A Case Study for Performance Optimization of Parallel Applications

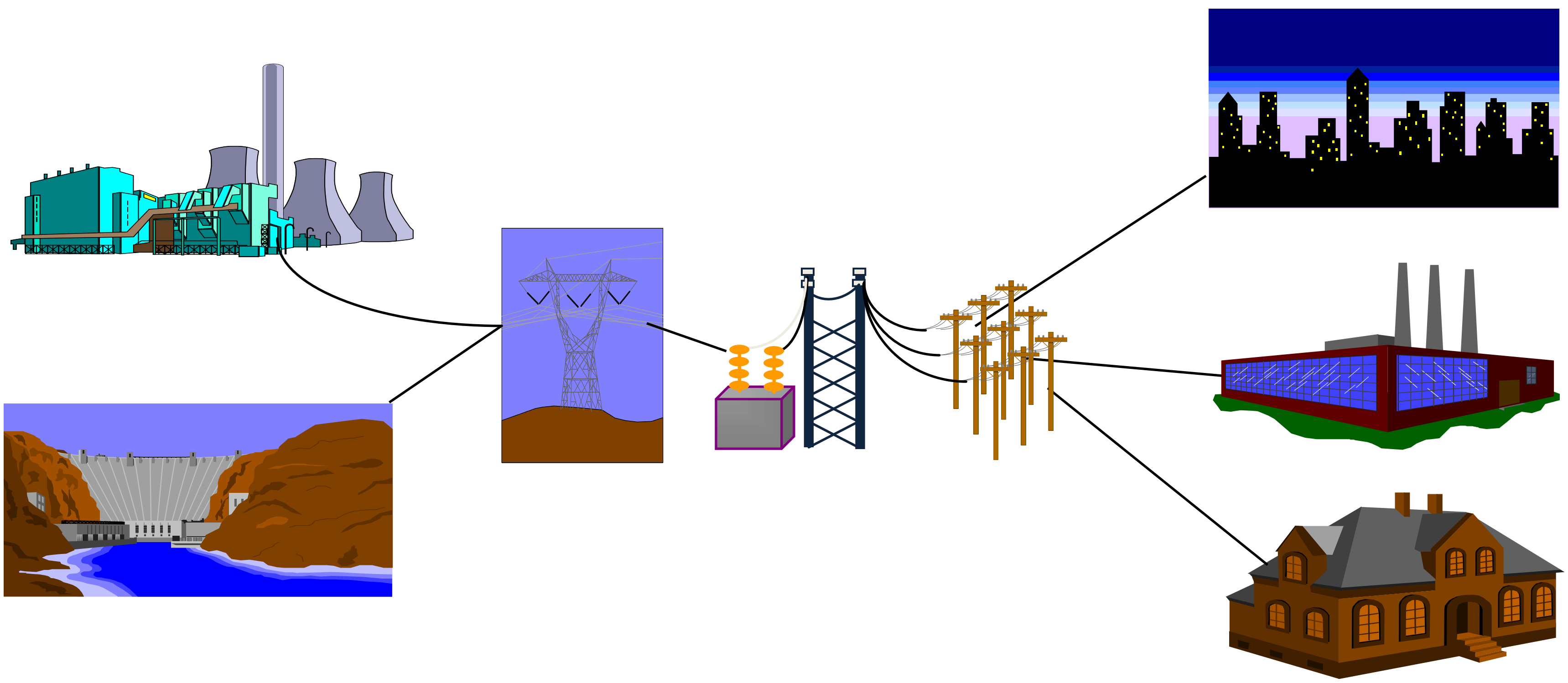
Wei Xue
Department of Computer Science and Technology
Tsinghua University
2013-4-10

Outline

- **A case study**
  - **Application introduction**
    - **Parallel Real-time Simulator** for large scale power grid
  - How to parallelism and tuning?
    - Parallel algorithm design and optimization
    - Code Tuning based on IPF
- Some words at the end

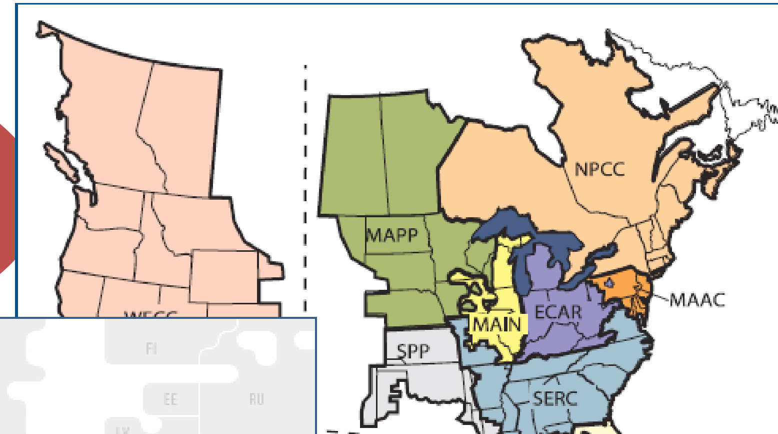
Background – What is it?

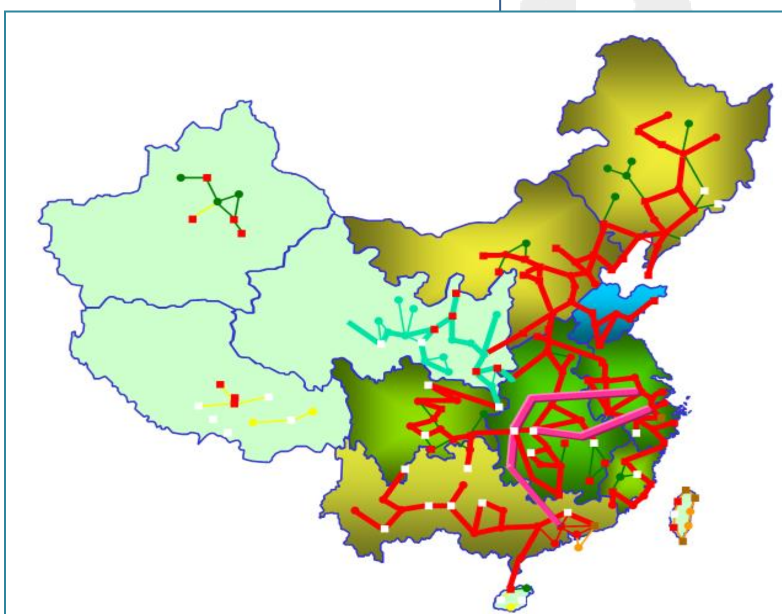
- Power system is the most important infrastructure of our society



It is a large system…

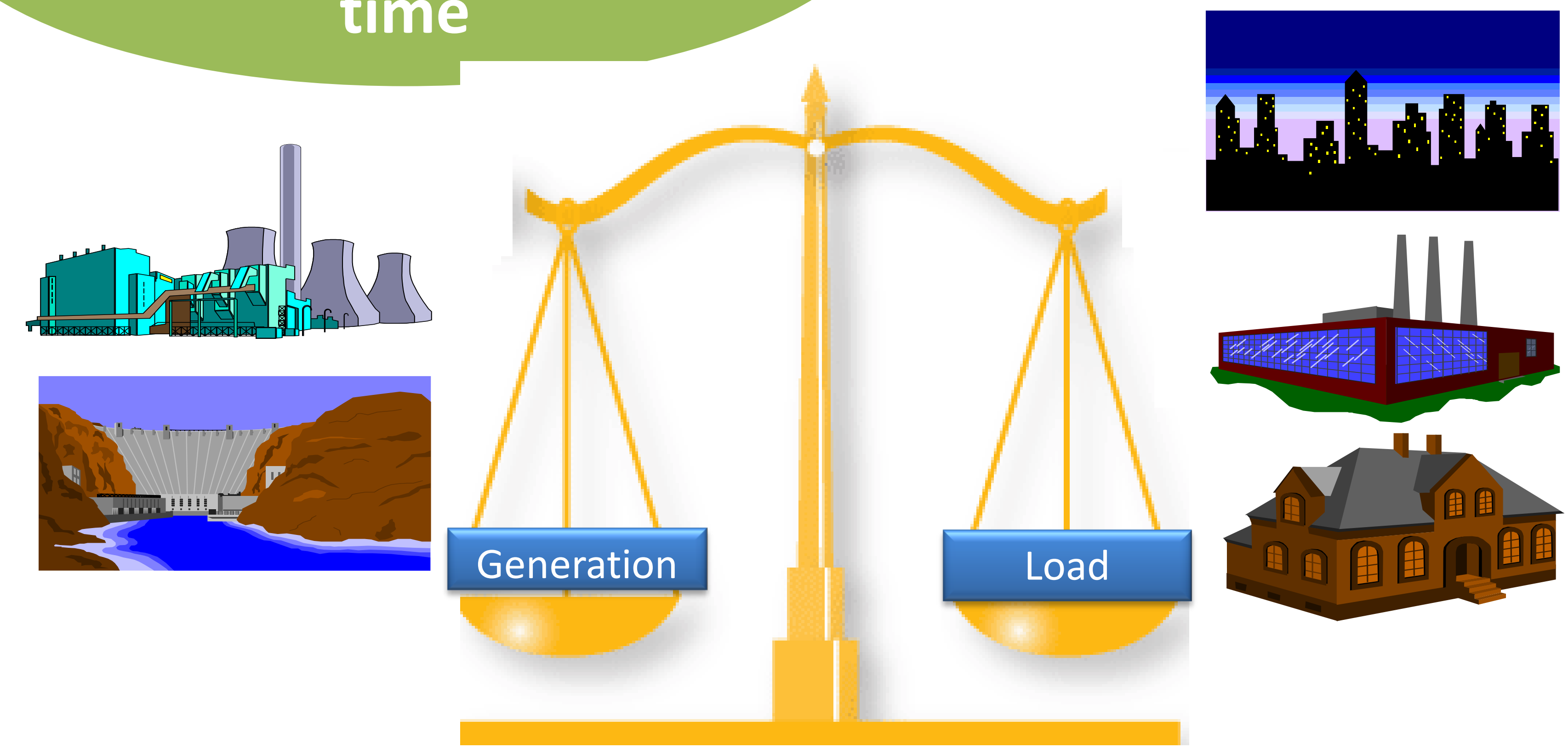
One of the largest man-made systems and becoming larger





It is a dynamic system…

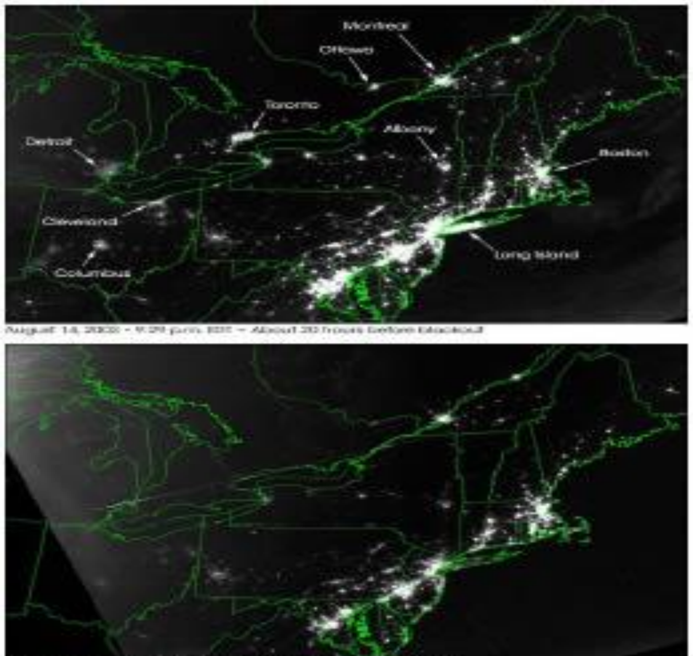
NOT stored: consume just after generation, needs balance time to time



It is a vulnerable system sometimes…

And the grid HAS transmission margin and violation is vulnerable to collapse

Blackout, US & CA
August 14, 2003



Shut off power for 50 million people in the Northeast and in Canada and caused financial losses of over $6 billion

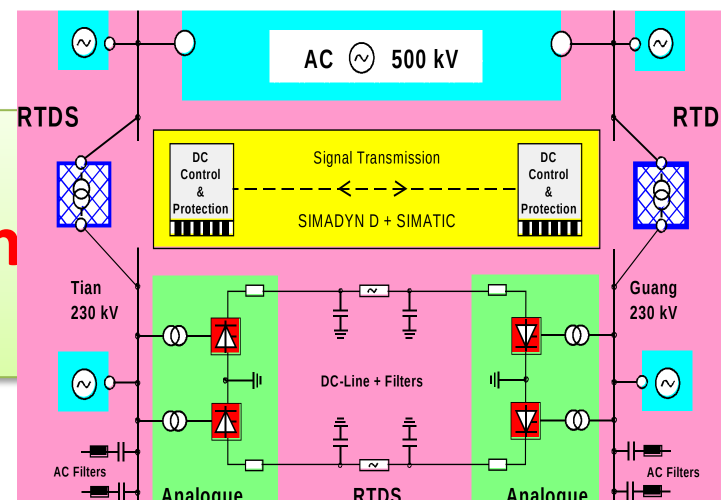
Huge Power System, Great Challenge

HAS operation marg

NOT store any power anywhere

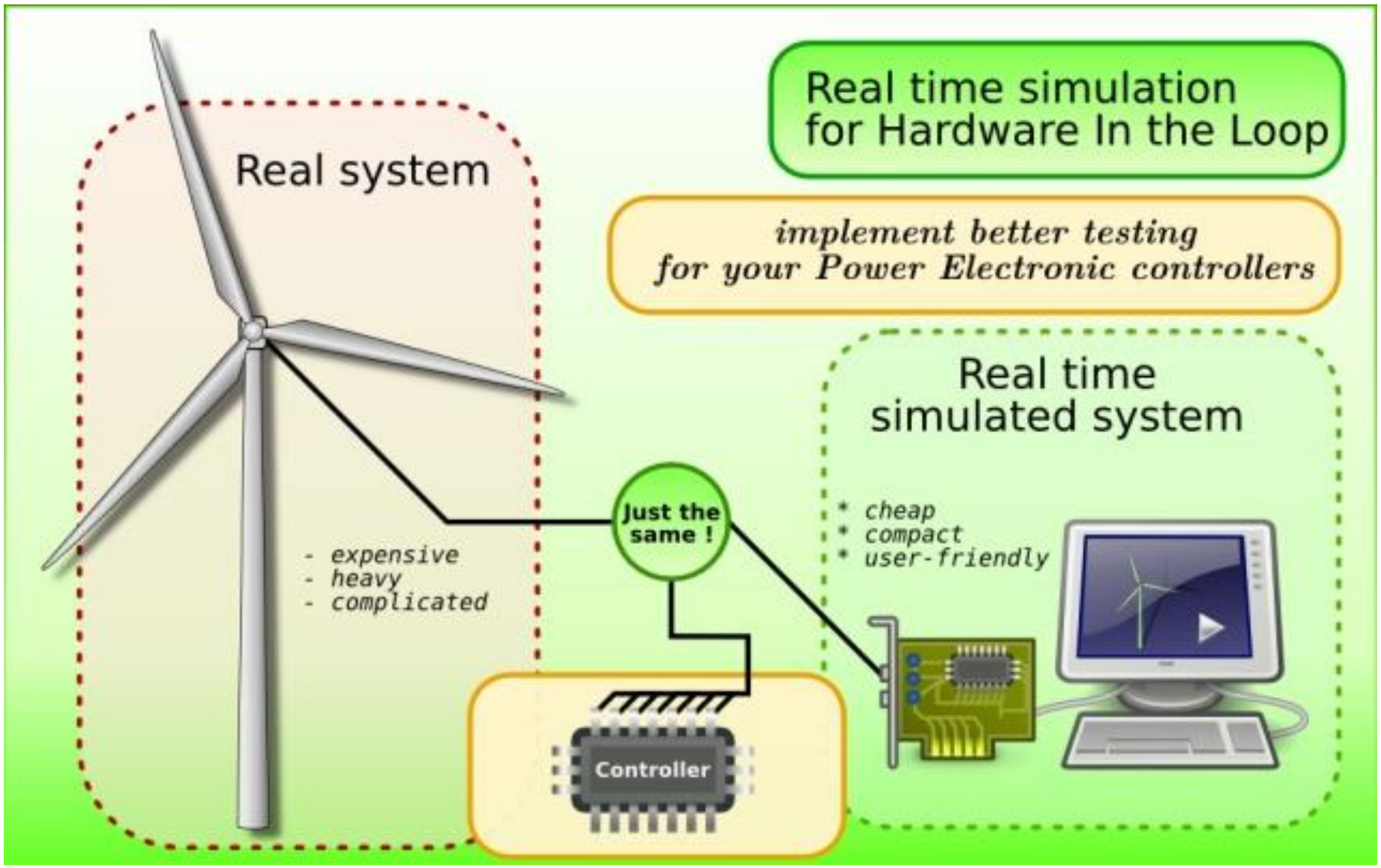
The largest man-made system

ontrol such



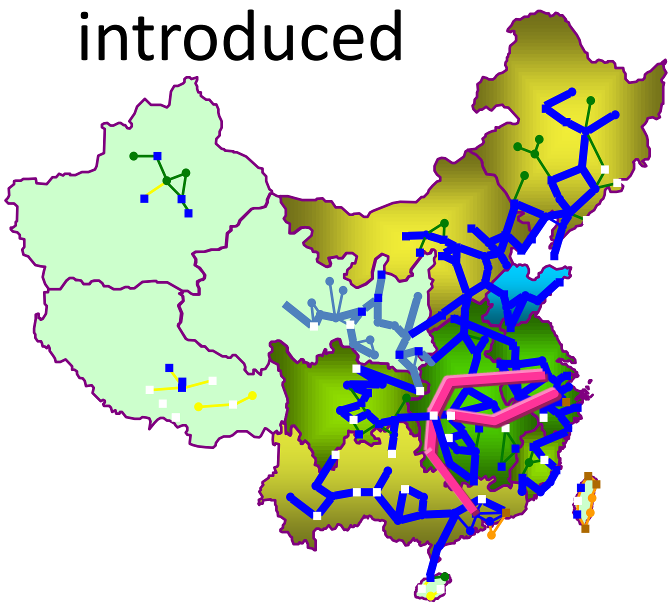
Brief Background

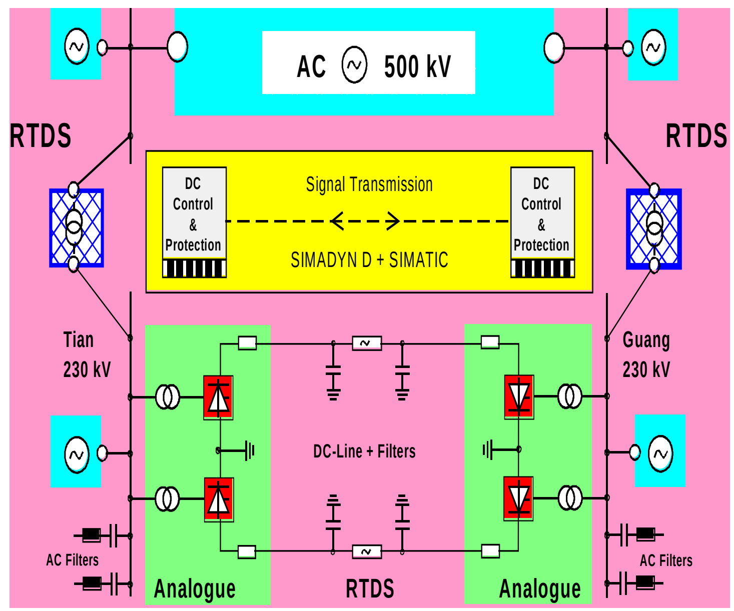
- Real-time simulators widely use in power system for device testing

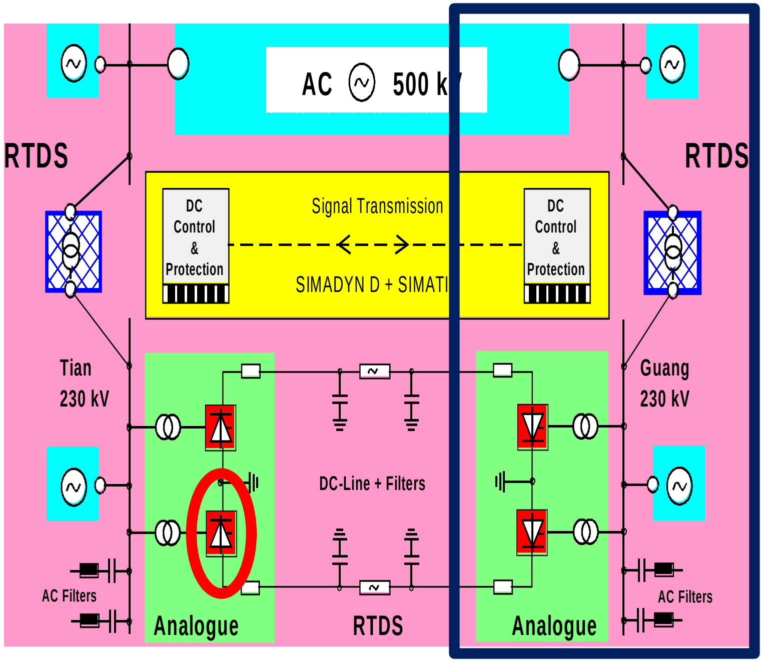


Brief Background

- Future power grid becomes more and more complex and bring more challenges to RT Sim.
  - Very large scale power grid emergence
  - More and more high volume power device introduced





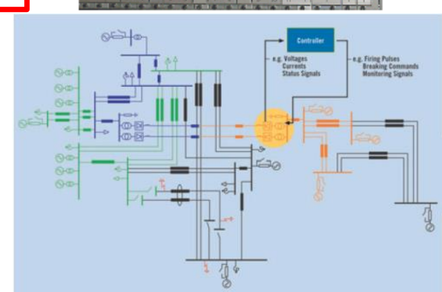

AC 500 kV
RTDS
RTDS
DC Control & Protection
Signal Transmission
SIMADYN D + SIMATI
DC Control & Protection
Tian 230 kV
Guang 230 kV
DC-Line + Filters
AC Filters
AC Filters
Analogue
RTDS
Analogue

Our solution

Transient simulation for external network

millisecond

Electromagnetic simulation for device

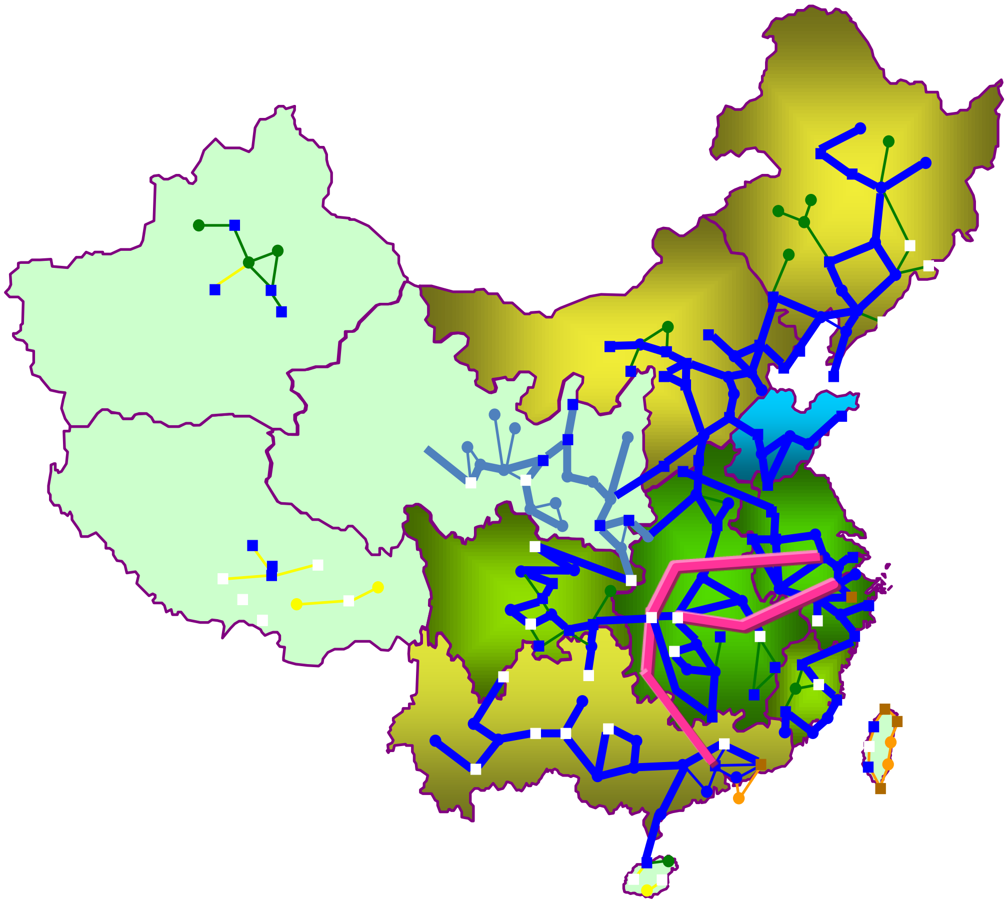


microsecond

Device controller system

Our Target

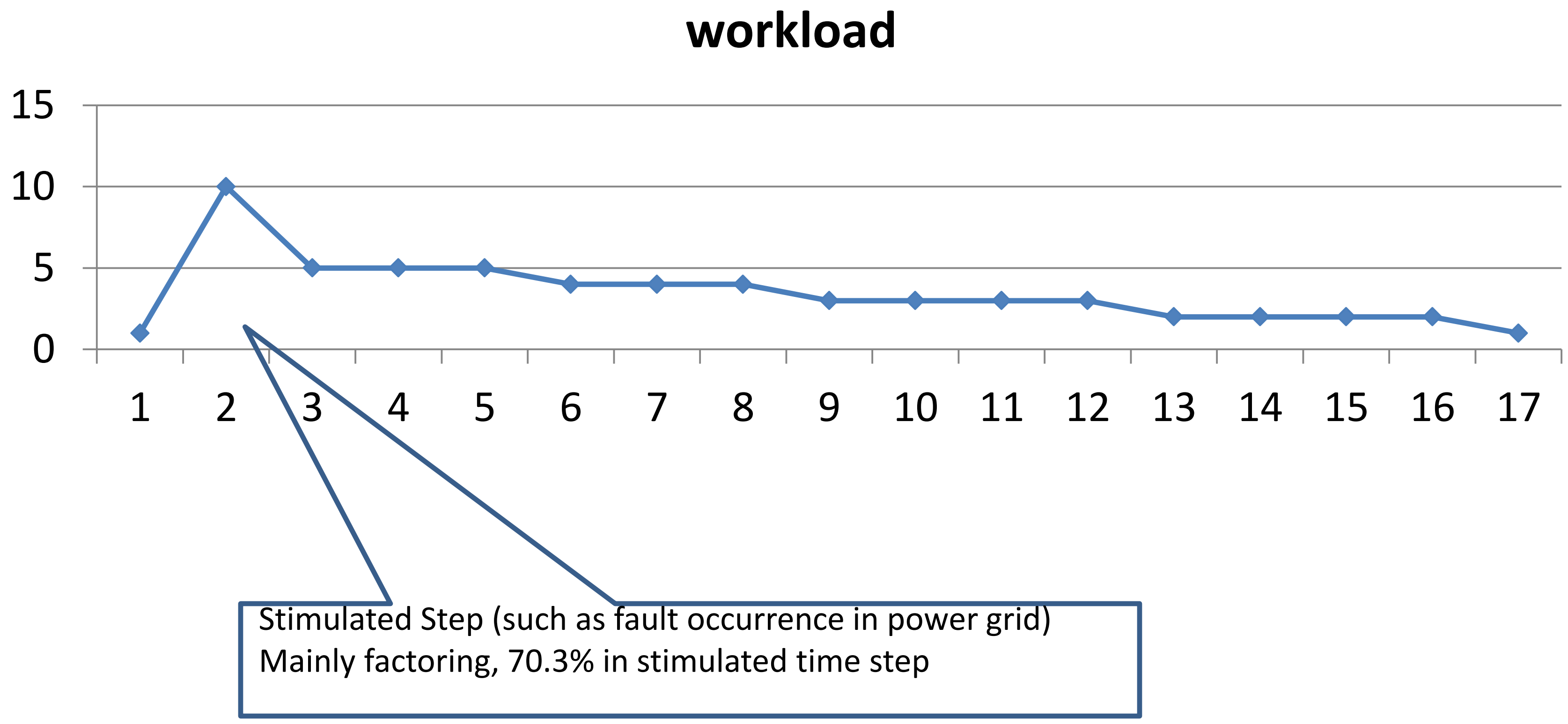
National Power System of China 2005



10188 nodes
1072 generators
3003 loads
13499 transmission lines
4 HVDCs

One case, serious fault,
10s dynamic process;
0.01s time step;
On Itanium2 1G CPU;
serial algorithm needs 25s totally
Stimulated step(0.01s) need **0.6s**

Workload analysis



- Where is the problem?
  - It is a computation intensive problem

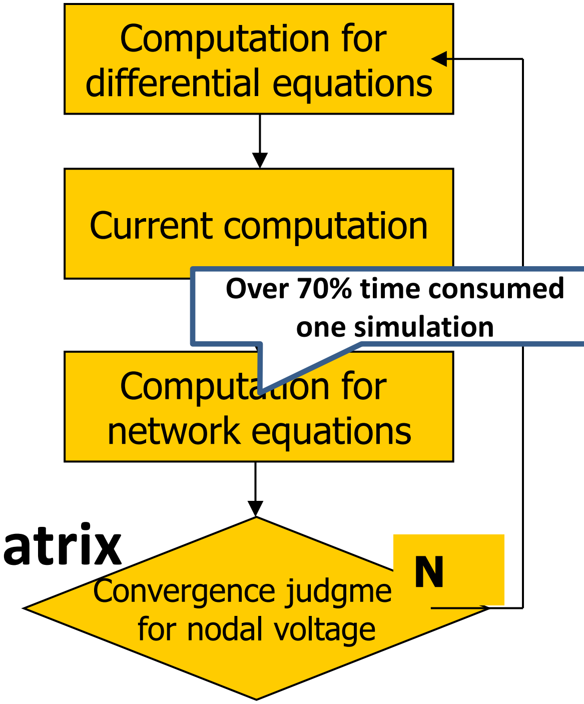
Stimulated step(0.01s) need 0.6s

- Our target: each time step < 0.01s

Mathematical Model

- **Computational Model**
  - Differential Algebra Equations

$$\dot{X} = f(X,V) = AX + Bu(X,V)$$
$$0 = I(X,V) - Y * V$$



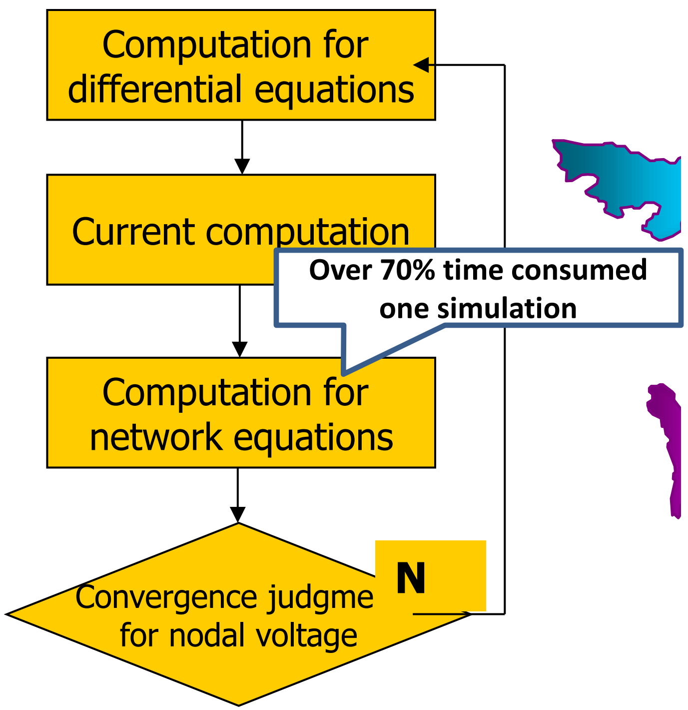
- Sparse linear equations
- **Y is unstructured and complex sparse matrix**

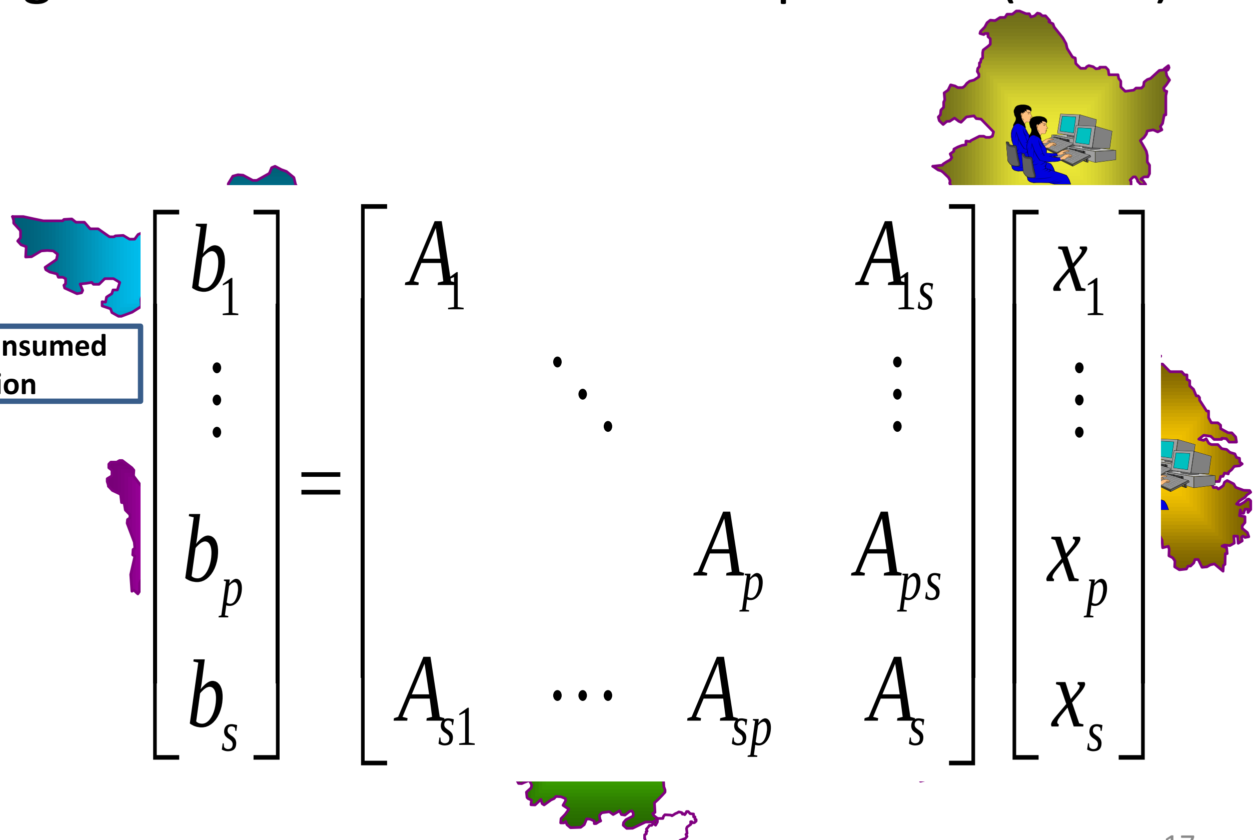
Outline

- Application introduction
- How to parallelism and tuning?
  - **Parallel algorithm design and optimization**
  - Code Tuning based on IPF
- Some words at the end

Design of Parallel Algorithm

- **Parallelism based on Domain Decomposition**
  - Differential equations are easy to parallel
  - Spatial parallel algorithm for linear network equations (AX=b)





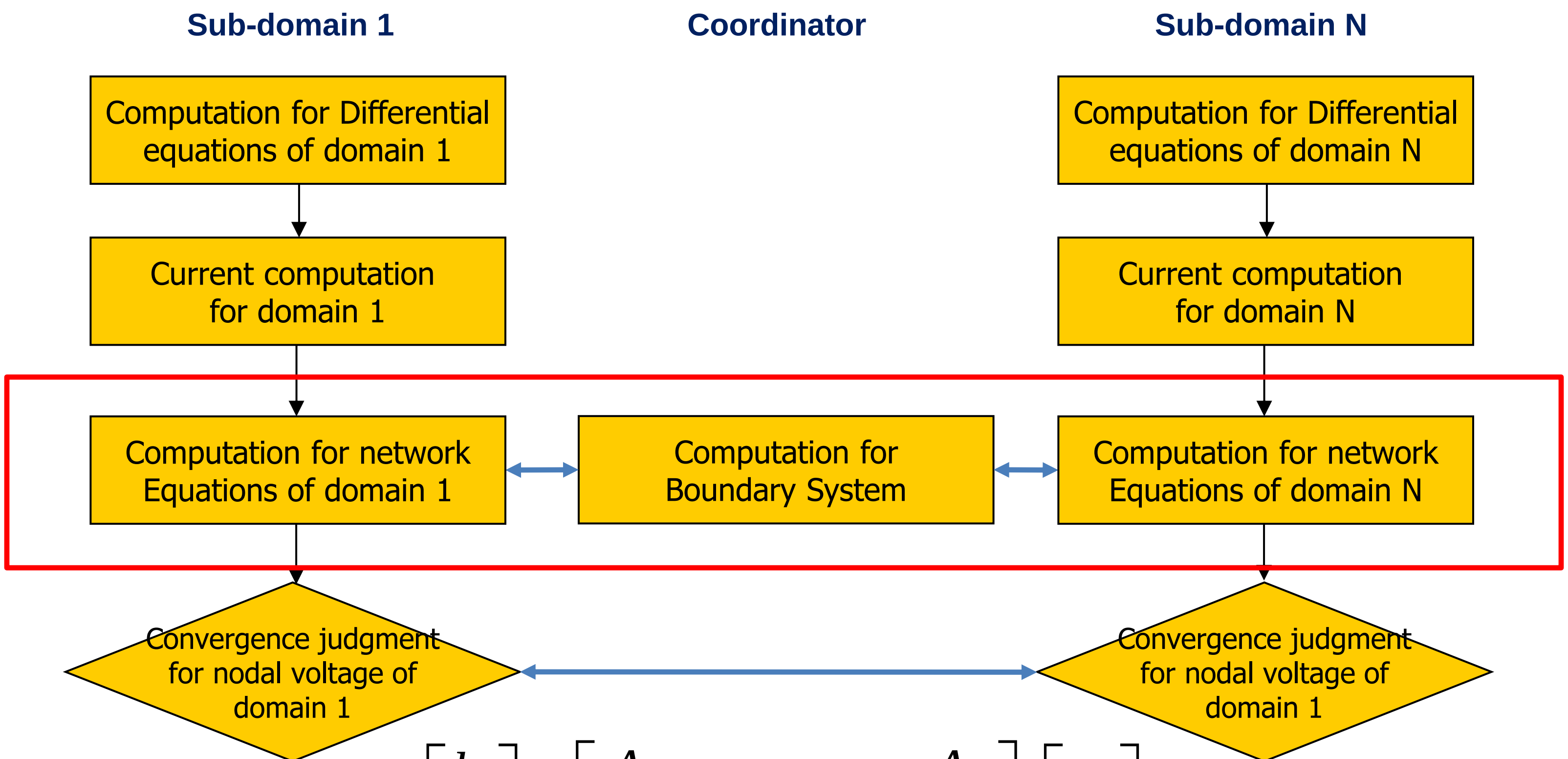
$$\begin{bmatrix} b_1 \\ \vdots \\ b_p \\ b_s \end{bmatrix} = \begin{bmatrix} A_1 & & & A_{1s} \\ & \ddots & & \vdots \\ & & A_p & A_{ps} \\ A_{s1} & \cdots & A_{sp} & A_s \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_p \\ x_s \end{bmatrix}$$

Parallel Implementation of Ax=b

$$\begin{bmatrix} b_1 \\ \vdots \\ b_p \\ b_s \end{bmatrix} = \begin{bmatrix} A_1 & & & A_{1s} \\ & \ddots & & \vdots \\ & & A_p & A_{ps} \\ A_{s1} & \cdots & A_{sp} & A_s \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_p \\ x_s \end{bmatrix}$$

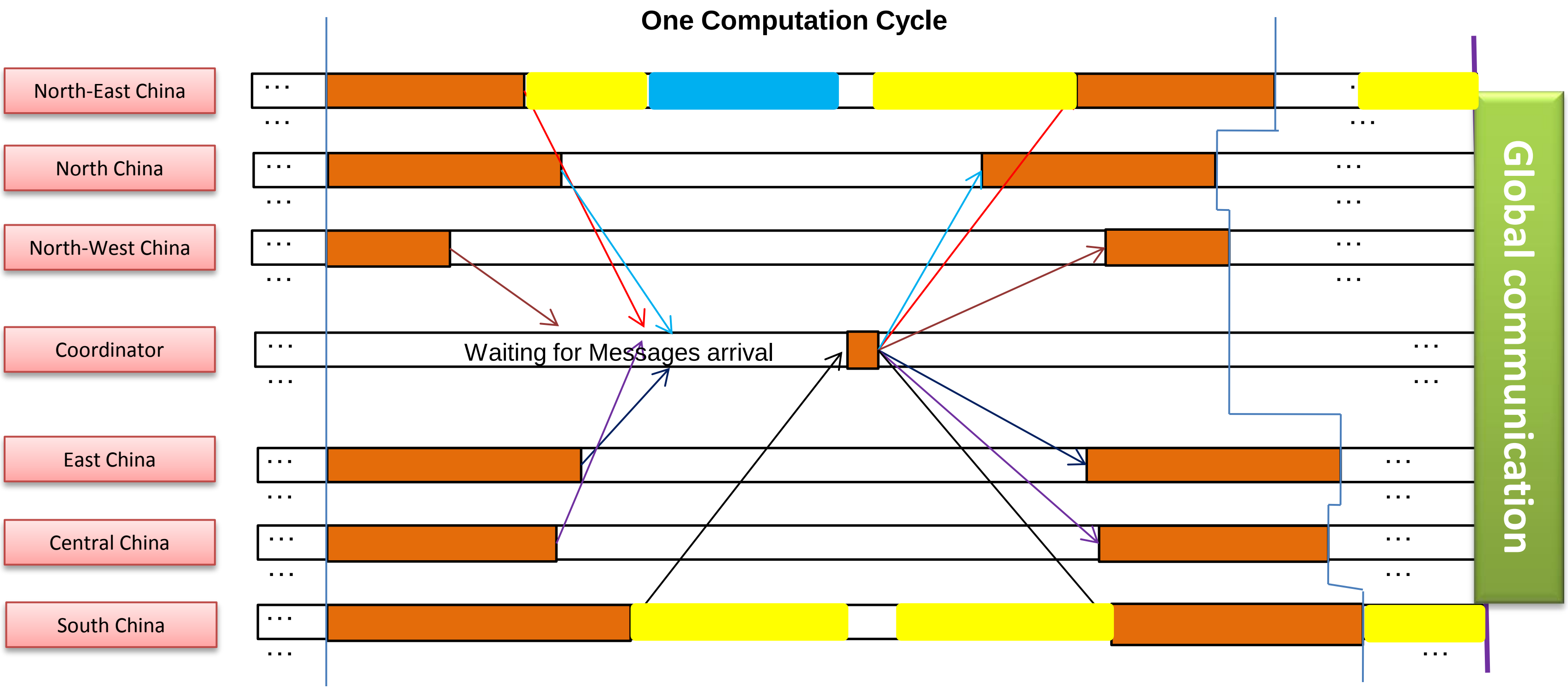
$$\begin{bmatrix} b_1 \\ \vdots \\ b_p \\ b_s - A_{s1}A_1^{-1}b_1 \cdots - A_{sp}A_p^{-1}b_p \end{bmatrix} = \begin{bmatrix} A_1 & & & A_{1s} \\ & \ddots & & \vdots \\ & & A_p & A_{ps} \\ 0 & \cdots & 0 & A_s - A_{s1}A_1^{-1}A_{1s} \cdots - A_{sp}A_p^{-1}A_{ps} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_p \\ x_s \end{bmatrix}$$

Problems of Traditional Parallel Algorithm



$$\begin{bmatrix} b_1 \\ \vdots \\ b_p \\ b_s \end{bmatrix} = \begin{bmatrix} A_1 & & & A_{1s} \\ & \ddots & & \vdots \\ & & A_p & A_{ps} \\ A_{s1} & \cdots & A_{sp} & A_s \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_p \\ x_s \end{bmatrix}$$

Baseline Parallel Algorithm

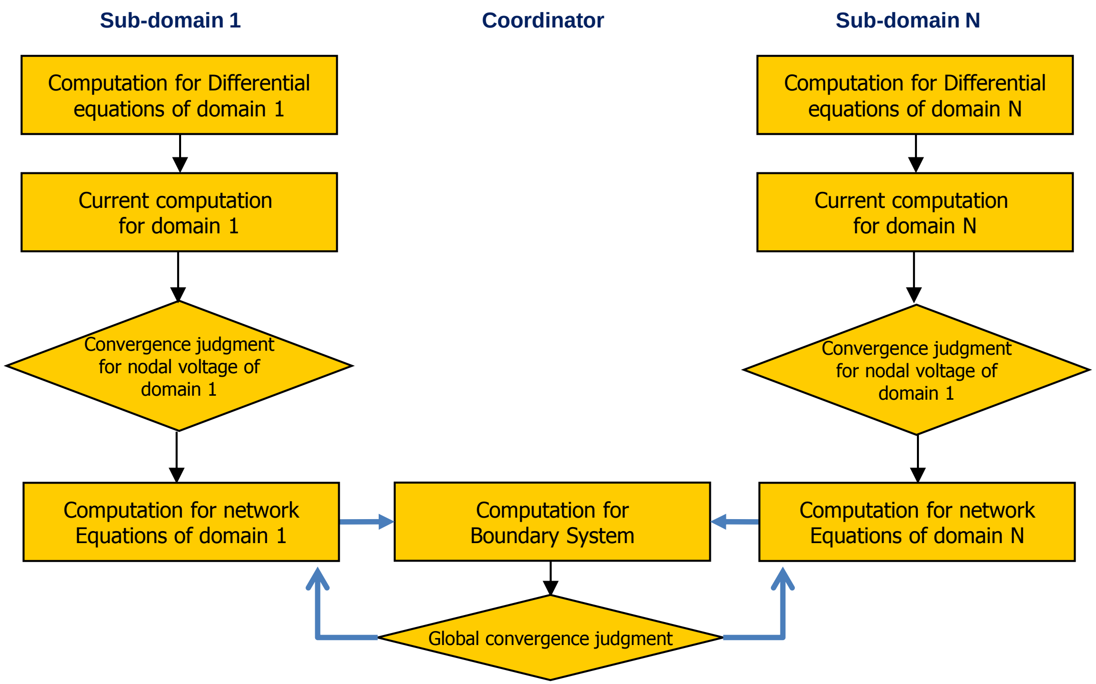


Where to find the performance?

Communication and task partitioning

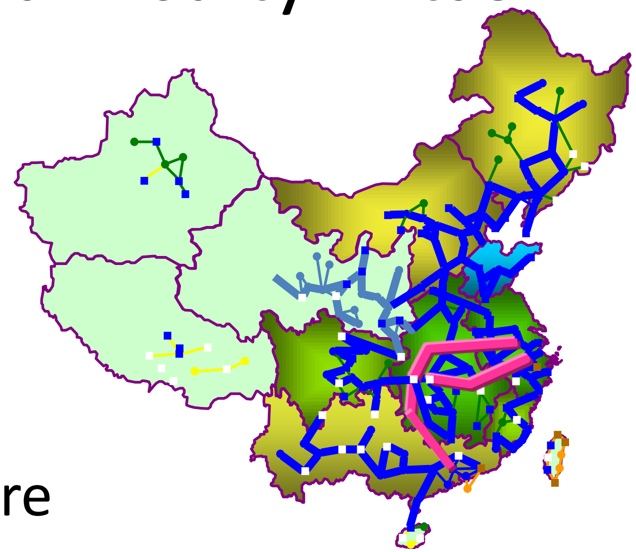
Communication Reduction

- Our solution
  - **Integrate two collective communications into one**
  - **Relative deviation of currents used for convergence**
  - **About 16% performance increase**



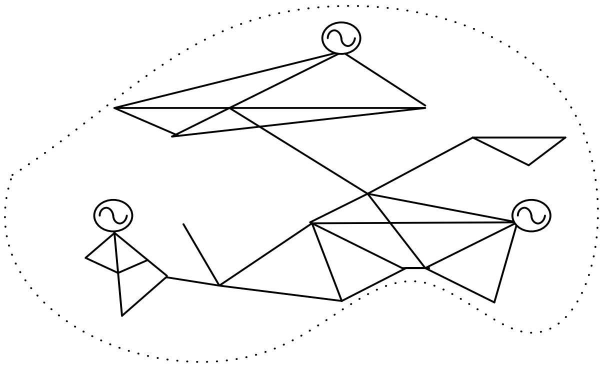
Task Partitioning

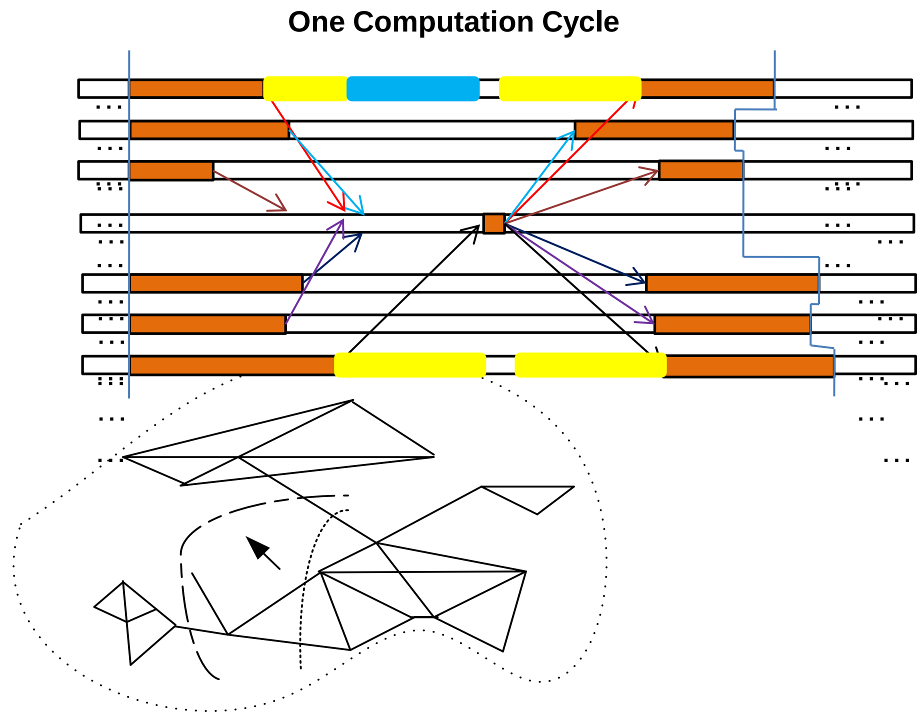
- Unstructured network brings difficulty in task partitioning
- Possible ways
  - Simple regional partition with little communication but load imbalance
  - Traditional Graph partition with more load balance but more cutting edges and more computation in boundary system
  - How to make balance between the two?

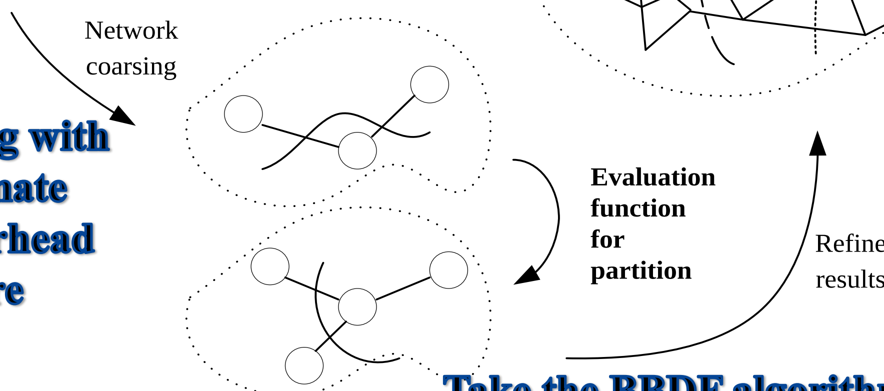


$$\begin{bmatrix} b_1 \\ \vdots \\ b_p \\ b_s \end{bmatrix} = \begin{bmatrix} A_1 & & & A_{1s} \\ & \ddots & & \vdots \\ & & A_p & A_{ps} \\ A_{s1} & \cdots & A_{sp} & A_s \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_p \\ x_s \end{bmatrix}$$

Task Partitioning







Use Network coarsing with regional info. to estimate the computation overhead of each partition more accurately

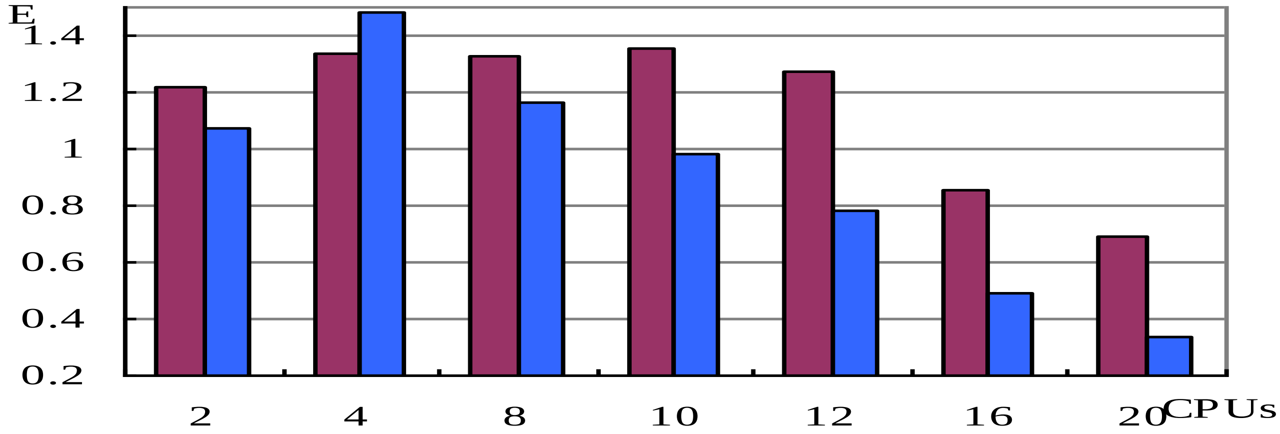
Take the BBDF algorithm into account during partition

$$F(p) = \underset{i=1,..,p}{Max}(CompCost_i) + CompCost_{Boundary}$$

Task Partitioning

- Performance improvement for our test case

  With 12 CPUs, the efficiency of our algorithm is about 63% higher than that of METIS.



Cluster: 4 Intel Xeon PIII700MHz CPUs/node + Myrinet

Outline

- Application introduction
- How to parallelism and tuning?
  - Parallel algorithm design and optimization
  - **Code Tuning based on IPF**
- Some words at the end

Why to do code tuning – App. View

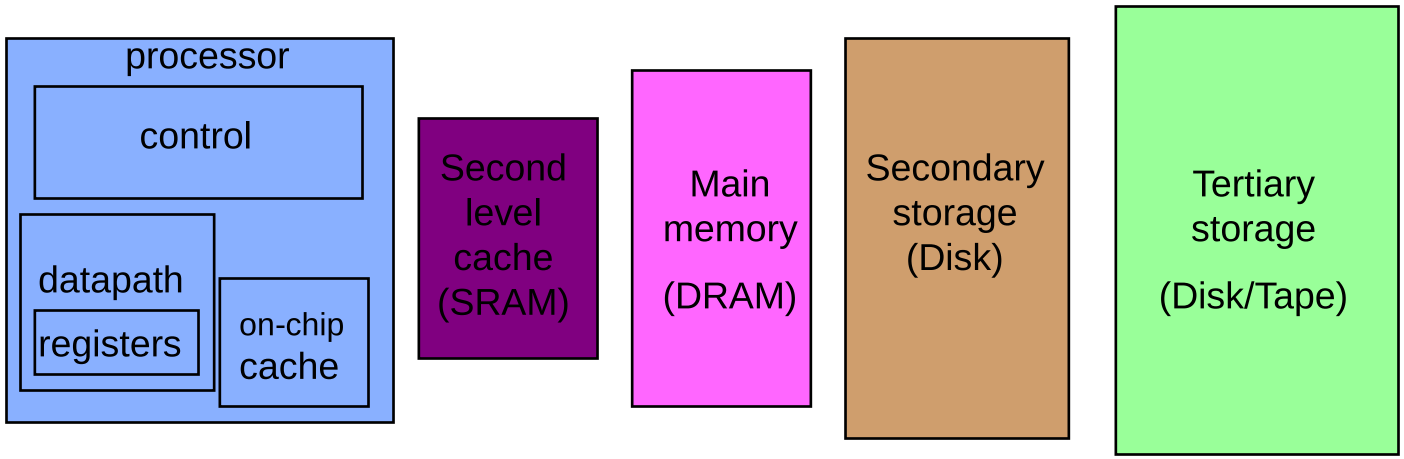
- Real-time simulation still can not be achieved

HP 4xItanium 2 1G node + Myrinet
Time step: 0.01s

Parallel Degree	1	2	4	8
Stimulated step	0.661s	0.196s	0.082s	0.024s
10s simulation	25.80s	11.83s	7.333s	4.314s

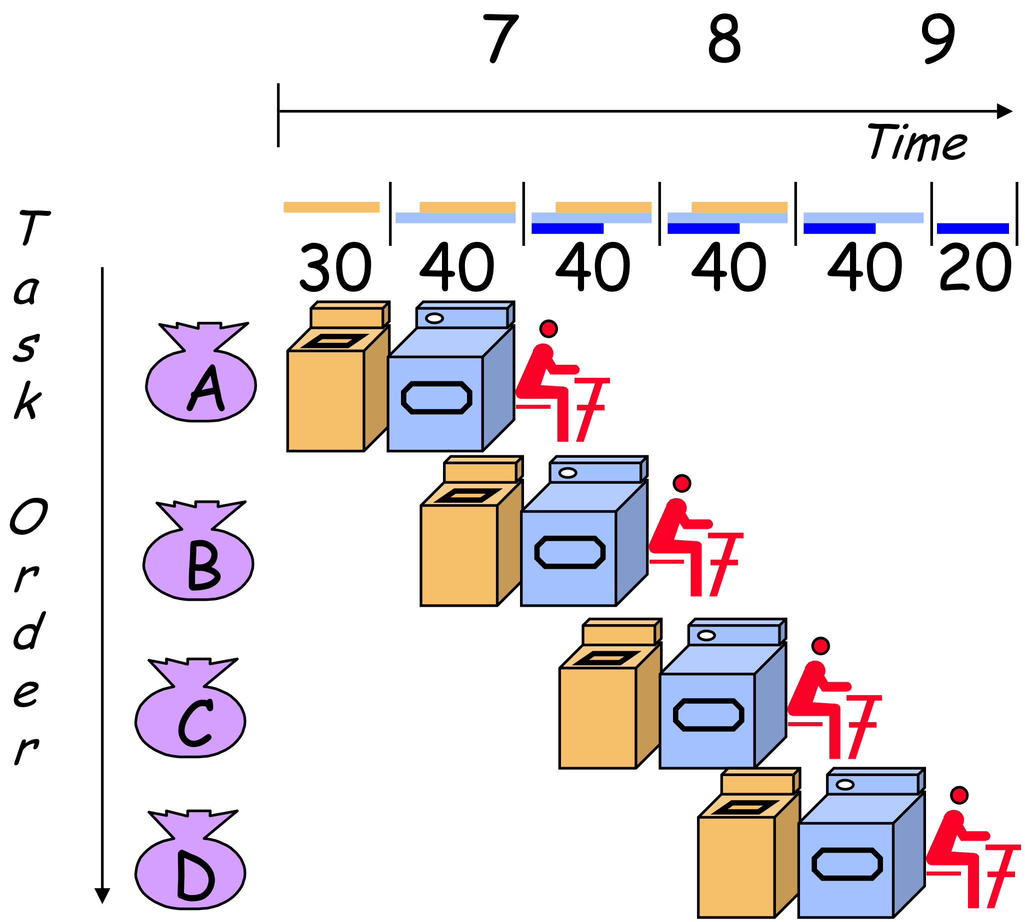
- The computation load increase of the stimulated step is due to the network equation reconfiguration and matrix refactoring of the sub-domain with fault

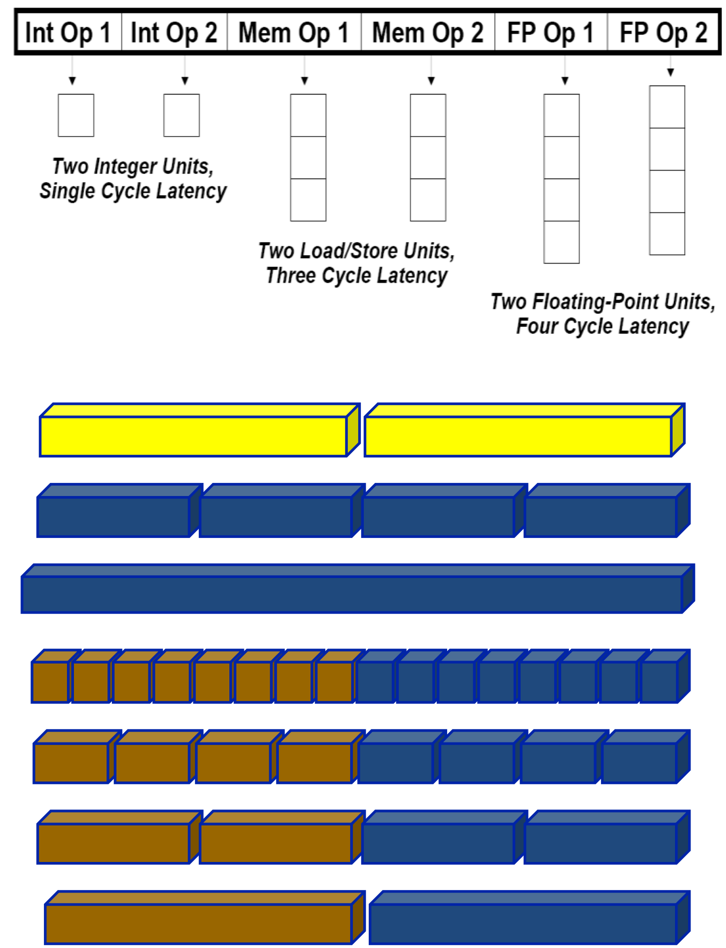
Why to do code tuning – Sys. View



Speed	1ns	10ns	100ns	10ms	10sec
Size	B	KB	MB	GB	TB

Why to do code tuning – Sys. View





Code analysis before tuning

- Global characteristics
  - Developed by C + MPI
  - Float point ops dominated
  - Memory malloc and free used for parallelism -> many pointer ops
- Very Sparse Network and Matrix computation is the bottleneck in computation
  - Avg. Sparse degree of A <10% -> many branches + vacant loops -> Poor CPU usage and memory access

Optimization Steps

1. Compiler perspective optimization

Code optimization

1. **One Code transformation Example**

2. MKL and Intel Compiler math library Used

Compiler perspective optimization

Comments	Compiler Switches	Results correct?	Performance improvement	alternative
Intel Compiler 9.0				
Baseline	-O2	Correct	/	
General Opt.	-O3	Correct	5%	
	+ -ipo	Correct	3%	
	+ -prof_gen/use	Correct	9.5%	
Floating point Opt.	+ -ftz -IPF_fltacc -IPF_fma -IPF-fp-relaxed	Correct	19%	
Addressing performance Opt.	+ -fno_alias	WRONG		-restrict (4%)

Example: Loop reconstruct for forward iteration

$$\begin{bmatrix} b_1 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & \ddots & & \\ L_{(n-1)1} & & 1 & \\ L_{n1} & \cdots & L_{n(n-1)} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix}$$

```
for(i=0;i<kk;i++)
    {
            ee = idiag[i+1]-1;
            for( ip=idiag[i]; ip<ee; ip++ )
            {
               e = jno[ip]-1;
               br[e] -= (br[i]*Lr[ip] - bi[i]*Li[ip]);
               bi[e] -= (br[i]*Li[ip] + bi[i]*Lr[ip]);
            }   // 计算inv(L)*b
    }
```

数组Lr和Li表示**按列连续存储**的矩阵元素；

数组idiag表示矩阵各列对角元所在的位置；

数组jno表示矩阵元素的行号；

kk表示矩阵维数。

Example: Loop reconstruct for forward iteration

```
for( ip=1; ip<kp; ip++ )
{
        if( ee != ip ) {
        e = jno[ip]-1;
        br[e] -= (br[i]*Lr[ip] - bi[i]*Li[ip]);
        bi[e] -= (br[i]*Li[ip] + bi[i]*Lr[ip]);
        }
        else {
        i++; ee = idiag[i+1]-1;
        }
}
```

$$\begin{bmatrix} b_1 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & \ddots & & \\ L_{(n-1)1} & & 1 & \\ L_{n1} & \cdots & L_{n(n-1)} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix}$$

Kp 矩阵非零元总数

Example: Loop reconstruct for forward iteration

- Using Local variables to reduce loads from memory

```
for( ip=1; ip<kk; ip++ )
{
        if( ee != ip ) {
        e = jno[ip]-1;
        br[e] -= (bri*Lr[ip] - bii*Li[ip]);
        bi[e] -= (bri*Li[ip] + bii*Lr[ip]);
        }
        else {
        i++; ee = idiag[i+1]-1;
        bri = br[i];
        bii= bi[i];
        }
}
```

Example: Loop reconstruct for forward iteration

- Complex operations introduced

```
__real__ ct1 = br[0]; __imag__ ct1 = bi[0];

for( ip=1; ip<kk; ip++ )
{
        if( ee != ip ) {
        __real__ ct2 = Lr[ip]; __imag__ ct2 = Li[ip];
        ct3 = ct1*ct2;
        e = jno[ip]-1;
        br[e] -= __real__ ct3;  bi[e] -= __imag__ ct3;
        }
        else {
        i++; ee = idiag[i+1]-1;
        __real__ ct1 = br[i]; __imag__ ct1 = bi[i];
        }
}
```

MKL and Intel Compiler math library

- Vector triangle functions in MKL used
- link Intel compiler math library instead of gcc math library

Final Result after tuning

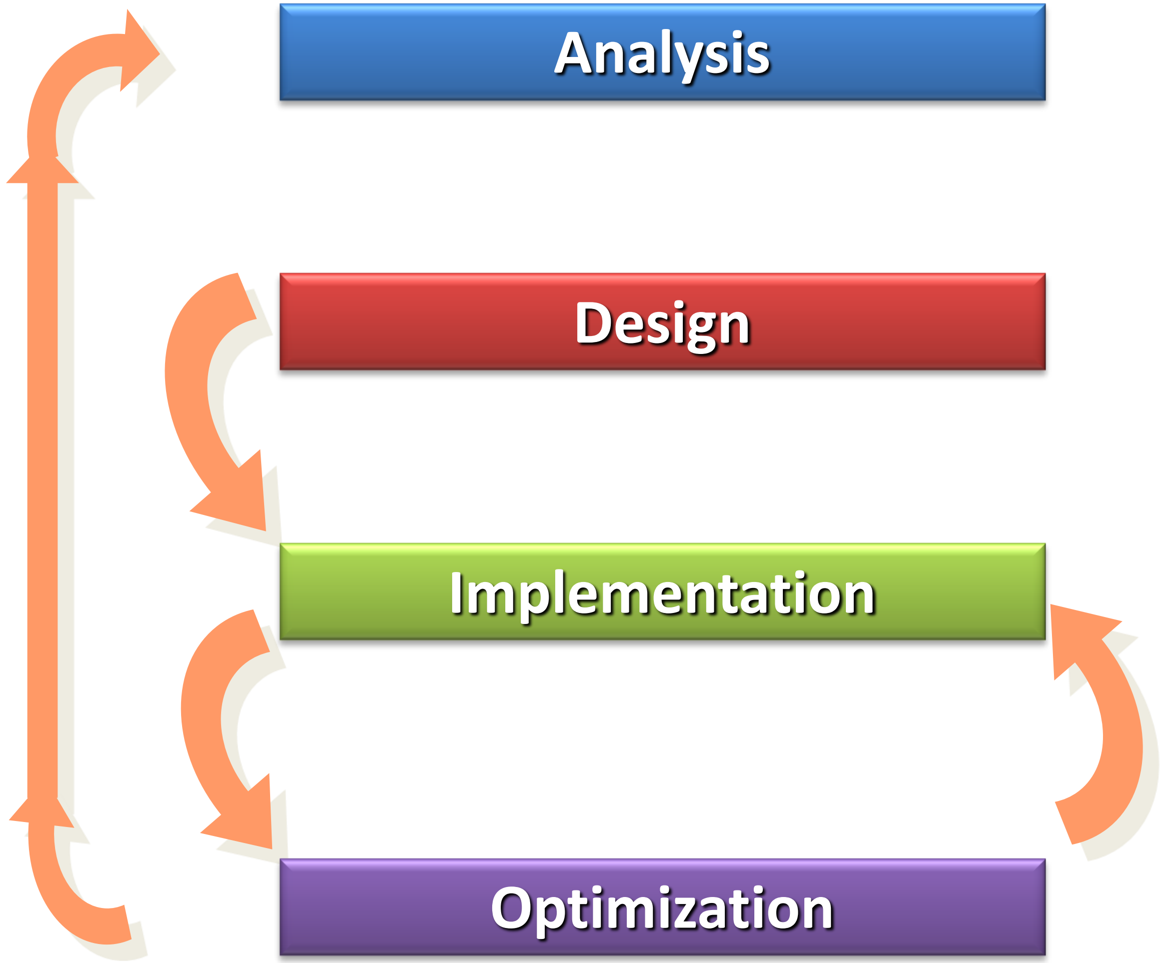
Parallel Degree (Partition number)	1	2	4	8
The stimulated step (before optimizations)	0.661s	0.196s	0.082s	0.024s
The stimulated step (after optimizations)	0.09s	0.03s	0.01s	0.006s
The long period simulation	25.8s	11.83s	7.33s	4.31s
The long period simulation	14.9s	6.63s	4.25s	2.46s

Meet the requirement of Real-time Simulation

Outline

- Application introduction
- How to parallelism and tuning?
  - Parallel algorithm design and optimization
  - Code Tuning based on IPF
- **Some words at the end**

Development Cycle



Questions I would like to ask myself

Analysis

1 Is it a valuable problem? IMPORTANT
2 Where is the gap?
3 Where is the bottleneck? your focus?

Design

1 Is there a similar model or pattern?
2 Where is the special characteristic?
3 What is the limitation/tradeoff for you?
4 Which tools can be used?

Implementation

Optimization

1 Where is the hotspots?
2 Where is the bottleneck?

Useful methods I would like to use

Analysis

1 PCAM and parallel patterns
2 Amdahl rule and …
3 known successful algorithms (examples)

Design

Implementation

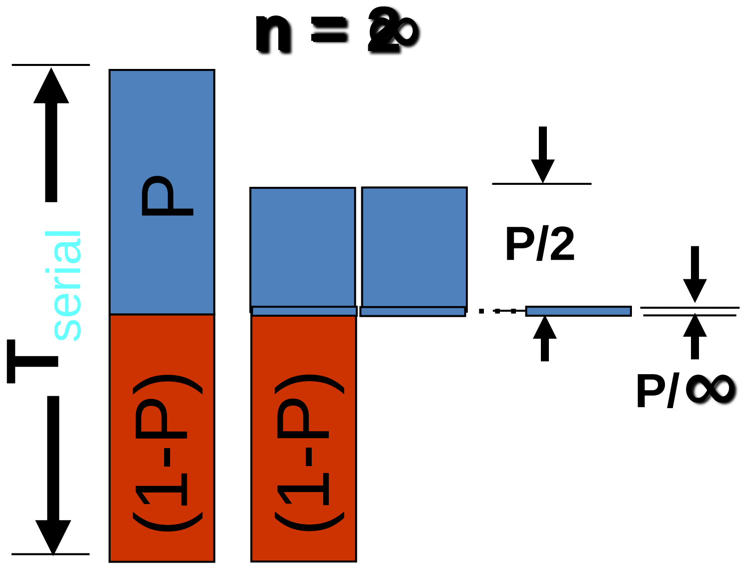
1 Reduce comm., especially global comm.
2 Comp. and comm. overlapping
3 Data reuse and …
4 tradeoff between readable and performance
5 Using libraries with standard interface

Optimization

1 Performance optimization cycle
2 Amdahl rule
3 Understanding computer system

Amdahl's Law

- Describes the upper bound of parallel execution speedup



0.5 + 0.25

$$T_{parallel} = \{(1-P) + P/n\}\, T_{serial}$$

n = number of processors

$$\text{Speedup} = T_{serial} / T_{parallel}$$

1.0/0.75 = 1.33

Serial code limits speedup

From SIGCSE 2010

Applications

Structural Patterns

Pipe-and-Filter
Agent-and-Repository
Process-Control
Event-Based/Implicit-Invocation
Puppeteer

Model-View-Controller
Iterative-Refinement
Map-Reduce
Layered-Systems
Arbitrary-Static-Task-Graph

Computational Patterns

Graph-Algorithms
Dynamic-Programming
Dense-Linear-Algebra
Sparse-Linear-Algebra
Unstructured-Grids
Structured-Grids

Graphical-Models
Finite-State-Machines
Backtrack-Branch-and-Bound
N-Body-Methods
Circuits
Spectral-Methods
Monte-Carlo

Concurrent Algorithm Strategy Patterns

Task-Parallelism
Recursive-splitting

Data-Parallelism
Pipeline

Discrete-Event
Geometric-Decomposition

Speculation

Implementation Strategy Patterns

SPMD
Strict-Data-Par.

Fork/Join
Actors
Master/Worker
Graph-Partitioning

Loop-Par.
BSP
Task-Queue

Program structure

Shared-Queue
Shared-Hash-Table

Distributed-Array
Shared-Data

Data structure

Parallel Execution Patterns

MIMD
SIMD

Thread-Pool
Speculation

Task-Graph
Data-Flow
Digital-Circuits

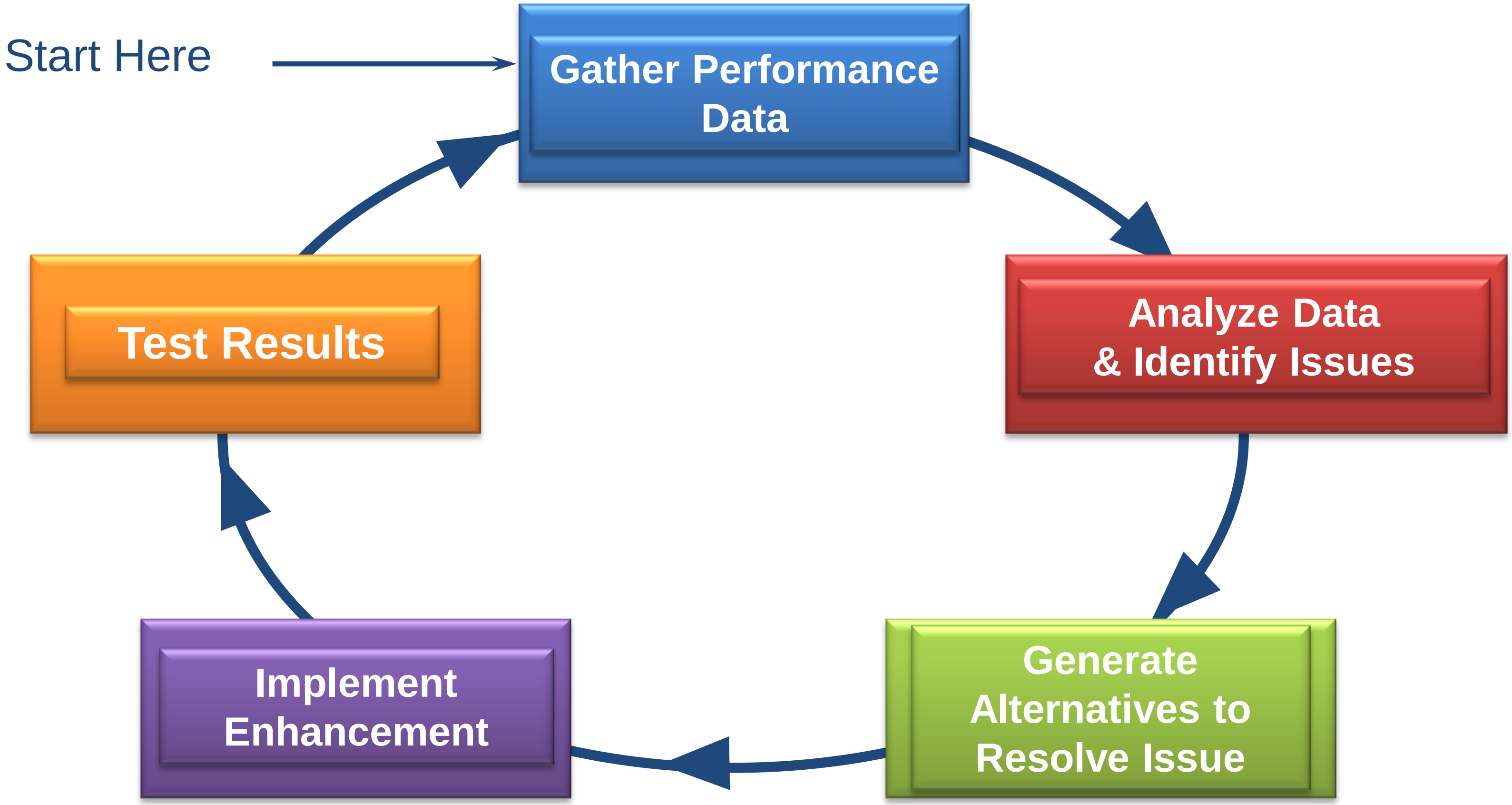
Advancing “program counters”

Message-Passing
Collective-Comm.
Mutual-Exclusion

Point-To-Point-Sync.
Collective-Sync.
Transactional-Mem.

Coordination

Performance Optimization Cycle



Useful tools I would like to use

Analysis

Performance tools such as Vtune, ITA, …

Design

1 MPI / OpenMP /…
2 Performance libraries: MKL, IPP,…
3 PETSC, METIS, …
4 TBB

Implementation

Debugger such as totalview, …

Optimization

Performance tools