Evaluation of Dataflow Process Networks Mapping on Multi-core Processors

Tania Shazadi

MSc. Computer Science

Supervisor & Examiner: Prof. Dr. Klaus Schneider

Second Supervisor: Yu Bai

9th December, 2015

Outline

- Motivation of the Research
- Problem Statement
- Mapping Techniques
- Experiments and Results
- Conclusion and Future Work

Motivation

- Several models of computation (MoC) have been developed
- Synchronous models are particularly well suited for analysis
- Translating them into a dataflow process network for efficient synthesis
- Desynchronization of synchronous systems
- Mapping of DPN on a multicore architecture is left

Problem Statement

“Evaluation of Dataflow Process Networks Mapping on Multi-core Processors”

- Dataflow process network model
- Mapping techniques
- Architecture model

Experimental Setup

- Language:
  For the representation of DPNs: CAL
- Tool:
  For the compilation of DPN applications: Orcc
- Mapping/partitioning techniques:
  Partitioning the DPN into as many threads as the processor has cores
- Scheduling techniques:
  Scheduling the processes of one partition into a single thread that can then run on a core

CAL

- CAL is an actor language
- Developed by Eker and Janneck at U.C. Berkeley
- A complete programming language for design and implementation of embedded software and FPGAs (field programmable gate array applications)

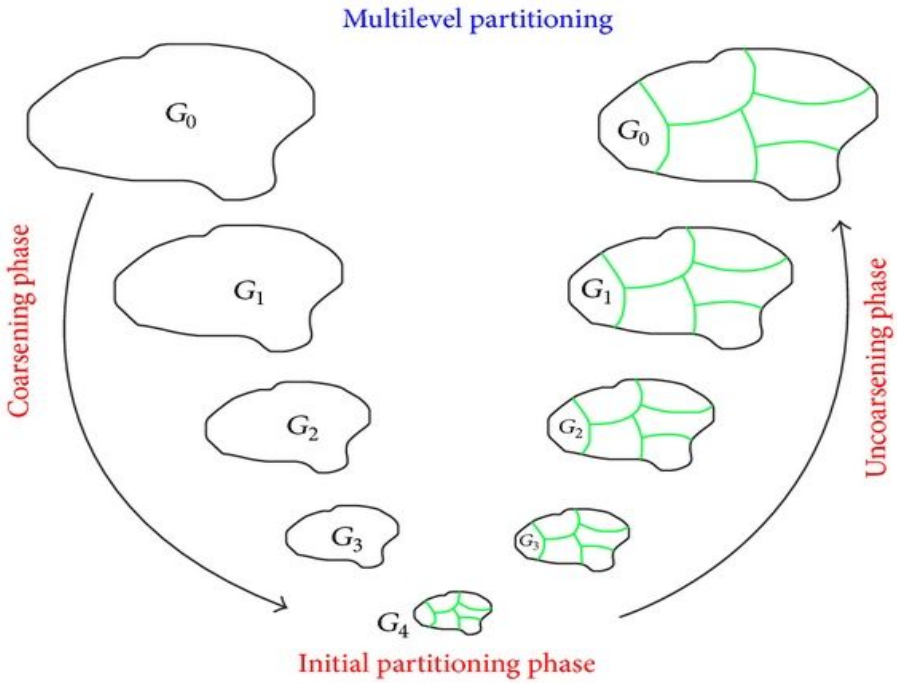
ORCC

- Open RVC-CAL Compiler (Orcc) is a plug-in in Eclipse
- Supports RVC-CAL code
- RVC-CAL is a subset of CAL that is standardized along MPEG
- Can compile RVC-CAL code for various platforms, such as C, Jade, TTA, and Xronos Verilog

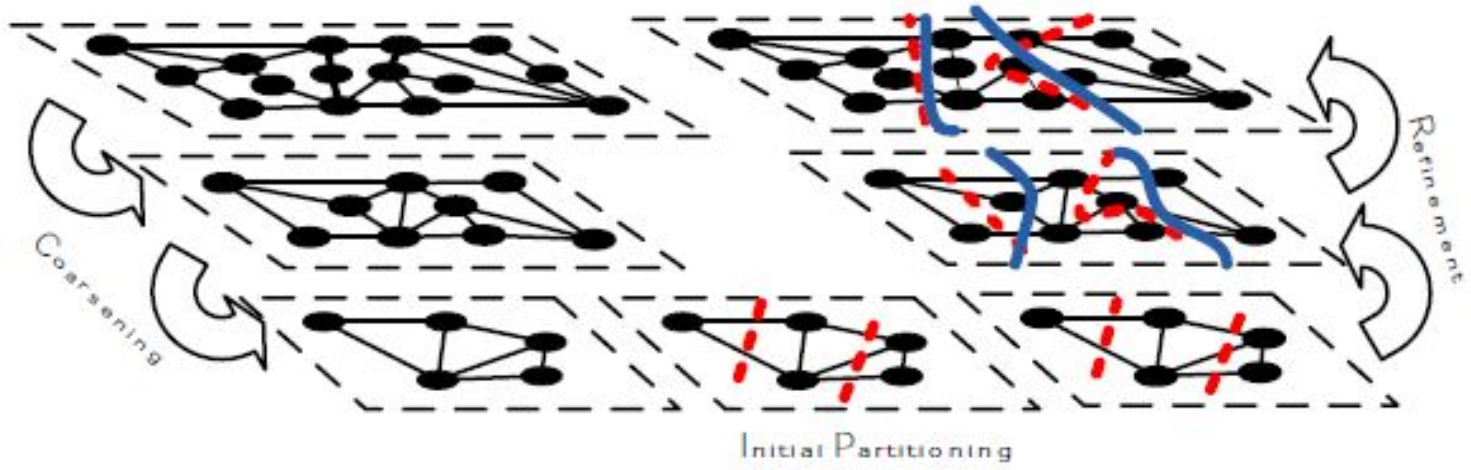
Mapping Techniques

- MR: METIS Recursive graph partition mapping
- MKCV: METIS *KWay* graph partition mapping (Optimize Com-
- munication volume)
- MKEC: METIS *KWay* graph partition mapping (Optimize Edge-cut)
- WLB: Weighted Load Balancing
- RR: A simple Round-Robin mapping
- KLR: Kernighan Lin Refinement Weighted Load Balancing

Metis Partitioning Process

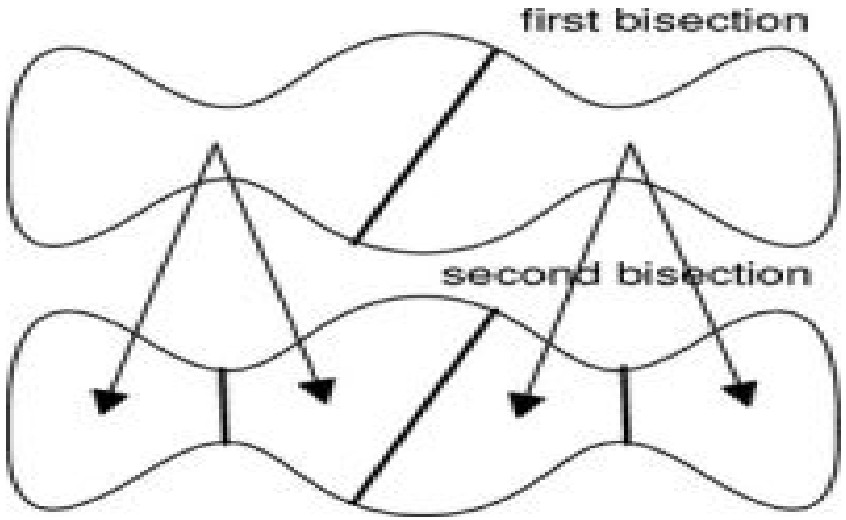


Continued..,


Coarsening
Refinement
Initial Partitioning

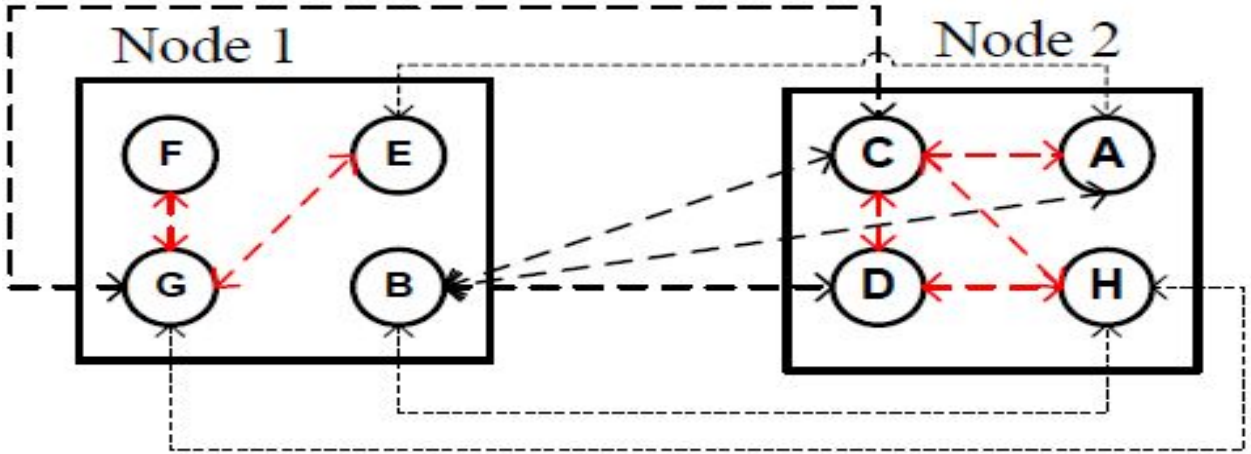

Fine Graph
Coarse Graph

MR



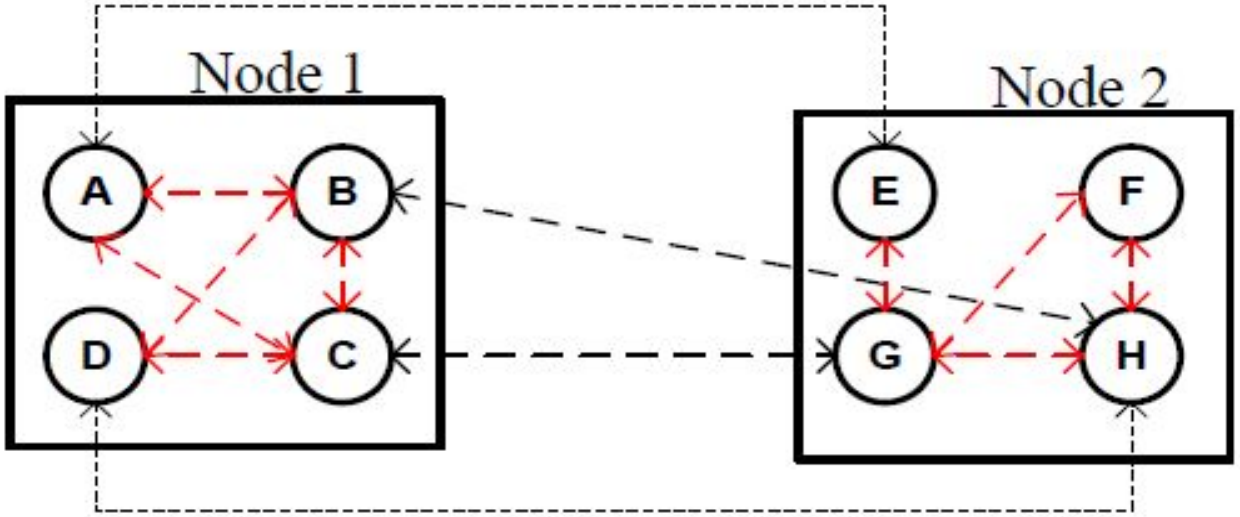
An example of recursive-bisection graph partitioning

MKCV



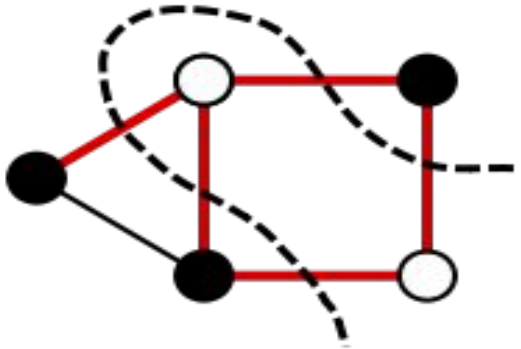
Example of reduced Inter-node communication before graph partitioning

Continued..,

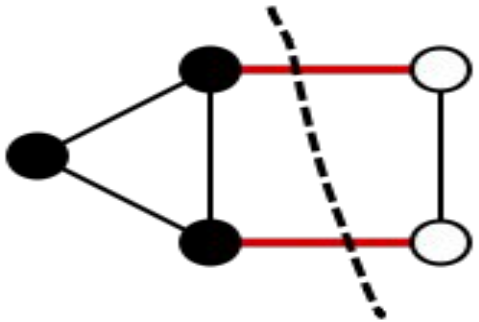


Reduced Inter-node communication after graph partitioning

MKEC

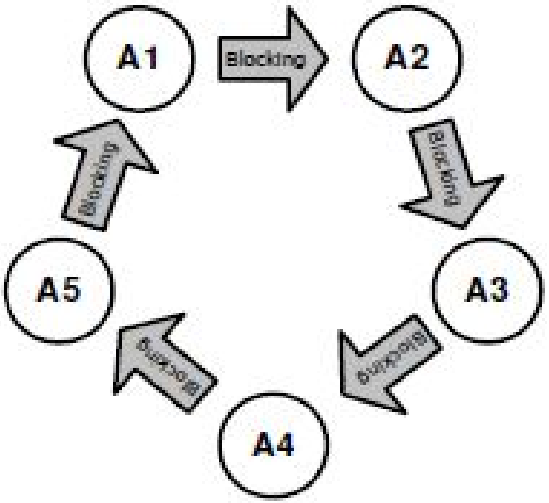


Maximum Edge-cut



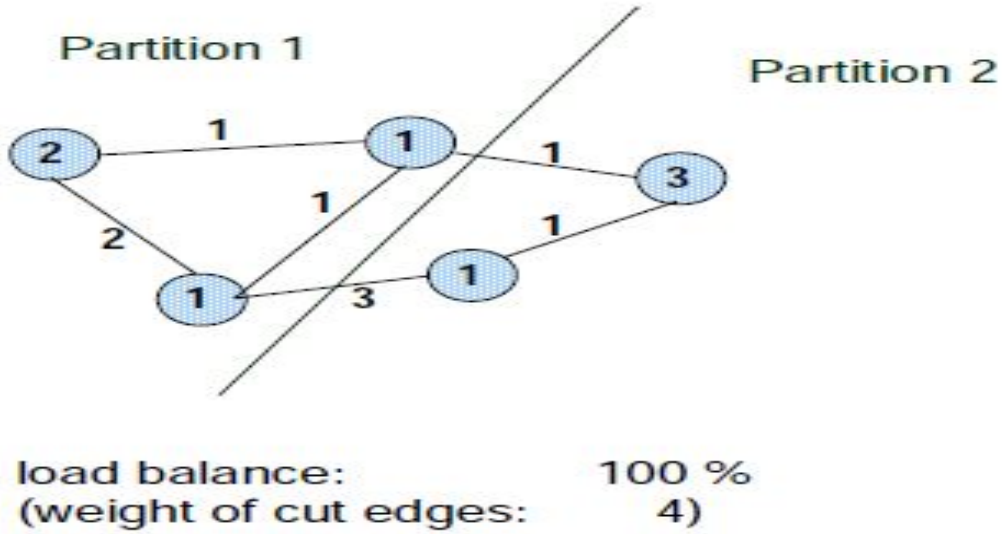
Minimum Edge-cut

RR



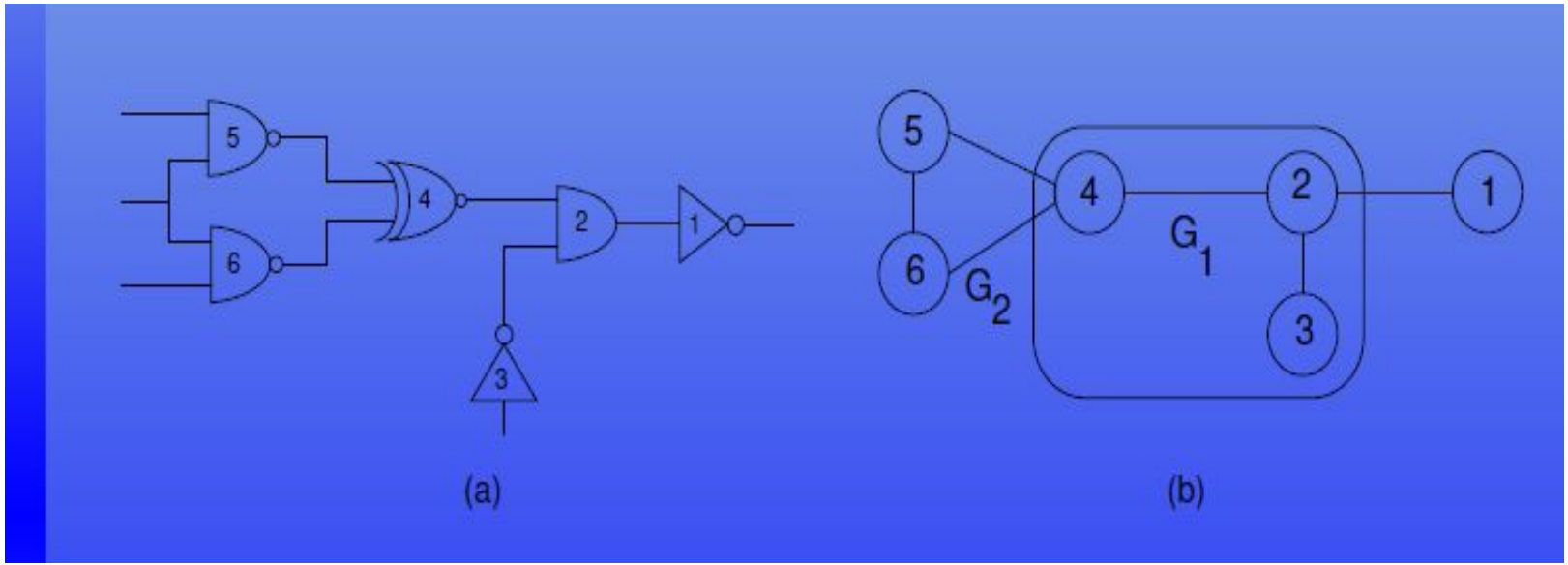
An example of RR mapping

WLB



Equal number of nodes within each partition

KLR



An example of KLR graph partitioning

Continued..,

Step 1: Initialization.

Let the initial partition be a random division of vertices into the partition $A=\{2,3,4\}$ and $B=\{1,5,6\}$.

$A' = A = \{2,3,4\}$, and $B' = B = \{1,5,6\}$.

Continued..,

- Step 2: Compute $D-$values.

$$
\begin{aligned}
D_1 &= E_1 - I_1 = 1 - 0 = +1 \\
D_2 &= E_2 - I_2 = 1 - 2 = -1 \\
D_3 &= E_3 - I_3 = 0 - 1 = -1 \\
D_4 &= E_4 - I_4 = 2 - 1 = +1 \\
D_5 &= E_5 - I_5 = 1 - 1 = +0 \\
D_6 &= E_6 - I_6 = 1 - 1 = +0
\end{aligned}
$$

Continued..,

- Step 3: Compute gains.

$$
\begin{aligned}
g_{21} &= D_2 + D_1 - 2c_{21} = (-1) + (+1) - 2(1) = -2 \\
g_{25} &= D_2 + D_5 - 2c_{25} = (-1) + (+0) - 2(0) = -1 \\
g_{26} &= D_2 + D_6 - 2c_{26} = (-1) + (+0) - 2(0) = -1 \\
g_{31} &= D_3 + D_1 - 2c_{31} = (-1) + (+1) - 2(0) = +0 \\
g_{35} &= D_3 + D_5 - 2c_{35} = (-1) + (+0) - 2(0) = -1 \\
g_{36} &= D_3 + D_6 - 2c_{36} = (-1) + (+0) - 2(0) = -1 \\
g_{41} &= D_4 + D_1 - 2c_{41} = (+1) + (+1) - 2(0) = +2 \\
g_{45} &= D_4 + D_5 - 2c_{45} = (+1) + (+0) - 2(1) = -1 \\
g_{46} &= D_4 + D_6 - 2c_{46} = (+1) + (+0) - 2(1) = -1
\end{aligned}
$$

- The largest g value is g_{41}. (a_1, b_1) is $(4, 1)$, the gain $g_{41} = g_1 = 2$, and $A' = A' - \{4\} = \{2,3\}$, $B' = B' - \{1\} = \{5, 6\}$.

Continued..,

- Step 4: Update $D-$values of nodes connected to (4,1).

  The vertices connected to (4,1) are vertex (2) in set A' and vertices (5,6) in set B'. The new $D-$values for vertices of A' and B' are given by

$$D_2^{'} = D_2 + 2c_{24} - 2c_{21} = -1 + 2(1-1) = -1$$
$$D_5^{'} = D_5 + 2c_{51} - 2c_{54} = +0 + 2(0-1) = -2$$
$$D_6^{'} = D_6 + 2c_{61} - 2c_{64} = +0 + 2(0-1) = -2$$

Continued..,

- Step 5: Determine k.
- We see that $g_1 = +2$, $g_1 + g_2 = -1$, and $g_1 + g_2 + g_3 = 0$.
- The value of k that results in maximum G is 1.
- Therefore, $X = \{a_1\} = \{4\}$ and $Y = \{b_1\} = \{1\}$.
- The new partition that results from moving X to B and Y to A is, $A = \{1, 2, 3\}$ and $B = \{4, 5, 6\}$.
- The entire procedure is repeated again with this new partition as the initial partition.

METIS Library

- A C program which can partition a graph, a finite element mesh, or reorder a sparse matrix
- Can be called from C/C++ and Fortran
- For large graph one may require machines with extra memory, example - 64 Gbytes of memory instead of 8 Gbytes

Other libraries: Chaco (Sandia), Scotch (INRIA), Jostle (now commercialized), Zoltan (Sandia)

Continued..,

- Associated executable programs with the **METIS** library
- **Kmetis** or **pmetis**, for partitioning the nodes of a graph
- **Partdmesh** or **partnmesh**, for partitioning the elements of a finite element mesh
- **Oemetis** or **onmetis**, for reordering the variables in a sparse matrix
- **Mesh2dual** or **mesh2nodal**, for converting a mesh into the dual or nodal graph
- **Graphchk**, for error checking a graph file

Scheduling Techniques

- A combined version of the round-robin and data-driven / demand-driven strategies
- To avoid starvation of distributed algorithm
- The scheduler applies the data-driven/demand-driven policy until its schedulable list is empty
- The round-robin policy is used until the schedulable list contains at least one actor

Dataflow Benchmarks

- Open-source applications described in a dynamic dataflow programming way using the CAL language
- The applications that are fully compliant with the Orcc toolset
- Compatible with Metis library support

Array Addition

- A simple network comprising of four actors
- Mapping on multi-core processors is not efficient
- Not much possibilities to partition the DPN

Digital Filtering

- Contains descriptions of a 4-tap FIR filter
- Few connections between 4-tap FIR filters
- The inter-processor communication overhead far greater than parallelization gain

ZigBee

- Implementation of the ZigBee protocol
- Works better with HDL-code generation
- Implementation is not still efficient on multi-core architecture, though have more actors
- Hence, the Predistortion applications considered

Predistortion

- A parallel Hammerstein digital predistortion filter with 5 FIR filter branches
- Each branch with 5 taps
- Native functions are provided: linux.c, linuxbin.c, linuxmulti.c
- First reads and writes text files and the latter binary files
- The FIR filter and the local oscillator leakage compensation coefficients
- Hardcoded into the RVC-CAL sources in the implementation

Experiments and Results

- To find out best possible graph partitioning strategy
- To run equal amount of work on multi-core processor architecture
- Different hardware architecture

CPU: Intel® CoreTM i5-3337U CPU @ 1.80GHz × 4

CPU: 2x Intel Xeon CPU X5450 (4x 3.0GHz) 64bit 16GB RAM, 950GB HDD

- Different input samples (1000, 5000)
- Different graph size (number of actors)

Predistrotion_application1 (No. of Actors: 10)						
Input Samples: 1000, FIFO Size: 512, Physical Cores: 4						
Partitioning Techniques	**No. of Logical Cores**					
	2	**4**	**6**	**8**	**10**	**12**
MR	Inf FPS	Inf FPS	Inf FPS	1025 FPS	256 FPS	256 FPS
MKCV	Inf FPS	Inf FPS	Inf FPS	Inf FPS	Inf FPS	Inf FPS
MKEC	Inf FPS	Inf FPS	Inf FPS	Inf FPS	1025 FPS	205 FPS
WLB	Inf FPS	Inf FPS	341 FPS	205 FPS	205 FPS	170 FPS
RR	Inf FPS	Inf FPS	341 FPS	256 FPS	256 FPS	205 FPS

MKCV works faster on any number of logical cores

KLR ?

Continued..,

Input Samples: 5000, FIFO Size: 512, Physical Cores: 4						
Partitioning Techniques	**No. of Logical Cores**					
	2	**4**	**6**	**8**	**10**	**12**
MR	Inf FPS	Inf FPS	Inf FPS	12 FPS	17 FPS	12 FPS
MKCV	Inf FPS	Inf FPS	Inf FPS	Inf FPS	Inf FPS	Inf FPS
MKEC	Inf FPS	Inf FPS	Inf FPS	22 FPS	11 FPS	12 FPS
WLB	Inf FPS	Inf FPS	16 FPS	11 FPS	10 FPS	10 FPS
RR	Inf FPS	Inf FPS	17 FPS	10 FPS	10 FPS	10 FPS

MKCV: fastest, MR, MKEC, WLB, RR: equal, KLR ?

Continued..,

Input Samples: 1000, FIFO Size: 512, Physical Cores: 8						
Partitioning Techniques	**No. of Logical Cores**					
	2	**4**	**6**	**8**	**10**	**12**
RR	Inf FPS	Inf FPS	205 FPS	205 FPS	341 FPS	146 FPS
WLB	Inf FPS	Inf FPS	341 FPS	205 FPS	146 FPS	146 FPS

KLR struggles - since the graph size is the same

Continued..,

Input Samples: 5000, FIFO Size: 512, Physical Cores: 8						
Partitioning Techniques	**No. of Logical Cores**					
	2	**4**	**6**	**8**	**10**	**12**
RR	Inf FPS	Inf FPS	205 FPS	205 FPS	341 FPS	146 FPS
WLB	Inf FPS	Inf FPS	341 FPS	205 FPS	146 FPS	146 FPS

RR, WLB map equal number of FPS, KLR doesn't work!

Predistortion_application2 (No. of Actors: 80)

Input Samples: 1000, FIFO Size: 512, Physical Cores: 4, FPS: frames per second

Partitioning Techniques	No. of Logical Cores					
	2	4	6	8	10	12
MR	Inf FPS	Inf FPS	800 FPS	960 FPS	992 FPS	480 FPS
MKCV	Inf FPS	Inf FPS	Inf FPS	864 FPS	864 FPS	464 FPS
MKEC	Inf FPS	Inf FPS	896 FPS	960 FPS	961 FPS	448 FPS
WLB	Inf FPS	Inf FPS	896 FPS	464 FPS	432 FPS	277 FPS
RR	Inf FPS	Inf FPS	432 FPS	432 FPS	298 FPS	277 FPS
KLR	Inf FPS	Inf FPS	960 FPS	480 FPS	288 FPS	298 FPS

MR, MKCV, MKEC: map more FPS, KLR works properly

Continued..,

Input Samples: 5000, FIFO Size: 512, Physical Cores: 4						
Partitioning Techniques	**No. of Logical Cores**					
	2	**4**	**6**	**8**	**10**	**12**
MR	Inf FPS	Inf FPS	Inf FPS	Inf FPS	Inf FPS	Inf FPS
MKCV	Inf FPS	Inf FPS	Inf FPS	Inf FPS	Inf FPS	Inf FPS
MKEC	Inf FPS	Inf FPS	Inf FPS	Inf FPS	Inf FPS	Inf FPS
WLB	Inf FPS	Inf FPS	Inf FPS	1521 FPS	1521 FPS	1521 FPS
RR	Inf FPS	Inf FPS	Inf FPS	1521 FPS	1521 FPS	1521 FPS
KLR	Inf FPS	Inf FPS	Inf FPS	1521 FPS	1521 FPS	1521 FPS

WLB, RR, KLR work better with more number of Input samples & large size of graph

Continued..,

Input Samples: 1000, FIFO Size: 512, Physical Cores: 8						
Partitioning Techniques	**No. of Logical Cores**					
	2	**4**	**6**	**8**	**10**	**12**
RR	Inf FPS	Inf FPS	Inf FPS	Inf FPS	Inf FPS	Inf FPS
WLB	Inf FPS	Inf FPS	Inf FPS	Inf FPS	Inf FPS	Inf FPS
KLR	Inf FPS	Inf FPS	Inf FPS	Inf FPS	Inf FPS	Inf FPS

RR, WLB, KLR give efficient results on Octa-core processor than Quad-core processor

Continued..,

Input Samples: 5000, FIFO Size: 512, Physical Cores: 8						
Partitioning Techniques	**No. of Logical Cores**					
	2	**4**	**6**	**8**	**10**	**12**
RR	Inf FPS	Inf FPS	1521 FPS	1521 FPS	1521 FPS	2560 FPS
WLB	Inf FPS	Inf FPS	Inf FPS	Inf FPS	2560 FPS	2560 FPS
KLR	Inf FPS	Inf FPS	Inf FPS	1521 FPS	1521 FPS	2560 FPS

WLB maps more number of FPS with the increase of input samples and physical cores

Conclusion

- Overall, all techniques can map maximum number of tasks with more number of logical cores
- Also, the results depend on:

  Size of graph, number of input samples, the hardware platform (multi-core processors)

- MR, MKCV and MKEC are faster than RR, WLB and KLR
- KLR struggles where graph size is small

Future Work

- Partitioning techniques for large structured graphs
- At heterogeneous hardware platforms
- Own library for the graph partitioning techniques
- Benchmarks for different types of DPNs
- Evaluation on different operating systems

Thank you
for your attention