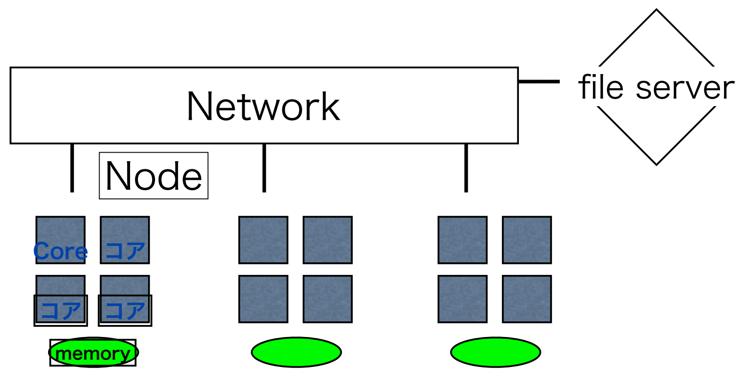
Top500 (June 2012)



Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom / 2011 IBM	1572864	16324.75	20132.66	7890.0
2	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu	705024	10510.00	11280.38	12659.9
3	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM	786432	8162.38	10066.33	3945.0
4	Leibniz Rechenzentrum Germany	**SuperMUC** - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR / 2012 IBM	147456	2897.00	3185.05	3422.7
5	National Supercomputing Center in Tianjin China	**Tianhe-1A** - NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 / 2010 NUDT	186368	2566.00	4701.00	4040.0
6	DOE/SC/Oak Ridge National Laboratory United States	**Jaguar** - Cray XK6, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA 2090 / 2009 Cray Inc.	298592	1941.00	2627.61	5142.0
7	CINECA Italy	**Fermi** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM	163840	1725.49	2097.15	821.9
8	Forschungszentrum Juelich (FZJ) Germany	**JuQUEEN** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom / 2012 IBM	131072	1380.39	1677.72	657.5
9	CEA/TGCC-GENCI France	**Curie thin nodes** - Bullx B510, Xeon E5-2680 8C 2.700GHz, Infiniband QDR / 2012 Bull	77184	1359.00	1667.17	2251.0

ローレンスリバモア研究所のセコイア

Basic structure



Data are distributed onto local memories

A cluster of PCs...

- 100 times faster if 100 nodes connected ?
- There are some *overhead*
- Some (many) problems intrinsically hard to be parallelized
- There are slow parts (bottlenecks) in every program

Parallel programs

- High-end compilers
- e.g., High Performance Fortran

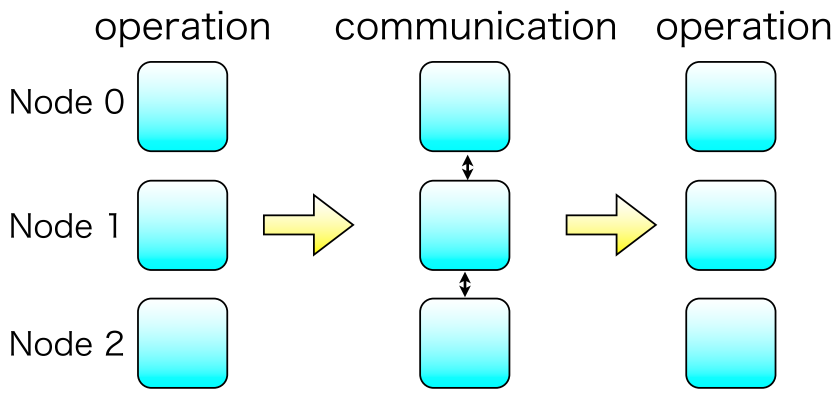
  Data distribution, management, work distribution are done automatically by HPF compilers

  Just a slightly different program (and compiler options) for users

Message Passing

- Data distribution, transfer, work distribution done explicitly by commands (i.e., by a programmer)
- A special programming skill is needed, but the product (programs) can be used on virtually all parallel machines.
- And, it's free!

Message Passing



Basic MPI commands

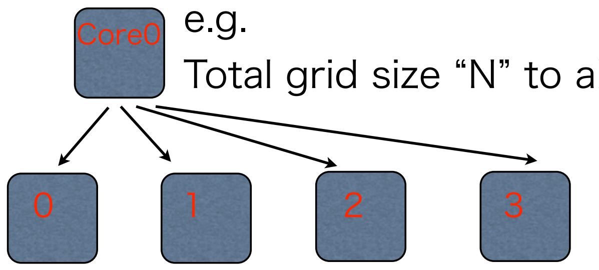
Not always core-to-core communication

MPI_Bcast : one to all

MPI_Allreduce : combine

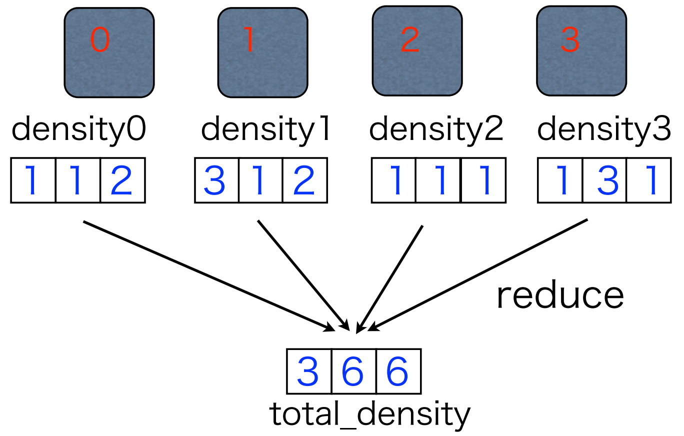
MPI_Allgather : collect and make a data vector

MPI_Bcast

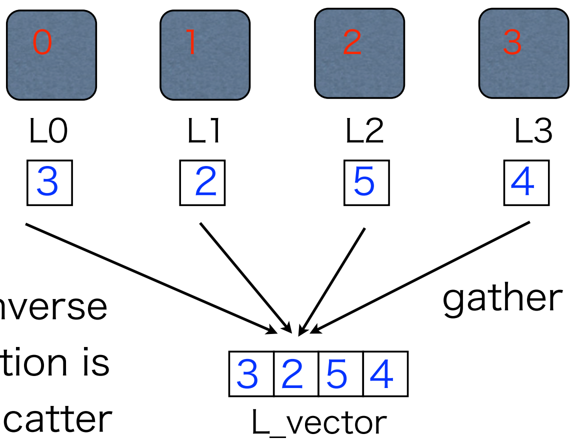


e.g.
Total grid size “N” to all cores

MPI_Allreduce

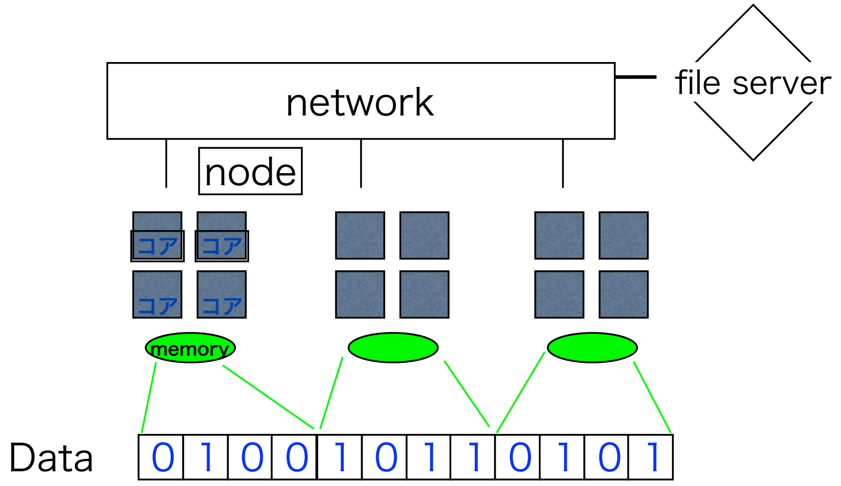


MPI_Allgather

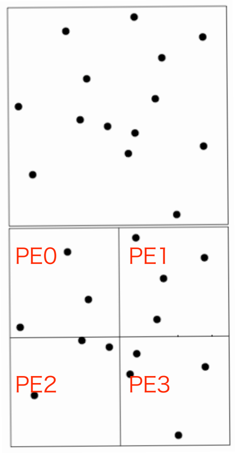


The inverse operation is MPI_Scatter

Local data



Domain decomposition

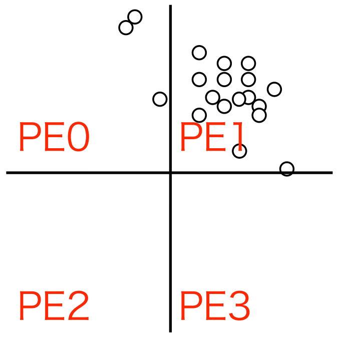


A simplest way is to devide the domain equally onto the number of processes.

For a homogenious distribution or regular grids, work-load is also balanced.

Domain decomposition

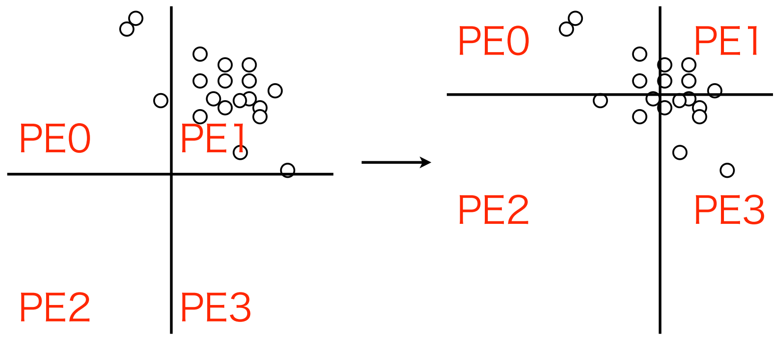
Systems evolve, however....



Think about the situation like in the left occurs when 10000 processes are used.

Domain decomposition itself is an important and hard problem.

Dynamic relocation

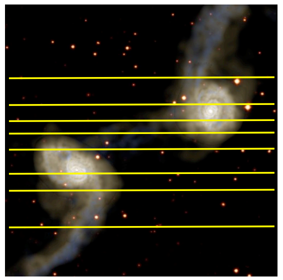
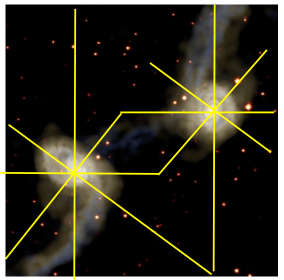


Move the domain boundaries such that the number of elements (and hence the work) is balanced.

Adaptive method

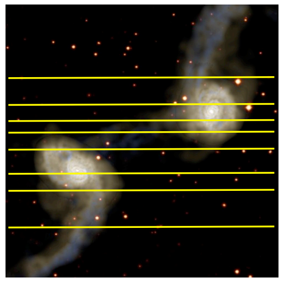
- Does "equal number of elements" mean an equal amount of work ?
- Every program has "slow" parts.
- It is important to balance the work done in the slowest part (say, gravity calculation).

Actual problems

Which is better ? There's no universal answer. On the other hand, a simpler algorithm works in many (if not all) cases.

Associated comm.



So far we have discussed the balance of local tasks.
Sometimes (often),
data exchanges accompany with dynamic load-balance.
Inter-node communication takes time... How often should a program check the balance ?

Note the above figures is in 2D.
In 3D, contact surfaces with other domains increase.

Problem dependence

- For systems with long-interaction, essentially all nodes need to know the status of active elements
- With regulars grids, things might appear easy. However, if there are many boundaries in 3D (imagine j, j+1, k, k+1 etc), significant data transfer is needed.

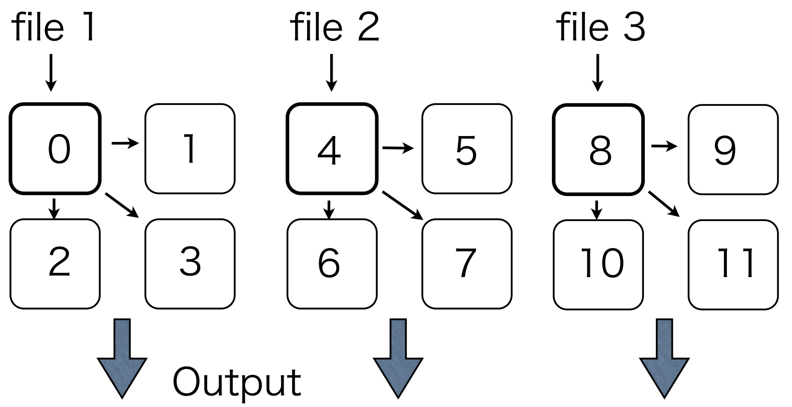
Incomplete parallelization

- Amdahl's law
  The efficiency is at most $\dfrac{1}{F + (1-F)/N}$
- Inter-node communication speed is much dependent of the architecture
- For each process, waiting time is really a waste of time.

Overall size

- Number of cores needed to be set appropriately, perhaps in proportion to the problem size.
  e.g.) You wouldn't use 100 cores for N=100 grids.
- Parallel computing intrinsically better suited for statistical studies. Evolution of a single object is hard to be followed with 1000 cores.

Process grouping



together, or piece by piece

Example MPI

```
No_PE_per_Group = NCore/FILE_PER_OUTPUT;

BossCore = (ThisCore/No_PR_per_Group)*No_PR_per_Group;

//readin the multiple output files
if(ThisCore == BossCore){
  i=read_output_multi(pathname, output_number, ThisCore);
  send_particle_data();
}else{
    receive_particle_data();
}


for(i=0; i< N_Local_Particle; i++){
```

local operation is here

BossCore

Core 0 does all the administration

```
 if(ThisCore == 0){

   //readin the header information
   i=read_header_information(pathname, Header);

   N_allocate=Header.Npart_all/NCore;
 }

 MPI_Bcast(&N_allocate,  1, MPI_INT, 0, MPI_COMM_WORLD);
 MPI_Bcast(&Header,  sizeof(struct io_head), MPI_BYTE, 0, MPI_COMM_WORLD);


if(ThisCore == BossCore){

 for(icomm=0; icomm<No_PE_per_group; icomm++){
     receiving_core = BossCore + icomm;

     MPI_Ssend(&N_send,  1, MPI_INT, receiving_core, tag_ngas,  MPI_COMM_WORLD);

     MPI_Ssend(&Part_read[ipointer].x, N_send*sizeof(struct particle), MPI_BYTE,
               receiving_core, tag_part, MPI_COMM_WORLD);

     ipointer += N_send;

 }
```

Child Core

```
}else  //other processors receive the particle info
  {

    for(icomm=0; icomm<No_PE_per_group; icomm++){
      receiving_core = BossCore + icomm;

        if(ThisCore == receiving_core){

          MPI_Recv(&N_part,  1, MPI_INT, BossCore, tag_ngas,  MPI_COMM_WORLD, &status);

          MPI_Recv(&Part[0].x, N_part*sizeof(struct particle), MPI_BYTE,
                   BossCore, tag_part, MPI_COMM_WORLD, &status);

        }
    }

  }

MPI_Barrier(MPI_COMM_WORLD);
```

message from BossCore

Wait for other processors