Center of Excellence: Supercomputing for AI & Big-Data

# Art of Parallel Programming: Think Parallel


**by: Tassadaq Hussain**

**Director Centre for AI and BigData**

**Professor Electrical Engiennering Department**

**Namal University Mianwali**


## Collaborations:

**Barcelona Supercomputing Center, Spain**

**European Network on High Performance and Embedded Architecture and Compilation**

**Pakistan Supercomputing Center**

# Why Parallel Computing

- Traditionally, parallel computing has been considered to be "the high end of computing" and has been motivated by numerical simulations of complex systems and "Grand Challenge Problems" such as:
  - weather and climate
  - chemical and nuclear reactions
  - biological, human genome
  - geological, seismic activity
  - mechanical devices - from prosthetics to spacecraft
  - electronic circuits
  - manufacturing processes

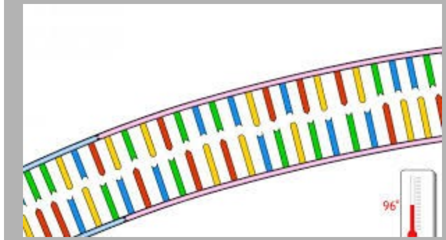
# The future

- During the past 10 years, the trends indicated by ever faster networks, distributed systems, and multi-processor computer architectures (even at the desktop level) clearly show that ***parallelism is the future of computing***.
- It will be multi-forms, mixing general purpose solutions (PC) and very speciliazed solutions as IBM Cells, ClearSpeed, GPGPU from Nvidia etc.

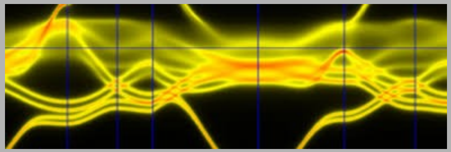
# Applications in Pakistan

- **Representative application domains requiring more than a Desktop PC Performance**

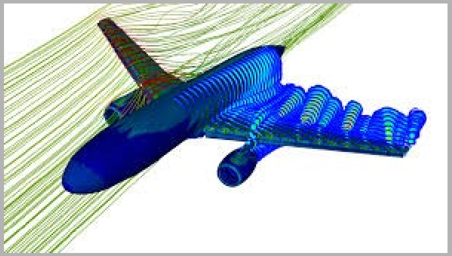
Biomedical
[ Alpha Genomic ]



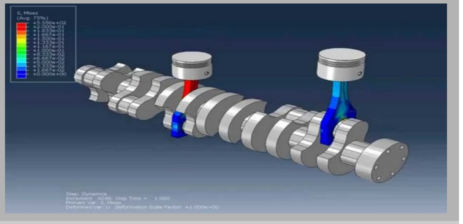
Control and
Simulation
[ CUST ]



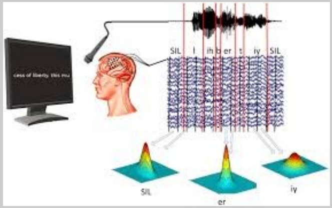
Aerodynamics
[Risalpur College]



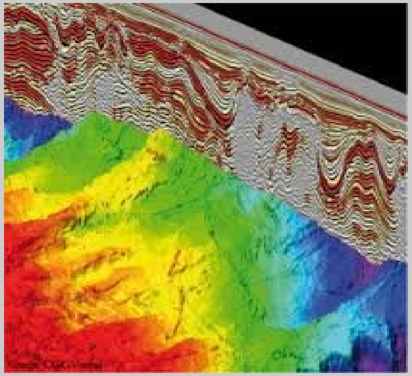
Mechanical Systems
Modeling and
Simulation
[HITech]



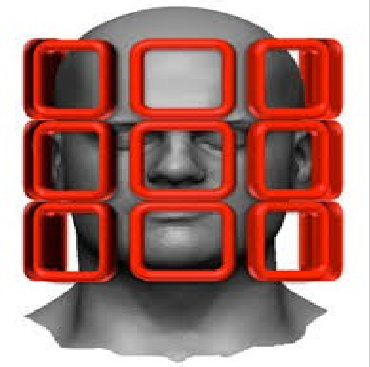
Brain Computer
Interface
[Riphah NewZeland
College of
Chiropractic]



Earth Sciences
(QAU)



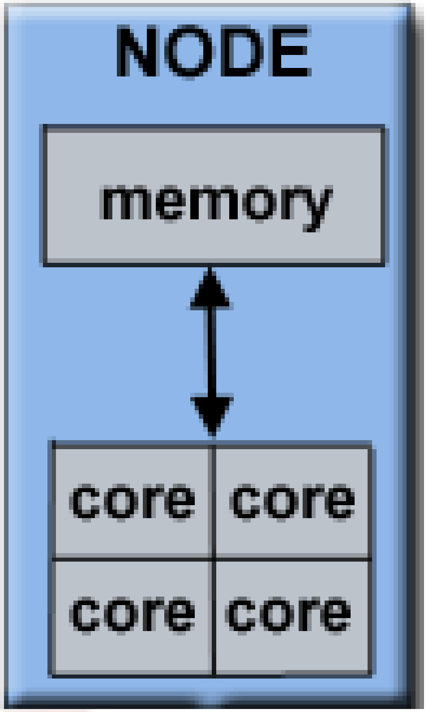
Parallel MRI
[NCP]



Artificial Intelligence

# Existing Solutions

- Supercomputer
- Server based computing
  - Shared Memory
  - Distributed Shared Memory
  - Centralized
  - Simulation Software Programs




HP ProLiant Server

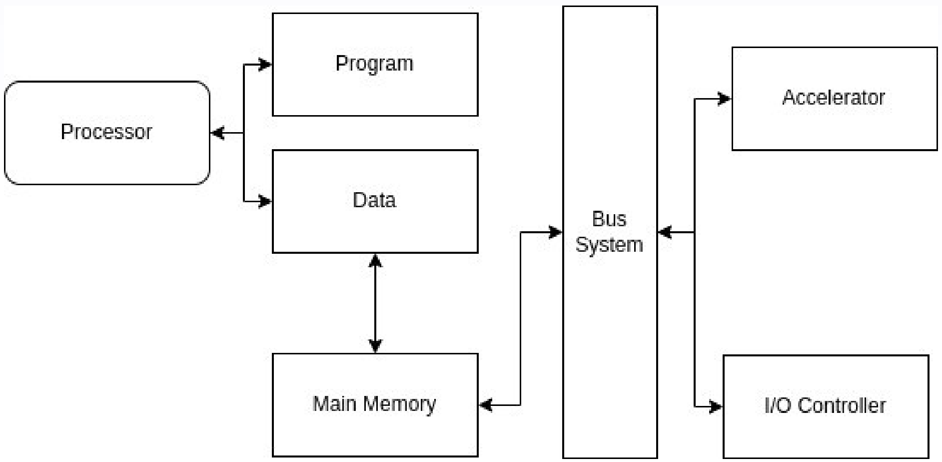
# So Whats Wrong with the existing solutions

- Based on **conventional micro-processor** architecture (Homogeneous) **=**> **Weak Compute Capability**
- Sequential Programming Models **=**> **no support for AI Applications**
- Performance depends upon Software Development Tools **=**> **not scalable.**
- **Not supporting Artificial Intelligence Frameworks**

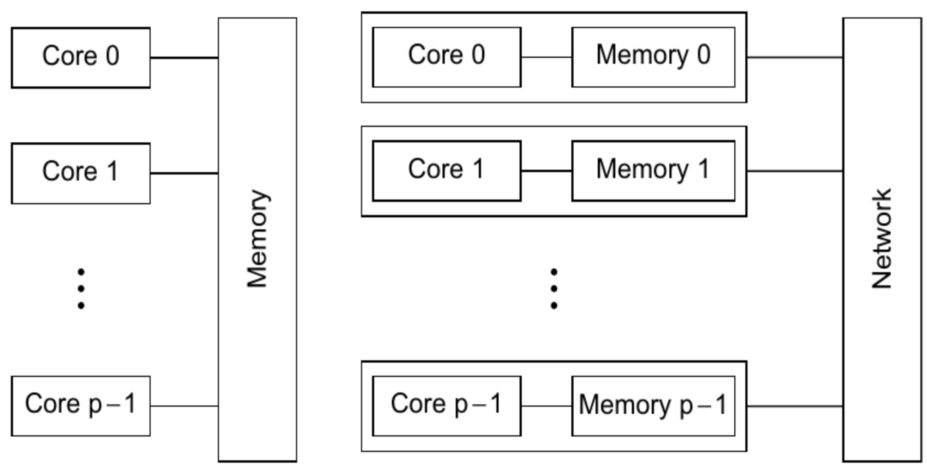
# Applications Problems

- Compute Bound
- Memory Bound
- I/O, Network Bound
- Complex and Irregular

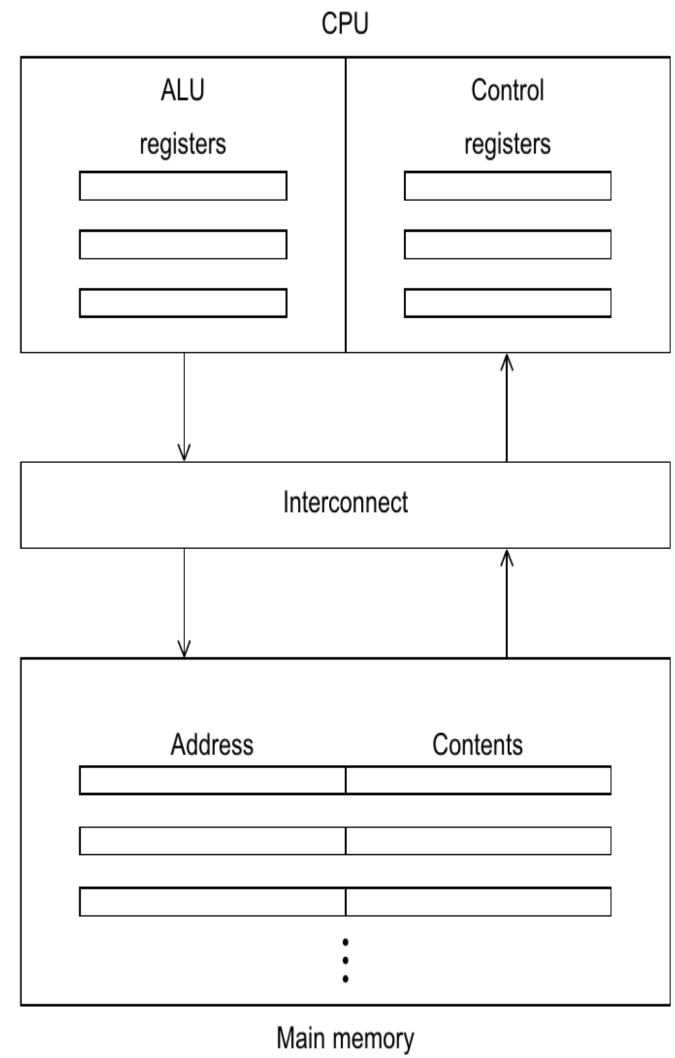
# Computer System Architecture




a

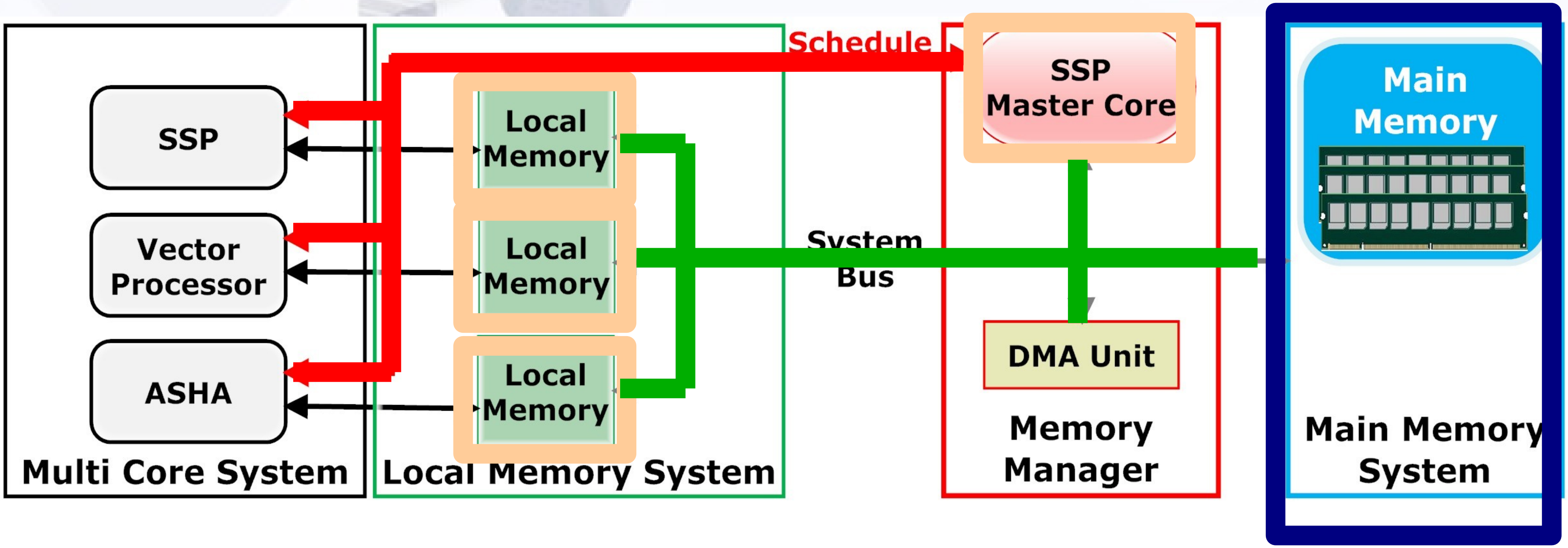



b

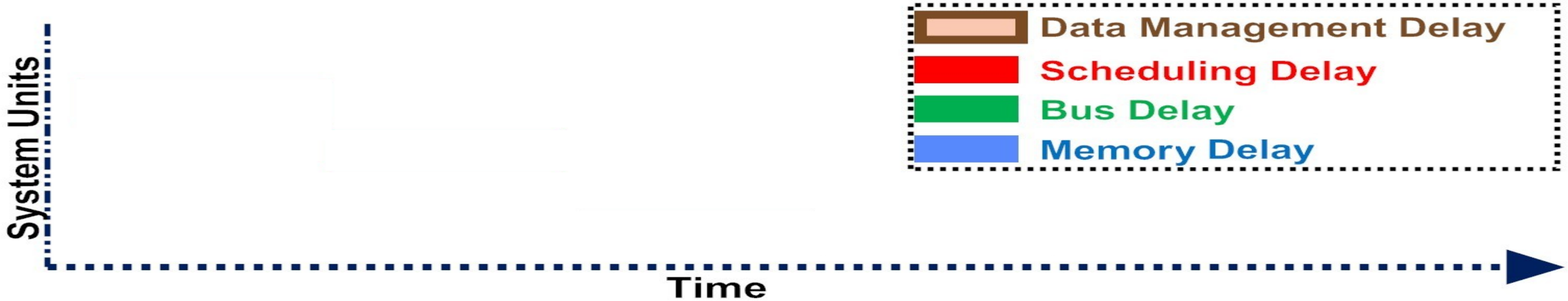



c

# System Level Delays




(a) Multi-core system




(b) Delays introduced

# Performance Requirement

- **FLOPS   Floating Point Operation Per Second**
  - Number of computation per second
- **FLOPS/Byte**
  - Operation on input data
- **FLOPS/watt**

# Parallel Computing: The Computational Problem

- The computational problem usually demonstrates characteristics such as the ability to be:
  - Broken apart into discrete pieces of work that can be solved simultaneously;
  - Execute multiple program instructions at any moment in time;
  - Solved in less time with multiple compute resources than with a single compute resource.
  - Amdahl's Law
    - Speedup = 1 / [(1 - P) + (P / N)]

# Speedup

Assume an algorithm that parallelized 80% by sorting task (P = 0.8) using N=4 number of processors.

- Speedup = 1 / [(1 - P) + (P / N)]
- Speedup = 1 / [(1 - .8) + (.8 / 4)]
- Speedup = 1 / [(.2) + (.2)]
- Speedup = 1 / [.4]
- Speedup = 2.5

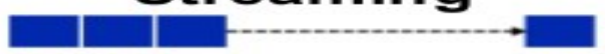
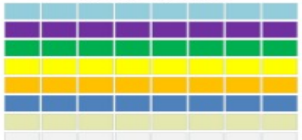
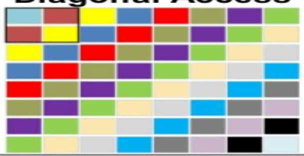
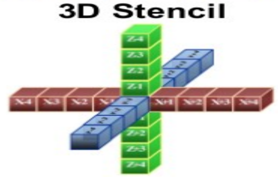
What if it improvers 100, P=1

# Key Considerations for Application Development

- I/O Architecture
- Data Architecture
- Hardware Architecture
- Software Architecture

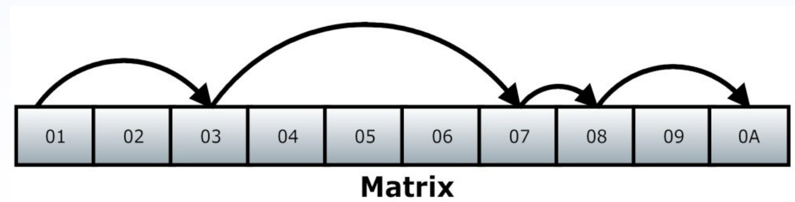
# Basic types of memory access patterns

- **Regular access**
  - Fixed stride
  - Predictable
  - Parallel
- **Irregular access**
  - Variable strides
  - Known
    - Predictable at compile-time
  - Unknown
    - Independent
    - Dependent

| Kernel | Description | Access Pattern |
|---|---|---|
| Rad_Con | Radian Converter converts degree into radian | Load/Store  |
| Thresh | Thresholding is an application of image segmentation, which takes streaming 8-bit pixel data and generates binary output. | |
| FIR | Finite Impulse Response calculates the weighted sum of the current and past inputs. | Streaming  |
| FFT | Fast Fourier Transform is used for transferring a time-domain signal into corresponding frequency-domain signal. | 1D Block  |
| Mat_Mul | Matrix Multiplication takes pair of tiled data and produce Output tile. Output= Row[Vector] × Column[Vector] X=Y×Z | Column & Vector Access  |
| Smith_W | Smith-Waterman determines the optimal local alignments between nucleotide or protein sequences. | Diagonal Access  |
| Lapl | Laplacian kernel applies discrete convolution filter that can approximate the second order derivatives. | 2D Tiled  |
| 3D-Sten | 3D-Stencil algorithm averages nearest neighbor points (size 8x9x8) in 3D. | 3D Stencil  |

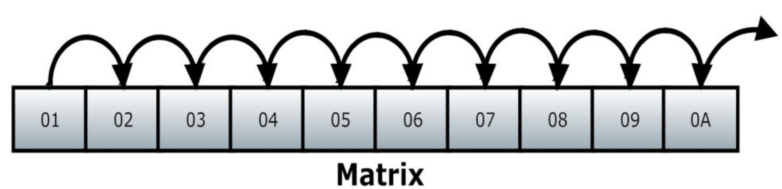
# *Basic types of memory access units*

- Load/store access
  - Conventional
  - Arbitrary access patterns
  - Fine granularity access
  - Low throughput




```
for ( inc=start_value; inc=end_value; inc++)
    {
    //Addresses Management by Microprocessor
    local_buffer[inc]=mem_buffer[inc+offset];
    }
```

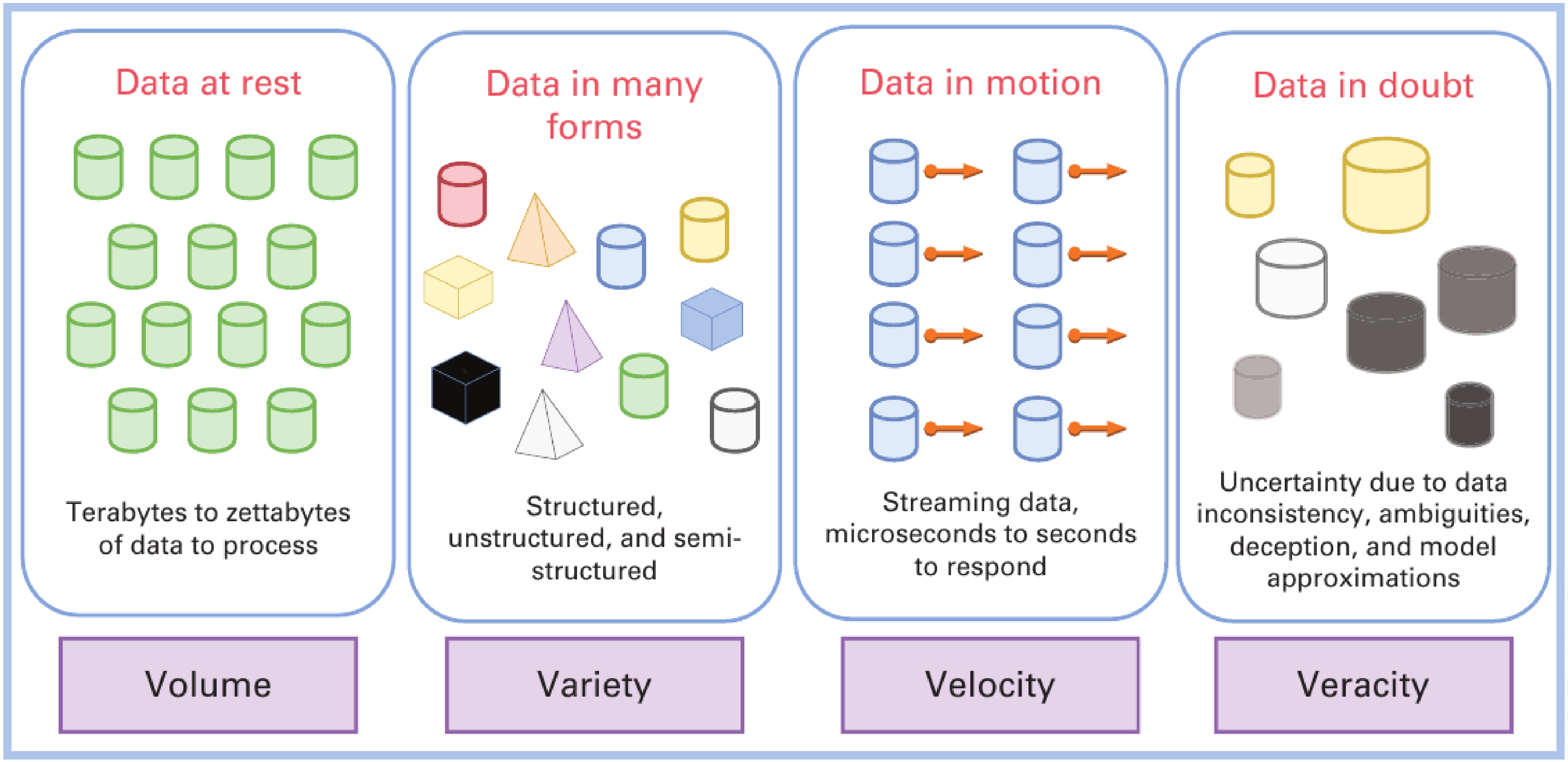
- DMA
  - Streaming access
  - Programmed with function call
  - High latency
  - High throughput




```
Read_DMA(source,destination,stream);
Write_DMA(source,destination,stream);
```

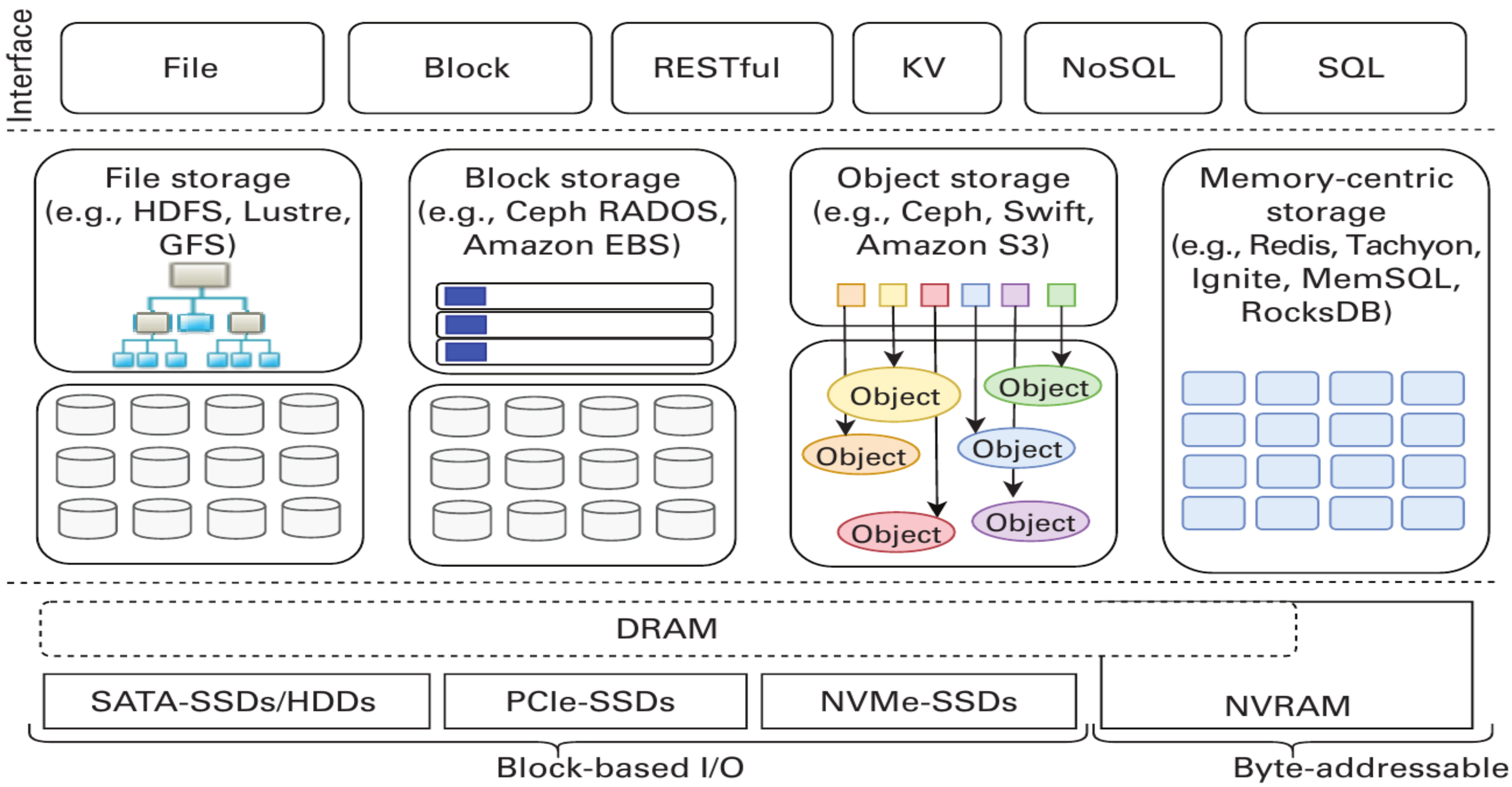
# Key Considerations for Application Development

- I/O Architecture
- **Data Architecture**
- Hardware Architecture
- Software Architecture



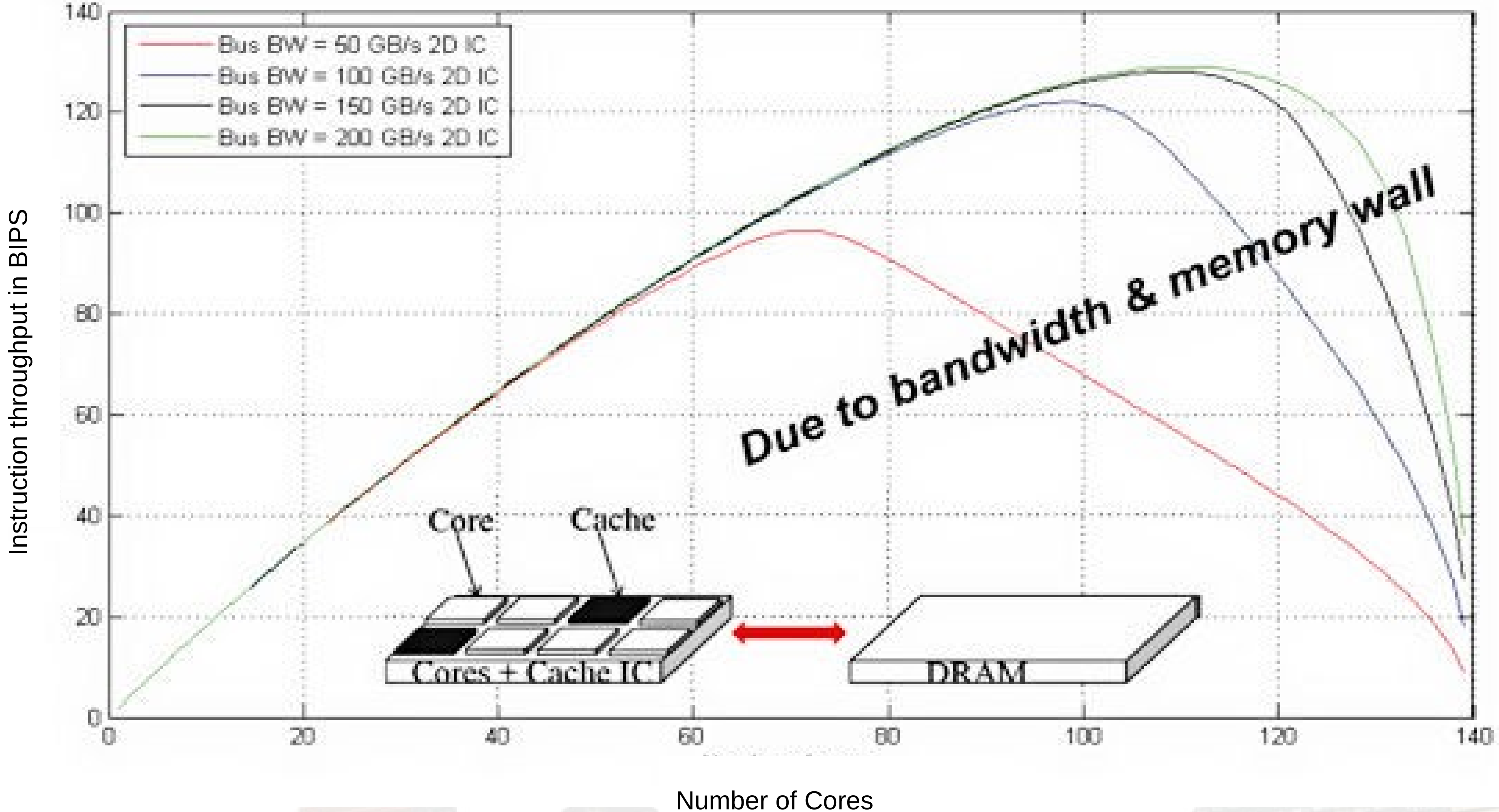
Data at rest
Terabytes to zettabytes of data to process
Volume
Data in many forms
Structured, unstructured, and semi-structured
Variety
Data in motion
Streaming data, microseconds to seconds to respond
Velocity
Data in doubt
Uncertainty due to data inconsistency, ambiguities, deception, and model approximations
Veracity

# Types: Memory Storage



# Memory Wall

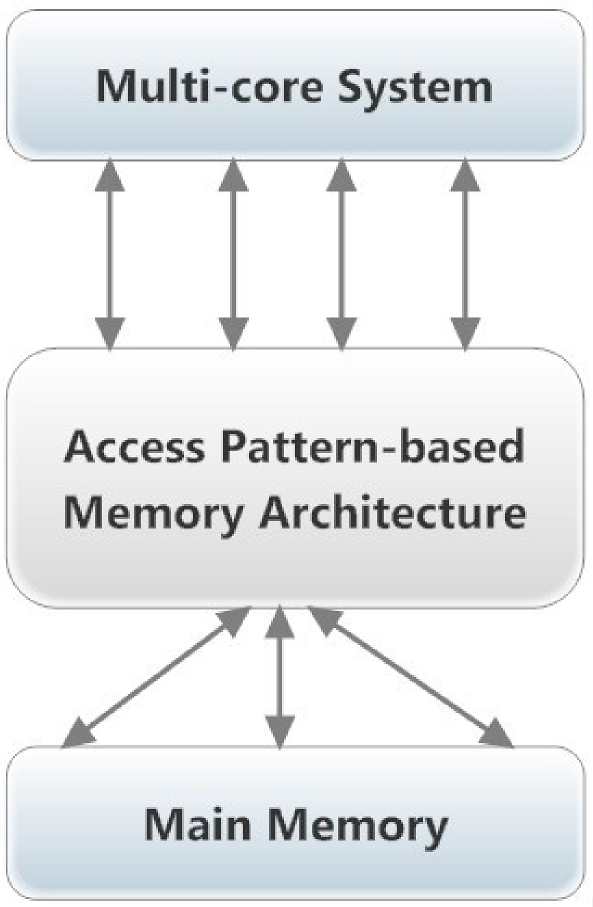
Bandwidth wall with increase in the number of cores [2]




**[2] Muhammad Bakir; Georgia Tech**

# *Focus to improve processor/memory performance*

- Multi-core system
  - RISC
  - Vector & hardware accelerator cores
- Access Pattern-based Data Architecture
  - Irregular/complex access patterns



# Key Considerations for Application Development

- I/O Architecture
- Data Architecture
- **Hardware Architecture**
- Software Architecture

# Processors for Supercomputers

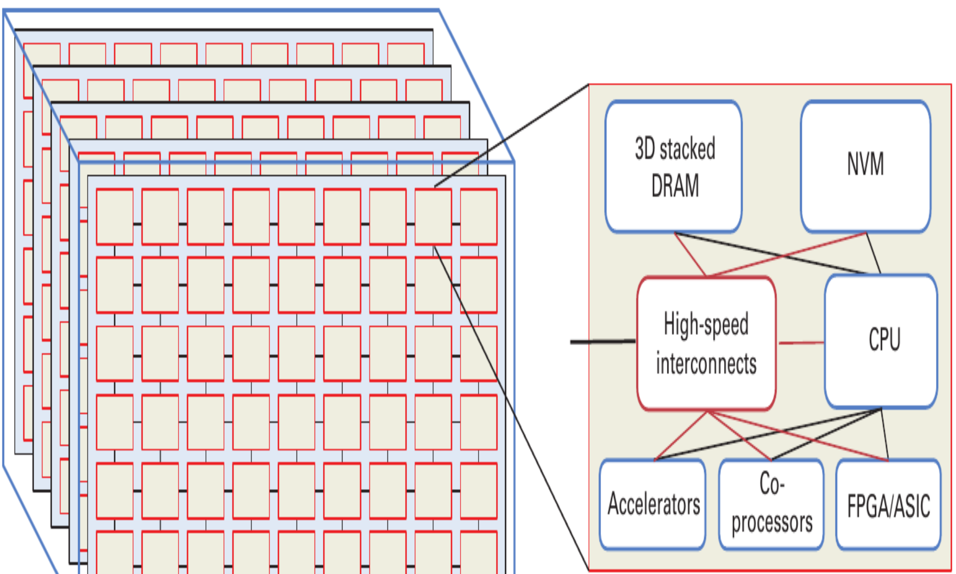
**Microprocessor development directions:**

- –Increasing of clock frequency and speed instruction stream processing
- –Processing of large collection of data in single processor instruction - SIMD
- –Control path multiplication – multi threading

- **RISC processors**
  - MIPS
  - IBM Power4
- **Pipeline Processor**
- **AlphaVector processors**
  - NEC SX-6
  - Cray (Cray X1)
- **CISC processors**
  - IA32
  - AMD x86-64
- **VLIW processors**
  - IA64
- **Multi-core Processor**
- **GPU**
- **FPGA**
  - SRAM Based

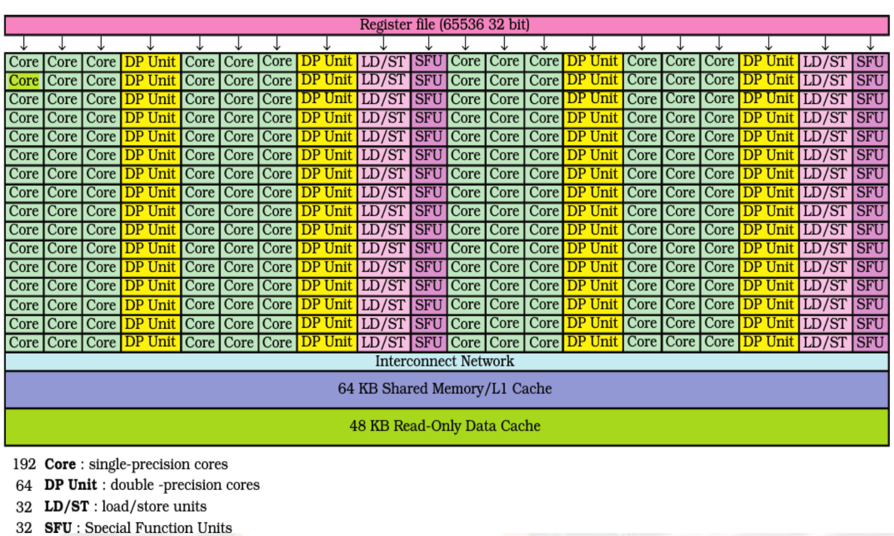
# Parallel Computing: Resources

- The compute resources can include:
  - A single computer with multiple processors;
  - A single computer with (multiple) processor(s) and some specialized computer resources (GPU, FPGA …)
  - An arbitrary number of computers connected by a network;
  - A combination of both.

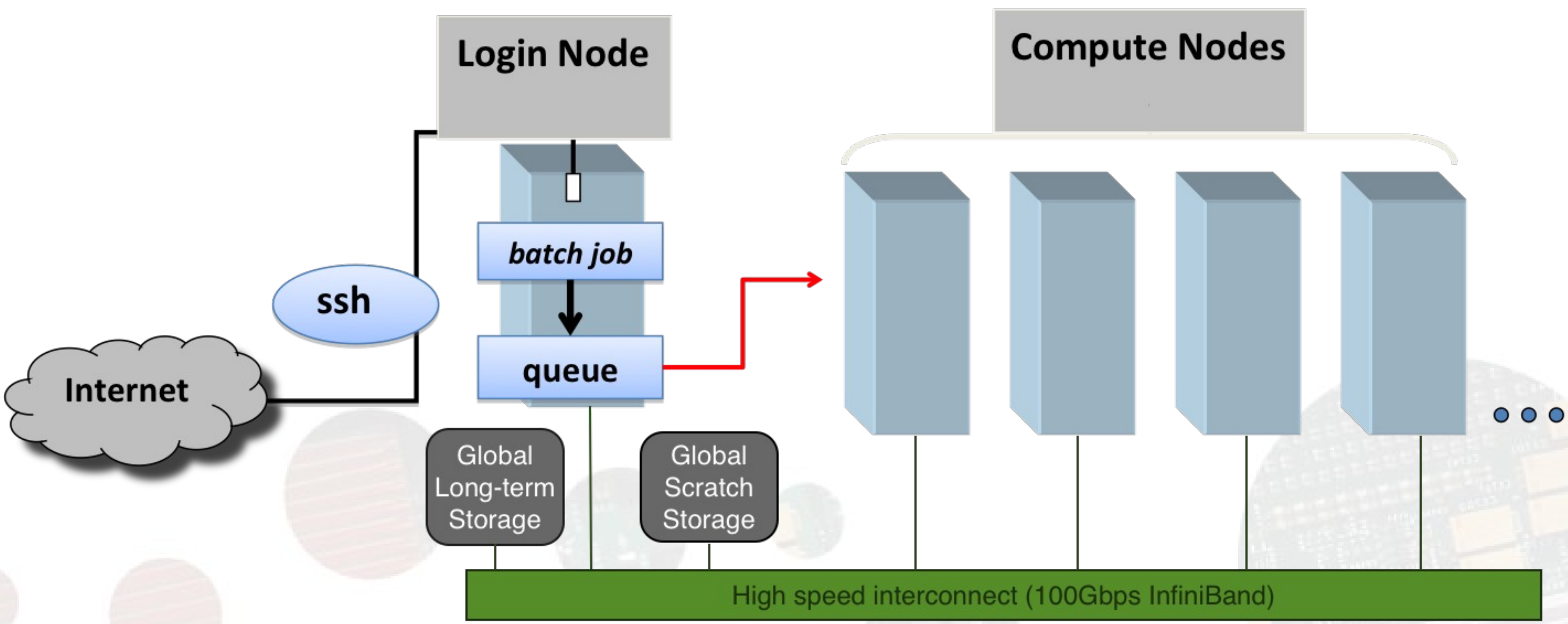


Terabit network connected system

Network-centric architecture



# Multi-Node based Architecture



# Key Considerations for Application Development

- I/O Architecture
- Data Architecture
- Hardware Architecture
- **Software Architecture**

# Types Parallel Processing?

**Instruction Level Parallelism**

- Pipelining and Superscalar Execution

**Data Level Parallelism**

- Vector Instruction or Specilizaed Accelerator

**Thread Level Parallelism**

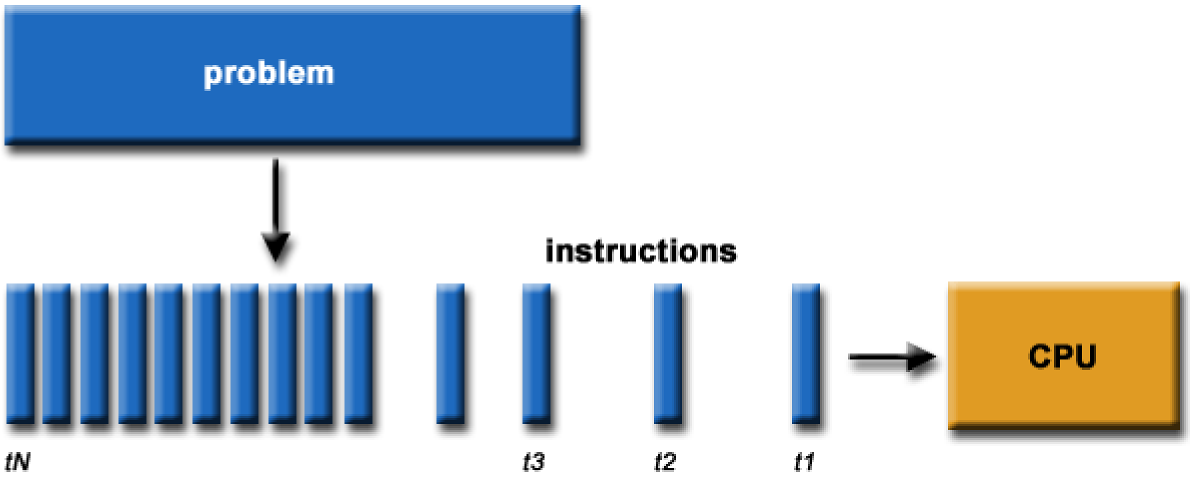
- Execute multiple function on different cores

**Task Level Parallelism**

- Breaking multiple tasks (Memory, Input Output etc) into subtasks and execute using TLP

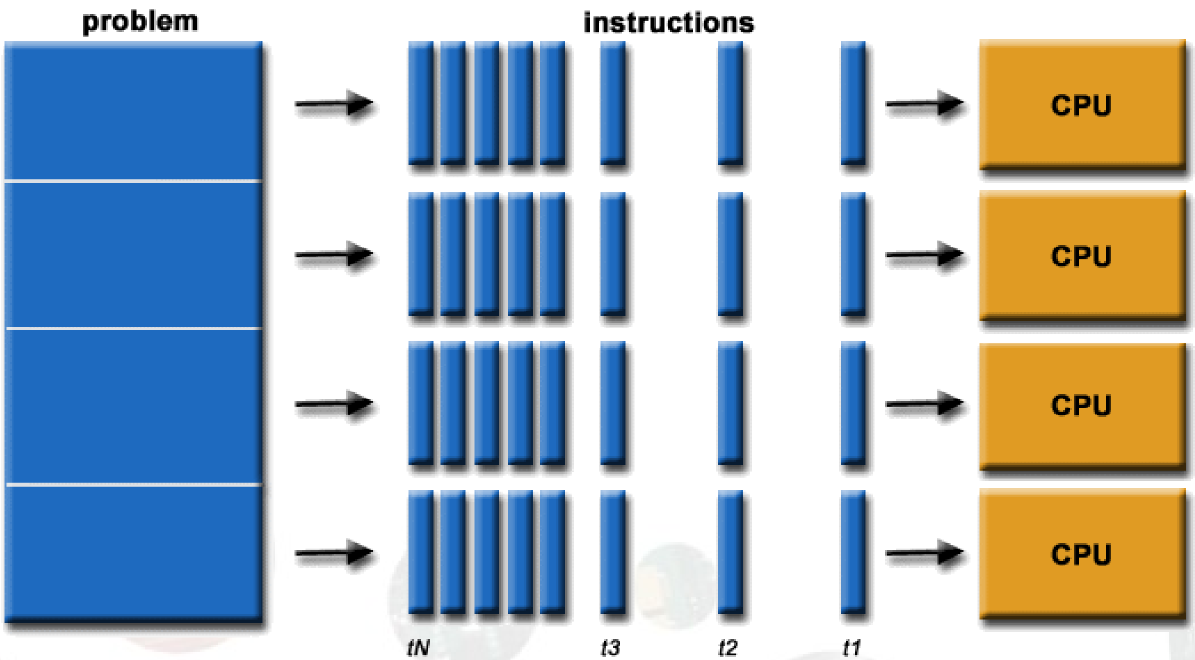
# Parallel Programming ?

- Traditionally, software has been written for ***serial*** computation:
  - To be run on a single computer having a single Central Processing Unit (CPU);
  - A problem is broken into a discrete series of instructions.
  - Instructions are executed one after another.
  - Only one instruction may execute at any moment in time.



# Parallel Computing

- In the simplest sense, ***parallel computing*** is the simultaneous use of multiple compute resources to solve a computational problem.
  - To be run using multiple CPUs
  - A problem is broken into discrete parts that can be solved concurrently
  - Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different CPUs



# Writing Parallel Application

**A=10;**

**B=20;**

C=A+B;

D=A*B;

E=C*A;

F=C*B;

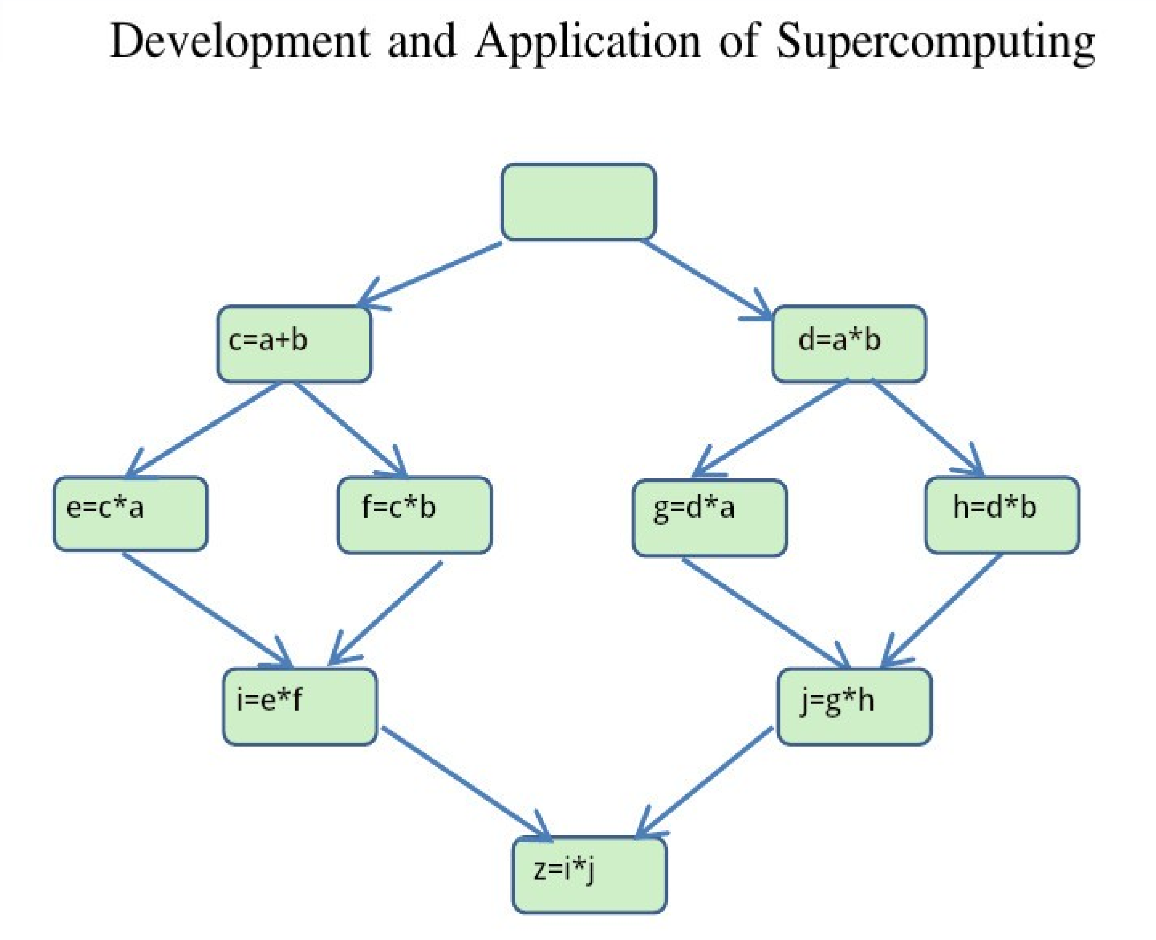
G=D*A;

H=D*B;

I=E*F;

J=G*H;

**Z=I*J;**



# Program Execution Model:

Process

Memory Management

Concurrency and Synchronization

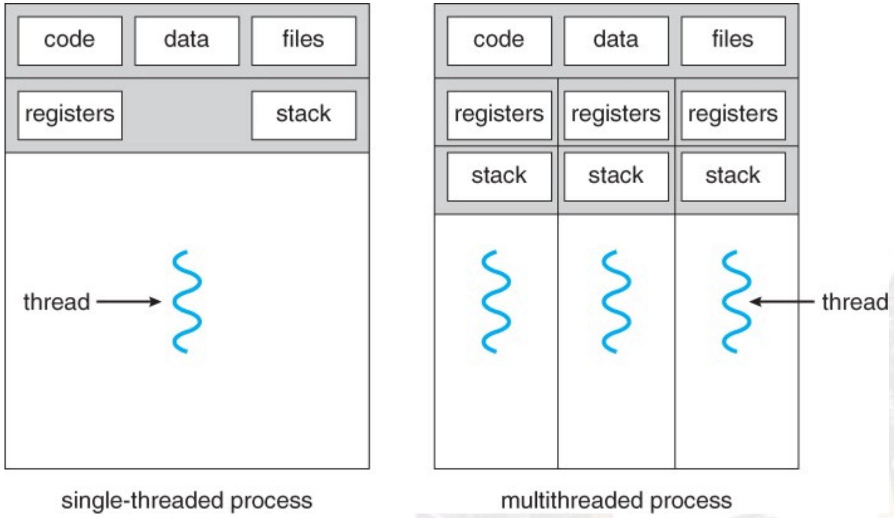
Inter-Process Communication (IPC)

# Processing

Process: Start by understanding that a program's execution begins as a process, which represents an instance of the program running on the operating system.

Thread: Threads allow concurrent execution of tasks within the same process.

Task: Define specific functions of work or tasks within the program that can be executed independently to leverage parallelism effectively.



# Memory Management

Memory Management:

Heap: Utilize the heap for dynamically allocating memory required by data structures or objects that are shared among threads or tasks.

Stack: Each thread has its own stack for managing local variables, function call information, and return addresses.

# Concurrency and Synchronization

Scheduler: Understand how the scheduler manages the execution of threads or tasks on the CPU, considering factors such as priority and time slicing.

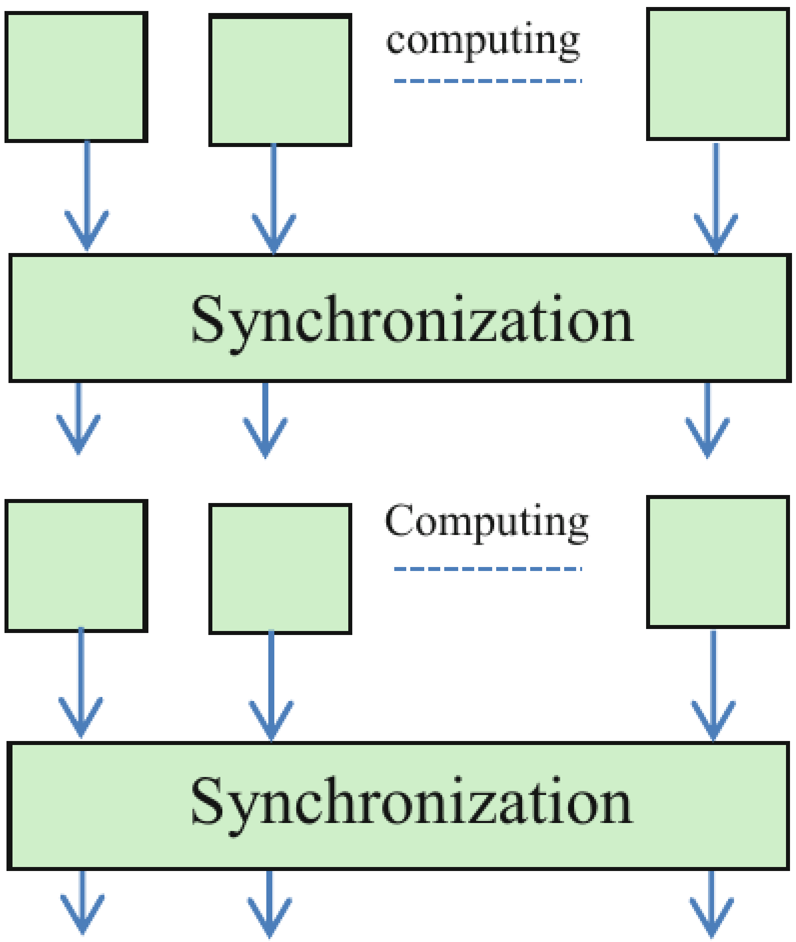
Synchronization: Implement synchronization mechanisms (e.g., mutexes, semaphores) to coordinate access to shared resources and ensure data integrity in concurrent execution.

# Inter-Process Communication (IPC):

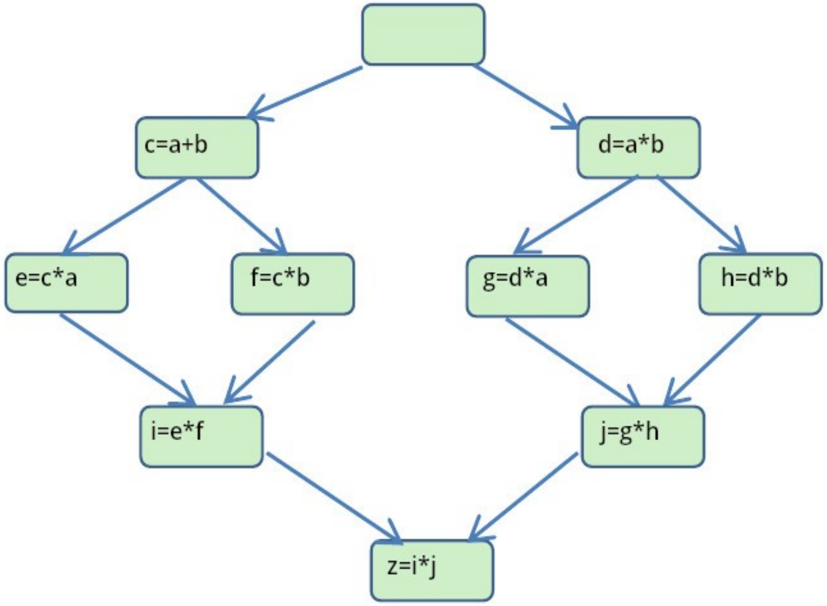
IPC mechanisms (e.g., pipes, shared memory, message queues) for communication and data exchange between processes or threads.

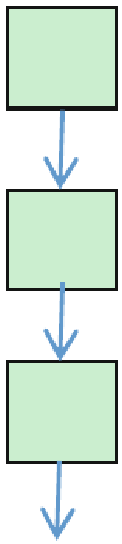
# Considerations for Parallel Programs

- Understand the Problem and the Program
  - Data Dependencies
  - Partitioning (Operations)
  - Communications (Distribution)
  - Synchronization (
  - Load Balancing
  - Granularity (Bit, Task, Thread)
  - Limits and Costs of Parallel Programming
- Automatic vs. Manual Parallelization
- Performance Analysis and Tuning

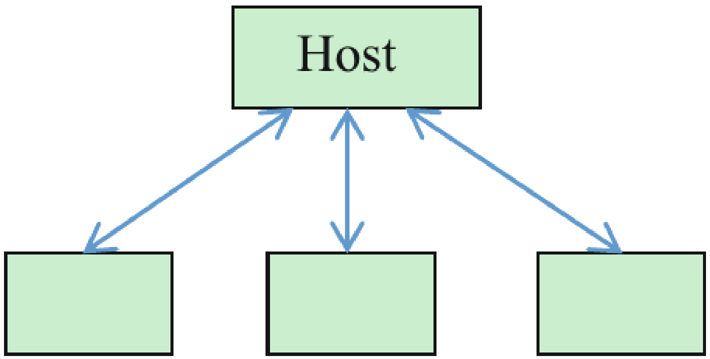



Development and Application of Supercomputing

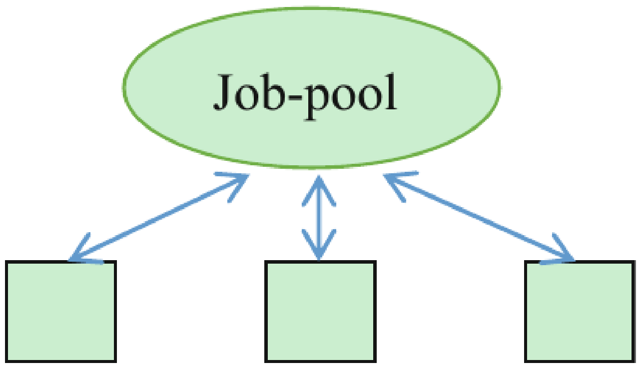




macro-pipeline

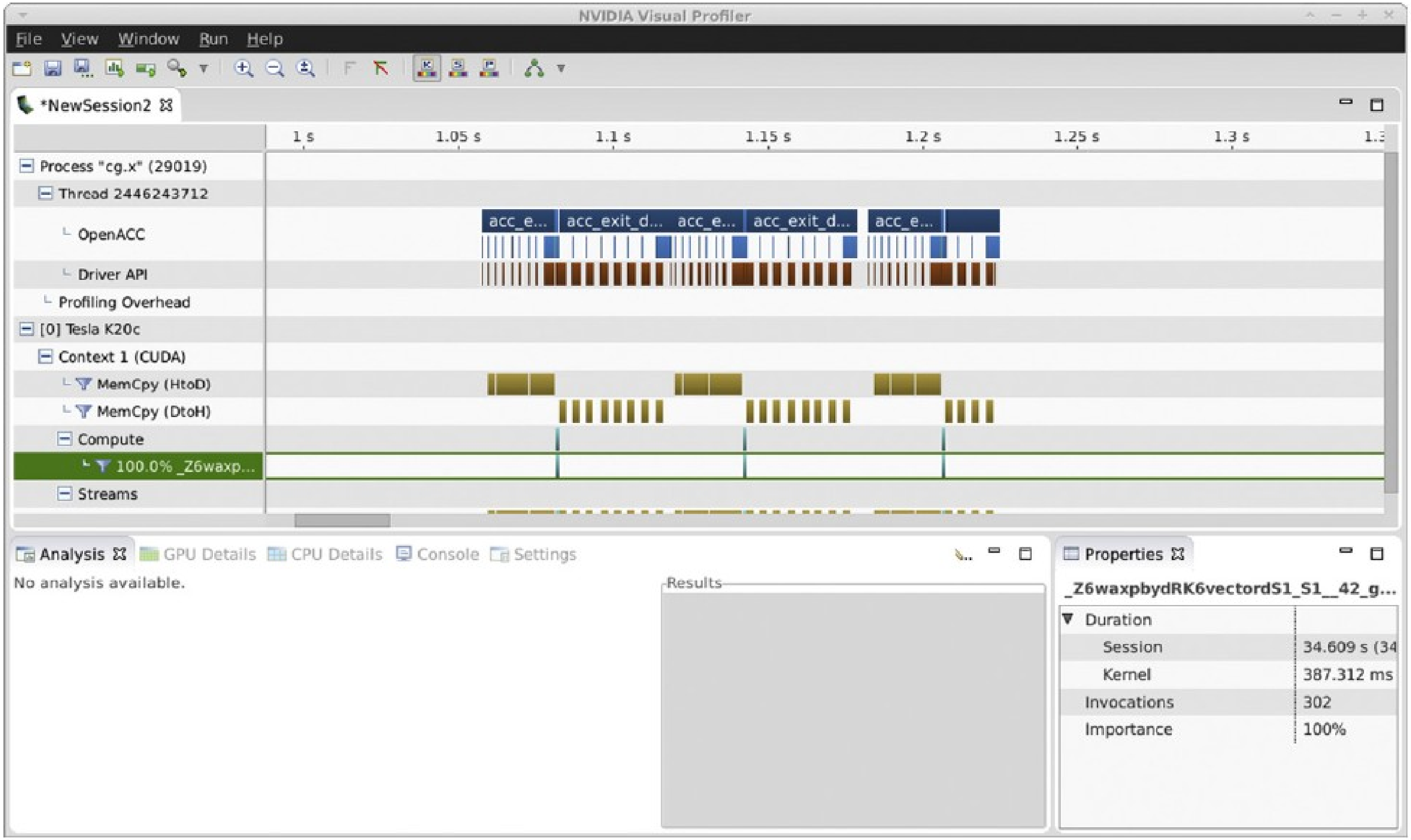



master-slave




job-pool

# Performance Analysis And Tuning



# Example

Improve Phase Unwrapping Algorithm

Scientific Pseudocode

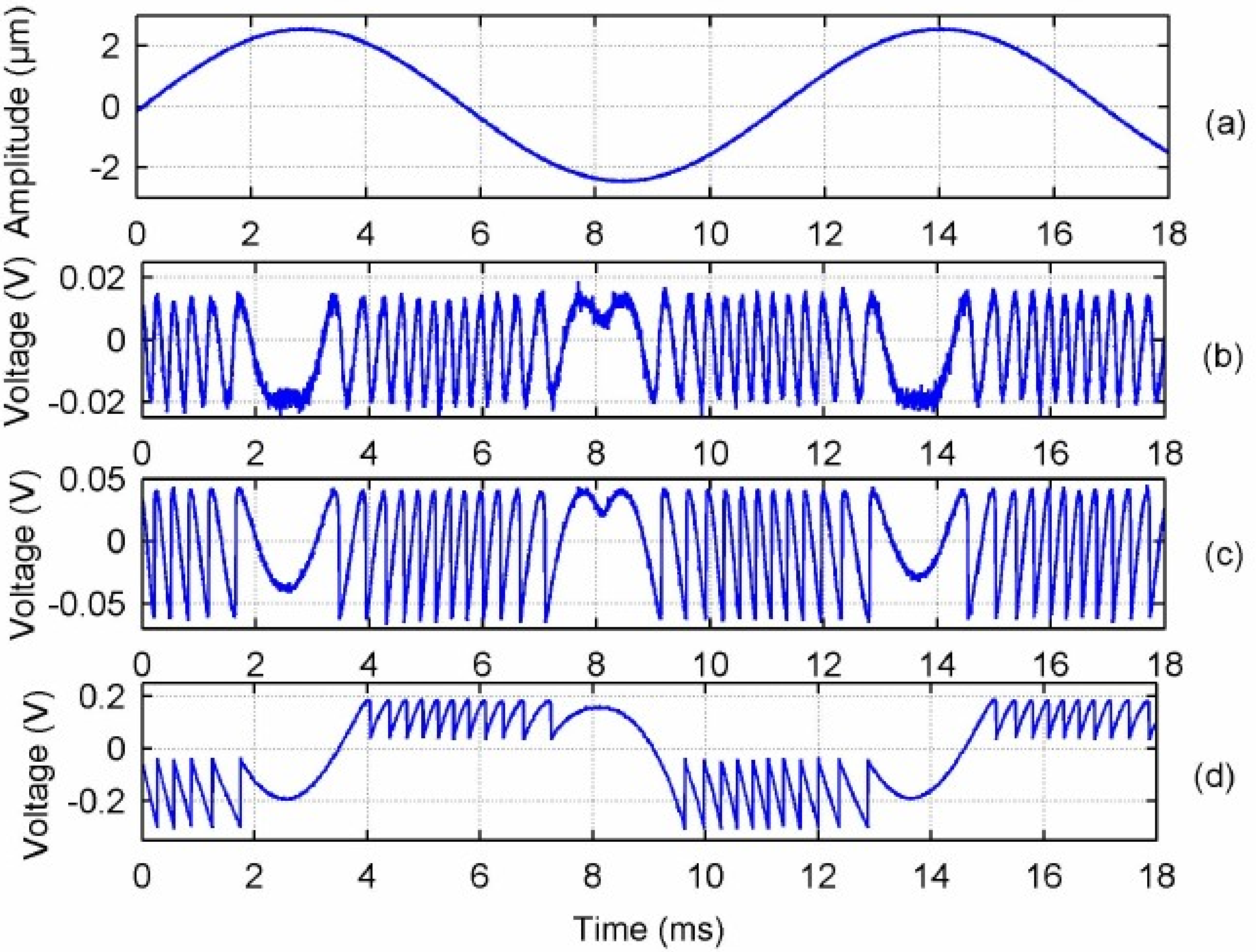
Mathematical Pseudocode

Control Data Flow Graph

Coding

Execution

# Understanding Problem: IPUM



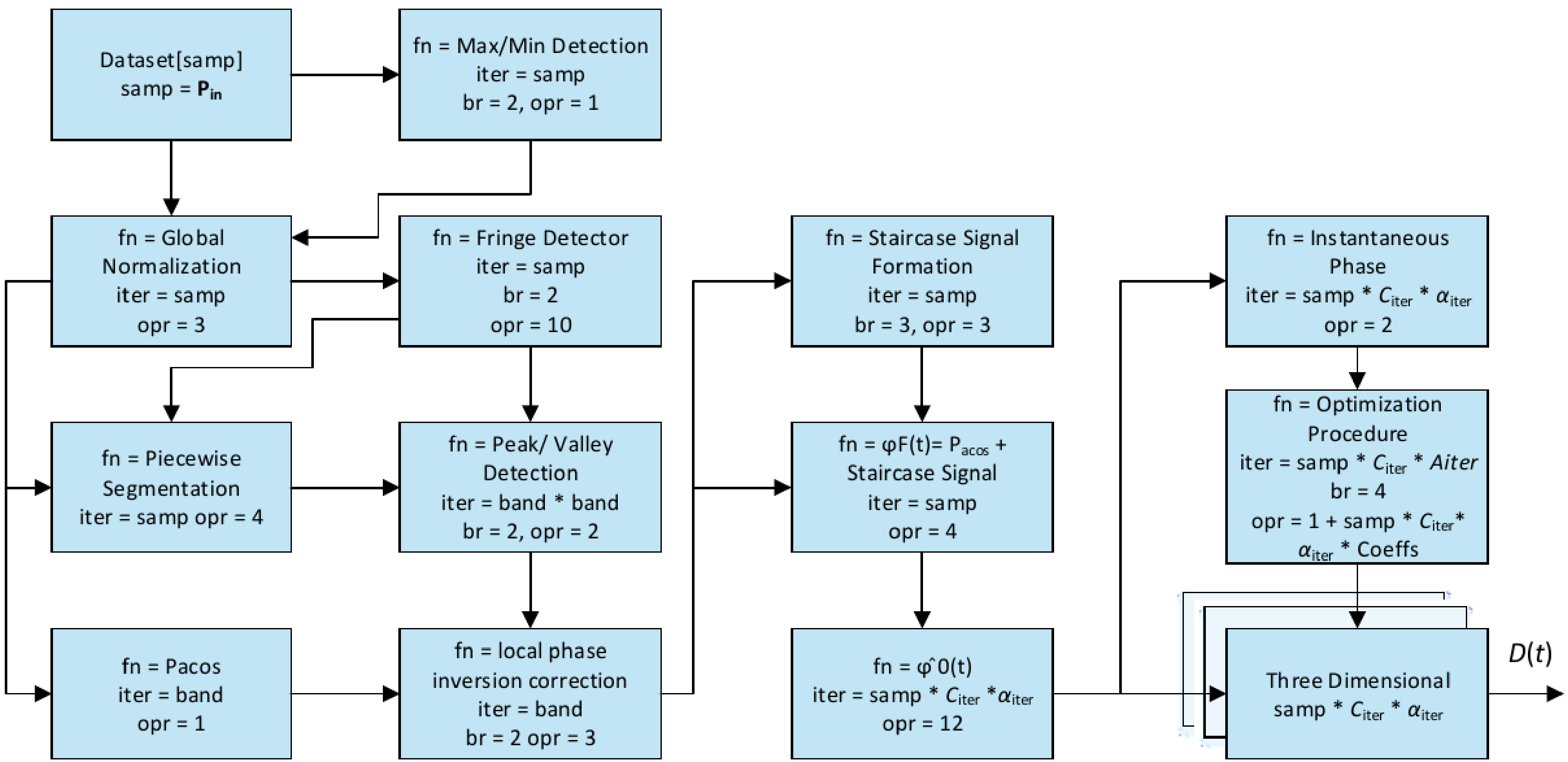
```
Begin
Input: P_in
Constant: FIR[coeffs]
T1: For j ← 0: i-1
    P[j] ← P_in
End T1
P_max, P_min ← 0
T2: For j ← 0: i-1
    If P[j] > P_max
        P_max ← P[j]
    Else If P[j] < P_min
        P_min ← P[j]
    End If
End T2
T3: For j ← 0: i-1
    P_norm[j] ← 2 * [(P[j]−P_min)/(P_max−P_min)] − 1
End T3
T4: For j ← 0: i-1
    φ_Fmodπ[j] ← arcos (P_norm[j])
End T4
T5: For j ← 1: i-1
    P_diff[j] ← P_norm[j] −P_norm[j-1]
End T5
Constant: th_pos ← A, th_neg ← B
Int K=0;
T6: For j ← 0: i-1
    IF P_diff [j] < th_neg
        Fringe_val[k] ← -1
        Fringe_loc[k] ← j
        k ← k+1
    ELSE IF P_diff [j] > th_pos
        Fringe_val[k] ← 1
        Fringe_loc[k] ← j
        k ← k+1
    End If
End T6
Constant: band ← C,
T7: For j ← 0: k
    Fringe_amp[j] ← 0;
    T7-1: For m ←Fringe_loc[j]– band : Fringe_loc[j] + band
        IF P[m] > Fringe_amp[j]
            Fringe_amp[j] ← P[m]
                Peak_loc[j] ← m
        End If
    End T7-1
    Fringe_amp[j] = 1;
    T7-2: For m ←Fringe_loc[j]– band :Fringe_loc[j] + band
        IF P[m] < Fringe_amp[j]
            Fringe_amp[j] ← P[m]
                Valley_loc[j] ← m
        End If
    End T7-2
End T7
T8: For j ← 0: k
    IF Fringe_val[j] ← -1
        T8-1: For m ← Valley_loc[j] : Peak_loc[j]
            φ_Fmodπ[m] ← arcos(-1* P[m])
        End T8-1
    Else IF Fringe_val[j] ← 1
        T8-2: For m ← Peak_loc[j] : Valley_loc[j]
            φ_Fmodπ[m] ← arcos(-1* P[m])
        End T8-2
    End IF
End T8
P_staircase[0] ← 0
T9: For j ← 1: i-1
    IF (φ_Fmodπ[j] - φ_Fmodπ[j-1]) > π/2 )
        P_staircase[j] ← P_staircase[j-1] - π
    Else IF (φ_Fmodπ[j] - φ_Fmodπ[j-1] < - π/2)
        P_staircase[j] ← P_staircase[j-1] + π
    Else
        P_staircase[j] ← P_staircase[j-1]
    End IF
End T9
T10: For j ← 1: i-1
    Φ̂_F[j] ← P_staircase[j] + φ_Fmodπ[j]
End T10
C_val ← C_start, α_val ← α_start
T11: Loop C_ind ← 0: i1-1
    T11-1: Loop α_ind ← 0: i2-1
        T11-1-1: Loop j ← 0: i-1
            Φ̂_0[j][C_ind][α_ind] ← Φ̂_F[j] + C_val * sin (Φ̂_F[j] + arctan(α_val))
        End T11-1-1
        T11-1-2: Loop j ← 1: i-1
            Φ̂_0_diff[j] ← Φ̂_0[j][C_ind][α_ind] − Φ̂_0[j − 1][C_ind][α_ind]
        End T11-1-2
        J[C_ind][α_ind] ← 0
        T11-1-3: Loop j ← 0: i-1
            IF j < i - coeffs+1      % filtering
                Accum = 0;
                T11-1-3-1: For f ← 0: coeffs
                    Accum = Accum + FIR[f] * Φ̂_0_diff[j + f]
                End T11-1-3-1
                Φ̂_0_diff[j] ← Accum
            Else
                Φ̂_0_diff[j] ← 0
            End IF
            J[C_ind][α_ind] ← J[C_ind][α_ind] + rms{ Φ̂_0diff[j]}
        End T11-1-3
        α_val ← α_val + α_step
    End T11-1
    C_val ← C_val + C_step
End T11
J_min ← P_max, C_opt ← 0 , α_opt ← 0
T12: Loop C_ind ← 0: i1-1
    T12-1: Loop2 α_ind ← 0: i2-1
        IF J[C_ind][α_ind] < J_min
            J_min ← J[C_ind][α_ind]
            C_opt ← C_ind*C_step,   α_opt ← α_ind*α_step
        End IF
    End T12-1
End T12
T13: For j ← 0: i-1
Output:     D[j] = (λ_0 / 4π) * Φ̂_0[j][C_opt][α_opt]
End T13
END
```

Scientific Pseudo Code

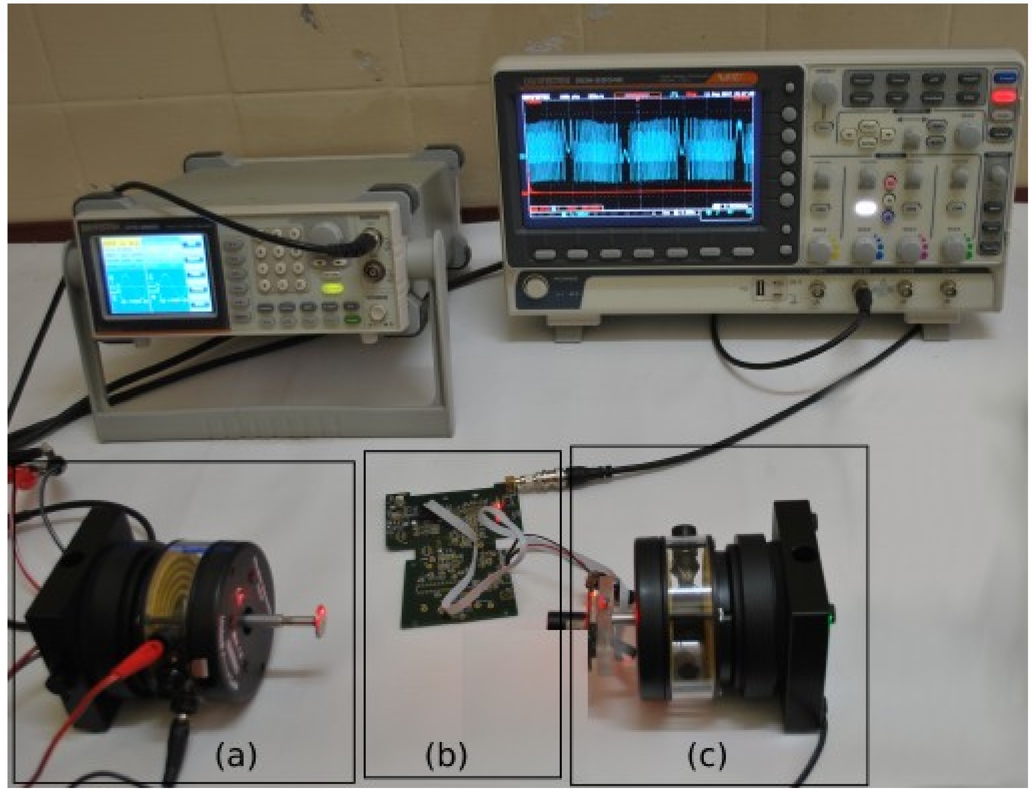
```
Begin
Input: P_in
Constant: FIR[coeffs]
P_max, P_min ← 0
T1: For j ← 0: i-1
    P[j] ← P_in
     If P[j] > P_max
          P_max ← P[j]
      Else If P[j] < P_min
           P_min ← P[j]
      End If
End T1
Constant th_pos ← A, th_neg ← B, band ← C
Int k=0, P_staircase[0] ← 0
T2: For j ← 0: i-1
     P_norm[j] ← 2 * [(P[n] − P_min)/(P_max − P_min)] − 1
     φ_Fmodπ[j] ← arcos(P_norm[j])
     If n>0 then
          P_diff[j] ← P_norm[j] – P_norm[j-1]
          IF P_diff [j] < th_neg
                 Fringe_val[k] ← -1
                 Fringe_loc[k] ← j
          Else IF P_diff [j] > th_pos
                 Fringe_val[k] ← 1
                 Fringe_loc[k] ← j
          End If
          Fringe_amp[k] ← 0;
          T2-1: For m ← Fringe_loc[k]–band:Fringe_loc[k] + band
                 IF P[m] > Fringe_amp[k]
                        Fringe_amp[k] ← P[m]
                        Peak_loc[k] ← m
                 End If
          End T2-1
            Fringe_amp[k] = 1;
          T2-2: For m ← Fringe_loc[k]–band:Fringe_loc[k] + band
                  IF P[m] < Fringe_amp[k]
                        Fringe_amp[k] ← P[m]
                        Valley_loc[k] ← m
                 End If
          End T2-2
          IF Fringe_val[k] ← -1
                 T2-3: For m ← Valley_loc[j] : Peak_loc[j]
                              φ_Fmodπ[m] ← arcos(-1* P[m])
                 End T2-3
           Else IF Fringe_val[k] ← 1
                 T2-4: For m ← Peak_loc[j] : Valley_loc[j]
                              φ_Fmodπ[m] ← arcos(-1* P[m])
                       End T2-4
              End IF
               K ← k+1
              IF (φ_Fmodπ[j] - φ_Fmodπ[j-1]) > π/2 )
                     P_staircase[j] ← P_staircase[j-1] - π
              Else IF (φ_Fmodπ[j] - φ_Fmodπ[j-1] < - π/2)
                     P_staircase[j] ← P_staircase[j-1] + π
              Else
                     P_staircase[j] ← P_staircase[j-1]
             End IF
             P_stair[j] ← 2π*Fringes[j] + P_stair[j-1]
             Φ̂_F[j] ← P_stair[j] + φ_Fmodπ[j]
        End If
End T2
C_val ← C_start, α_val ← α_start, J_mii ← P_max, C_opt ← 0 , α_opt ← 0
T3: Loop C_ind ← 0: i1-1
     T3-1: Loop α_ind ← 0: i2-1
      J[C_ind][α_ind]   ← 0
         T3-1-1: Loop j ← 0: i-1
     Φ̂_0[j][C_ind][α_ind] ← Φ̂_F[j] + C_val * sin(Φ̂_F[j] + arctan(α_val))
          If n >0 then
            Φ̂_0_diff[j] ←   Φ̂_0[j][C_ind][α_ind] −  Φ̂_0[j − 1][C_ind][α_ind]
                IF j < i - coeffs+1     % filtering
                    Accum = 0;
                    T3-1-1-1: For f ← 0: coeffs
                      Accum = Accum + FIR[f] * Φ̂_0_diff[j + f]
                   End LT3-1-1-1
                  Φ̂_0_diff[j] ←  Accum
                Else
                  Φ̂_0_diff[j] ←  0
                End IF
            J[C_ind][α_ind] ← J[C_ind][α_ind] + rms{Φ̂_0_diff[j]}}
          End If
        End T3-1-1
        IF J[C_ind][α_ind] < J_min then
          J_min   ← J[C_ind][α_ind]
         C_opt ← C_ind*C_step,   α_opt ← α_ind*α_step
       End IF
       α_val ← α_val + α_step
    End T3-1
  C_val ← C_val + C_step
End T3
T4: For j ← 0: i-1
Output:              D[j] = (λ_0/4π) * Φ̂_0[j][C_opt][α_opt]
End T4
END
```

Metathetical Pseudo Code

# Control Data Flow Graph



# Performance Results: BSC PowerCTE




PARALLEL-HYBRID IPUM ALGORITHM SCALABLITY: EXECUTION TIME AGAINST DIFFERENT NUMBER OF DISTRIBUTED NODES FOR $v_t(t) = 100x10^{-3}\ meter/sec$, INPUT SAMPLES = $4\times10^6$

| Nodes | 1 | 2 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|---|
| Execution Time (Sec) | 22.1 | 11.9 | 6.38 | 3.21 | 1.66 | 0.98 |

# Revese Time Migration Kernel

- Sequential application program and converts into parallel program.
- Understand Algorithm/Application data access, data structure, data dependencies and CFG.

$$for\ (y = stencil;\ y < NY - stencil;\ y++)$$

$$for\ (x = stencil;\ x < NX - stencil;\ x++)$$

$$for\ (z = stencil;\ z < NZ - stencil;\ z++)$$

$$P_3(x,y,z) = \sum_{l}^{s} w_l^1\ [P_2(x-l,y,z) + P_2(x+l,y,z)]$$

$$+\sum_{l}^{s} w_l^2\ [P_2(x,y-l,z) + P_2(x,y+l,z)]$$

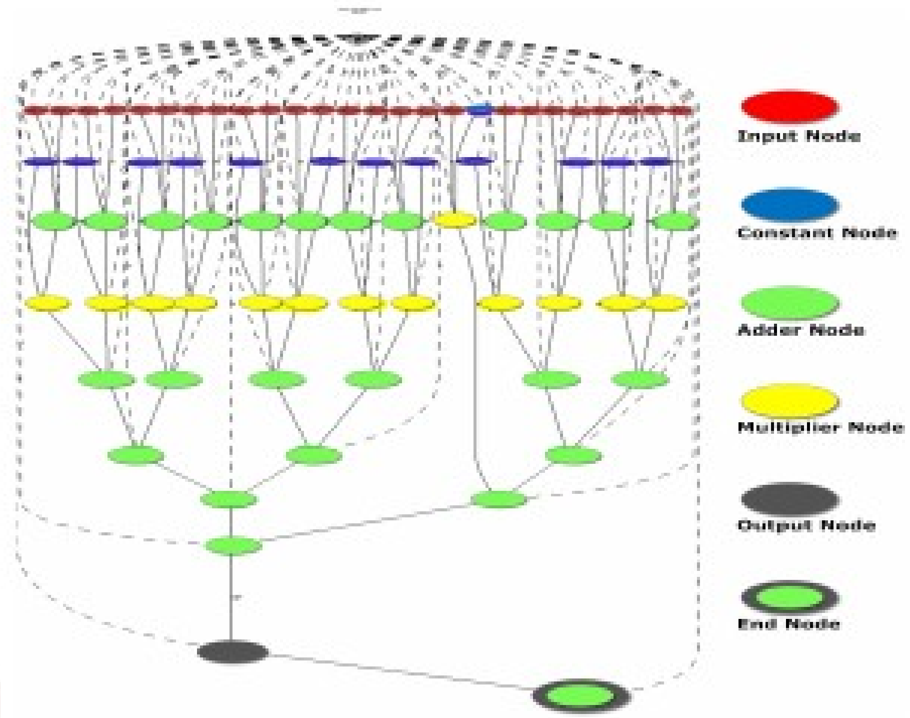
$$+\sum_{l}^{s} w_l^3\ [P_2(x,y,z-l) + P_2(x,y,z+l)] + c^{\circ} P_2(x,y,z))$$

$$+(V(x,y,z) \times dt)^2 + (2 \times P_2(x,y,z)) - P_1(x,y,z)$$

Mathematical Model

# RTM

- Sequential application program and converts into parallel program.
- Understand Algorithm/Application data access, data structure, data dependencies and CFG.




CDFG

# RTM

- ➔ Sequential application program and converts into parallel program.
- ➔ Understand Algorithm/Application data access, data structure, data dependencies and CFG.

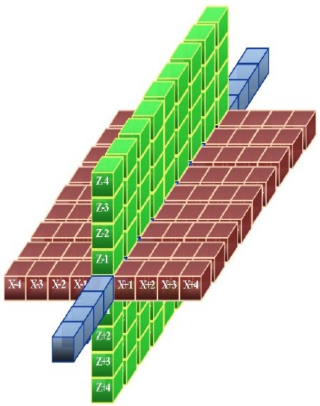
```
#define MX 64
#define MY 64
#define MZ 64
for ( k = Stencil ; k < MY - Stencil ; k++ )
for ( j = Stencil ; j < MZ - Stencil ; j++ )
for ( i = Stencil ; i < MX - Stencil ; i++ )
{
iter = k*(MX*MZ) + (j*MX) + i;
tmp =
Y1*(P2_linear[i+j*iter_j+(k-1)*iter_k] + P2_linear[i+j*iter_j+(k+1)*iter_k]) +
Y2*(P2_linear[i+j*iter_j+(k-2)*iter_k] + P2_linear[i+j*iter_j+(k+2)*iter_k]) +
Y3*(P2_linear[i+j*iter_j+(k-3)*iter_k] + P2_linear[i+j*iter_j+(k+3)*iter_k]) +
Y4*(P2_linear[i+j*iter_j+(k-4)*iter_k] + P2_linear[i+j*iter_j+(k+4)*iter_k]) +
c00 * P2_linear[iter] +
X4*(P2_linear[i+(j-4)*iter_j+k*iter_k] + P2_linear[i+(j+4)*iter_j+k*iter_k]) +
X3*(P2_linear[i+(j-3)*iter_j+k*iter_k] + P2_linear[i+(j+3)*iter_j+k*iter_k]) +
X2*(P2_linear[i+(j-2)*iter_j+k*iter_k] + P2_linear[i+(j+2)*iter_j+k*iter_k]) +
X1*(P2_linear[i+(j-1)*iter_j+k*iter_k] + P2_linear[i+(j+1)*iter_j+k*iter_k]) +
Z4*(P2_linear[(i-4)+j*iter_j+k*iter_k] + P2_linear[(i+4)+j*iter_j+k*iter_k]) +
Z3*(P2_linear[(i-3)+j*iter_j+k*iter_k] + P2_linear[(i+3)+j*iter_j+k*iter_k]) +
Z2*(P2_linear[(i-2)+j*iter_j+k*iter_k] + P2_linear[(i+2)+j*iter_j+k*iter_k]) +
Z1*(P2_linear[(i-1)+j*iter_j+k*iter_k] + P2_linear[(i+1)+j*iter_j+k*iter_k]);
P3_linear[iter] = tmp ;
}
```

C/C++ Program

# Programing Example: 3D-Stencil

```
// Stencil Structure
#define Sten_size 4
// 128x128x128 Main Memory Data Set
#define WIDTH 128
#define HEIGHT 128
#define BANK 128
main () {
int X,Y,Z;
X = HEIGHT;
Y = WIDTH*HEIGHT;
Z =0;
float Sten[WIDTH*HEIGHT*BANK];
for ( k = Stencil_size ; k < BANK - Sten_size ; k++ )
  for ( j = Stencil_size ; j < HEIGHT - Sten_size ; j++ )
    for ( i = Stencil_size ; i < WIDTH - Sten_size ; i++ )
{
   Z = k*(WIDTH*HEIGHT) + (j*WIDTH) + i;
   Sten[i+j*X+(k-1)*Y] + Sten[i+j*X+(k+1)*Y] +
   Sten[i+j*X+(k-2)*Y] + Sten[i+j*X+(k+2)*Y] +
   Sten[i+j*X+(k-3)*Y] + Sten[i+j*X+(k+3)*Y] +
   Sten[i+j*X+(k-4)*Y] + Sten[i+j*X+(k+4)*Y] +
   Sten[Z] +
   Sten[i+(j-4)*X+k*Y] + Sten[i+(j+4)*X+k*Y] +
   Sten[i+(j-3)*X+k*Y] + Sten[i+(j+3)*X+k*Y] +
   Sten[i+(j-2)*X+k*Y] + Sten[i+(j+2)*X+k*Y] +
   Sten[i+(j-1)*X+k*Y] + Sten[i+(j+1)*X+k*Y] +
   Sten[(i-4)+j*X+k*Y] + Sten[(i+4)+j*X+k*Y] +
   Sten[(i-3)+j*X+k*Y] + Sten[(i+3)+j*X+k*Y] +
   Sten[(i-2)+j*X+k*Y] + Sten[(i+2)+j*X+k*Y] +
   Sten[(i-1)+j*X+k*Y] + Sten[(i+1)+j*X+k*Y];}
}
```

Conventional 3D stencil access

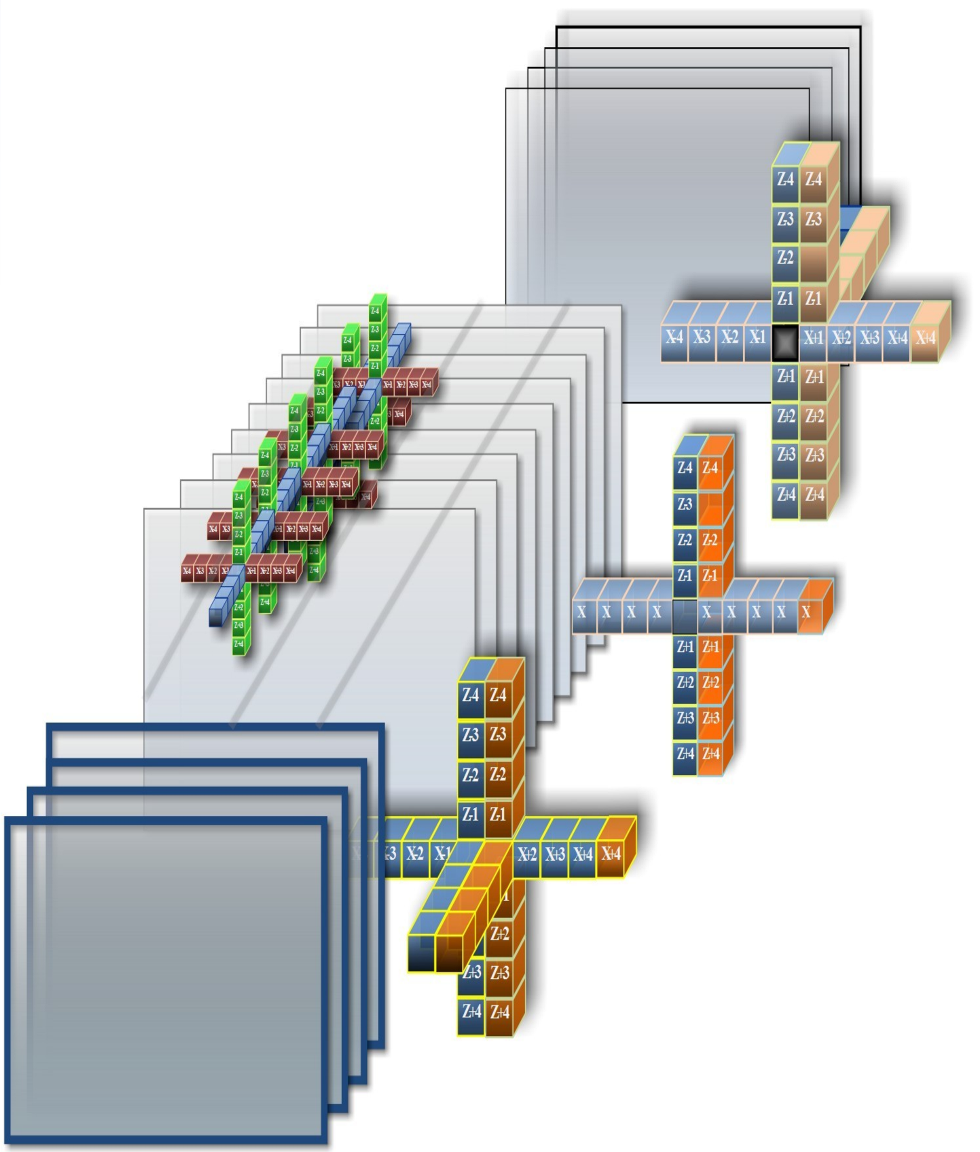



```
 #define stencil_size 4
 #define PRIORITY1 1
 #define PRIORITY2 2
// Main Program
   PMC_SCRATCHPAD STENCIL;
   PMC_SCRATCHPAD SSM_3D;
   MAIN_MEMORY DATASET_3D;
// Part I : Local SSM
// Single Stencil Buffer
   STENCIL.ADDRESS=0X10000000;
   STENCIL.WIDTH=9;
   STENCIL.HEIGHT=3;
   STENCIL.BANK=1;
// 3D 32x32x32 SSM
   SSM_3D.ADDRESS=0X11000000;
   SSM_3D.WIDTH=32;
   SSM_3D.HEIGHT=32;
   SSM_3D.BANK=32;
// Part II : Main Memory
// 3D-Data set
   DATASET_3D.ADDRESS=0X00100000;
   DATASET_3D.WIDTH=128;
   DATASET_3D.HEIGHT=128;
   DATASET_3D.BANK=128;
//PART III : DATA TRANSFER

3D_STENCIL (STENCIL, DATASET_3D, PRIORITY1);

3D_STENCIL_VECTOR (SSM_3D, DATASET_3D, PRIORITY2);
```

PMC 3D stencil access

# 3D Memory Architecture

- Noncontiguous data access to contiguous formate
- Plane size = Nx * Nz
- Number of parallel ports = Ny * 2



Center of Excellence: Supercomputing for AI & Big-Data

# Art of Parallel Programming: Think Parallel


**by: Tassadaq Hussain**

**Director Centre for AI and BigData**

**Professor Electrical Engiennering Department**

**Namal University Mianwali**


**Collaborations:**

**Barcelona Supercomputing Center, Spain**

**European Network on High Performance and Embedded Architecture and Compilation**

**Pakistan Supercomputing Center**