AMS 290 B  Advanced Topics in the Numerical Solution of PDEs

2007:
An Introduction to Parallel Computing and Large Fluid Dynamics Codes.

Syllabus

PART A: CONCEPTS
1/ Intro to parallel computing
   Parallel machine models
   Parallel programming models
2/ Designing parallel algorithms
   (do performance and pumping things together)
   2 weeks

PART B: TOOLS
1/ MPI
2/ OpenMP
3/ Debugging, performance analysis, visualization (IDL, Vapour)
   3 weeks

PART C: CASE STUDIES
1/ ITPS (spectral methods)
2/ Finite volume (Spalart-Allmaras)
3/ Glatmaier? More?
   3 weeks

PART D: SPECIAL TOPICS IN FLUID TURBULENCE
1/ DNS/LES
2/ AMR
PART A

CONCEPTS
1.1 Overview of parallel programming

Need to solve bigger problems:
- more memory intensive (big problem)
- more computation (complex problem, churning)
- more data intensive (simple problem, lots of data)

Parallel computing provides:
- more CPU
- more memory
- solve problems that were not possible in serial
- solve problems more quickly
- more I/O

Parallel computer - set of processors that are able to work co-operatively to solve a computationally problem.

This also includes:
- parallel machines with 100,000+ processors
- networks of workstations (now)
- multiprocessor single workstation
- embedded systems

Parallel computing is no longer an exotic sub-area of computing!

Trends in applications, architecture, networking = not possible parallelism is becoming ubiquitous!
Trends in computing applications

Technology seems to move ahead very quickly.
Tending to assume that computers will eventually become "fast enough" to solve anything.

History shows that as a particular technology satisfies known applications, new applications appear that demand further new technology!

(e.g. 1940's Bhabha government report: "requirements -- 2/3 computers")

Used primarily for ballistics. Didn't expect e.g. climate, genomes, all the new technologies, (plasma physics, space, stealth ship)

e.g. Cray predicted market for 10 supercomputers!

Traditional developments driven by numerical simulation of complex systems:

weather / climate
chemical reactions (nuclear)
electronic circuits

Now commercial applications are emerging as significant drivers:

collaborative working environments (ndto conference)
computer-aided medical diagnosis
advanced graphics (games, entertainment)

e.g. ndto servers (Youtube) integrate multimedia, hpc & networking (datastreaming @ mbps/sec, simultaneous requests, encode/decode)
Example: Scientific / entertainment visualisation:

3-D dataset, $1024^3 = 10^9$ volume elements
200 operations (adds, divides, multiplies) per element
Display 30 frames/sec

$= 6.4 \times 10^{12}$ operations per sec !!!

Commercial may drive but scientific will always be important
- experiments on complex systems expensive
  (use models in a wind tunnel not real plane!)
- experiments may be impossible
  (e.g. astrophysics)
- analysis becomes intractable quickly in nonlinear regime.

Computational costs go (at least) to the 4th power of the resolution $\sim N^4$ = insurmountable demand!

E.g. climate simulation:

\[
\begin{array}{c}
\text{10 year simulation} \\
\text{1000 cubed resolution}
\end{array}
\]

\[
\begin{array}{c}
\text{1000 ops/voxel} \\
\text{1,000,000 hme steps}
\end{array}
\]

\[
\begin{array}{c}
10^{18} \text{ floating point operations total} \\
\sim 5 \text{ mm of resolution}
\end{array}
\]

\[
\begin{array}{c}
\text{Computer 1 TeraFLOPS (}10^{12} \text{ fops)} \\
\text{(1199)}
\end{array}
\]

\[
\begin{array}{c}
= \sim 10 \text{ days continuous computing.} \\
\Rightarrow \text{DATA: } 1000^3 \times 5 \times \frac{1 \text{ million}}{100} = 50 \text{ Terabytes}
\end{array}
\]

However, easy to imagine increasing thus:

100 km on each resolution $\rightarrow 10$ km

Imported process representations

\[
\begin{array}{c}
\text{(ocean-atmosphere coupling, atmoiph chemistry)}
\end{array}
\]

\[
\begin{array}{c}
\text{10 year simulation} \rightarrow 100 \text{ yr sim}
\end{array}
\]

\[
\begin{array}{c}
\text{factor of } 10 \text{ each}
\end{array}
\]

\[
\begin{array}{c}
\text{factor of } 10^4 - 10^5
\end{array}
\]
Trends in Computer Design

Computer performance grows exponentially from 1945 to present day.
- Factor of 10 every 5 years. (See figure)

Earliest computer: 10 ylops
Current fastest: ~500 Teraflops (10^12 ylops)

Not only supercomputers — also small-scale
(calculators, PCs, workstations)

E.g., Commodore 64, 1982, 64 kbytes memory
Phone today: 2 kbytes memory!

Speed-up flattened in 1980s, then sped-up again
with introduction of massively-parallel machines
Performance depends directly on time to perform
basic operation (8 no. that can be performed concurrently)
- Limited by clock cycle of processor
- Clock cycle times are decreasing slowly now
and approaching physical limit (speed of light)

Attempt to use on-chip concurrency:
E.g., operate simultaneously on all 64 bits of two nos.
that are to be multiplied.

However, this is EXPENSIVE:
Time to perform op \( T \sim \frac{1}{\sqrt{A}} \) chip area
Decrease \( T \) must increase chip area by square of factor
(factor of 4)
Ways to overcome on a chip:

- pipelining (different stages of instruction or concurrency)
- multithreaded units (several multiplexes on one chip)
  (+ cache + memory bandwidth)
  (+ multiprocessing)
  (+ prefetching)

VLSI (Very Large-Scale Integration)

- integrate more of a computer onto a chip

Trends in networking

4 node ARPA/NET 1969 50 kbps

Current fastest (1984) 8 Gbps (factor of 10,000)
(Internet land speed record)

University of Tokyo, Japan

Trend is to use physically distributed resources
as if they were part of same computer
- process on one machine
- use database from another
- render results (visualize) on a graphics computer
- real-time input/control on local workstation

Major issues of reliability, security, heterogeneity, ...

Developing code that can do this is a PARALLEL COMPUTING problem
"Parallel" and "distributed" computing are converging.

Summary

Parallelism pervades through all scale:
- chip
- distributed networks

=> need for concurrence programming

Exponential increase => need for scalability (resilience to actual processor counts)
Why not vector computers anymore

- Higher segmentation not useful for general program and data structures
- Higher segmentation increases pipeline-startup
- Performance improvement needs higher clock rates

Evolution of supercomputers

Peak Performance (GFLOPS)

Parallel Computer

- Improving cost performance
- Increase of parallel programs and low cost hardware

Vector Supercomputer

- Difficulty of performance improvement
- Focusing of new development

Hardware Architectures & Parallel Programming Models
Hochleistungsrechenzentrum Stuttgart

2. Hardware Architectures and Parallel Programming Models — 2-6

The earliest available commercial machines, were the IBM 7030 Stretch and Sperry Rand UNIVAC LARC, delivered in the early 1960s. These two machines established a pattern often observed in subsequent decades: The government-funded supercomputers were produced in very limited numbers and delivered primarily to government users. But the technology pioneered in these systems would find its way into the industrial mainstream a generation or two later in commercial systems. For example, one typical evaluation holds that “while the IBM 7030 was not considered successful, it spawned many technologies incorporated in future machines that were highly successful. The transistor logic was the basis for the IBM 7090 line of scientific computers, then the 7040 and 1400 lines. Multiprogramming, memory protection, generalized interrupts, the
At this point, there was no such thing as a commodity processor. All computer processors were custom produced. The high computational performance of the CDC 6600 at a relatively low cost was a testament to the genius of its design team. Additionally, the software tools that were provided by CDC made it possible to efficiently deliver this performance to the end user.

Although the 6600 gave CDC economic success at the time, simply delivering theoretical computational power at a substantially lower price per computation was not sufficient for CDC to dominate the market. Then, as now, the availability of applications software, the availability of specialized peripherals and storage devices tailored for specific applications, and the availability of tools to assist in programming new software were just as important to many customers.

The needs of the government users were different. Because the specific applications and codes they ran for defense applications were often secret, frequently were tied to special-purpose custom hardware and peripherals built in small numbers, and changed quickly over time, the avail-
good penetration of TOP500 platforms in industry does not necessarily indicate that applications in industry have scaled up in proportion to the scaling of TOP500 platforms over the years; the size of academic platforms is a better indicator of the scale of applications running on them.

Keeping those caveats in mind, many things can be learned from studying the TOP500 data.

There has been continuing rapid improvement in the capability of high-performance systems over the last decade (see Figure 3.3). Mean Linpack performance has improved fairly steadily, by roughly an order of magnitude every 4 years (about 80 percent improvement annually). The performance of the very fastest machines (as measured by the $R_{\text{max}}$ of the machine) has shown much greater unevenness over this period but on average seems roughly comparable. Interestingly, the performance of the least capable machines on the list has been improving more rapidly than

---

26 ASCI White and ASCI Red are two supercomputers installed at DOE sites as part of the ASC strategy. Information on all of the ASC supercomputers is available at [http://www.llnl.gov/asci/platforms/platforms.html](http://www.llnl.gov/asci/platforms/platforms.html).

27 The $R_{\text{max}}$ is the maximal performance achieved on the Linpack benchmark—for any size system of linear equations.
a rather high degree of problem complexity. There may be multiple time and space scales, different component sub-models (e.g., magnetic, hydrodynamic, or biochemical), different types of equations (e.g., nonlinear partial differential equations and ordinary differential equations), and different algorithms (spectral, finite-difference, finite-element, algebraic) covering a range of problems being studied in each area.

It is clear from the summary above that a 1,000-fold increase in computing power is needed almost immediately and a 1,000,000-fold increase will ultimately be needed by the current major applications. Some of this increase can be expected on the basis of Moore's law and greater numbers of processors per machine. Any increase in raw computing power in terms of raw flops will have to be accompanied by larger memories to accommodate larger problems, and internal bandwidth will have to increase dramatically. As problems become more data-oriented, more effective parallel I/O to external devices will be needed, which will themselves have to be larger than today's disks and mass storage systems.

Table 4.1 summarizes six supercomputing system bottlenecks that often limit performance on important applications and gives examples of the applications. It should be noted that the limitations/bottlenecks in application areas are heavily dependent on the problem-solving strategies and the algorithms used.

The ability of applications to be mapped onto hardware effectively is critically dependent on the software of the overall system, including both the operating system and the compilers. Application programmers and users will need software that exploits the features of any given machine without heroic efforts on the programmer's part. Software ideally should

<table>
<thead>
<tr>
<th>Limitation/Bottleneck</th>
<th>Typical Areas of Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating-point performance</td>
<td>Astrophysics, defense radar cross-sections, climate modeling, plasma physics</td>
</tr>
<tr>
<td>Memory size</td>
<td>Intelligence, materials science, genomics, automobile noise, vibration, and harshness</td>
</tr>
<tr>
<td>Memory bandwidth</td>
<td>Intelligence, climate modeling, materials science, astrophysics, biological systems modeling</td>
</tr>
<tr>
<td>Memory latency</td>
<td>Intelligence, nuclear simulation, climate modeling, astrophysics, biological systems modeling</td>
</tr>
<tr>
<td>Interconnect bandwidth</td>
<td>Intelligence, climate modeling, materials science, astrophysics, biological systems modeling</td>
</tr>
<tr>
<td>Interconnect latency</td>
<td>Intelligence, nuclear simulation, climate modeling, astrophysics, biological systems modeling</td>
</tr>
</tbody>
</table>
Trying to find the parallel equivalent of the single machine or von Neumann model:

![Diagram of a von Neumann computer]

Multicomputer model:
- A kind of von Neumann computer or nodes linked by a interconnection network.

- Each computer executes its own program.
- Program may access local memory.
- Messages across network used to communicate with other computers (read/write remote memory).
- Ideally, cost of sending msg:
  - independent of node to ahon / traffic (UMA)
  - dependent on msg length

![Diagram of a multicomputer system]

- Access to local memory cheaper than access to remote.
- Fundamental parallel concepts = concurrency, availability, locality.
Multicomputer is a
- distributed memory (DMP)
- MIMD multiple instruction multiple data

model. e.g. IBM SP, Thinking Machine, CMS, CAG TPE

Each processor executes a separate stream of instructions on its own local data, and the total memory of the machine is distributed amongst the processors. Not centrally located.

However, real distributed MIMD machines may not have msg passing costs that are independent of node(s) & traffic. (more later)

Other models
- (SMP)

Multiprocessor = shared memory MIMD
- All processors have access to a common memory
- Usually via a bus (or hierarchy of buses)
- Usually mean need memory hierarchy:
  - Store copies of frequently-used data in
    - Local memory = CACHE
    - Faster access time than shared memory

![Diagram](https://via.placeholder.com/150)

All CPUs can access
- Global memory
- Local memory
- Crossbar (independent access from each CPU)

Take Note:
- Uniform Memory Access (UMA)
- Memory interconnect
Multi-computer algorithms usually work on multi-processors 
(shared memory = efficient message passing!)

More specialized:
- SIMD: single instruction, multiple data
- All processors do same instruction on different data
- Reduce memory complexity
- Only for certain very regular problems
  (image processing)
  - e.g. MasPar

More specialized:
- across networks
  - "Grid" computing
  - LAN: local area network
  - WAN: wide area network
- network technology
  - ethernet
  - ATM (asynchronous transfer mode)
  - globus (middleware)
  - heterogeneous - not all one kind of computer.

BECOMING more and more common:

Hierarchical memory systems
= combination of DMP & SMP

= "clusters"
  of SMP nodes
  + node interconnect
Parallel Machine Models: von Neumann

von Neumann computer

- (named after Hungarian mathematician John von Neumann)
- “stored program” concept: CPU executes sequence of instructions = read/write to memory
- Memory stores both program and data
- Instructions are coded data
- CPU gets instructions, decodes, and sequentially performs them
Parallel Machine Models: Shared Memory

- Many different varieties
- Multiple processors operate independently but access global memory space
- Changes in a memory location visible to all processors
- UMA – uniform memory access = SMP (symmetric multiprocessor)
  - Everything identical
  - Equal memory access
- NUMA – non-uniform
  - unequal access to memory
  - linked SMPs
- ISSUES: cache coherence

✓ User friendly
✓ Fast access: O(1) connectivity
× Lack of scalability (buses)
× Synchronisation
× Expense for large machines
Parallel Machine Models: Distributed Memory

- Many different varieties
- Processors have local memory
- Require communication network to connect processor memory
- No global address space: all separate
- Data from one proc must be communicated to another if required
- Synchronisation is programmer’s responsibility
- Network fabric varies greatly!

![Distributed memory diagram]

- ✓ Very scalable
- ✓ LOCAL memory access rapid
- ✓ No cache coherence
- ✓ Use commodity processors
- × Lot of programmer responsibility!
- × NUMA times
Parallel Machine Models: Hybrid Memory

- Biggest machines today have both.
- Shared memory component of cc-UMA SMP
- Distributed network of these
- Advantages and disadvantages are those of the individual parts!
Concepts and Terminology

Flynn's Classical Taxonomy

- There are different ways to classify parallel computers. One of the more widely used classifications, in use since 1966, is called Flynn's Taxonomy.

- Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of Instruction and Data. Each of these dimensions can have only one of two possible states: Single or Multiple.

- The matrix below defines the 4 possible classifications according to Flynn.

<table>
<thead>
<tr>
<th>S I S D</th>
<th>S I M D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Instruction, Single Data</td>
<td>Single Instruction, Multiple Data</td>
</tr>
<tr>
<td>M I S D</td>
<td>M I M D</td>
</tr>
<tr>
<td>Multiple Instruction, Single Data</td>
<td>Multiple Instruction, Multiple Data</td>
</tr>
</tbody>
</table>

**Single Instruction, Single Data (SISD):**

- A serial (non-parallel) computer
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest and until recently, the most prevalent form of computer
- Examples: most PCs, single CPU workstations and mainframes

**Single Instruction, Multiple Data (SIMD):**

- A type of parallel computer
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units.
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays and Vector Pipelines
- Examples:
  - Processor Arrays: Connection Machine CM-2, Maspar MP-1, MP-2
  - Vector Pipelines: IBM 9000, Cray C90, Fujitsu VP, NEC SX-2, Hitachi S820

**Multiple Instruction, Single Data (MISD):**

- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams.
- Few actual examples of this class of parallel computer have ever existed. One is the experimental Carnegie-Mellon C.mmp computer (1971).
- Some conceivable uses might be:
  - multiple frequency filters operating on a single signal stream
  - multiple cryptography algorithms attempting to crack a single coded message.
**Multiple Instruction, Multiple Data (MIMD):**

- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, networked parallel computer "grids" and multi-processor SMP computers - including some types of PCs.

---

**Concepts and Terminology**

**Some General Parallel Terminology**

Like everything else, parallel computing has its own "jargon". Some of the more commonly used terms associated with parallel computing are listed below. Most of these will be discussed in more detail later.

**Task**
A logically discrete section of computational work. A task is typically a program or program-like set of instructions that is executed by a processor.

**Parallel Task**
A task that can be executed by multiple processors safely (yields correct results)

**Serial Execution**
Execution of a program sequentially, one statement at a time. In the simplest sense, this is what happens on a one processor machine. However, virtually all parallel tasks will have sections of a parallel program that must be executed serially.

**Parallel Execution**
Execution of a program by more than one task, with each task being able to execute the same or different statement at the same moment in time.

**Shared Memory**
From a strictly hardware point of view, describes a computer architecture where all processors have direct (usually bus based) access to common physical memory. In a programming sense, it describes a model where parallel tasks all have the same "picture" of memory and can directly address and access the same logical memory locations regardless of where the physical memory actually exists.

**Distributed Memory**
In hardware, refers to network based memory access for physical memory that is not common. As a programming model, tasks can only logically "see" local machine memory and must use communications to access memory on other machines where other tasks are executing.

**Communications**
Parallel tasks typically need to exchange data. There are several ways this can be accomplished, such as through a shared memory bus or over a network, however the actual event of data exchange is commonly referred to as communications regardless of the method employed.

**Synchronization**
The coordination of parallel tasks in real time, very often associated with communications. Often implemented by establishing a synchronization point within an application where a task may not proceed further until another task(s) reaches the same or logically equivalent point.

Synchronization usually involves waiting by at least one task, and can therefore cause a parallel application's wall clock execution time to increase.

**Granularity**
In parallel computing, granularity is a qualitative measure of the ratio of computation to communication.
- **Coarse**: relatively large amounts of computational work are done between communication events
- **Fine**: relatively small amounts of computational work are done between communication events

**Observed Speedup**
Observed speedup of a code which has been parallelized, defined as:

---

Control Structure of Parallel Programs

- Processing units in parallel computers either operate under the centralized control of a single control unit or work independently.

- If there is a single control unit that dispatches the same instruction to various processors (that work on different data), the model is referred to as single instruction stream, multiple data stream (SIMD).

- If each processor has its own control control unit, each processor can execute different instructions on different data items. This model is called multiple instruction stream, multiple data stream (MIMD).
SIMD and MIMD Processors

PE: Processing Element

A typical SIMD architecture (a) and a typical MIMD architecture (b).
SIMD Processors

- Some of the earliest parallel computers such as the Illiac IV, MPP, DAP, CM-2, and MasPar MP-1 belonged to this class of machines.

- Variants of this concept have found use in co-processing units such as the MMX units in Intel processors and DSP chips such as the Sharc.

- SIMD relies on the regular structure of computations (such as those in image processing).

- It is often necessary to selectively turn off operations on certain data items. For this reason, most SIMD programming paradigms allow for an "activity mask", which determines if a processor should participate in a computation or not.
Conditional Execution in SIMD Processors

if (B == 0)  
    C = A;  
else  
    C = A/B;

(a)

Initial values

Processor 0

```
A  5
B  0
C  0
```

Processor 1

```
A  4
B  2
C  0
```

Processor 2

```
A  1
B  1
C  0
```

Processor 3

```
A  0
B  0
C  0
```

Step 1

Processor 0

```
A  5
B  0
C  5
```

Processor 1

```
A  4
B  2
C  0
```

Processor 2

```
A  1
B  1
C  0
```

Processor 3

```
A  0
B  0
C  0
```

Step 2

Processor 0

```
A  5
B  0
C  5
```

Processor 1

```
A  4
B  2
C  2
```

Processor 2

```
A  1
B  1
C  1
```

Processor 3

```
A  0
B  0
C  0
```

(b)

Executing a conditional statement on an SIMD computer with four processors: (a) the conditional statement; (b) the execution of the statement in two steps.
MIMD Processors

- In contrast to SIMD processors, MIMD processors can execute different programs on different processors.

- A variant of this, called single program multiple data streams (SPMD) executes the same program on different processors.

- It is easy to see that SPMD and MIMD are closely related in terms of programming flexibility and underlying architectural support.

- Examples of such platforms include current generation Sun Ultra Servers, SGI Origin Servers, multiprocessor PCs, workstation clusters, and the IBM SP.
SIMD-MIMD Comparison

- SIMD computers require less hardware than MIMD computers (single control unit).

- However, since SIMD processors are specially designed, they tend to be expensive and have long design cycles.

- Not all applications are naturally suited to SIMD processors.

- In contrast, platforms supporting the SPMD paradigm can be built from inexpensive off-the-shelf components with relatively little effort in a short amount of time.
Communication Model of Parallel Platforms

- There are two primary forms of data exchange between parallel tasks – accessing a shared data space and exchanging messages.

- Platforms that provide a shared data space are called shared-address-space machines or multiprocessors.

- Platforms that support messaging are also called message passing platforms or multicomputers.
Shared-Address-Space Platforms

- Part (or all) of the memory is accessible to all processors.

- Processors interact by modifying data objects stored in this shared-address-space.

- If the time taken by a processor to access any memory word in the system global or local is identical, the platform is classified as a uniform memory access (UMA), else, a non-uniform memory access (NUMA) machine.
NUMA and UMA Shared-Address-Space Platforms

Typical shared-address-space architectures: (a) Uniform-memory-access shared-address-space computer; (b) Uniform-memory-access shared-address-space computer with caches and memories; (c) Non-uniform-memory-access shared-address-space computer with local memory only.
NUMA and UMA Shared-Address-Space Platforms

- The distinction between NUMA and UMA platforms is important from the point of view of algorithm design. NUMA machines require locality from underlying algorithms for performance.

- Programming these platforms is easier since reads and writes are implicitly visible to other processors.

- However, read-write data to shared data must be coordinated (this will be discussed in greater detail when we talk about threads programming).

- Caches in such machines require coordinated access to multiple copies. This leads to the cache coherence problem.

- A weaker model of these machines provides an address map, but not coordinated access. These models are called non cache coherent shared address space machines.
Shared-Address-Space vs. Shared Memory Machines

- It is important to note the difference between the terms shared address space and shared memory.

- We refer to the former as a programming abstraction and to the latter as a physical machine attribute.

- It is possible to provide a shared address space using a physically distributed memory.
Message-Passing Platforms

- These platforms comprise of a set of processors and their own (exclusive) memory.

- Instances of such a view come naturally from clustered workstations and non-shared-address-space multicomputers.

- These platforms are programmed using (variants of) send and receive primitives.

- Libraries such as MPI and PVM provide such primitives.
Message Passing vs. Shared Address Space Platforms

- Message passing requires little hardware support, other than a network.

- Shared address space platforms can easily emulate message passing. The reverse is more difficult to do (in an efficient manner).
Interconnect

Memory interconnect:
- bus
- cross-bar

Node interconnect:
- bus-based (lots of buses)
- multi-link networks
  - e.g., ring
- e.g., 2-D / 3-D torus - ring of rings

- cross-bar
- hierarchical - multi-level cross bars
- full interconnect

Network topology is an issue.

Massive topic

See notes from