GPU Acceleration for Fixed Complexity Sphere Decoder in Large MIMO Uplink Systems

Presented by Tianpei Chen Tianpei.chen@mail.mcgill.ca

Supervisor : Harry Leib Harry.Leib@mcgill.ca

Outline

Introduction

Introduce large MIMO uplink system, General Purpose Graphic Processing Unit Computing (GPGPU) , Compute Unified Device Architecture (CUDA) and microarchitecture of GPU.

Channel Model and Fixed Complexity Sphere Decoder (FCSD)

Provide background knowledge of discrete time MIMO channel model and details of Fixed Complexity Sphere Decoder.

GPU Based Acceleration of FCSD

Present implementation details of CUDA-FCSD

Performance and Analysis

Present computer simulation results, compare GPU and CPU implementation under different circumstances, analyze the factors that impact Acceleration performance

Conclusions

Give conclusions and discussions

Introduction

- Large MIMO system

- General Purpose Graphics Processing Units Computing (GPGPU)

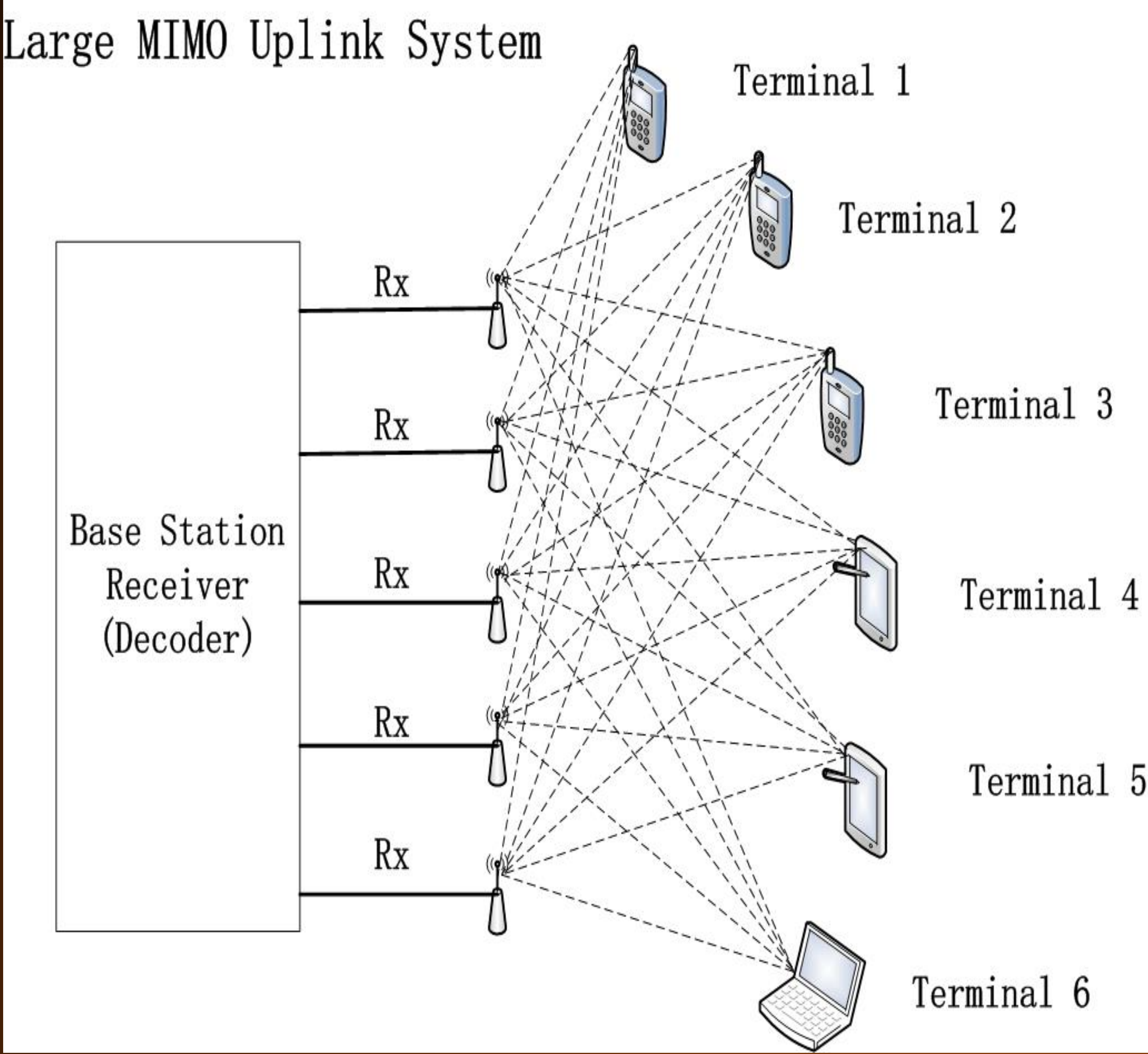
- Compute Unified Device Architecture (CUDA)

- Microarchitecture of GPU

Introduction

Large MIMO system

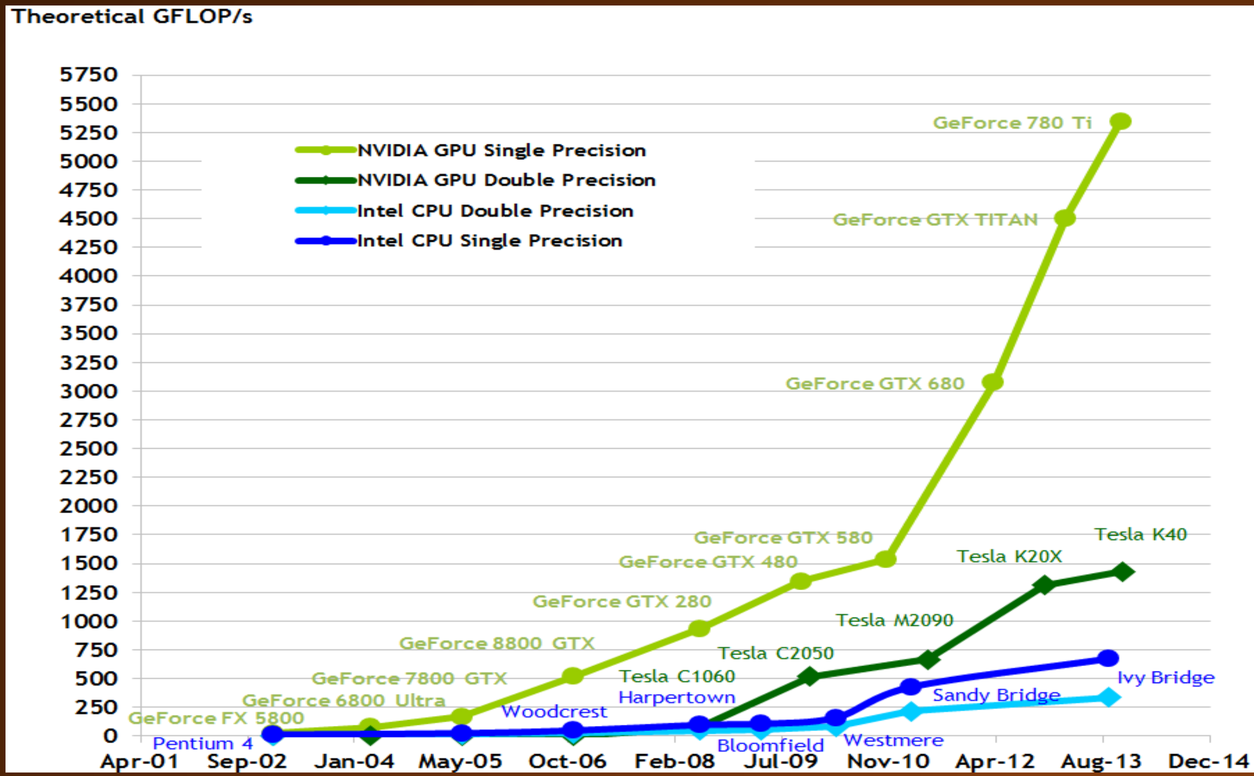
● Multiple Input Multiple Output (MIMO) technology has attracted immense research interests since it can improve spectrum efficiency as well as Quality of Service (QoS). Large Scale MIMO systems employ tens to hundreds of low-power low-price antennas at base station, serving several tens of terminals at the same time. Large MIMO system can achieve full potential of conventional MIMO systems while providing additional advantages.



Introduction

General Purpose Graphics Processing Units Computing (GPGPU)

General Purpose Graphics Processing Units (GPGPU) computing is a common trend in industry and research area that using Graphics Processing Units (GPU) to handle massive computation applications which are traditionally implemented by Central Processing Units (CPU). Compared to a CPU, a GPU can naturally cope with computational intensive tasks, since most of the hardware resources (transistors) are allocated for data processing and parallel computation, rather than caching and flow control.

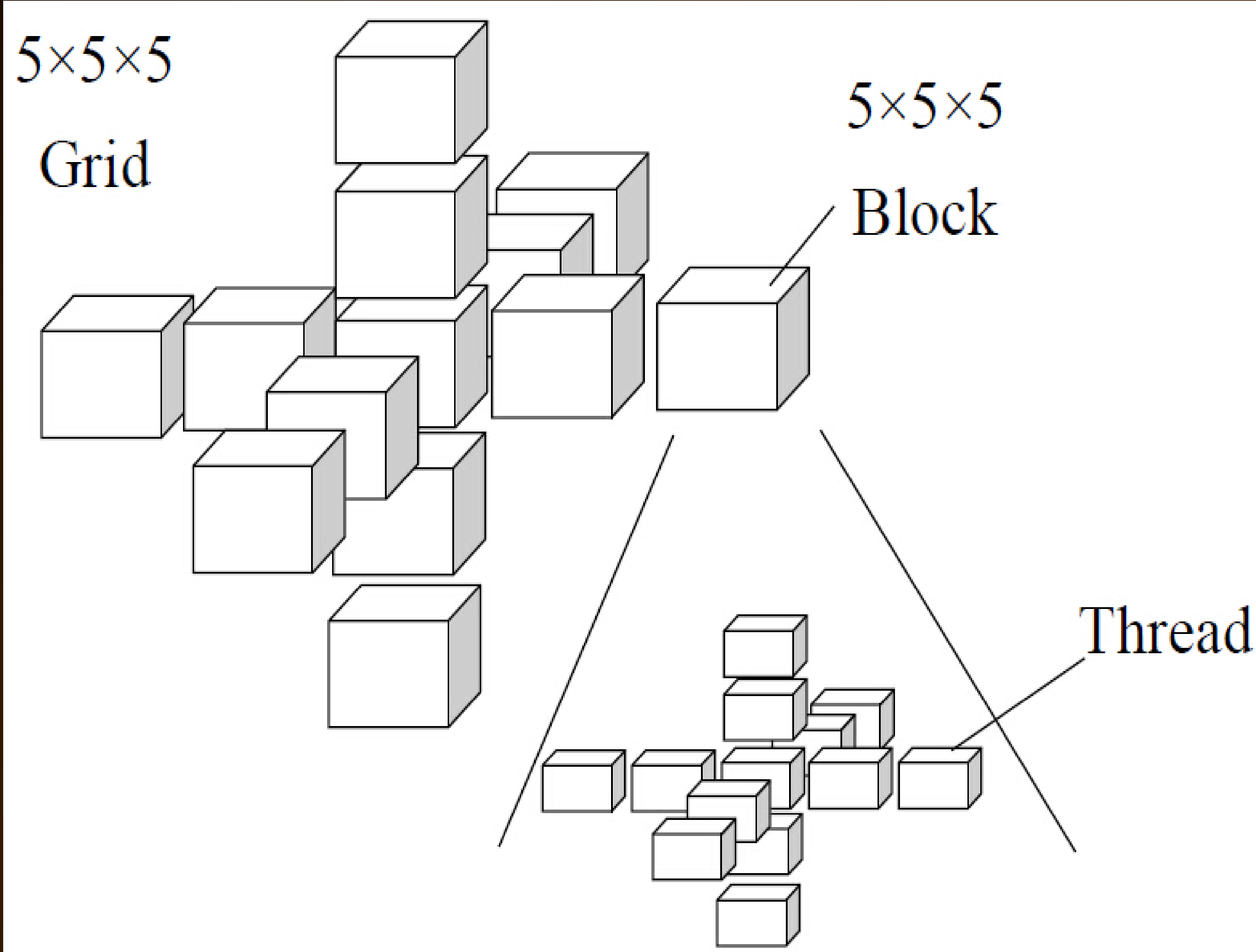


[1]

Introduction

Compute Unified Device Architecture (CUDA)

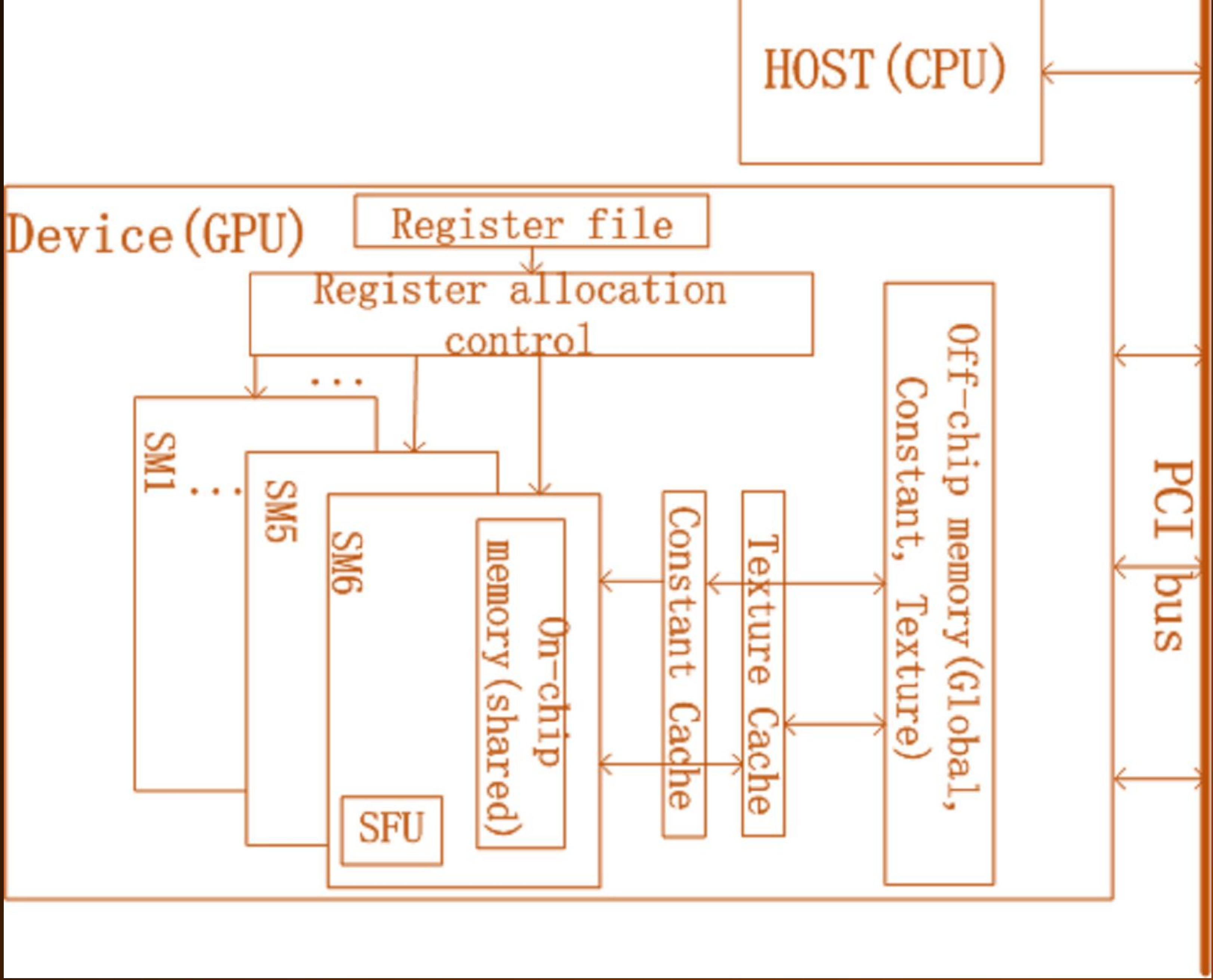
Compute Unified Device Architecture (CUDA) is a programming model that enables developers to use popular programming languages such as C, C++, Java and Python for GPU programming. It was introduced by NVIDIA in 2006.



Introduction

Microarchitecture of GPU

GPU works as a coprocessor system. The major hardware resources are located at Stream Multiprocessors (SMs), which performs basic arithmetic operations, each SM has a special functional unites (SFU), which performs more complex arithmetic operations. A GPU contains five types of memory. registers, shared memory, global memory, constant memory and texture memory.



Channel Model and Fixed Complexity Sphere Decoder (FCSD)

- MIMO channel Model

- Fixed Complexity Sphere Decoder

Channel Model and Fixed Complexity Sphere Decoder (FCSD)

MIMO channel Model

We consider a complex uncoded spatial multiplexing MIMO system with N_r receive and N_t transmit antennas, The discrete time MIMO channel model is given by

$$y = Hs + n \qquad (1)$$

$y \in C^{N_r \times 1}$ denotes the received symbol vector, $s \in C^{N_t \times 1}$ denotes the transmitted symbol vector, with components that are mutually independent and taken from signal constellation alphabet **O (e.g. 4QAM, 16QAM, 64QAM).** $\mathbf{n} \in \boldsymbol{C}^{N_r \times 1}$ denotes additive white Gaussian noise (AWGN) vector with zero mean components and variance N_o . $\boldsymbol{H} \in \boldsymbol{C}^{N_r \times N_t}$, denotes Rayleigh flat fading channel matrix, with independent identically distributed (i.i.d) circularly symmetric complex Gaussian components of unit variance. Let E_s denotes the average energy of transmitted symbols. Hence, $\frac{E_s}{N_o}$ is signal to noise ratio.

Assume the receiver has perfect channel state information (CSI), meaning that H is known, as well as the SNR. The task of the MIMO decoder is to recover s based on y and H.

Channel Model and Fixed Complexity Sphere Decoder (FCSD)

Fixed Complexity Sphere Decoder

From (1), the maximum likelihood detector (MLD) can be specified by

$$s_{ML} = arg \max_{s \in O^{N_t}} p(\mathrm{y}|s, H) \quad (2)$$

Which can be rewritten as

$$s_{ML} = arg \min_{s \in O^{N_t}} ||y - Hs||^2 \quad (3)$$

Perform the QR factorization to H, we have

$$H = QR \quad (4)$$

Where $Q \in C^{N_t \times N_t}$ denotes unitary matrix and $\mathrm{R} \in C^{N_t \times N_t}$ denotes upper triangular matrix. For sake of brevity we gave the object function directly

$$s_{ML} = arg \min_{s \in O^{N_t}} ||R(\hat{S} - S)||^2 \quad (5)$$

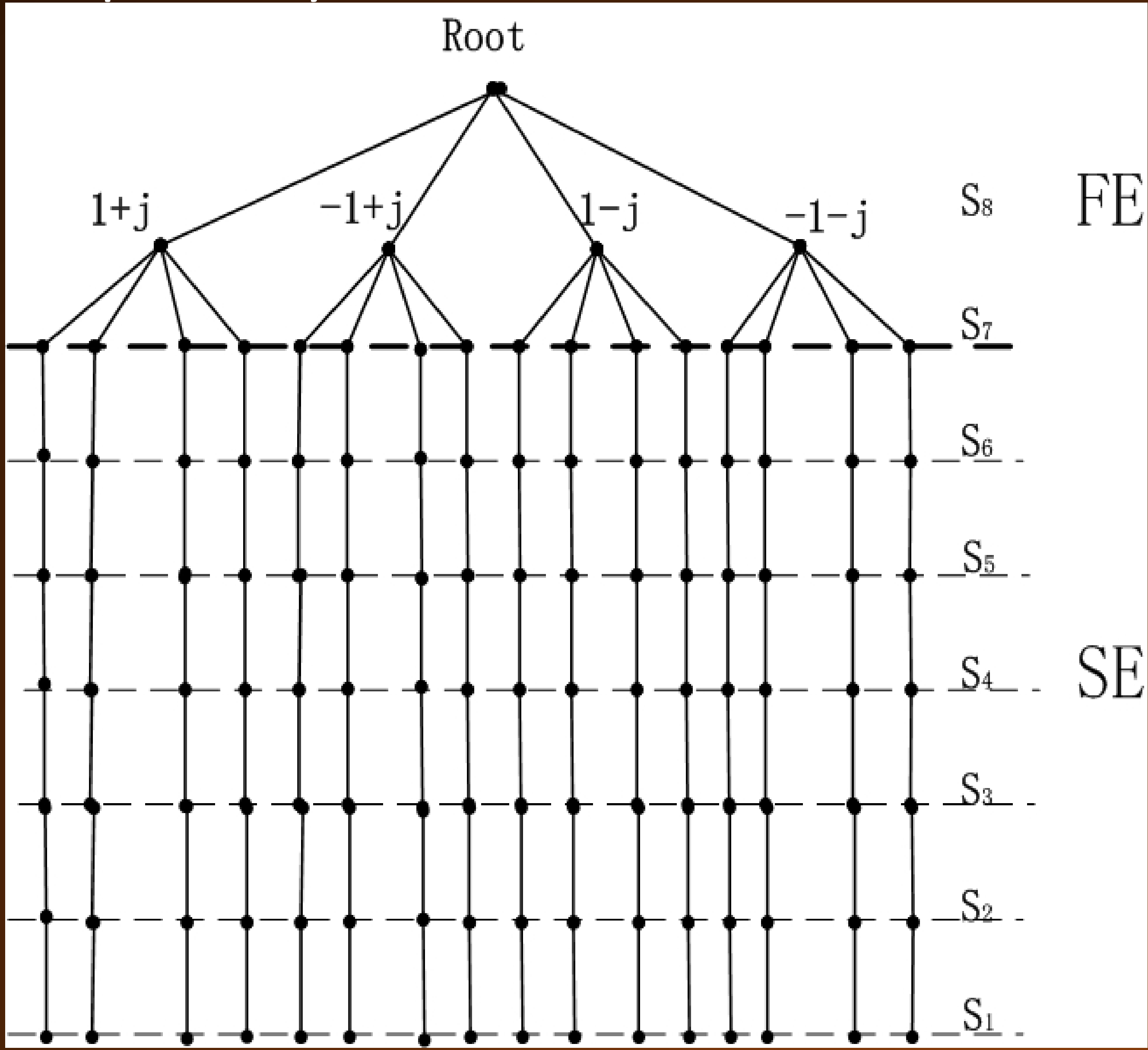
Where $\hat{s} = (H^H H)^{-1} H^H y$ denotes unconstrained estimation of s.

Channel Model and Fixed Complexity Sphere Decoder (FCSD)

Fixed Complexity Sphere Decoder

FCSD is a tree searching algorithm performs depth first searching.

At the first ρ node levels FCSD searches all the possible signal symbols exhaustively in constellation alphabet O, a process called Full Expansion (FE). at the remaining $N_t - \rho$ levels of nodes, the FCSD employs decision feedback, therefore there are only one branch expansion for each symbol node at this stage. This stage is called Single Expansion (SE).



Channel Model and Fixed Complexity Sphere Decoder (FCSD)

Fixed Complexity Sphere Decoder

FE stage:

$$s_i^F \in O^{N_t}\ if\ i = N_t, N_t\text{-}1, \ldots N_t - \rho + 1 \quad (6)$$

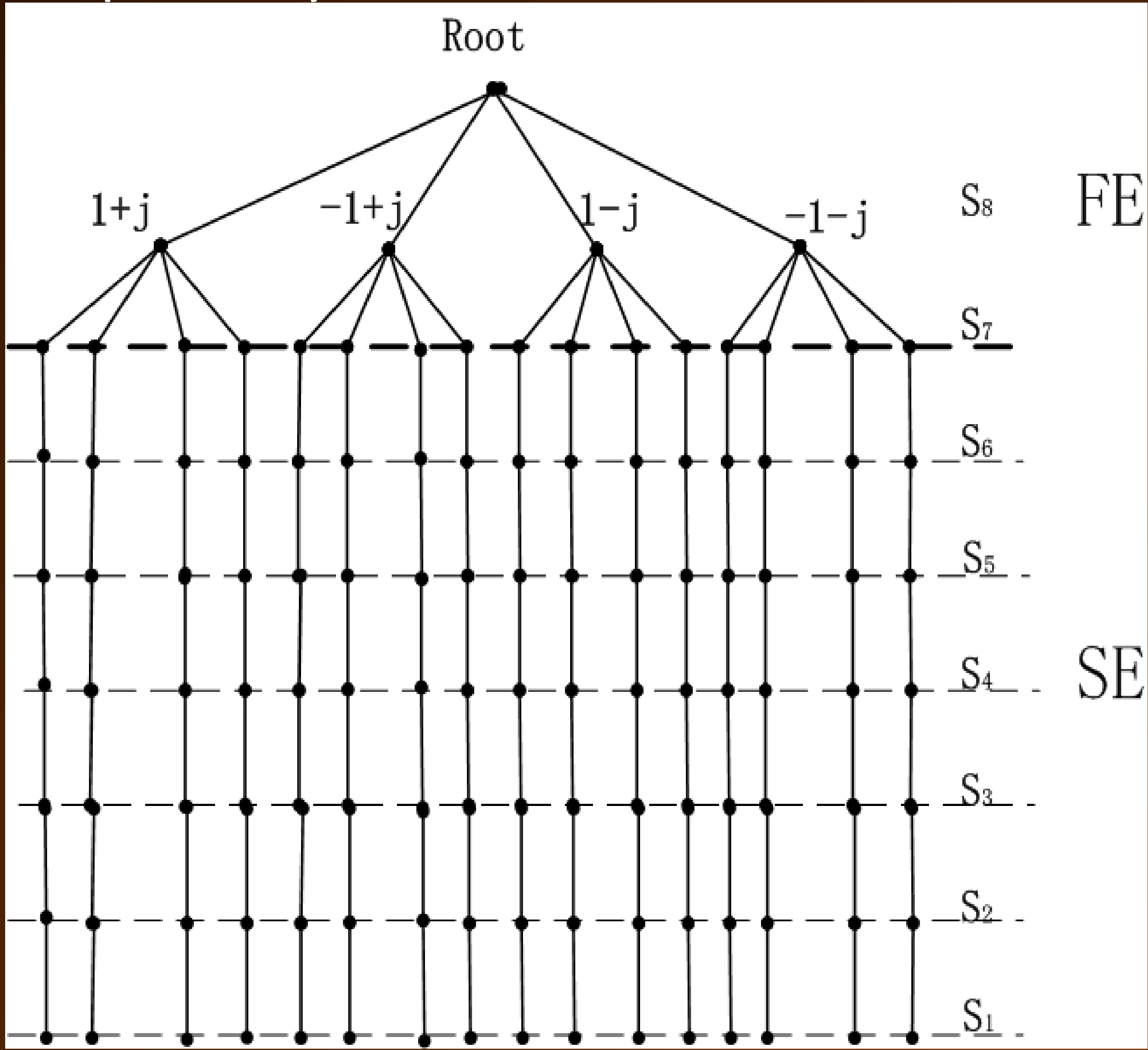
SE stage:

$$s_i^F = \mathbb{Q}[\hat{s}_i + \sum_{j=i+1}^{N_t} \frac{r_{ij}}{r_{ii}} (\hat{s}_j - s_j^F)] \quad (7)$$

Where s^F denotes one of solution candidate. $\mathbb{Q}$ denotes constellation quantization operation.

- At the post processing step, FCSD compares Euclidean distance and choose the optimal vector candidate.

$$E_u = ||R(\hat{s} - s)||^2 \quad (8)$$



GPU Based Acceleration of FCSD

- Preprocessing
- Integrated Parallel Acceleration of Paths Searching
- Memory Coalescing

GPU Based Acceleration of FCSD

Preprocessing

Preprocessing process of FCSD includes calculation of unconstrained estimation ŝ, QR factorization as well as iterative channel ordering.

A FCSD channel ordering strategy is used in order to avoid error propagation in the serial path searching process. This channel ordering strategy is based on post processing SNR [2]

$$\varphi_m = \frac{E_s}{N_o(H^H H)_m^{-1}} \quad (9)$$

Where φ_m denotes the post processing SNR of m th data stream.

The FCSD channel ordering works iteratively based on the following rule

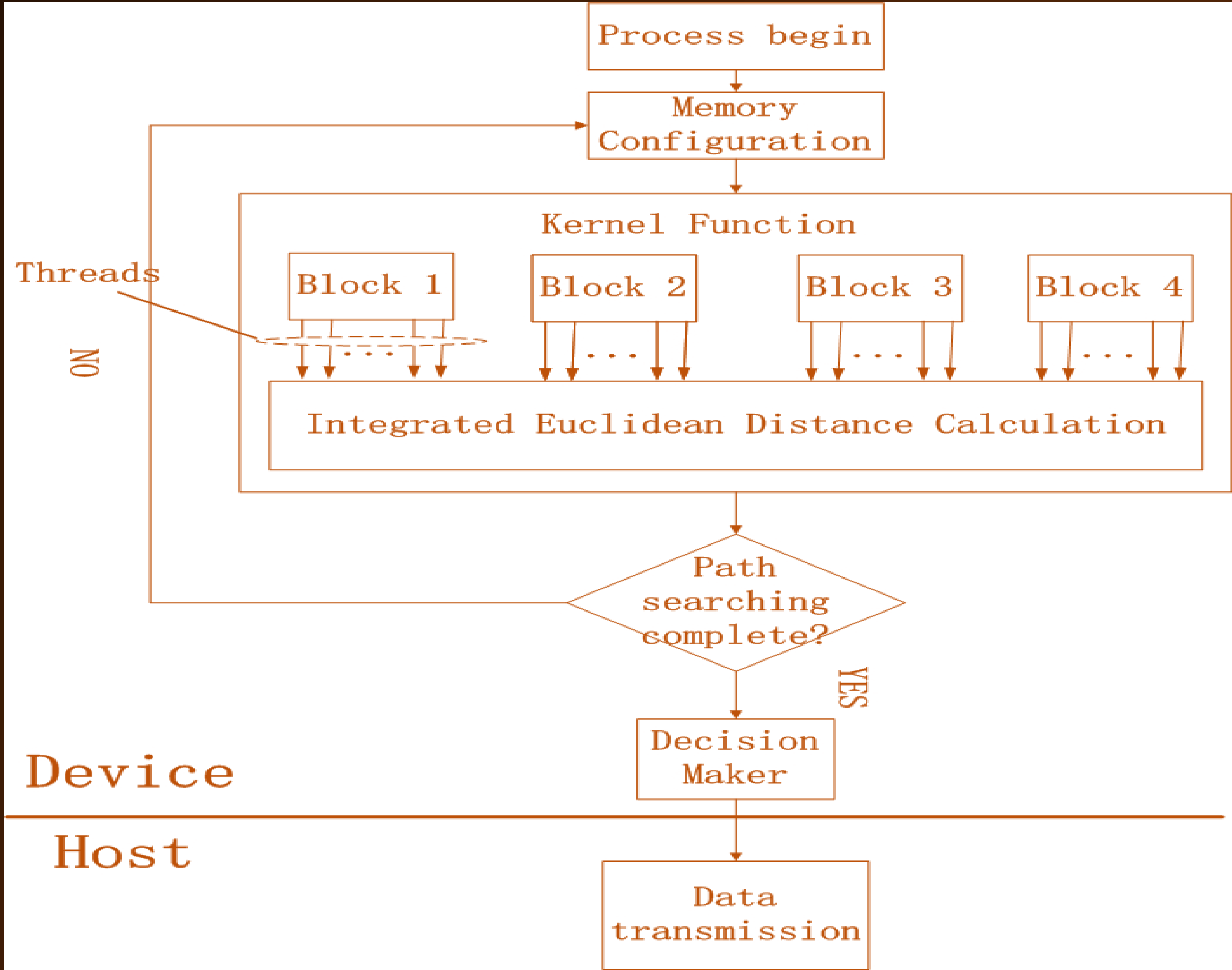
$$p = \begin{cases} \arg\max_k (H_j^H H_j)_k^{-1} & FE\ stage \\ \arg\min_k (H_j^H H_j)_k^{-1} & SE\ stage \end{cases}$$

We make use of the high performance CUDA basic linear algebra subroutines (cuBLAS) [3] to accelerate preprocessing process.

GPU Based Acceleration of FCSD

Integrated Parallel Acceleration of Paths Searching

- Since the decision feedback searching paths has a serial nature, we match one path to one thread. In order to have the largest number of threads that can be parallelized, we use one dimensional blocks for widest expansion, and organize all the paths into several parallel blocks.



GPU Based Acceleration of FCSD

Integrated Parallel Acceleration of Path Searching

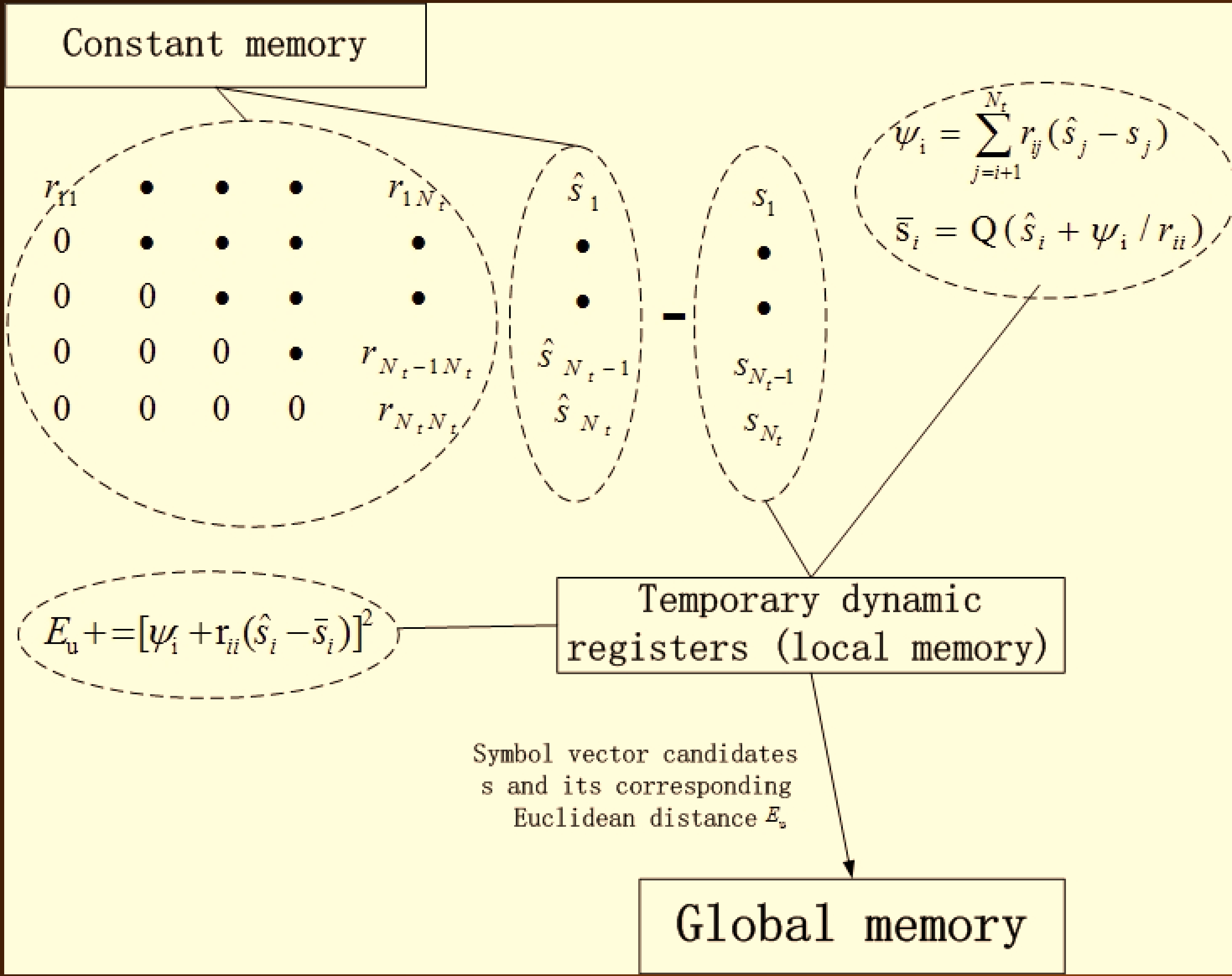
● The utilization of memory bandwidth is the bottleneck of overall performance.

● At post processing stage, Euclidean distance of all the possible candidates are calculated and compared.

$$E_u = \sum_{i=1}^{N_t} \left| \sum_{j=i}^{N_t} r_{ij}(\widehat{s_j} - s_j) \right|^2$$

if we look at each searching path, we can find the value is partly calculated, at each node we have

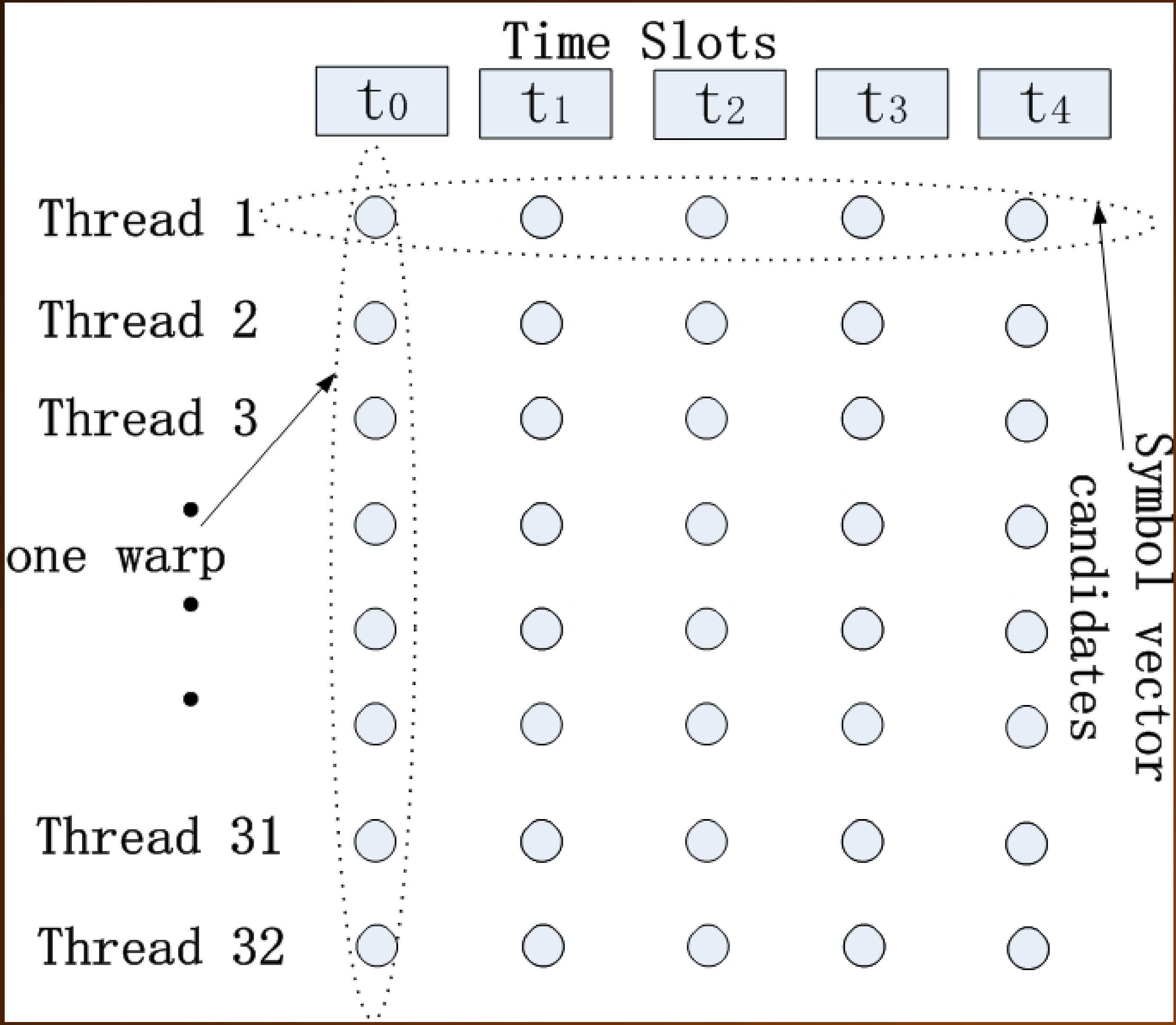
$$\psi_i = \sum_{j=i+1}^{N_t} r_{ij}(\widehat{s_j} - s_j)$$



GPU Based Acceleration of FCSD

Memory Coalescing

In a modern DRAM the R/W operations are performed by accessing a piece of consecutive memory location during one cycle. In fact the threads are managed and scheduled in the unit of wrap. Currently 32 threads formed a warp. Memory space is accessed warp by warp. Thus the most effective global memory access pattern is to ensure a warp of threads can access a consecutive memory space.



Performance and Analysis

- Acceleration Performance

- Bit Error Rate-Signal to Noise Ratio Performance

Performance and Analysis

Acceleration Performance

Comparisons are made between GPU implementation on GeForce 760 and CPU implementation on two different desktops. Operation time is calculated based on 1000 channel realizations.

- Graphic Processing Units: GeForce GTX 760
- Central Processing Units-Modigliani: Intel Core I5-4 th generation, 4 physical cores.
- Central Processing Units-Monet : Intel Core I7-3rd generation, 6 physical cores.

TABLE II
SPEEDUP PERFORMANCE OF DIFFERENT MIMO SYSTEMS USING 4 QAM

Array size	Time/s			Speedup	
	GeForce GTX 760	Modigliani	Monet	$\vartheta_{GTX760}/\vartheta_{Modigliani}$	$\vartheta_{GTX760}/\vartheta_{Monet}$
8×8	7.39	0.10	0.11	0.01	0.01
16×16	16.26	0.92	0.98	0.06	0.06
32×32	38.32	31.27	34.60	0.82	0.90
48×48	71.95	262.00	285.14	3.64	3.96
64×64	322.90	1753.30	1940.10	5.43	6.01
72×72	1285.41	8727.69	9641.80	6.79	7.50
84×84	6454.02	47095.33	49962.01	7.30	7.74

TABLE III
SPEEDUP PERFORMANCE OF DIFFERENT MIMO SYSTEMS USING 16 QAM

Array size	Time/s			Speedup	
	GeForce GTX 760	Modigliani	Monet	$\vartheta_{GTX760}/\vartheta_{Modigliani}$	$\vartheta_{GTX760}/\vartheta_{Monet}$
8×8	13.20	0.67	0.93	0.05	0.07
16×16	26.80	31.68	41.95	1.18	1.57
20×20	192.70	740.50	978.26	3.84	5.08
32×32	4980.13	28209.53	38106.00	5.66	7.65
36×36	5568.26	35240.16	47290.00	6.33	8.61

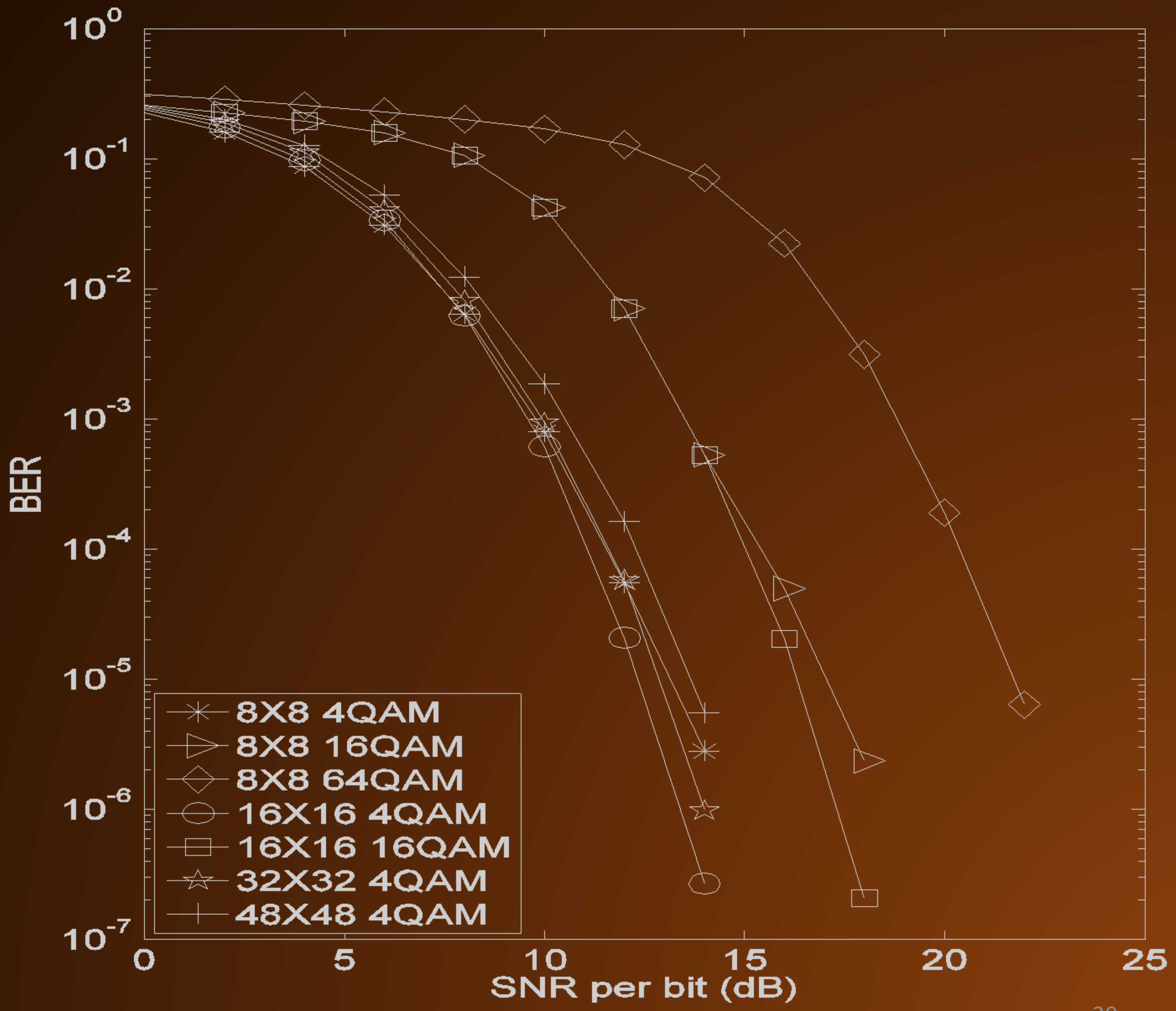
TABLE IV
SPEEDUP PERFORMANCE OF DIFFERENT MIMO SYSTEMS USING 64 QAM

Array size	Time/s			Speedup	
	GeForce GTX 760	Modigliani	Monet	$\vartheta_{GTX760}/\vartheta_{Modigliani}$	$\vartheta_{GTX760}/\vartheta_{Monet}$
8×8	12.28	10.69	13.38	0.87	1.09
16×16	468.92	2066.08	2689.00	4.41	5.73

Performance and Analysis

BER-SNR Performance

BER-SNR performances are evaluated by Monte-Carlo simulations, with at least 100000 channel realizations and at least 500 symbol errors accumulated.



Conclusion

We present a GPU implementation of a single fixed complexity sphere decoder (FCSD) for large MIMO uplink systems. In order to exploit the computational capability of GPU in the simulation of large MIMO systems, the utilization of a heterogeneous programming method, accounting for resource limitations and memory configurations, must be considered. The simulation results show that significant accelerations of the GPU implementation of FCSD over CPU implementation can be obtained for large MIMO systems and signal constellation sizes, while maintaining the same BER performance. This shows the potential of GPU computing to reduce the time associated with intensive simulations of large MIMO systems. Since the GPU and CPU can work independently, the data processing power of GPU computing is an additional advantage provided by the traditional simulation platforms.

Thank you!

References

- [1] NVIDIA Corporation, “CUDA C Programming Guide,” 2014. [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz3RI9FkiM6
- [2] P. W. Wolniansky, G. J. Foschini, G. Golden, and R. Valenzuela, “VBLAST: An architecture for realizing very high data rates over the rich-scattering wireless channel,” in Signals, Systems, and Electronics, 1998. ISSSE 98. 1998 URSI International Symposium on. IEEE, 1998, pp. 295–300.
- [3] NVIDIA Corporation, “Basic linear algebra subroutines (cuBLAS) library,” 2014. [Online]. Available: http://docs.nvidia.com/cuda/cublas/#axzz3RI9FkiM6
- [4] C. Oestges and B. Clerckx, MIMO wireless communications: from real-world propagation to space-time code design. Academic Press, 2010.
- [5] L. G. Barbero and J. S. Thompson, “Fixing the complexity of the sphere decoder for MIMO detection,” Wireless Communications, IEEE Transactions on, vol. 7, no. 6, pp. 2131–2142, 2008.
- [6] S. Cook, CUDA programming: a developer’s guide to parallel computing with GPUs. Newnes Press, 2013.