Exploiting Multi-Core SoC Architecture for MU-MIMO Schedulers

Ganesh Venkatraman, Janne Janhunen, and Markku Juntti
Email: {gvenkatr, janne.janhunen, markku.juntti}@ee.oulu.fi

Centre for Wireless Communications (CWC),
Department of Communications Engineering (DCE),
University of Oulu, Oulu, FI-90014

Outline

Abstract

- Problem Studied - Implementing multi-user MIMO scheduler schemes on TI TCI6636K2H eight core SoC
- Issues addressed -
  - Complexity involved in implementing scheduling algorithms - low complex algorithm design
  - How to partition scheduler processing among eight cores in TI TCI6636K2H eight core SoC
- Summary - Proposed implementation supports up to 100 users in the system with 4×4 MIMO configuration

Introduction and Motivation

- Current standards are moving towards multi-antenna systems due to its numerous advantages
- To avail the benefits, spatially multiplexing multiple user streams are considered
- In order to do so, efficient precoding and user subset are to be identified
- In this work, we analyze the computational needs of different MU-MIMO scheduling algorithms for a single scheduling block
- We evaluate algorithm complexity by implementing on TI TCI6636K2H eight core SoC

Notations used

- We consider a single-cell multi-user MIMO scenario
- Let K be the total number of users with N_R antenna elements
- Let κ be the total available spatial streams for a user k, given by $\kappa = \min(N_T, N_R)$
- $\mathbf{H}_{\hat{k}} \in \mathbb{C}^{N_R \times N_T}$ be the channel between BS and user $\hat{k}, \forall k \in \mathcal{U}$
- Let $\mathcal{A} \subset \mathcal{U}$ be the subset of users chosen by scheduling algorithm

System Model

- Let $\mathbf{H}_{\hat{k}} = \mathbf{U}_{\hat{k}} \mathbf{D}_{\hat{k}} \mathbf{V}_{\hat{k}}^{\mathrm{H}}$ be singular value decomposition of $\mathbf{H}_{\hat{k}}$
- Let $k = \kappa \hat{k} + i$ be the virtual user corresponding to the spatial stream $i \in \{0, \ldots, \kappa - 1\}$
- Using this, we denote virtual channel $\mathbf{h}_k = \mathbf{U}_{\hat{k}}(i)^{\mathrm{H}} \mathbf{H}_{\hat{k}}$, where $\mathbf{U}_{\hat{k}}(i)$ corresponds to the column i of $\mathbf{U}_{\hat{k}}$
- Now, the received symbol $\hat{d}_k$ of virtual user k is given as

$$\hat{d}_k = \mathbf{h}_k \mathbf{m}_k d_k + \sum_{i \in \mathcal{A} \setminus \{k\}} \mathbf{h}_k \mathbf{m}_i d_i + n_k$$

- where $\mathbf{m}_k \in \mathbb{C}^{N_T \times 1}$ is the transmit precoder of user k

Overview of Scheduling Algorithms

- To minimize interference, only a subset of users are allowed for transmission
- Subset selection with certain objective requires exhaustive search
- Scheduling can inherently be performed by precoder designs - efficient iterative algorithms are available
- However, as the user count increases, complexity scales up significantly
- Hence, precoders are to be designed only for a subset of users chosen by scheduling algorithms

Successive Projections[†]

- Based on Gram-Schmidt Orthogonalization Procedure
- In each iteration, user channel vectors are projected on to the subspace orthogonal to the span of channel vectors already chosen
- Upon projecting on to the orthogonal subspace, resulting vector with maximum norm is chosen as the candidate user

$$\mathbf{N}(\mathcal{A}) = \mathbf{I}_{N_T} - \mathbf{F}\left(\mathbf{F}^{\mathrm{H}}\mathbf{F}\right)^{-1}\mathbf{F}^{\mathrm{H}}$$

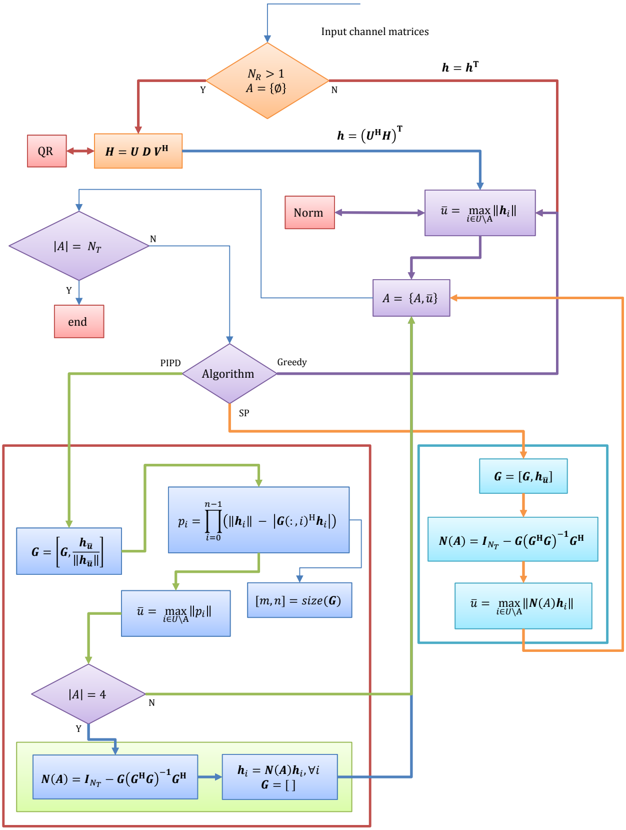
- where **F** is the matrix formed by stacking channel vector of already chosen users in $\mathcal{A}$

[†] T. Yoo and A. Goldsmith, "On the Optimality of Multi-Antenna Broadcast Scheduling using zero-forcing Beamforming, in *IEEE J. Sel. Areas Commun.*, vol. 24, no. 3. IEEE, march 2006.

Product of Independent Projections (PIPD)[†]

- As compared to the subspace projection in previous algorithm, vector projections are considered
- Each user channel is projected on to unit vector in the direction of already chosen users channel
- Selection is based on the product of independent vector projections
- Performs significantly closer to successive projections method
- Due to vector projections, inverse calculation is not required - low complexity

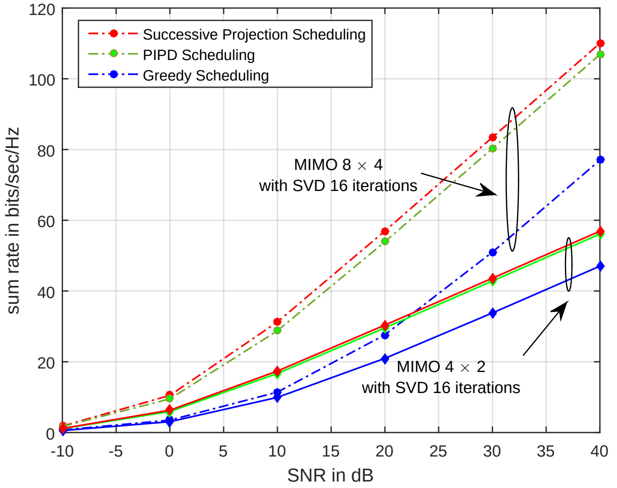
[†]Venkatraman, G., Tolli, A., Janhunen, J., and Juntti, M. "Low Complexity Multi-User MIMO Scheduling for Weighted Sum Rate Maximization", in *Proc. of European Signal Process. Conference (EUSIPCO)*, pp. 820–824, 2013



Input channel matrices
$N_R > 1$
$A = \{\emptyset\}$
Y
N
$\boldsymbol{h} = \boldsymbol{h}^{\mathrm{T}}$
QR
$\boldsymbol{H} = \boldsymbol{U}\,\boldsymbol{D}\,\boldsymbol{V}^{\mathrm{H}}$
$\boldsymbol{h} = (\boldsymbol{U}^{\mathrm{H}}\boldsymbol{H})^{\mathrm{T}}$
Norm
$\bar{u} = \max_{i \in U \setminus A} \|\boldsymbol{h}_i\|$
$|A| = N_T$
N
Y
end
$A = \{A, \bar{u}\}$
PIPD
Algorithm
Greedy
SP
$\boldsymbol{G} = \left[\boldsymbol{G}, \frac{\boldsymbol{h}_{\bar{u}}}{\|\boldsymbol{h}_{\bar{u}}\|}\right]$
$p_i = \prod_{i=0}^{n-1} \left(\|\boldsymbol{h}_i\| - |\boldsymbol{G}(:,i)^{\mathrm{H}}\boldsymbol{h}_i|\right)$
$[m,n] = size(\boldsymbol{G})$
$\bar{u} = \max_{i \in U \setminus A} \|p_i\|$
$|A| = 4$
N
Y
$\boldsymbol{N}(\boldsymbol{A}) = \boldsymbol{I}_{N_T} - \boldsymbol{G}(\boldsymbol{G}^{\mathrm{H}}\boldsymbol{G})^{-1}\boldsymbol{G}^{\mathrm{H}}$
$\boldsymbol{h}_i = \boldsymbol{N}(\boldsymbol{A})\boldsymbol{h}_i, \forall i$
$\boldsymbol{G} = [\,]$
$\boldsymbol{G} = [\boldsymbol{G}, \boldsymbol{h}_{\bar{u}}]$
$\boldsymbol{N}(\boldsymbol{A}) = \boldsymbol{I}_{N_T} - \boldsymbol{G}(\boldsymbol{G}^{\mathrm{H}}\boldsymbol{G})^{-1}\boldsymbol{G}^{\mathrm{H}}$
$\bar{u} = \max_{i \in U \setminus A} \|\boldsymbol{N}(A)\boldsymbol{h}_i\|$

Figure: Block diagram of the scheduler algorithms: greedy, PIPD and successive projections

Figure: Comparison of Scheduler Algorithms for $K = 100$ users.



Overview of TIC6636K2H Eight Core SoC

- Four ARM Cortex A15 operating at 1.4GHz
- Eight C66x CorePacs DSP Core Subsystems 1.2GHz
- 1024K Byte Local L2 Per CorePac
- 6 MB MSM SRAM Memory Shared by DSP CorePacs and ARM CorePac
- TeraNet Fabric interconnect between core subsystems and peripherals
- DDR3 memory interface

Partitioning of Algorithm

- Computationally, SVD is the most demanding operation
- SVD is performed by repeated QR factorization (16 iterations)
- In order to utilize the SoC efficiently, SVD is shared among eight C66x cores
- SVD processing begins with a chip level interrupt controller (CIC) interrupt from Core(0)
- Channel matrices are stored in MSM SRAM memory, which is accessible to all C66x cores
- Upon completion, Core(0) carries out scheduler design until completion

Implementation of Scheduler Algorithm

- Each core runs separate copy of SYS-BIOS
- Inter core communication is facilitated using MCSDK 3.0 software stack
- Storage address of the channel buffer in MSM SRAM is fixed across cores using #pragma location
- Avoids the usage of *SharedMem* and *Notify* modules to attain the same
- *IPC* module is used to synchronize the cores upon BIOS_Start() function call

Core(0) Implementation

- Signed Q1.15 format for real and imaginary entries
- CIC interrupt from Core (0) is used to notify the availability of channel buffer to other cores
- Cache *write-back* is performed upon completing SVD processing by all cores
- Upon completion, Core (0) proceeds with scheduling algorithm processing
- However, in a dynamic scenario, CIC interrupt can be used to notify the completion from other cores
- Number of SVD's per core -
  - Core(0) - $\left\lfloor \frac{K}{N_C} \right\rfloor + \left(K - \left\lfloor \frac{K}{N_C} \right\rfloor \times N_C \right)$
  - Other cores - $\left\lfloor \frac{K}{N_C} \right\rfloor$

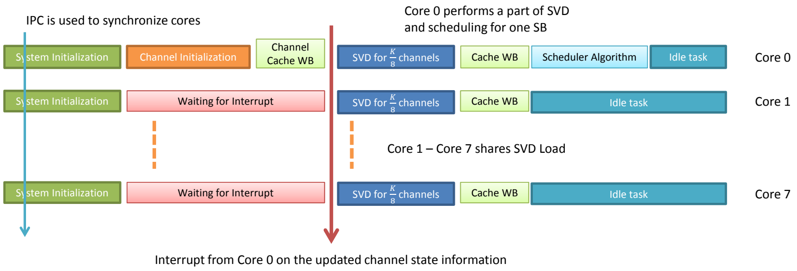

IPC is used to synchronize cores
Core 0 performs a part of SVD and scheduling for one SB
System Initialization
Channel Initialization
Channel Cache WB
SVD for $\frac{K}{8}$ channels
Cache WB
Scheduler Algorithm
Idle task
Core 0
System Initialization
Waiting for Interrupt
SVD for $\frac{K}{8}$ channels
Cache WB
Idle task
Core 1
Core 1 – Core 7 shares SVD Load
System Initialization
Waiting for Interrupt
SVD for $\frac{K}{8}$ channels
Cache WB
Idle task
Core 7
Interrupt from Core 0 on the updated channel state information

Figure: Task scheduling over $N_C = 8$ cores.

Table: Scheduling Complexity for $K = 100$ users (msec) with C66x operating at 1.2GHz

$N_T \times N_R$	λ	SVD (1)	SVD (8)	Greedy	SP	PIPD
8×4	4	22.68	2.90	0.075	0.524	0.469
8×4	2	22.68	2.90	0.064	0.325	0.268
8×2	2	6.055	0.79	0.063	0.325	0.266
8×2	1	6.055	0.79	0.058	0.226	0.166
4×4	4	15.81	2.07	0.045	0.168	0.167
4×4	2	15.81	2.07	0.034	0.102	0.098
4×2	2	4.844	0.64	0.034	0.102	0.097
4×2	1	4.844	0.64	0.029	0.069	0.063

Conclusion from Implementation Results

- Current design can handle all scheduling algorithms within 0.5 msec duration
- Moreover, with 8 parallel cores, it can support 8 scheduling blocks (SBs) within 0.5 msec
- However, the complexity is mainly attributed by the SVD processing
- With current implementation, it can support the MIMO configuration of 8×2 system for $K = 100$ users
- It is valid only if the channel changes onces in every radio frame of 10 msec duration
- In 4×2 configuration, current implementation supports 2 SBs using earlier assumption

Conclusions

- We studied the implementation of different state-of-the-art MU-MIMO scheduling algorithms on TIC6636K2H
- Complexity is mainly attributed to SVD decomposition of channel matrices
- Using parallel implementation, current design can support 100 SVD of 8×2 matrices in 6.055 msec
- We have demonstrated that with the current implementation, all scheduling schemes meet the real-time requirements
- Even though we considered only single SB, the above implementation is scalable.