Independent Component Analysis (ICA)

A parallel approach

Motivation

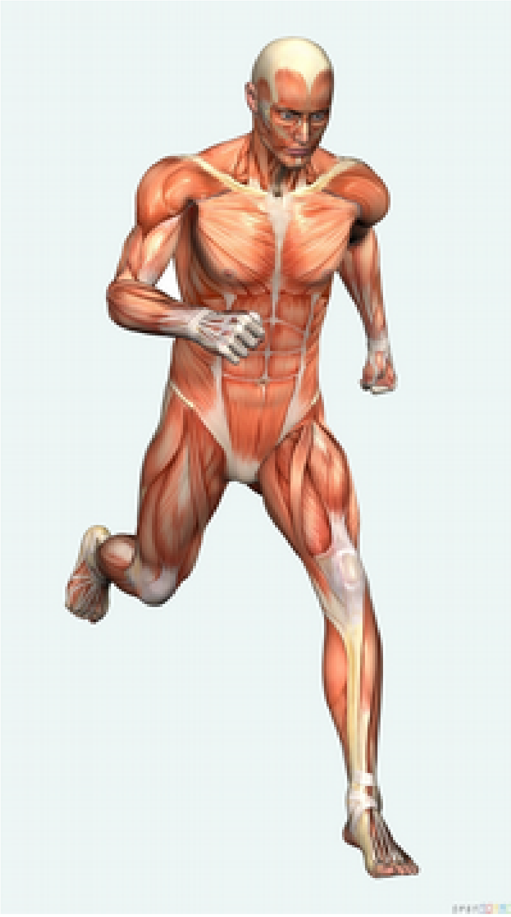
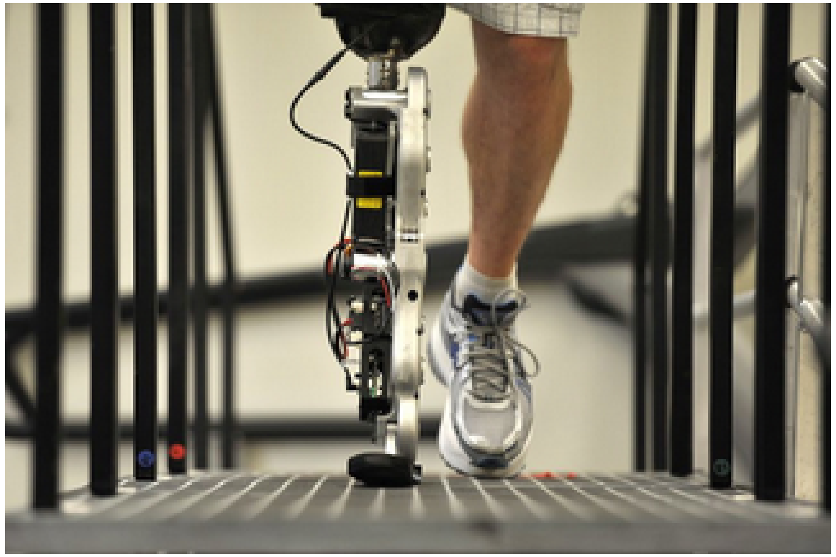


Image from new.homeschoolmarketplace.com

Suppose a person wants to run

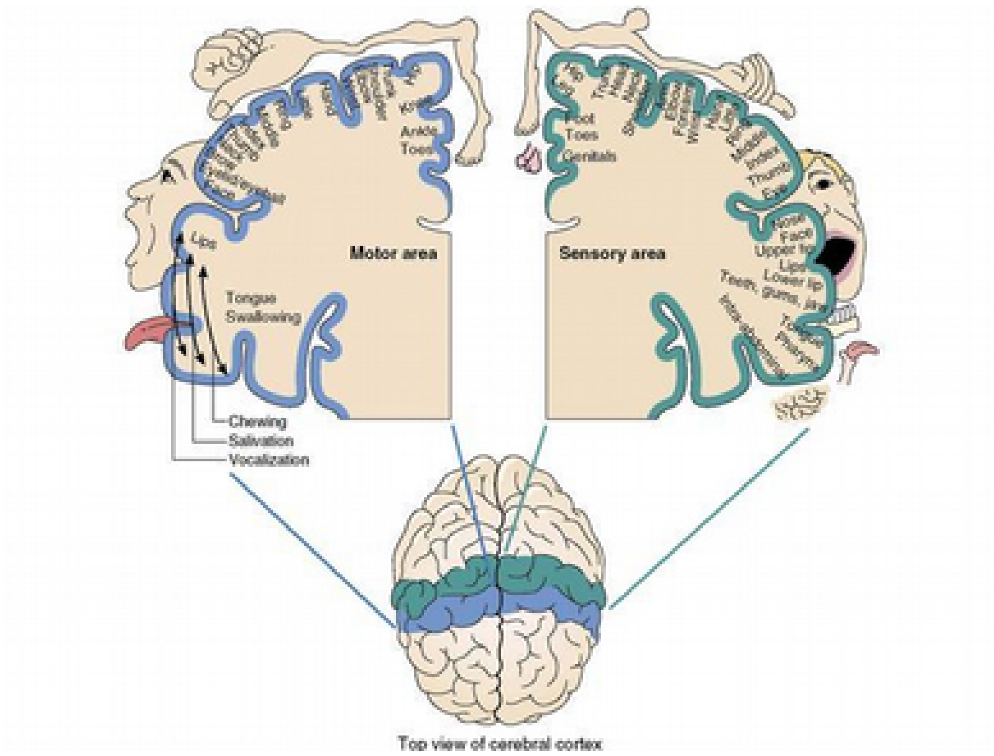
But he doesn’t have a leg



http://images.dailytech.com/

We need a computerized mechanism to control this bionic

A characteristic of the brain



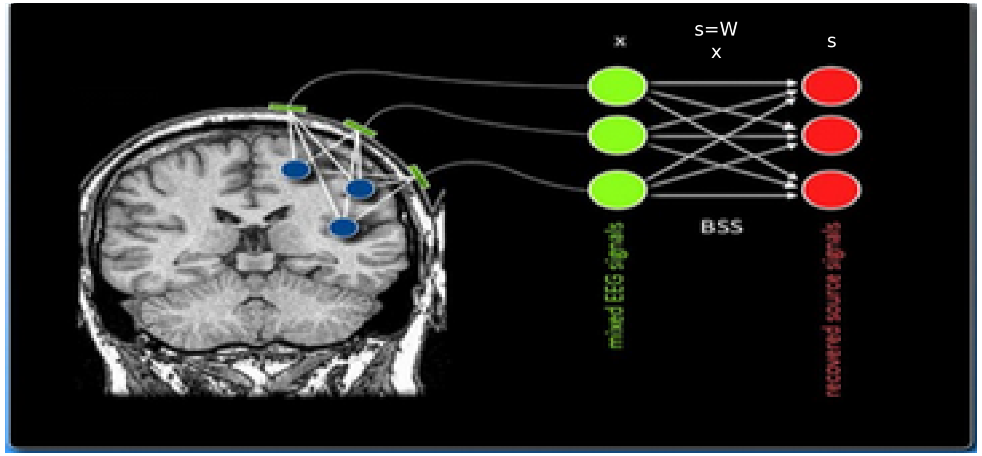
Right side of brain controls left leg, left side controls right leg

Problem we should solve

- We need to identify the original source signals
- We need to identify it fast

Problem one - Identifying signal Independent Component Analysis(ICA)

Blind Source Separation



$x = As$

We don't know both (A) and (s)

X- what we observe
A - Mixing matrix(Depends on the source position)
S – Unknown original sources

$x = As$

$A^{-1}x = A^{-1}As$

$A^{-1}x = s$

$s = Wx$

Where $W = A^{-1}$

Problem two - Identifying it fast

- FastICA algorithm
- Fast and accurate

FastICA is still slow

- Dataset - 118 sources and 15,000 samples taken within 15 seconds

- Result - FastICA took about 4,700 seconds to solve this

- This is about One and Half hours !

We improved FastICA

- Used parallelism to improve the performance
- Implemented in threading, parallel processing and hybrid version of threads and processes

FastICA algorithm

Pre processing	Centering data Singular Value Decomposition Initialize W matrix
Main loop	Dot products Symmetric Decorrelation Apply non-linear function(g) to the input matrix (x) ex:- Cube(x) , tanh(x), log cosh(x)
Post Calculations	Wx = s

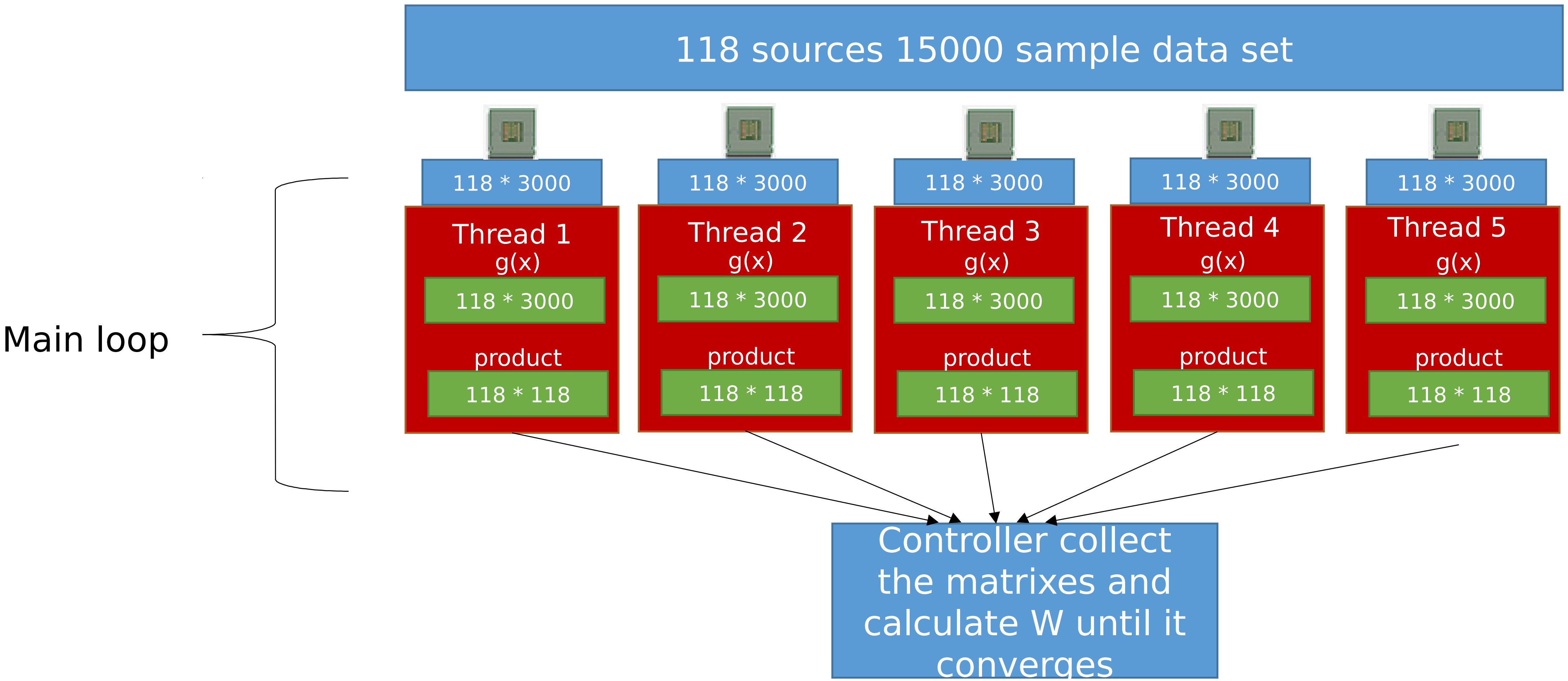
Amdahl's law

- If we need to improve an algorithm, we need to improve the most time consuming serial part

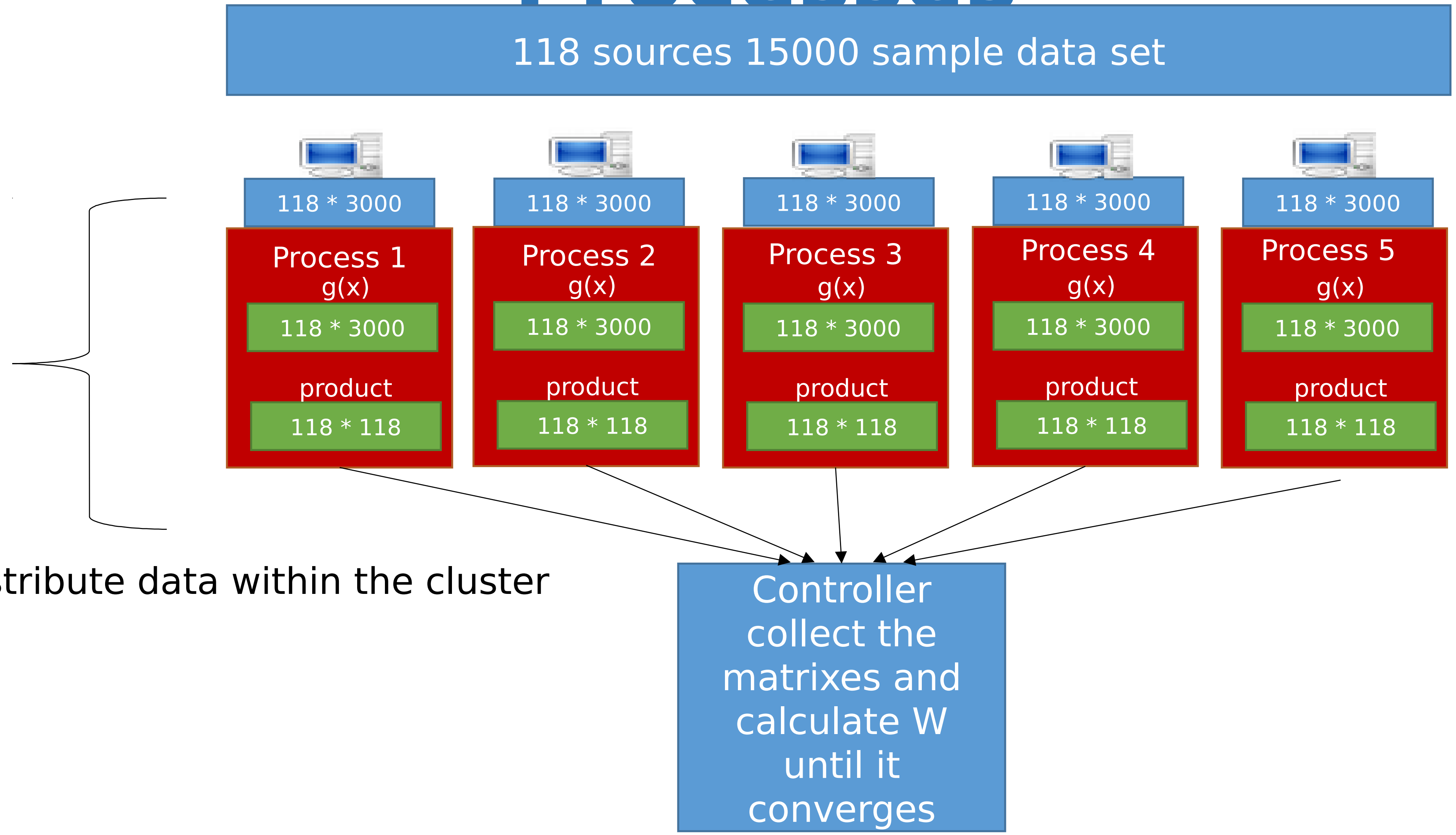
- ICA main loop consumed about 90% of total time for 118 source, 15000 sample set

Paralleling the FastICA - Threading

(Apply non-linear functions to the input matrix (x) parallelly)



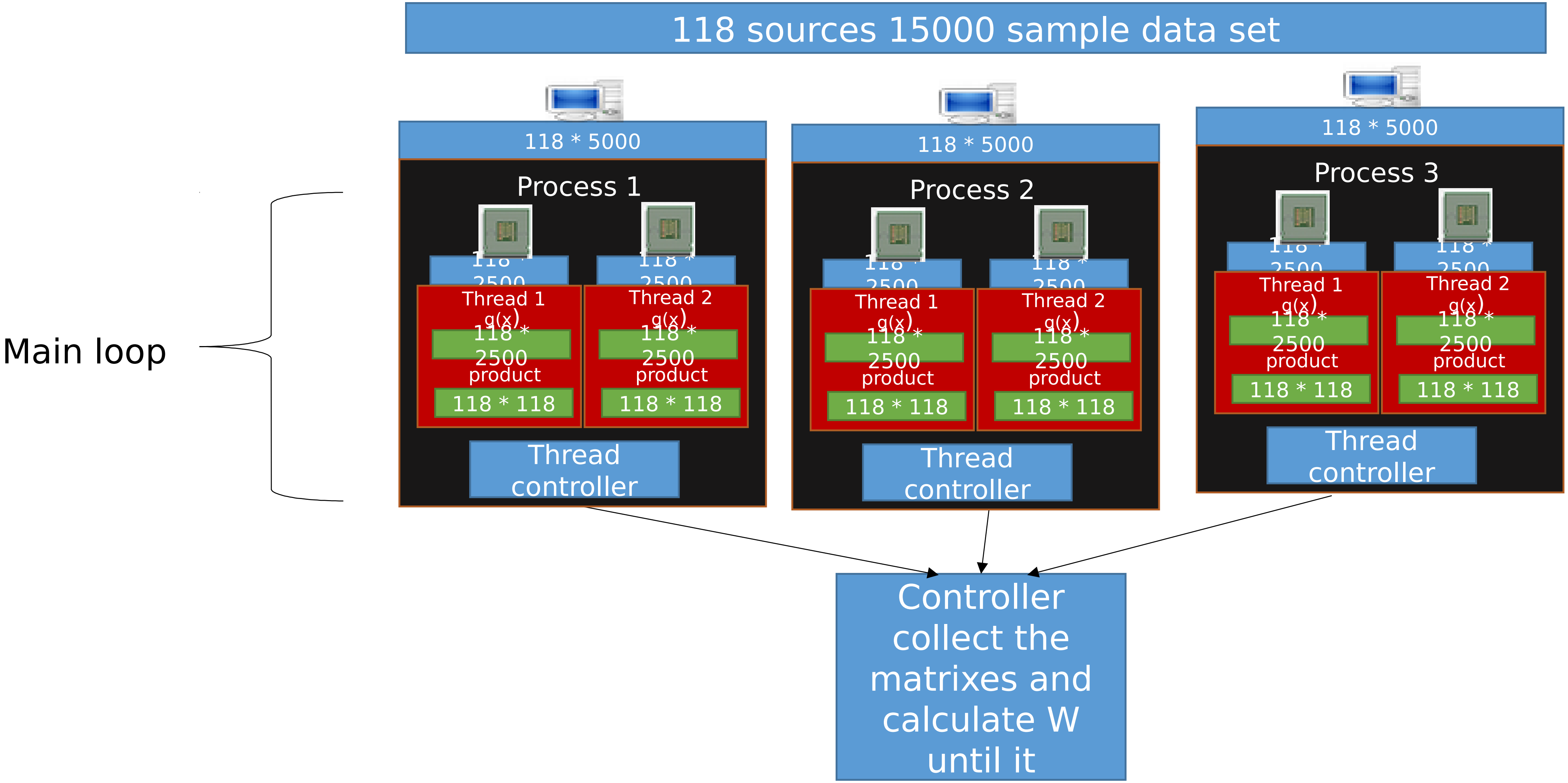
Paralleling the FastICA - Processes



Main loop

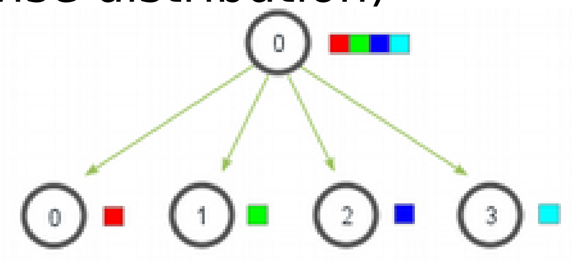
used to distribute data within the cluster

Paralleling the FastICA - Hybrid



To Achieve High Levels of Parallelism

- Input data decomposition (columns wise distribution)



- What kind of granularity is matter ?

Critical Path

The longest directed path between any pair of start and finish nodes

Degree of Concurrency

Maximum degree of concurrency

Average degree of concurrency

$$\text{avg degree of concurrency} = \frac{\text{total amount of work}}{\text{critical path length}}$$

To Achieve High Levels of Parallelism

- Maximize data locality
- Minimize volume of data exchange between threads or processes
- Management of access of shared data

Experimental Setup

Test data set

118 sources and 298458 samples
taken from BCI Competition III (**http://bbci.de/competition**)

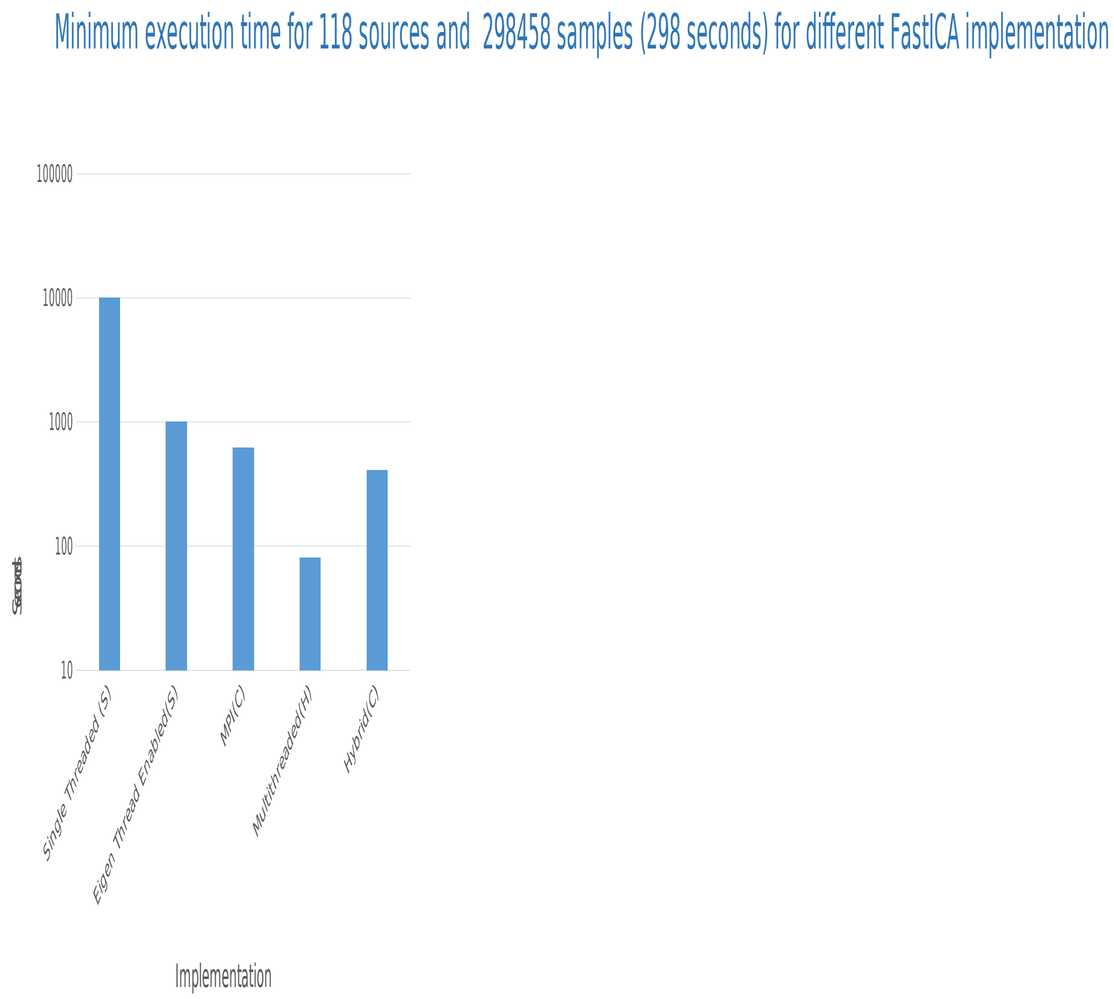
We used following machines to test our solution.

- Single Node (S) - 4 cores 8 threads
- High performance computer (HPC/H) - 16 cores and 32 threads
- MPI Cluster (M) - four single node machines

High Performance Computer
Intel(R) Xeon(R) CPU E5-2670 0 @2.60GHz cpu
MHz : 2601.000
Cache size : 20MB
Memory Total: : 256GB

Single Node Computer
Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz cpu
MHz : 1600.000
Cache size : 6MB Memory Total: : 4 GB

Result



The graph is in log scale

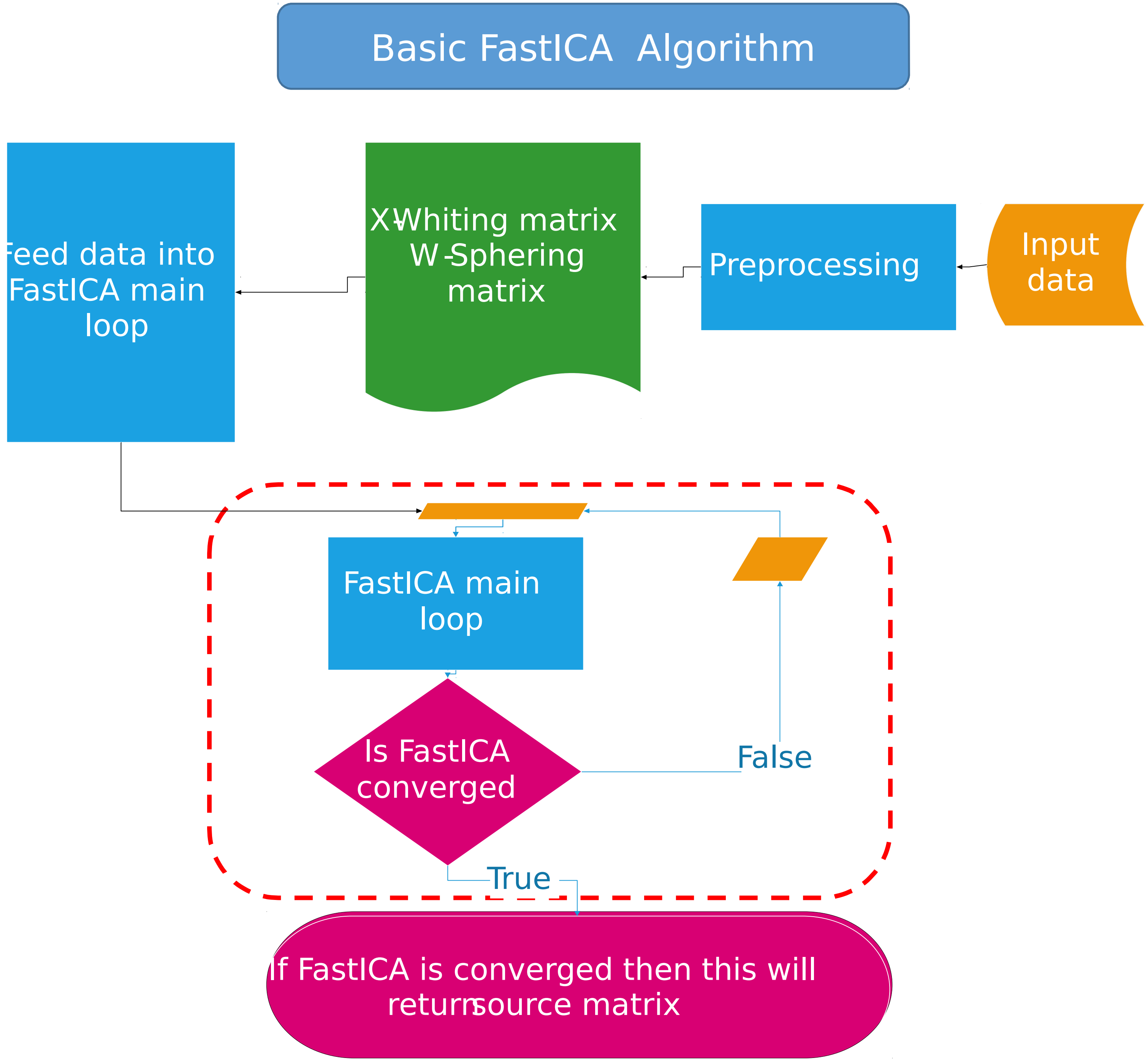
Implementation	Execution time(Seconds)	Maximum Parallelism
Single Threaded (S)	10069.9272	1 threads
Eigen Thread Enabled(S)	1012.9345	32 threads
MPI(C)	625.0176	8 processes
Multithreaded(H)	81.0965	32 threads
Hybrid(C)	408.8416	2 processes + 8 threads

Conclusion

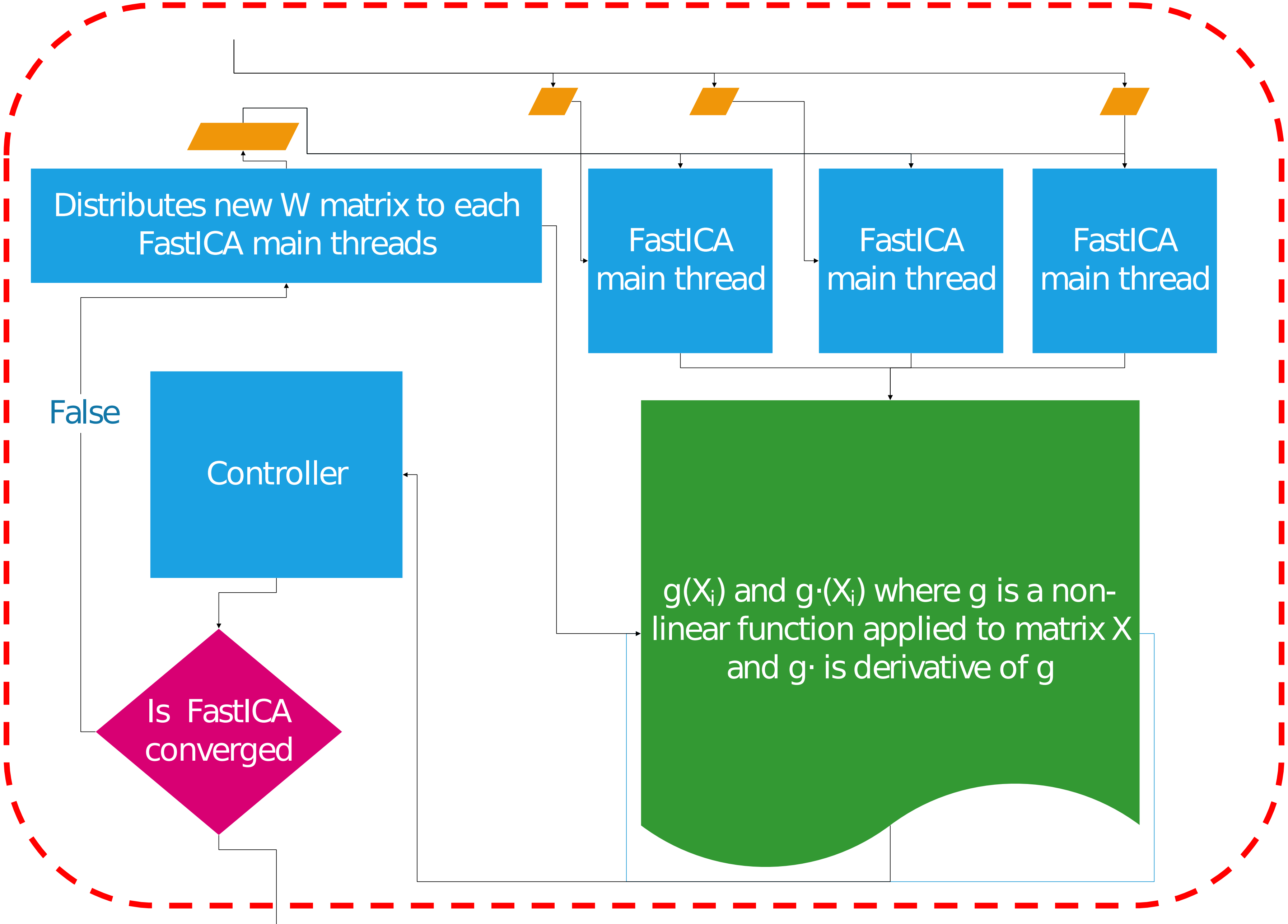
- We need to identify the original source signals
  - FastICA gives the correct result
- We need to identify it fast
  - For 300 seconds sample the calculation only took 81 seconds in high performance computer
  - But this is not a feasible solution
  - We need to find a cheap solution : Nvidia-CUDA

Future!

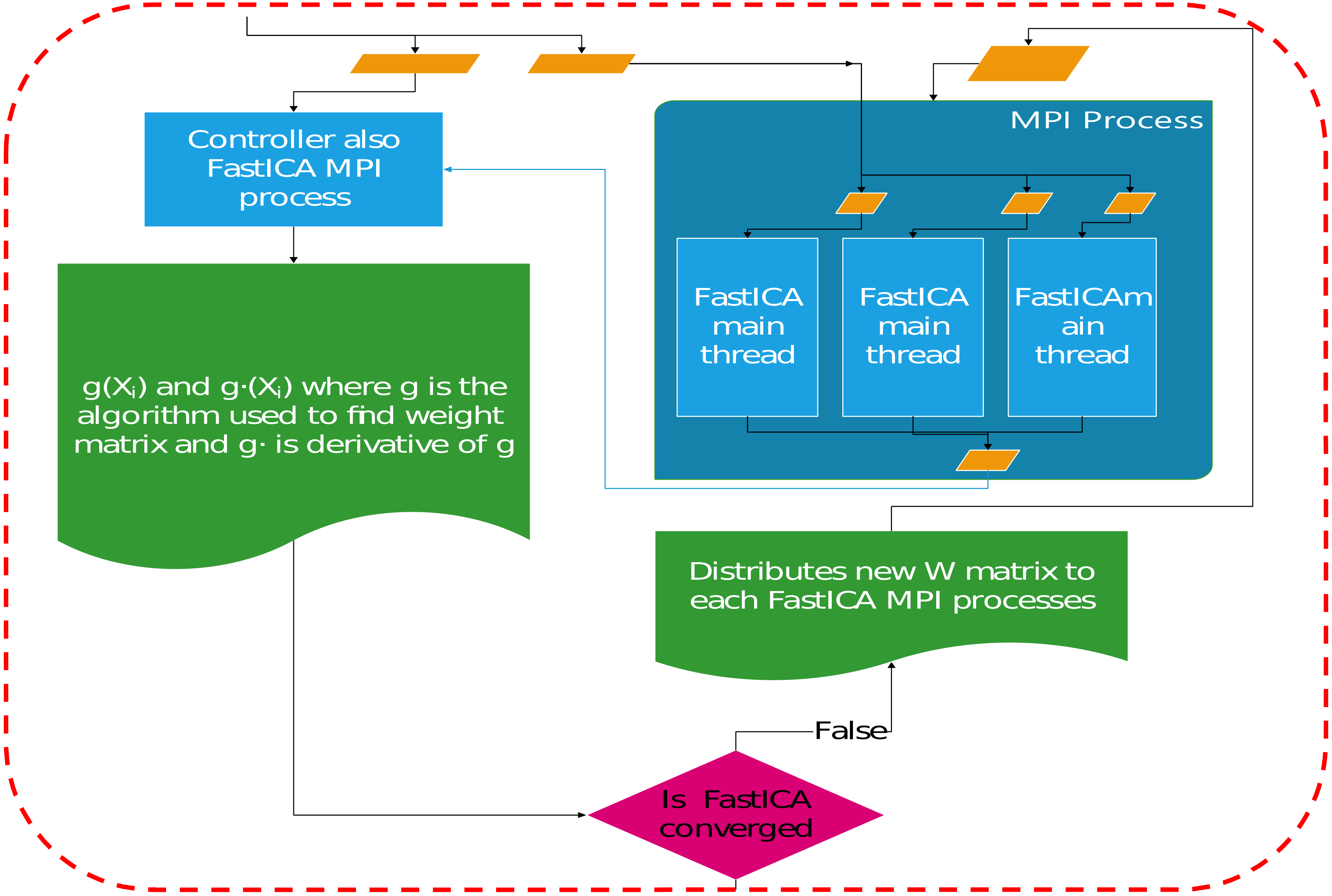
Thank you


Basic FastICA Algorithm
Feed data into FastICA main loop
X-Whiting matrix W-Sphering matrix
Preprocessing
Input data
FastICA main loop
Is FastICA converged
False
True
If FastICA is converged then this will return source matrix

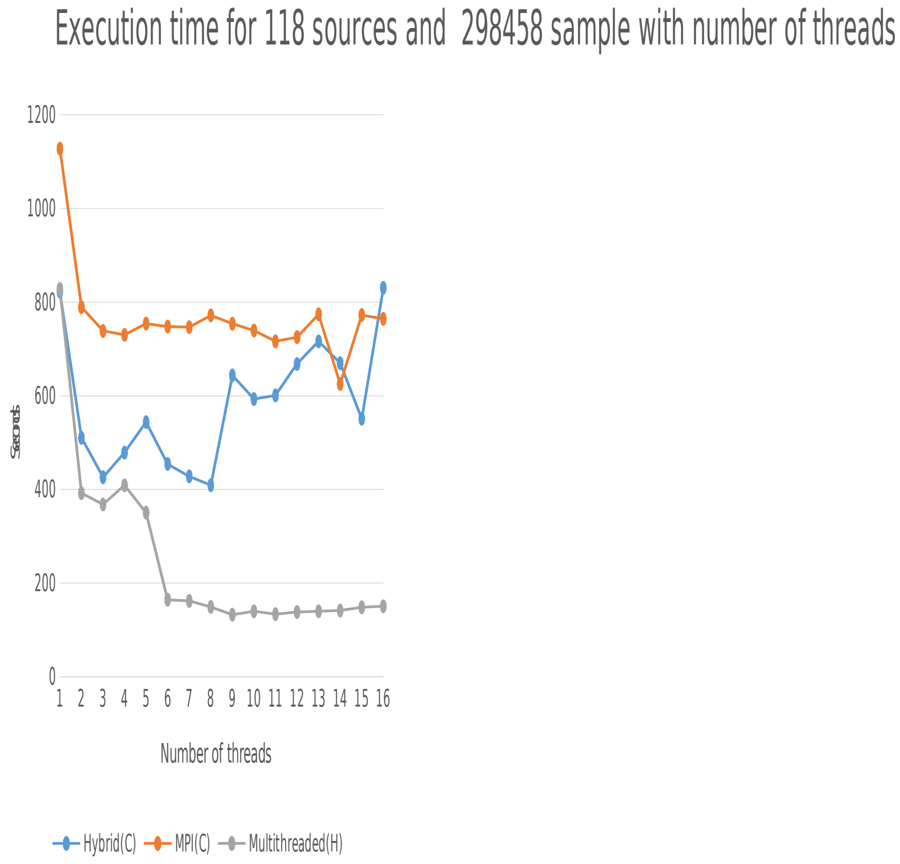
Multi-threaded FastICA algorithm



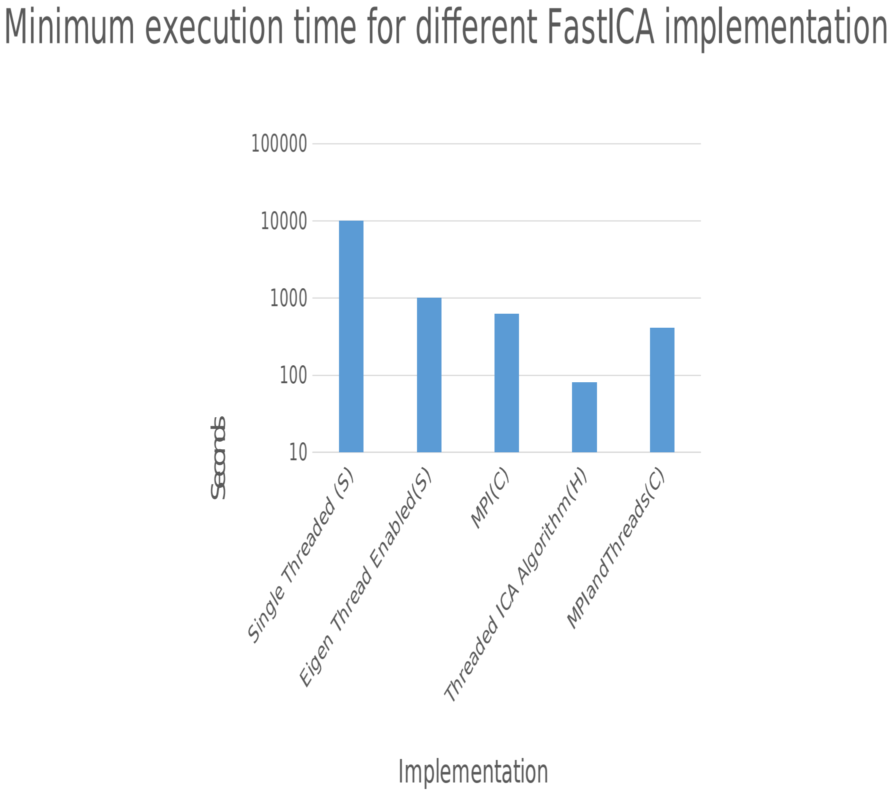
MPI and Multithreaded Algorithm



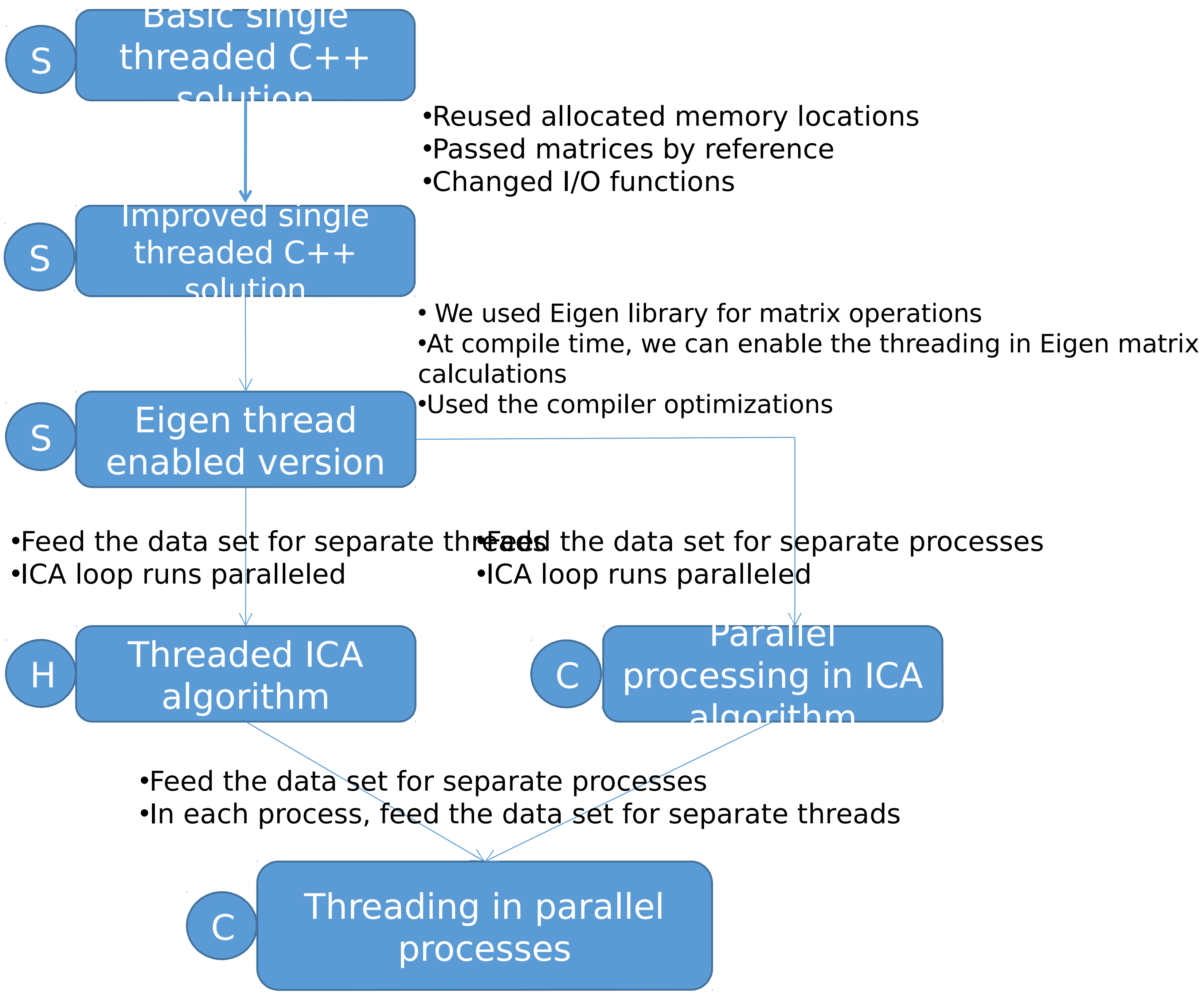
Results



Execution time for 118 sources and 298458 sample with number of threads
1200
1000
800
600
400
200
0
Seconds
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Number of threads
Hybrid(C)
MPI(C)
Multithreaded(H)



Minimum execution time for different FastICA implementation
100000
10000
1000
100
10
Seconds
Single Threaded (S)
Eigen Thread Enabled(S)
MPI(C)
Threaded ICA Algorithm(H)
MPIandThreads(C)
Implementation


S
Basic single threaded C++ solution
•Reused allocated memory locations
•Passed matrices by reference
•Changed I/O functions
S
Improved single threaded C++ solution
• We used Eigen library for matrix operations
•At compile time, we can enable the threading in Eigen matrix calculations
•Used the compiler optimizations
S
Eigen thread enabled version
•Feed the data set for separate threads
•ICA loop runs paralleled
•Feed the data set for separate processes
•ICA loop runs paralleled
H
Threaded ICA algorithm
C
Parallel processing in ICA algorithm
•Feed the data set for separate processes
•In each process, feed the data set for separate threads
C
Threading in parallel processes

Technologies/Tools and Libraries

scikit learn

MPICH2

NVIDIA CUDA

Algorithm FastICA

Input: C Number of desired components

Input: $\mathbf{X} \in \mathbb{R}^{N \times M}$ Matrix, where each column represents an *N*-dimensional sample, where $C < N$

Output: $\mathbf{W} \in \mathbb{R}^{C \times N}$ Un-mixing matrix where each row projects X onto into independent component.

Output: $\mathbf{S} \in \mathbb{R}^{C \times M}$ Independent components matrix, with M columns representing a sample with C dimensions.

for p **in** 1 to C:

  $\mathbf{w_p} \leftarrow$ *Random vector of length N*

  while $\mathbf{w_p}$ changes

$$\mathbf{w_p} \leftarrow \frac{1}{M}\mathbf{X}g(\mathbf{w_p}^T\mathbf{X}) - \frac{1}{M}g'(\mathbf{w_p}^T\mathbf{X})\mathbf{1}\mathbf{w_p}$$

$$\mathbf{w_p} \leftarrow \mathbf{w_p} - \sum_{j=1}^{p-1} \mathbf{w_p}^T\mathbf{w_j}\mathbf{w_j}$$

$$\mathbf{w_p} \leftarrow \frac{\mathbf{w_p}}{\|\mathbf{w_p}\|}$$

Output: $\mathbf{W} = \begin{bmatrix} \mathbf{w_1} \\ \vdots \\ \mathbf{w_C} \end{bmatrix}$

Output: $\mathbf{S} = \mathbf{W}\mathbf{X}$