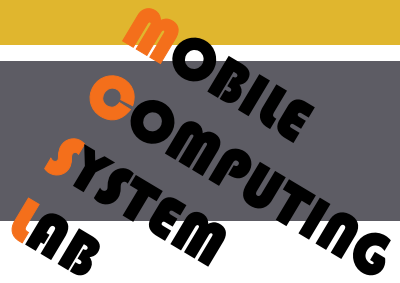
Network-on-Chip Traffic Patterns Based on Real MPSoC Applications

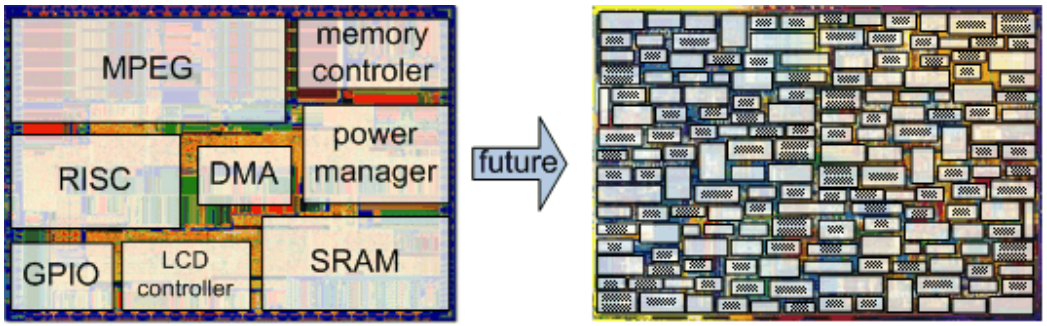
Jiang Xu



THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Multiprocessor System-on-Chip

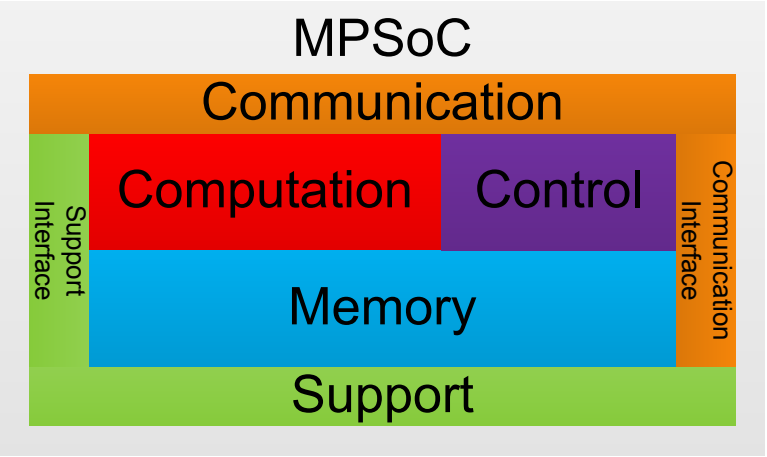
- Put all or most part of a complex system on a single chip
  - High performance, high reliability, low power consumption, compact
  - Cost-effective when volume is high
- Attractive platform for computing systems
  - The de facto choice for low-power, high-performance, high-volume, and mobile computing systems



Subsystems in MPSoC

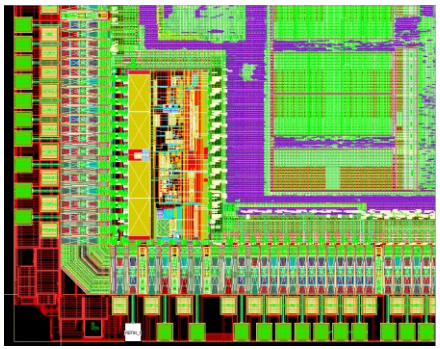
- Computation: process data, implement major functions
- Control: coordinate all the subsystems
- Memory: temporally store data, instructions, and system status
- Communication: transfer data, instructions, and other information inside, into, and out of an MPSoC
- Support: maintain appropriate operating conditions, such as power supply, clock, temperature, *etc.*

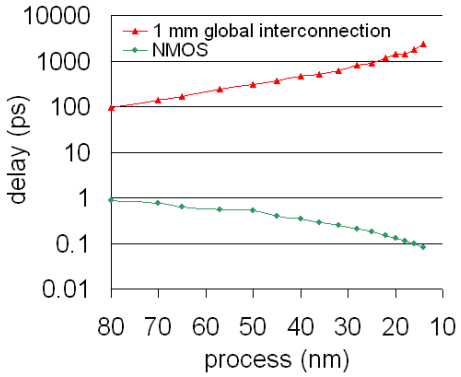
- Can physically overlap
  - E.g. computation and control
- There are grey areas
  - E.g. communication interfaces



Communication Subsystem

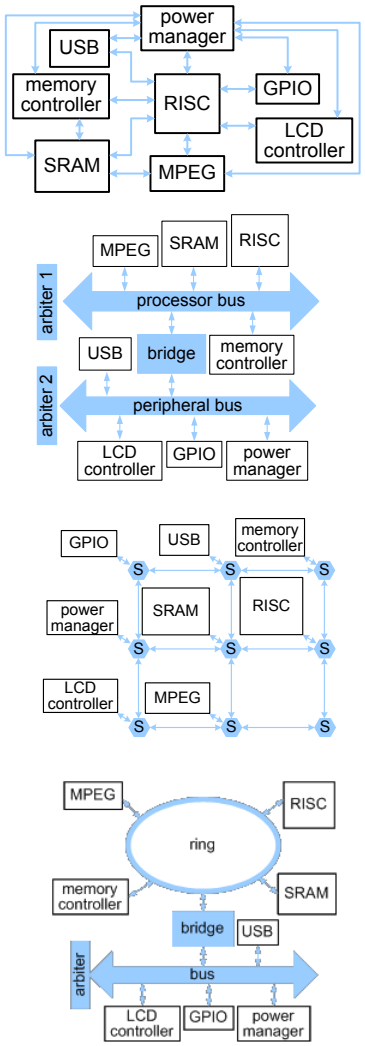
- Performance
  - Cooperation among functional units and chips
  - Communication latency
- Power consumption
  - Significant dynamic power used by communications
  - Considerable leakage of interconnect drivers and buffers
- Cost and yield
  - Metal layers for interconnects
  - Device layer for drivers and buffers
- Challenges
  - Longer communication delay
  - More on-chip communications
  - Higher power consumption





Communication Architecture

- Ad hoc interconnects
  - Dedicated point-to-point interconnections
  - Intuitive, but not cost-effective for complex systems
- Bus
  - Shared media communication architectures
  - Mature, but limited throughput and high power consumption
- Network-on-Chip (NoC)
  - Based on switching and routing techniques
  - High-throughput, scalable, and energy-efficient
  - But complex to design
- Hybrid networks
  - Very flexible
  - Complex to design and analyze
- Future NoC directions
  - Optical, nano-interconnects, wireless, *etc.*



NoC Traffic Patterns

- NoC traffic patterns are essential for
  - NoC architecture exploration
  - NoC performance evaluation
- Random/synthetic traffic patterns
  - Pro: simple to implement, pinpoint specific aspects of NoCs
  - Con: often cut off from computation and memory requirements, cannot show overall application performance and power, and application-specific issues
- Realistic traffic patterns are based on real applications
  - Pro: consider computation and memory requirements, show overall application performance and power, and application-specific issues
  - Con: difficult to get

MCSL Traffic Patterns

- MCSL stands for Multi-Constraint System-Level
  - Capture communication behaviors as well as their temporal and spatial dependencies
  - Include computation and memory requirements
- Available at www.ece.ust.hk/~eexu/index_files/traffic.htm

Application	Description	Tasks	Communication Links
FFT-1024_complex	Fast Fourier transform with 1024 inputs of complex numbers	16,384	25,600
H264-1080p_dec	H.264 video decoder with a resolution of 1080p	5,191	7,781
H264-720p_dec	H.264 video decoder with a resolution of 720p	2,311	3,461
FPPPP	SPEC95 Fpppp is a chemical program performing multi-electron integral derivatives	334	1145
RS-32_28_8_dec	Reed-Solomon code decoder with codeword format RS(32,28,8)	278	390
RS-32_28_8_enc	Reed-Solomon code encoder with codeword format RS(32,28,8)	248	328
SPARSE	Random sparse matrix solver for electronic circuit simulations	96	67
ROBOT	Newton-Euler dynamic control calculation for the 6-degrees-of-freedom Stanford manipulator	88	131

MCSL Traffic Patterns

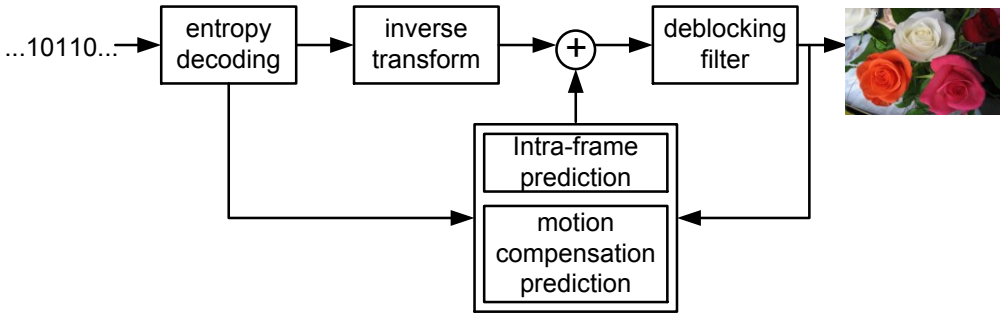
- Support different topologies and network sizes
  - Currently cover mesh, torus, and fat tree
- Two types of traffic patterns
  - Recorded traffic pattern (RTP) for detailed analysis
  - Statistical traffic pattern (STP) for average, worst, and best case studies
- RTP includes three stages
  - Ramp-up, stable, and ramp-down
- More applications will be released soon

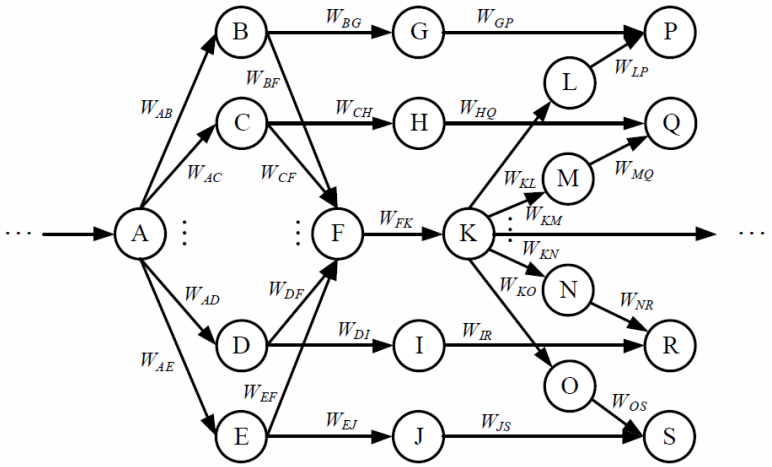
Generation Flow

- Based on HW/SW codesign methodology
- Optimized to maximize overall application performance
  - Memory space allocation
  - Application mapping and scheduling
- Modular design for future expansions and customizations

Computation and Communication Behavior Modeling

- The computation and communication requirements are captured by the application model
- Application model is a task communication graph (TCG)
  - A directed graph $G_t = (V, E)$, where V is the set of computation tasks, and E is the set of communication links between tasks. A task v has an execution time t, which is a function of task executions. A directed edge $e = (v_s, v_d, w)$ has a source task v_s, a destination task v_d and the amount of data w that sends from v_s to v_d. w is a function of communication transactions.





Memory Allocation

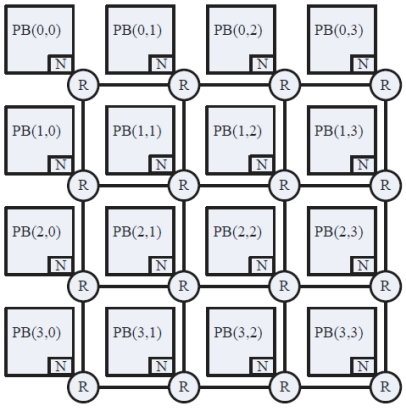
- Genetic algorithm is used to reduce memory size while maintain maximum application performance

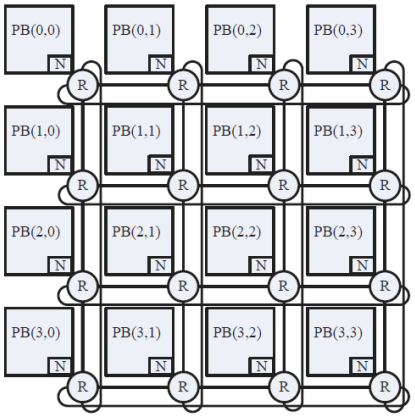
Require: application model $G_t(V, E)$
1: define pop_size, gen_num, $best\_sol$
2: $pop\_parents$ = Initialization(pop_size)
3: **while** gen_num is not reached **do**
4: $pop\_offspring = pop\_parents$
5: Crossover($pop\_offspring$)
6: Mutation($pop\_offspring$)
7: $pop\_parents$ = Selection($pop\_offspring$, pop_size)
8: $best\_sol$ = UpdateBestSolution($pop\_parents$)
9: **end while**
10: **return** the best memory space allocation $best\_sol$

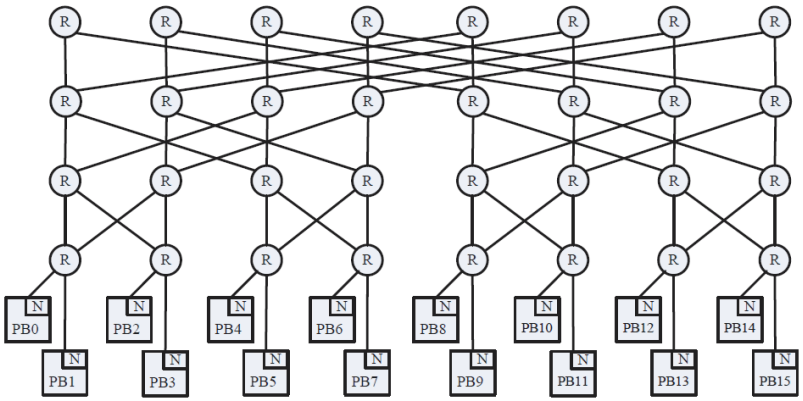
- An application memory space is created as the result
  - Different physical memory space configurations can be used

Architecture Model

- Capture hardware resources
- $G_p = (P, N)$, where P is a set of heterogeneous processing blocks (PB), and N is the topology of an on-chip communication architecture
  - PB can be processors, accelerators, memories, and/or memory interfaces
- Heterogeneous processors/accelerators
  - Using acceleration factor
- NoC topologies and sizes
  - Mesh, torus, fat-tree are covered from 4-core to 256-core in current version
  - Topology coverage can be expanded







Application Mapping and Scheduling

Require: application model $G_t(V, E)$, architecture model $G_p(P, N)$

1: $time = 0$
2: **while** there is unscheduled task in G_t **do**
3: $readyQueue$ = UpdateReadyQueue(G_t)
4: **for** each task v in $readyQueue$ **do**
5: $minWeight = \infty$
6: **for** each PB $p \in P$ **do**
7: **if** $w(v,p) < minWeight$ **then**
8: $selectedProc = p$
9: $minWeight = w(v,p)$
10: **end if**
11: **end for**
12: $m(v) = selectedProc$
13: $s(v, selectedProc)$ = GetOrder($selectedProc$)
14: $procAvailTimes$ = GetEarliestAvailableTimes(P)
15: **end for**
16: $time$ = TimeAdvance($procAvailTimes$)
17: **end while**
18: **return** mapping M, schedules S

Scaling factors

Cost function:

$$w(v,p) = c_1 t(v,p) + q c_2 n(v,p)$$

in which

Task execution

$$t(v,p) = f(p) + t(v) * a(p),$$

Task communication

$$n(v,p) = \sum_{(v,u)\in E} k(v,u) \times l(p, m(u))$$

$w(v,p)$	The weight of mapping task v to PB p
$t(v,p)$	The required time for task v to finish on PB p
$f(p)$	the time for executing previously assigned tasks on PB p
$n(v,p)$	The total amount of network transmission generated by task v when it is assigned to PB p
$k(v,u)$	The number of packets that task v generates to edge (v,u)
$m(v)$	The mapping of task v
$l(p,q)$	The distance between PB p and q (predicted according to routing policies on different NoC topologies)

Recorded Traffic Pattern

- Generated through cycle-accurate simulations
  - All task execution and task communication events are recorded
  - Temporal and spatial communication dependencies (not their exact timing) are kept
  - Distributed shared memory is used

$T_r = \{V_r(p) \mid p \in P\}$

The traffic is given by the set of tasks allocated on the PBs

$V_r(p) = \{\big(s(v), t(v), IS(v), OS(v)\big) \mid v \in V\}$

The task schedule with dependencies and exact execution times

$IS(v) = \{(v_i(e), n_i(e), m_i(e)) \mid e \in E_i(v)\}$

$OS(v) = \{(v_o(e), p_o(e), m_o(e), d_o(e)) \mid e \in E_o(v)\}$

The input and output sets with memory space allocation

RTP Data Structure

```
class RecEdge {
  int                 id;                   // the id of the edge
  RecTask*            src_task;             // the source task
  RecTask*            dst_task;             // the destination task

  vector<int>         mem_addr_list;        // the list of memory addresses
  vector<double>      rec_msg_size_list;    // the list of recorded message sizes
};

class RecTask {
  int                 id;                   // the id of the task
  RecProc*            proc;                 // the PB the task is assigned
  vector<int>         schedule_list;        // the list of task schedule sequence numbers
  vector<int>         rec_time_list;        // the list of recorded execution times

  vector<RecEdge*>    incoming_edge_list;   // each entry is an incoming edge
  vector<RecEdge*>    outgoing_edge_list;   // each entry is an outgoing edge
};

class RecProc {
  int                 id;                   // the id of the PB
  int                 row_index;            // the row index in mesh/torus
  int                 col_index;            // the column index in mesh/torus
  vector<RecTask*>    task_list;            // the list of scheduled tasks
};

class RecNOCTraffic {
  int                 topology;             // the topology code
  int                 num_row;              // the number of rows in mesh/torus
  int                 num_col;              // the number of columns in mesh/torus
  int                 num_iter;             // the number of iterations the graph executes for
  vector<RecProc>     proc_list;            // the list of PBs
  vector<RecTask>     task_list;            // the list of tasks
  vector<RecEdge>     edge_list;            // the list of edges

  vector<RecTask*>    starting_task_list;   // the list of starting tasks
  vector<RecTask*>    finishing_task_list;  // the list of finishing tasks
};
```

Statistical Traffic Pattern

- Use statistical distributions to capture the uncertainty in task execution time, data size, and packet interval

$T_s = \{V_s(p) \mid p \in P\}$

The traffic is given by the set of tasks allocated on the PBs

$V_s(p) = \{\big(s(v), D_t(v), IS(v), OS(v)\big) \mid v \in V\}$

The tasks are scheduled, with dependencies and execution times

$IS(v) = \{(v_i(e), n_i(e), m_i(e)) \mid e \in E_i(v)\}$

$OS(v) = \{(v_o(e), p_o(e), m_o(e), D_d(e), D_i(e)) \mid e \in E_o(v)\}$

The input and output sets, with memory space allocation

$D_t(v) = (\mu_t(v), \sigma_t(v))$

Gaussian distribution of task execution time

$D_d(e) = (\mu_d(e), \sigma_d(e))$

Gaussian distribution of data size

$D_i(e) = \lambda_i(e)$

Negative exponential distribution of packet generation interval

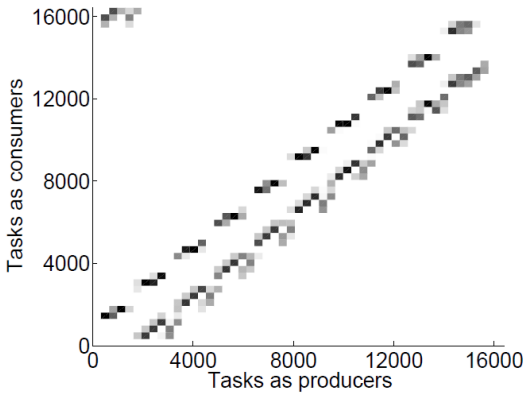
STP Data Structure

```
class StatEdge {
  int                 id;                    // the id of the edge
  StatTask*           src_task;              // the source task
  StatTask*           dst_task;              // the destination task

  int                 mem_start_addr;        // the starting address of the memory
  int                 mem_size;              // the size of the memory
  double              mu_msg_size;           // the mean of the message size
  double              sigma_msg_size;        // the sd of the message size
  double              lambda_pkt_interval;   // the rate parameter, the inverse of
                                             // the average packet generation interval
};

class StatTask {
  int                 id;                    // the id of the task
  StatProc*           proc;                  // the PB the task is assigned
  int                 schedule;              // the task schedule sequence number
  double              mu_time;               // the mean of the task execution time
  double              sigma_time;            // the sd of the task execution time

  vector<StatEdge*>   incoming_edge_list;    // each entry is an incoming edge
  vector<StatEdge*>   outgoing_edge_list;    // each entry is an outgoing edge
};

class StatProc {
  int                 id;                    // the id of the PB
  int                 row_index;             // the row index in mesh/torus
  int                 col_index;             // the column index in mesh/torus
  vector<StatTask*>   task_list;             // the list of scheduled tasks
};

class StatNOCTraffic {
  int                 topology;              // the topology code
  int                 num_row;               // the number of rows in mesh/torus
  int                 num_col;               // the number of columns in mesh/torus
  vector<StatProc>    proc_list;             // the list of PBs
  vector<StatTask>    task_list;             // the list of tasks
  vector<StatEdge>    edge_list;             // the list of edges

  vector<StatTask*>   starting_task_list;    // the list of starting tasks
  vector<StatTask*>   finishing_task_list;   // the list of finishing tasks
};
```

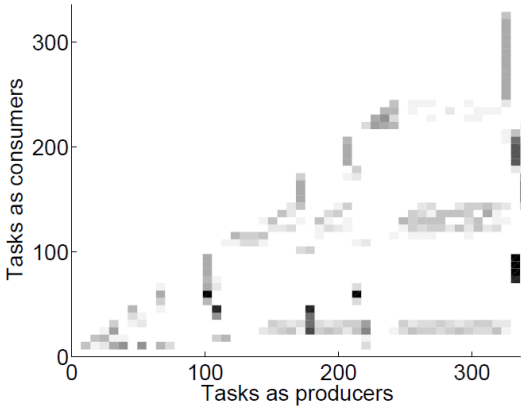
RTP vs. STP

- Tradeoffs between accuracy and coverage
- RTP is accurate but low coverage
  - Exact record of every task instance and communication transaction
  - Include ramp-up, stable, and ramp-down stages
  - Useful for detail analysis, such as debugging
- STP is less accurate but cover more cases
  - Using statistical models to cover wide possibilities
  - Useful for average and worst/best case studies which require extensive simulation runs

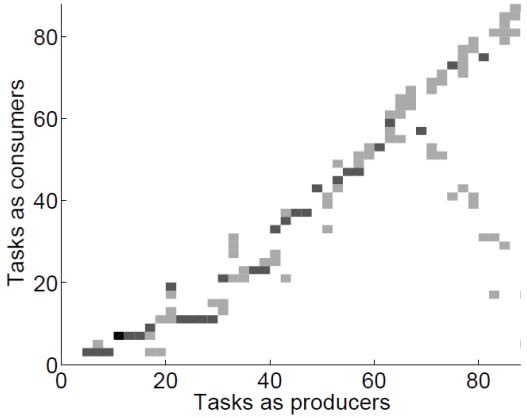
Communications Among Tasks



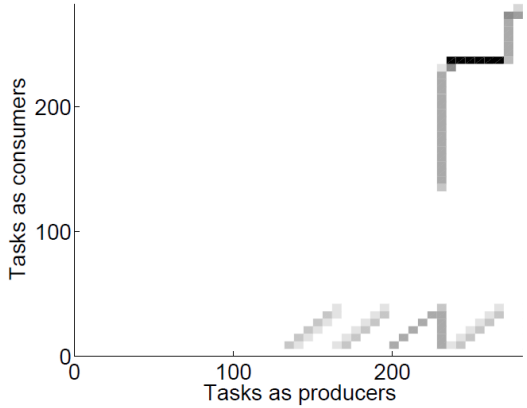
FFT-1024_complex



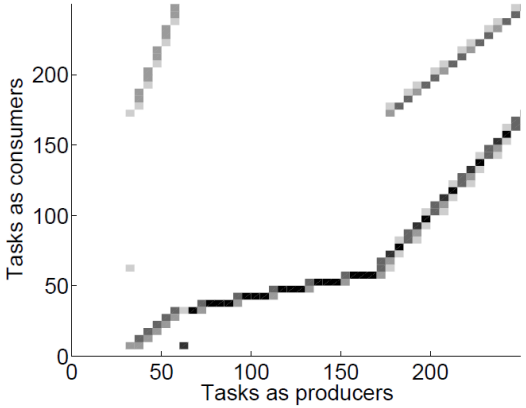
Fpppp



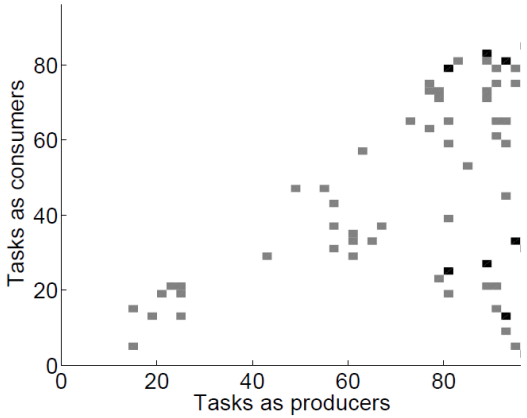
Robot



RS-32_28_8_dec

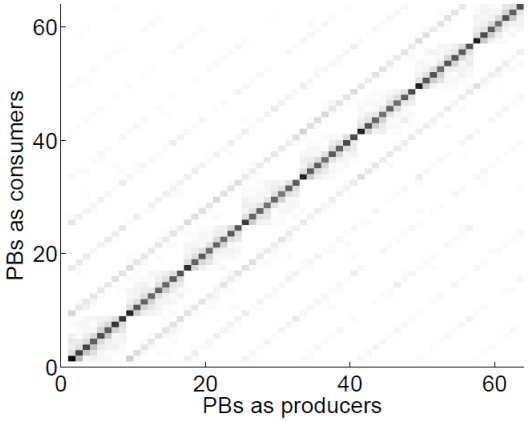


RS-32_28_8_enc

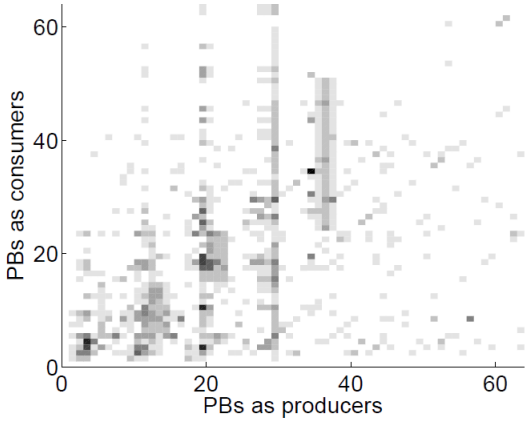


Sparse

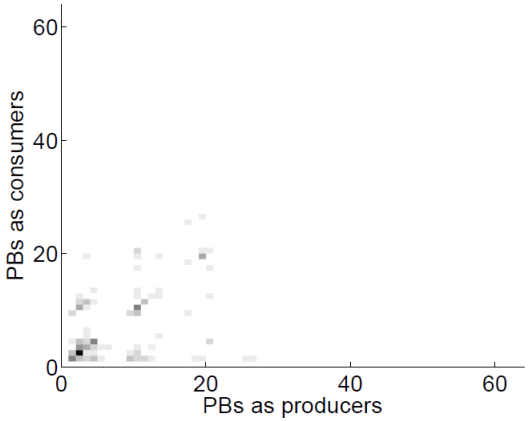
Communications Among PBs on 8x8 Mesh-based NoC



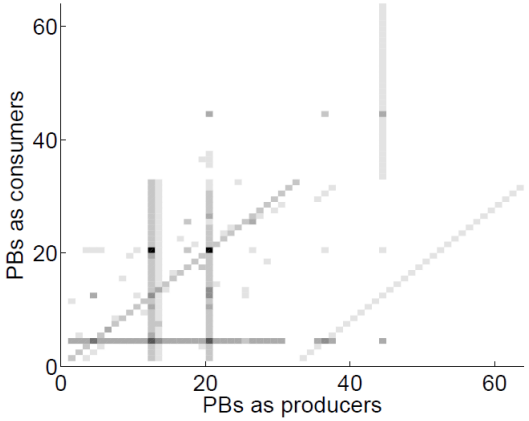
FFT-1024_complex



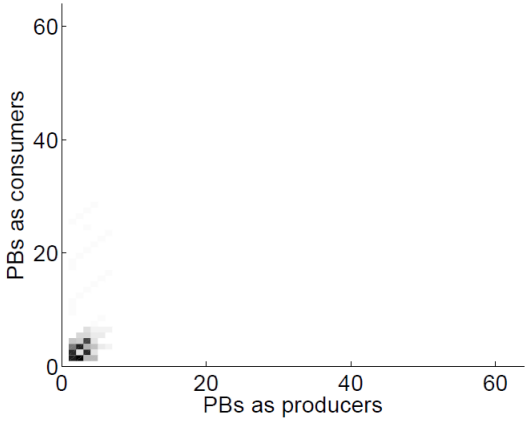
Fpppp



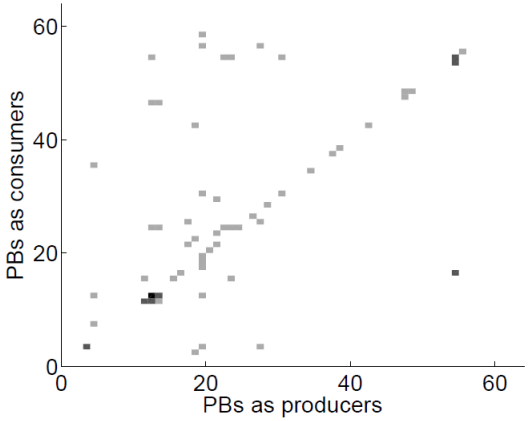
Robot



RS-32_28_8_dec

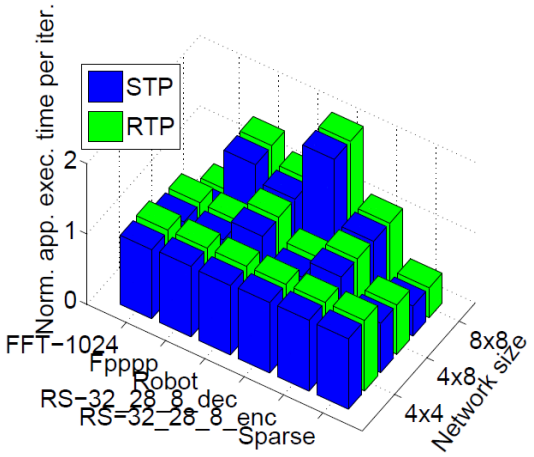


RS-32_28_8_enc

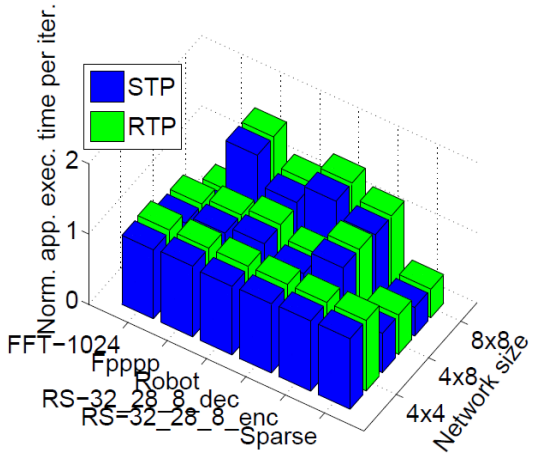


Sparse

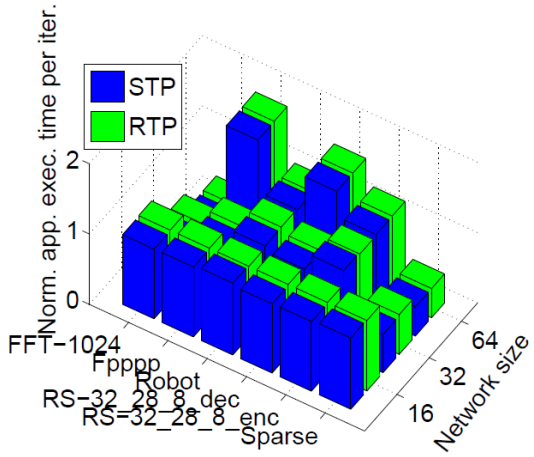
RTP vs. STP



Mesh



Torus

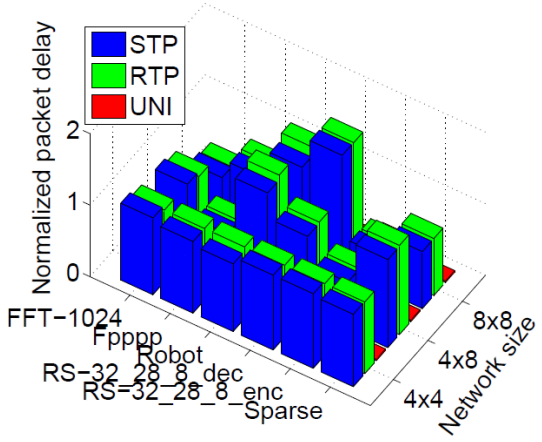


Fat tree

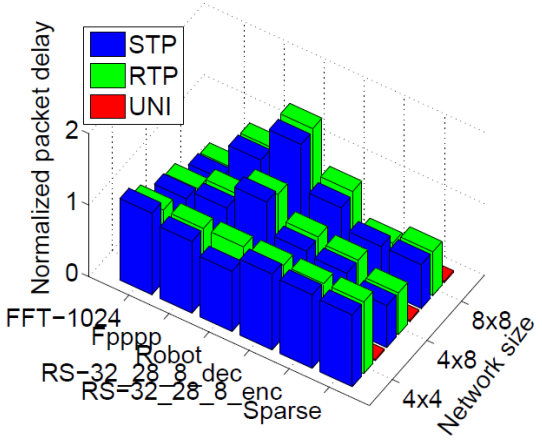
- Performance results showed by RTP and STP are consistent

	Average difference of RTP vs. STP
Network throughput	1.3%
Packet delay	6.0%
Application throughput	2.7%
Application execution time	0.9%

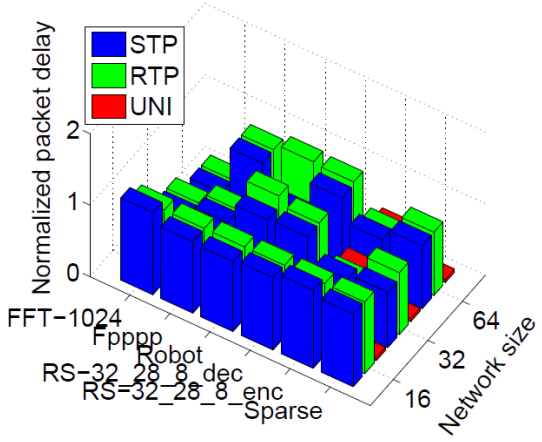
Comparing to Random Traffics



Mesh

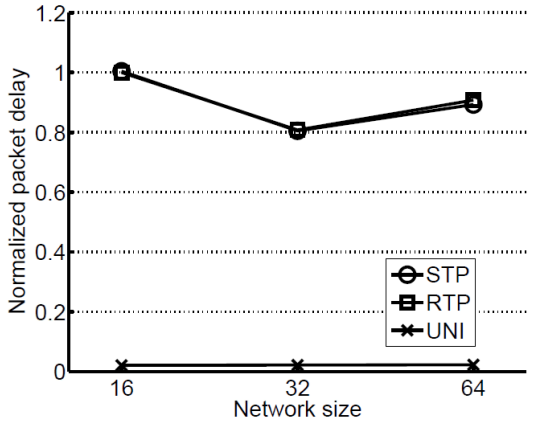


Torus



Fat tree

- Significantly different NoC performance results reported by random traffic patterns
  - E.g. uniform traffic patterns report 34X smaller packet delay on average



Average

Acknowledgement

- Zhe Wang, Weichen Liu, Xiaowen Wu, Bin Li, Ravi Iyer, Ramesh Illikkal, Wei Zhang, Yaoyao Ye, Zhehui Wang, Mahdi Nikdast

intel®

UGC 大學教育資助委員會 University Grants Committee

Reference

- Weichen Liu, Jiang Xu, Xiaowen Wu, Yaoyao Ye, Xuan Wang, Wei Zhang, Mahdi Nikdast, Zhehui Wang, “A NoC Traffic Suite Based on Real Applications”, in Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI), July 2011.
- Weichen Liu, Zhe Wang, Xiaowen Wu, Jiang Xu, Bin Li, et al., ”A Network-on-Chip Benchmark Suite Based on Real Applications,” Workshop on SoCs, Heterogeneous Architectures and Workloads (SHAW), February 2013.
- Weichen Liu, Jiang Xu, Xiaowen Wu, Yaoyao Ye, Xuan Wang, Wei Zhang, Mahdi Nikdast, Zhehui Wang, “MCSL: A Realistic Traffic Benchmark Suite for Network-on-Chip Studies”, Design Automation Conference (DAC), June 2011 (Poster).
- Santanu Dutta, Jens Rennert, Tiehan Lv, Jiang Xu, Shengqi Yang, Wayne Wolf, “MPSoC Architectures for Video”, Multiprocessor Systems-on-Chips, Morgan Kaufmann, 2004.
- Weichen Liu, Zonghua Gu, Jiang Xu, Xiaowen Wu, Yaoyao Ye, “Satisfiability Modulo Graph Theory for Task Mapping and Scheduling on Multiprocessor Systems”, IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 8, August 2011.
- Weichen Liu, Jiang Xu, Jogesh Muppala, Wei Zhang, Xiaowen Wu, Yaoyao Ye, “Coroutine-based Synthesis of Efficient Embedded Software from SystemC Models”, IEEE Embedded Systems Letters, vol.3, no.1, March. 2011.
- Xing Wen, Oscar Au, Jiang Xu, Lu Fang, Run Cha, “Novel RD-optimized VBSME with Matching Highly Data Re-usable Hardware Architecture”, IEEE Transactions on Circuits and Systems for Video Technology, vol.21, no.2, pp.206-219, Feb. 2011.
- Jiang Xu, Wayne Wolf, Wei Zhang, “Double-Data-Rate, Wave-Pipelined Interconnect for Asynchronous NoCs”, IEEE Micro, vol. 29 no. 3, pp. 20-30, 2009.
- Weichen Liu, Zonghua Gu, Jiang Xu, “Efficient Software Synthesis for Dynamic Single Appearance Scheduling of Synchronous Dataflow Graphs”, IEEE Embedded Systems Letters, vol. 1 no. 3, pp. 69-72, 2009.
- Zonghua Gu, Weichen Liu, Jiang Xu, Jin Cui, Xiuqiang He, Qingxu Deng, “Efficient 2D Area Management and Online Task Placement on Runtime Reconfigurable FPGAs”, Microprocessors and Microsystems, vol. 33, pp. 374-387, August 2009.
- Jiang Xu, Wayne Wolf, Joerg Henkel, Srimat Chakradhar, “A Design Methodology for Application-Specific Networks-on-Chip”, ACM Transactions on Embedded Computing Systems, July 2006.
- Tiehan Lv, I. Burak Ozer, Srimat Chakradhar, Jiang Xu, Wayne Wolf, Joerg Henkel, “A Methodology for Architectural Design of Multimedia Multiprocessor SoCs”, IEEE Design & Test of Computers, January 2005.
- Yuan Xie, Jiang Xu, Wayne Wolf, "Augmenting Platform-based Design with Synthesis Tools", Journal of Circuits, Systems, and Computers, April 2003.
- Xing Wen, Oscar C. Au, Jiang Xu, Lu Fang, Run Cha, “Sub-pixel Downsampling of Video with Matching Highly Data Re-use Hardware Architecture”, in Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS), Rio de Janeiro, Brazil, May 2011.
- Xing Wen, Oscar C. Au, Jiang Xu, Lu Fang, Jiali Li, “A Highly Data Reusable and Standard-Compliant Motion Estimation Hardware Architecture”, in Proceedings of IEEE International Conference on Multimedia and Expo (ICME), 2010.
- Weichen Liu, Zonghua Gu, Jiang Xu, Yu Wang, Mingxuan Yuan, “An Efficient Technique for Analysis of Minimal Buffer Requirements of Synchronous Dataflow Graphs with Model Checking”, International Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS), 2009.
- Xing Wen, Oscar C. Au, Jiang Xu, Zhiqin Liang, Yi Yang, Weiran Tang, “A Novel Multiple Description Video Coding based on H.264/AVC Video Coding Standard”, IEEE International Symposium on Circuits and Systems (ISCAS), 2009.
- Jiang Xu, Wei Zhang, Mo Kwai Hung, Zili Shao “Design ASNoC for Low-Power SoCs”, in Proceedings of International System-on-Chip Design Conference (ISOCC), 2008.
- Jiang Xu, Wayne Wolf, Joerg Henkel, Srimat Chakradhar, “A Methodology for Design, Modeling, and Analysis of Networks-on-Chip”, in Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS), Kobe, Japan, May 2005.
- Jiang Xu, Wayne Wolf, Srimat Chakradhar, Joerg Henkel, “H.264 HDTV Decoder Using Application-Specific Networks-on-Chip”, in Proceedings of IEEE International Conference on Multimedia and Expo (ICME), Amsterdam, the Netherlands, July 2005
- Jiang Xu, Wayne Wolf, Joerg Henkel, Srimat Chakradhar, Tiehan Lv, “A Case Study in Networks-on-Chip Design for Embedded Video”, in Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE), Paris, France, February 2004
- Jiang Xu, Wayne Wolf, “Platform-Based Design and the First Generation Dilemma”, in Proceedings of Electronic Design Processes Workshop (EDP), Monterey, CA, April 2002.

Thanks!