Åbo Akademi University

Decoding of DVB-T2 LDPC Codes on a TILE Processor: Optimisations and Performance Comparisons

Sudeep Kanur[1], **Stefan Grönroos**[2,1], Kristian Nybom[1], Jerker Björkqvist[1], and Johan Lilius[1,2]

[1]Åbo Akademi University, Turku, Finland.

[2]TUCS- Turku Centre for Computer Science, Turku, Finland

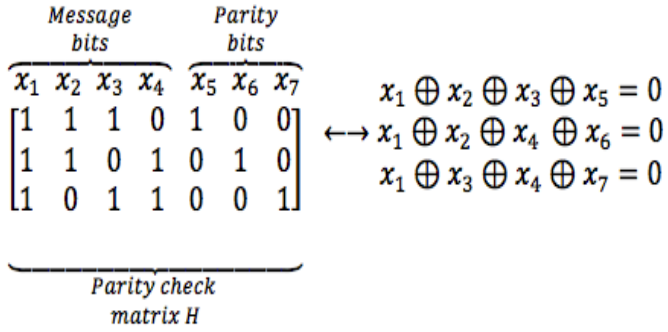
Background

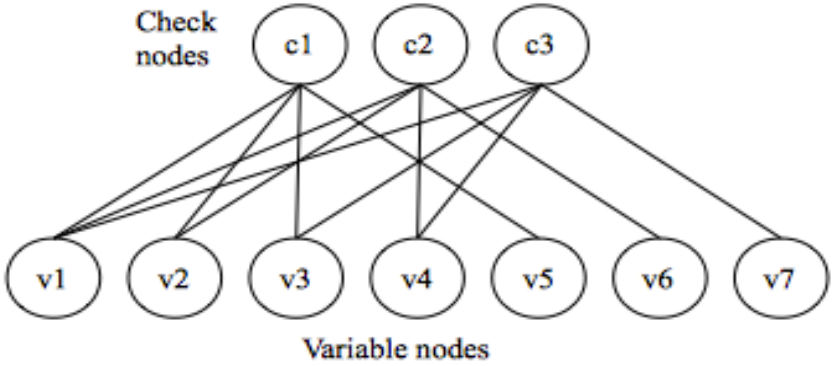
- This paper is part of a series of papers
- 2010: Measured complexity of the signal processing blocks of DVB-T2 standard
  - Conclusion: LDPC FEC decoding is very computationally demanding
- 2011-2012: Presented a real-time capable LDPC decoder on a GPU, and a fast decoder on a desktop Intel CPU
- 2013: Measured the performance of LDPC decoding on multi-core ARM Cortex-A9 processor
- Now: Analysis of LDPC decoding on a Tilera TilePro 64-core tile processor

Introduction

- **LDPC** codes are extensively used in second generation Digital Video Broadcasting standards for **forward error correcting** schemes
- **Decoding** of LDPC codes is a **NP hard** problem
- **Iterative decoding** scheme is common
- Multiple code rates + Codeword lengths of 16200 bits & 64800 bits further complicate decoding problem

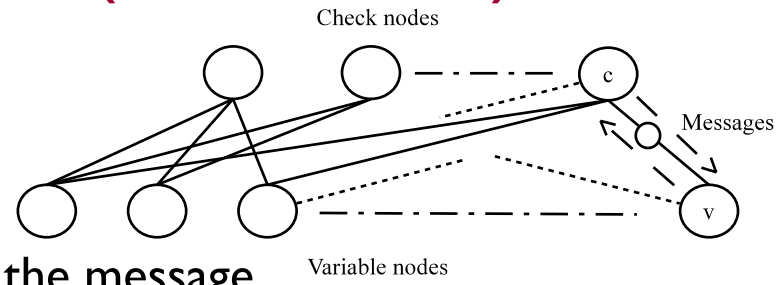
Low Density Parity Check codes





- LDPC code of length *n* bits consists of *k* bits of information & $n - k$ bits of redundancy called parity bits.
  - Code rate: k/n
- Relationship between information & parity bits is linear given by matrix called parity-check matrix
- Parity-check matrix can be represented as bipartite graph – Tanner graph
- Tanner graphs help us understand decoding algorithm

Decoding Algorithm (Min Sum)



Four step iterative algorithm, for iteration j

1. Initialisation – Each variable node v sends the message

$$L_{v\to c}(x_v) = LLR(v)$$

2. Check node update – Each check node c sends the message

$$L_{c\to v}(x_v) = \left(\prod_{v'\in V(c)\setminus v} \operatorname{sign}(L_{v'\to c}(x_{v'})) \right) \times \min_{v'\in V(c)\setminus v} |L_{v'\to c}(x_{v'})|$$

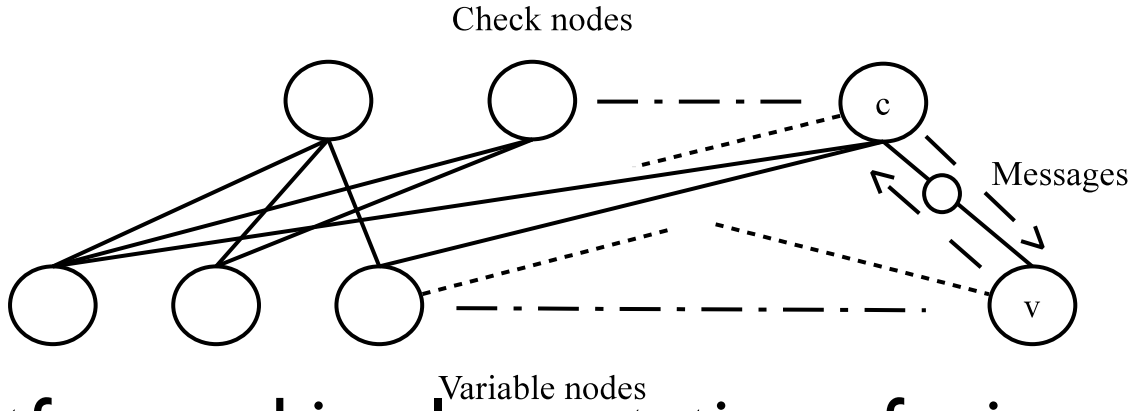
3. Variable node update – Each variable node v sends the message

$$L_{v\to c}(x_v) = LLR(v) + \sum_{c'\in C(v)\setminus c} L_{c'\to v}(x_v)$$

$$L_v(x_v) = LLR(v) + \sum_{c\in C(v)} L_{c\to v}(x_v)$$

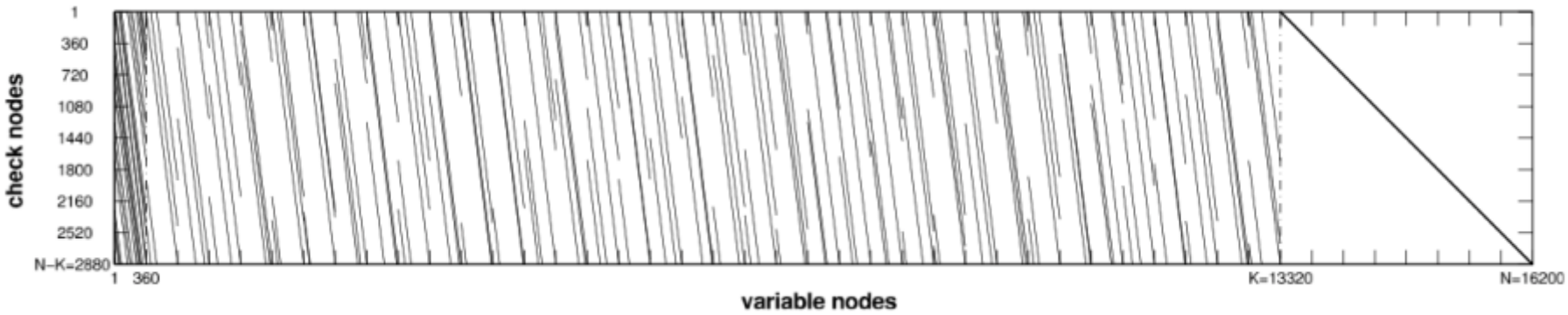
4. Decision – Quantize xv such that xv = 1if Lv(xv) < 0, and xv = 0 otherwise. If $H.x^T = 0$, x is a valid codeword and the decoder outputs x. Otherwise go to step 2.

Implementation I – Baseline method



- Straightforward implementation of min-sum algorithm
- Messages are stored in an array and are indexed from check node's perspective. There are as many messages as there are edges
- Hence from variable node's perspective its results in irregular memory access

Implementation II – Strand Method

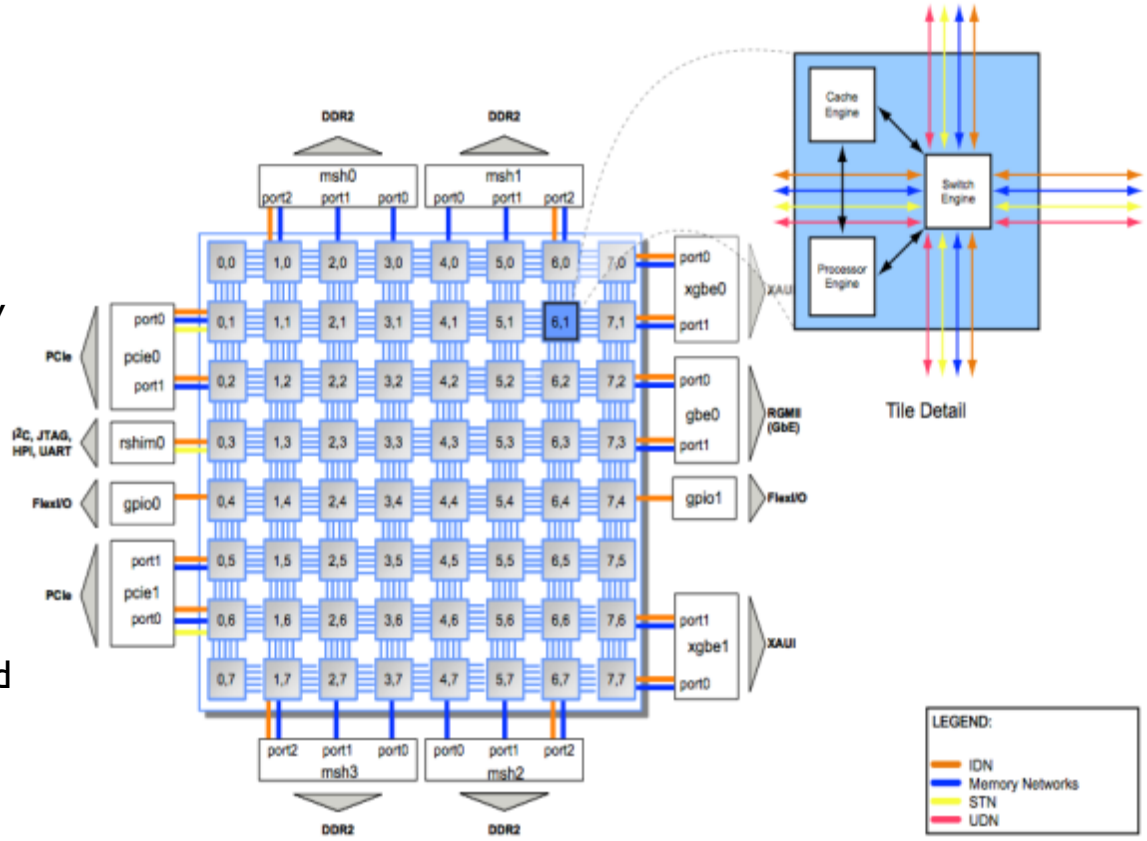


- This methodology behind Strand implementation is influenced by [1].
- Utilizes block-circulant nature of H matrix defined by standard
- Grouped by 360 nodes, with each diagonal (strands) given by $S_{q,d} = (r_{c_q}(q) + jQ) mod((n-k), j)$
- Where j = 1 … 360, q = 1 …. Q, Q = rate dependent constant defined in standard

[1] A.Jimenez-Pacheco and O.Dabeer, "A novel conflict free memory and processor architecture for DVB-T2 LDPC decoding", in Ultra Modern Telecommunications and Control Systems Workshops (ICUMT), 2011 3rd International Congress on, 2011.

TILEPRO64 Processor

- (32bit, 3 instruction wide integer VLIW engine with instruction fetch unit, execution units, memory management unit incl TLBs, 64 entry register file, two level cache) x 64 connected in 2D mesh
- Cache coherent shared memory maintained by hardware
- 4 DDR2 RAM with 64 bit interfaces and operates in striped memory configuration
- Has DMA engine & supports vector operations
- C/C++ applications with pthread library support
- SMP Linux run in Zero Overhead Linux Mode – one thread/tile & no interrupt overhead.



Experimental Setup (1/2)

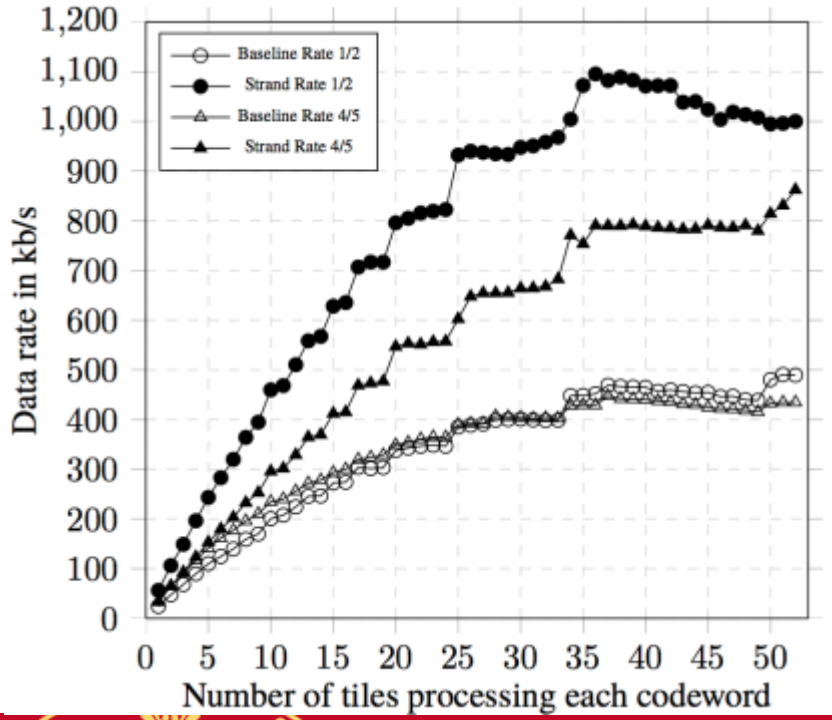
- Throughput rate is product of performance per tile and scaling across multiple tiles
  - 100 codewords are decoded & average taken
- Thread pools used and are pinned to tiles starting from 0.
- Throughput rates measured for worst case performance, i.e. for **30 iterations**
- Feedback based optimization used to decrease code footprint, i.e. improve I-cache hit ratio
- Hash for home strategy enables fetching data from tiles of other caches (acts as L3 cache)
- Data structures aligned to TLB boundaries

Experimental Setup (2/2)

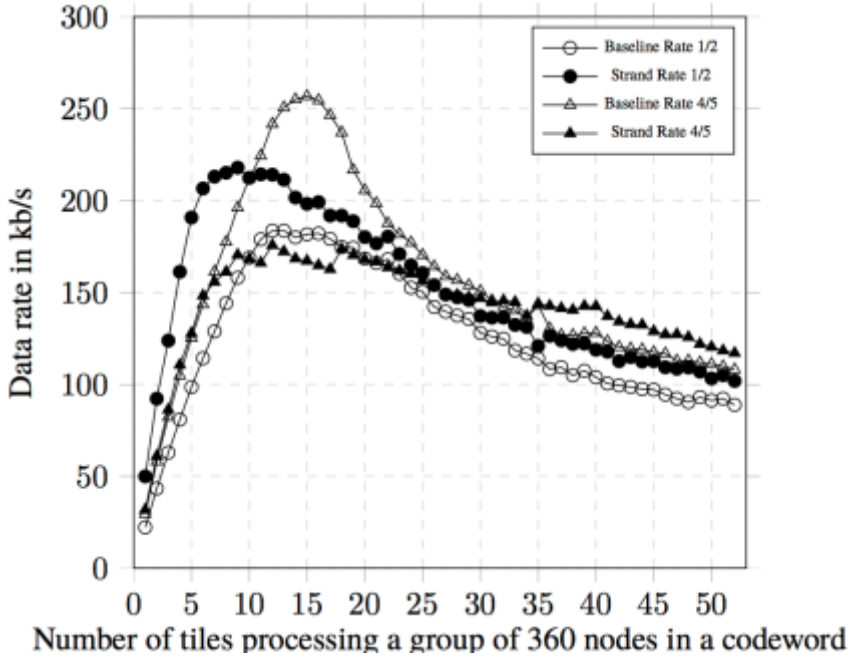
- Data & task parallelism used. Best strategy decided by 4 experiments. Code rates 4/5, 1/2 characterized by 32000x64800 and 12960x64800 used.
  1. Pure data parallelism
  2. Pure task parallelism. 360 nodes grouped together
  3. Data & task parallelism. 2 level thread pools in **blocking** mode. First level data parallel, second level task parallel. When no threads are available in second level, first level waits.
  4. Data & task parallelism. 2 level thread pools in **non-blocking** mode. Same as above. But when no threads are available in second level, first level continues.

Experiment & Results (1/2)

- Data parallel only
- Maximum Throughput of 5
- Bumps are seen as distance from RAM decreases.
- Memory bound

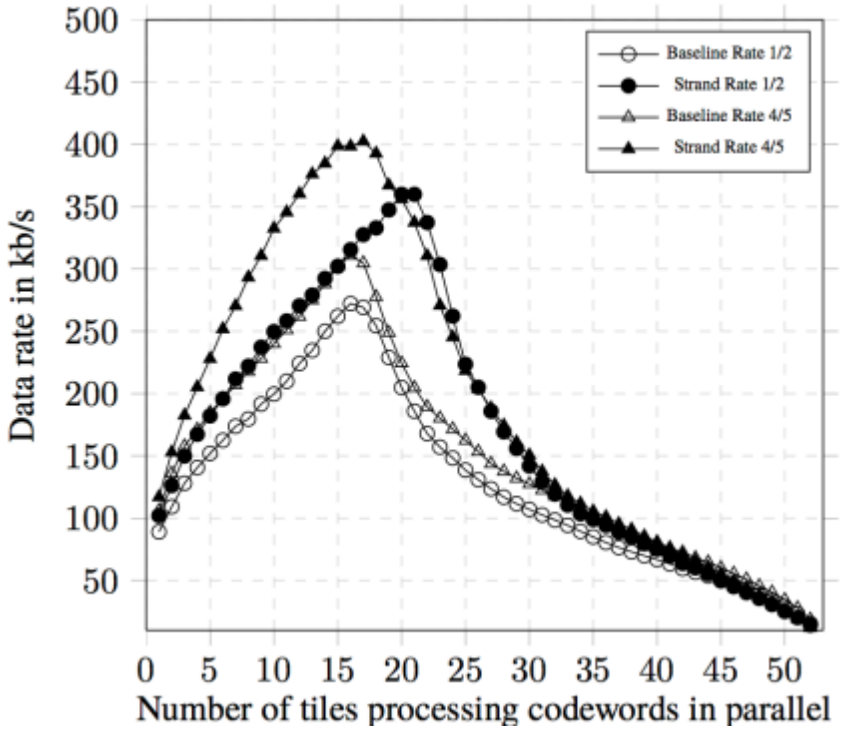


- Task parallel only
- Worst performance of 5
- No gain in performance beyond a certain point
- Irregular memory access stalls the processors

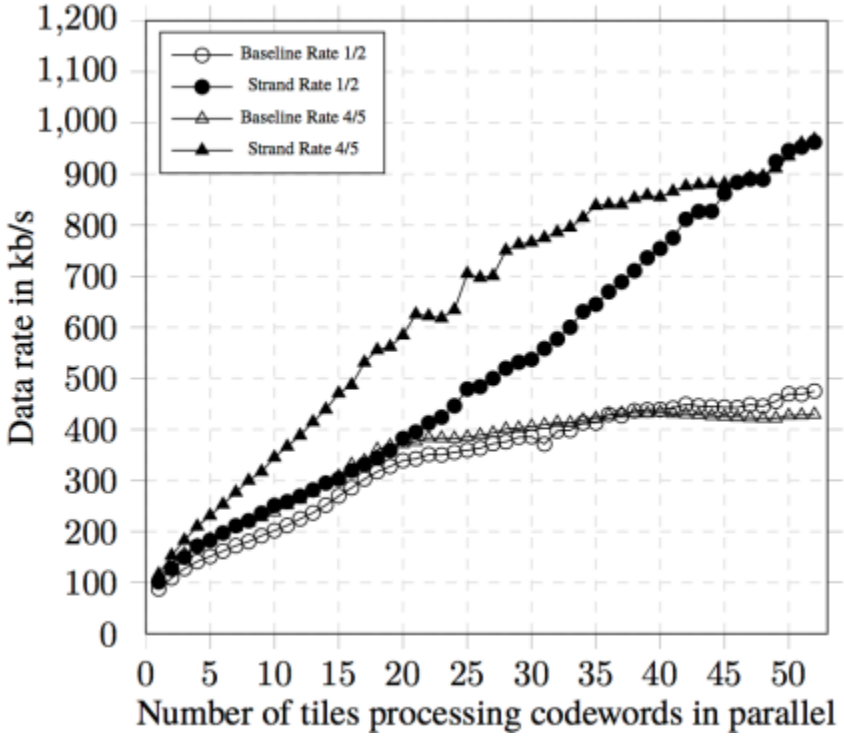


Experiments & Results (2/2)

- Similar to task parallel execution
- Inverse relation between data tiles & task parallel tiles, only peak point shifts. Curve largely remains same



- Best of both worlds
- Almost linear increase
- Performance still less than 1st Exp
- No bumps, tiles are efficiently used



Summary (1/2)

- Max throughput of 1.09 Mbps for rate ½ and 970 kbps for rate 4/5
  - In data parallel only scenario
- Though non-real time, some insights were obtained about scalability on NoC platforms
- Strand method performs **better** to Baseline
- The trend in Exp 4 shows further scalability with introduction of cores
- Irregular Memory accesses still a problem

Summary (2/2)

- Implementation is still far from being complete.
- Optimizations such as vector processing, efficient utilization of DMA engine is left out.
- Throughput per watt is still to be measured

Åbo Akademi University

THANK YOU!

Sudeep Kanur[1], Stefan Grönroos[2,1], Kristian Nybom[1], Jerker Björkqvist[1], and Johan Lilius[1,2]

[1]Åbo Akademi University, Turku, Finland.

[2]TUCS- Turku Centre for Computer Science, Turku, Finland

Email: firstname.lastname@abo.fi

References used in presentation

1. A.Jimenez-Pacheco and O.Dabeer, "A novel conflict free memory and processor architecture for DVB-T2 LDPC decoding", in Ultra Modern Telecommunications and Control Systems Workshops (ICUMT), 2011 3rd International Congress on, 2011, pp.1-7

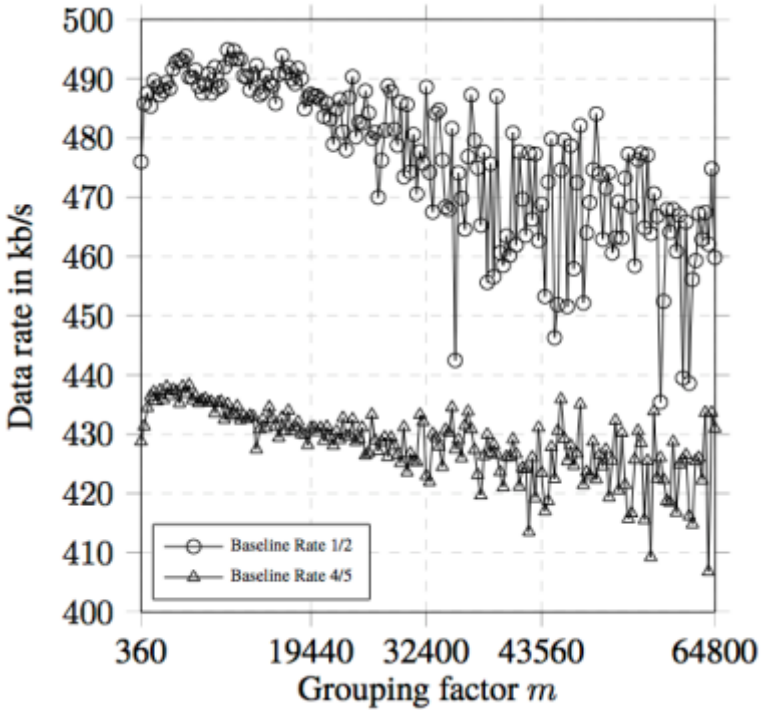
Introduction (2/2)

- FPGA & ASICs provide low power, real time solution but lack programmability
- CPU+GPU also achieve real time throughputs & provide programmability but with high power consumption
- Low-power **Multicore processors** provide middle ground

Experiments & Results (3/3)

- **Grouping provides no noticeable advantage for Baseline method**
- **Its irrespective of number of tiles present for data parallelism & task parallelism**

Purely data parallel



Data & task parallel – non blocking with ratio 1:1