Support of Programming Models and Tools for Embedded Multi-core Platforms

Jenq Kuen Lee
Brian Kun-Yuan Shieh, Chung-Wen Huang
Department of Computer Science
National Tsing-Hua University
Hsinchu, Taiwan

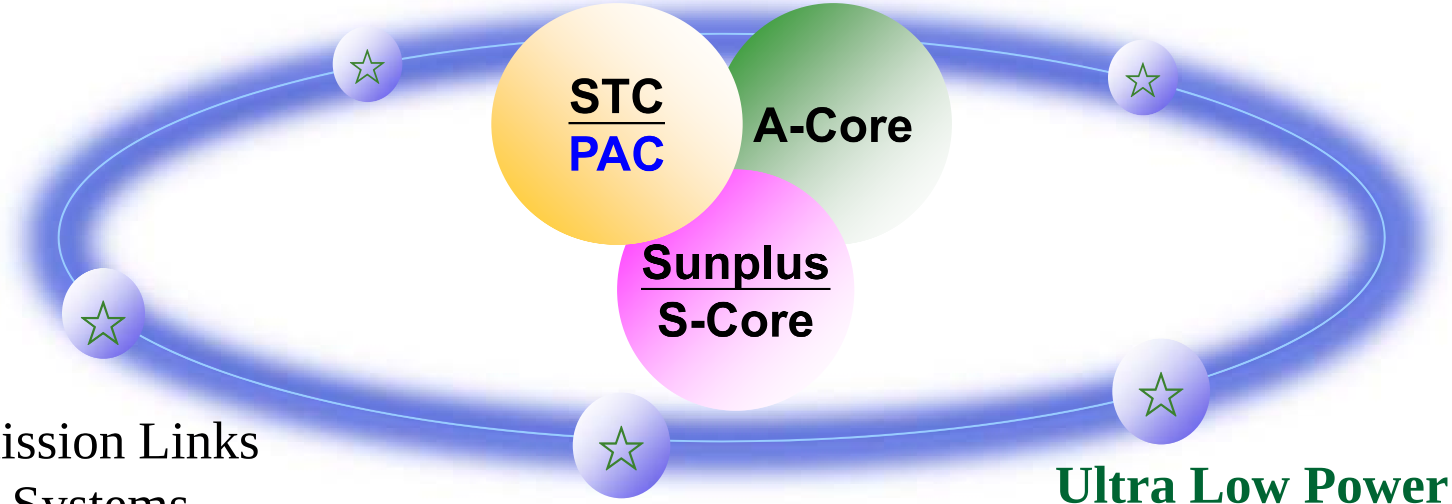
Outline

- Multi-Core Programming Model
    - Streaming RPC (with Kuen-Yuan)
- Optimization
    - Optimizing internal handshaking (with Kuen-Yuan)
    - Virtual channel supports (with Wan-Shu and Kuen-Yuan)
- Streaming RPC with IDE Tools (with Chien-Hong, Chung-Wen, and Jia-Jer)
- Experimental results
- Conclusion

StarIP Programs

Key Techniques in Chip Systems

Messenger – Distributed Radio Transmitter System



STC

PAC

A-Core

Sunplus

S-Core

Transmission Links in Chip Systems

Low-Power Embedded Multi-Core Programming Tools

Ultra Low Power (ULP) DSP Core

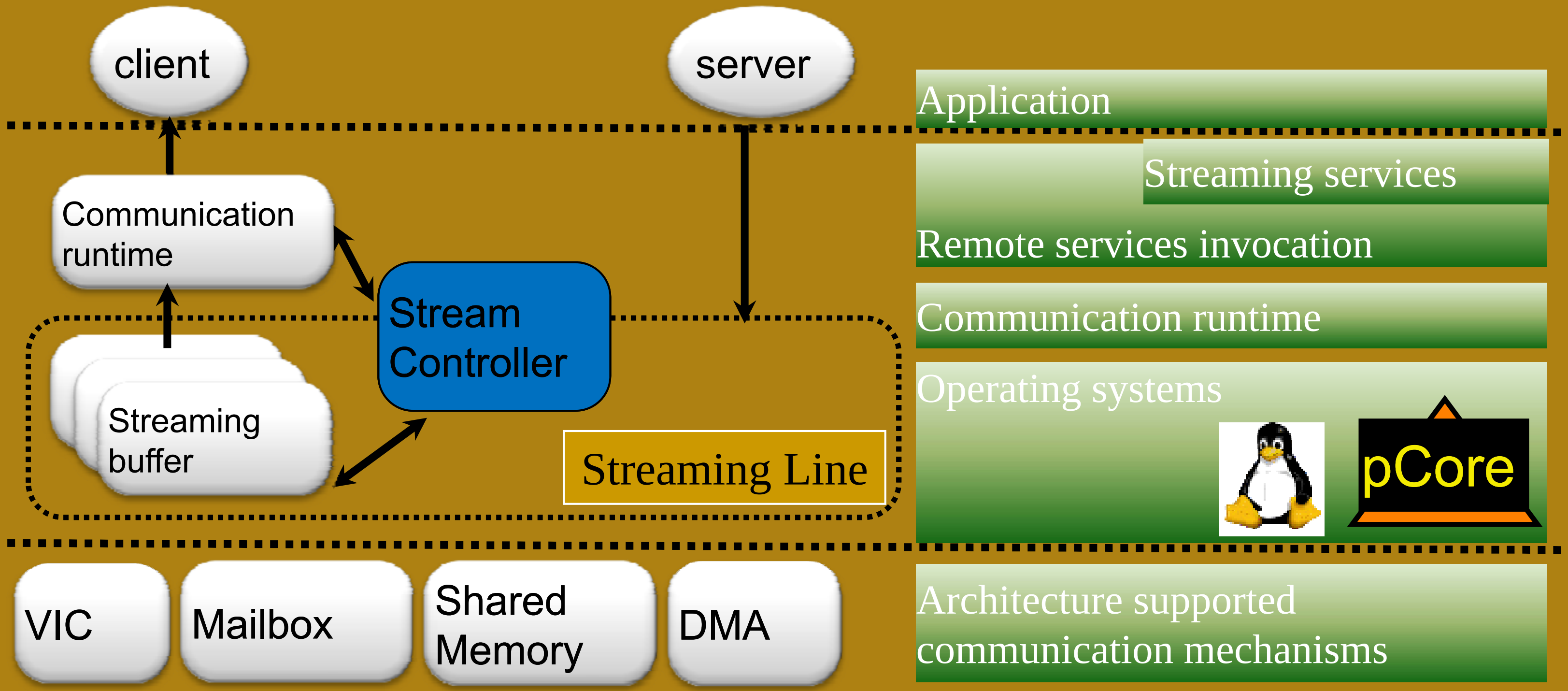
Streaming RPC (Streaming Remoting)

- An enhanced form of RPC (RMI)
- Support streaming flow vs. point-wise RPC
- It’s a higher level abstraction than message-passing programming (such as MPI or MCAPI).
- It can co-work with other multi-core programming model.
  - Multi-threading
  - SIMD & Clustered
- It’s a form of coarse-grained parallelism.
- Asynchronous parallelism and support for overlapping of communication and computation.
- It’s done with APIs and without language extensions.
  - C + Streaming RPC
  - Java +Streaming RMI

Programming Models for Multi-Core: Programming with Streaming RPC

Key components

- Streaming channel: A streaming channel is associated with an RPC request for transmitting data by setting the predefined stream identifier. The streaming channel provides a communication channel between the RPC client and server.
- Streaming buffer: associated to a streaming channel for providing data buffering
- Stream controller: monitoring and managing the streaming buffers .



client

server

Communication runtime

Stream Controller

Streaming buffer

Streaming Line

VIC

Mailbox

Shared Memory

DMA

Application

Streaming services

Remote services invocation

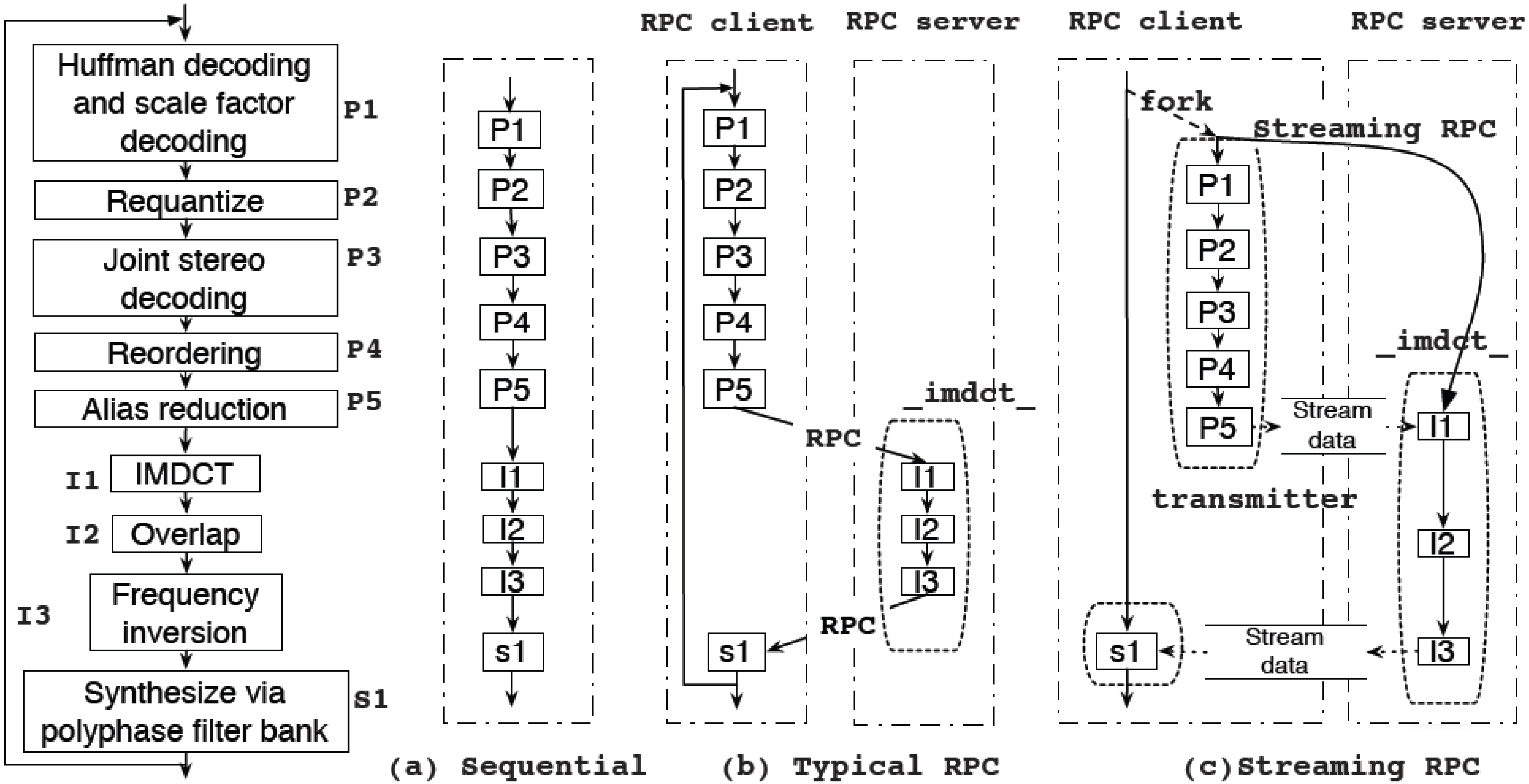
Communication runtime

Operating systems

pCore

Architecture supported communication mechanisms

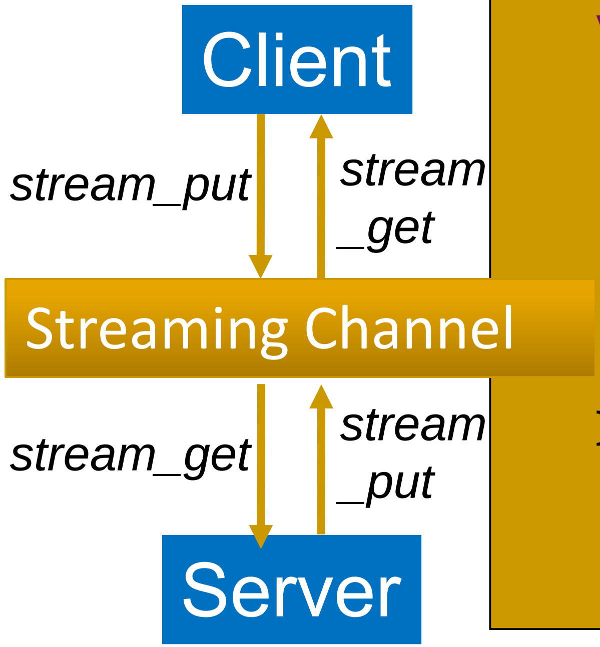
Application Example: MP3



(a) Sequential (b) Typical RPC (c)Streaming RPC

Example program

- An RPC is associated with streaming channels
- The client and server can send/get data to/from the channel
- Streaming operations
    - stream_get
    - stream_put
    - stream_push
    - stream_pop
    - stream_create
    - stream_rpc



Client

stream_put

stream _get

Streaming Channel

stream_get

stream _put

Server

```
/* Streaming RPC client */
void MP3_decoder(){
        stream_rpc(_imdct_, _transmitter_);
}
void _transmitter_(){
        STREAM_ID id = 4;
        /* Initializing streaming channel */
        stream_create(id);
     /* Pushing data to streaming channel */
        stream_put(id, DATA);
        stream_push(id);
        ...
}


    /* Streaming RPC server */
    void _imdct_(){
            STREAM_ID id = 4;
            /* Initializing streaming channel */
            stream_create(id);
            /* Aggregating data from streaming
             channel */
            stream_get(id, DATA);
            stream_pop(id);
            ...
    }
```

Optimization Issue: Buffer Managements and Internal Hand-Shaking

- Case 1: Producer (sender) is producing data much faster than the consumer (receiver).
- Case 2: Consumer (receiver) is consuming data much faster than the producer (sender).
- Difference in processing speed can result in frequent suspension and waking up!
  - Increase amount of internal RPC handshakings
  - Ex. when δ A > δ T, the receiver is suspended frequently

Sender

δT

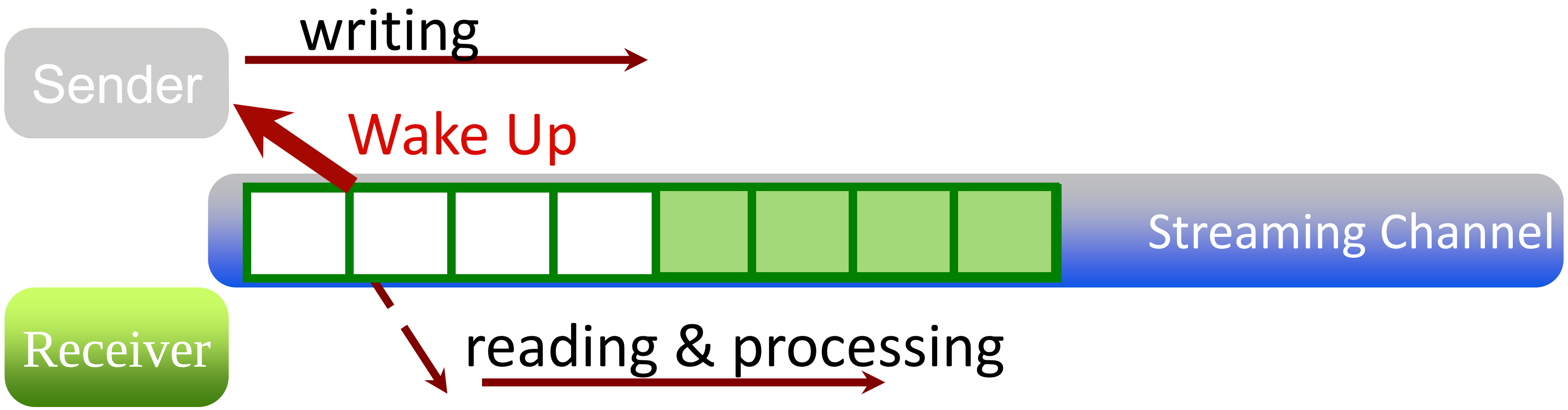
Streaming Channel

δA

Receiver

Setting Threshold Number

- To avoid frequent suspension and waking up!
- Assigning a threshold value to a streaming channel
  - The stream controller only wakes up the sender/receiver when a streaming channel satisfies the threshold criterion
  - ex. threshold value = 4



writing

Sender

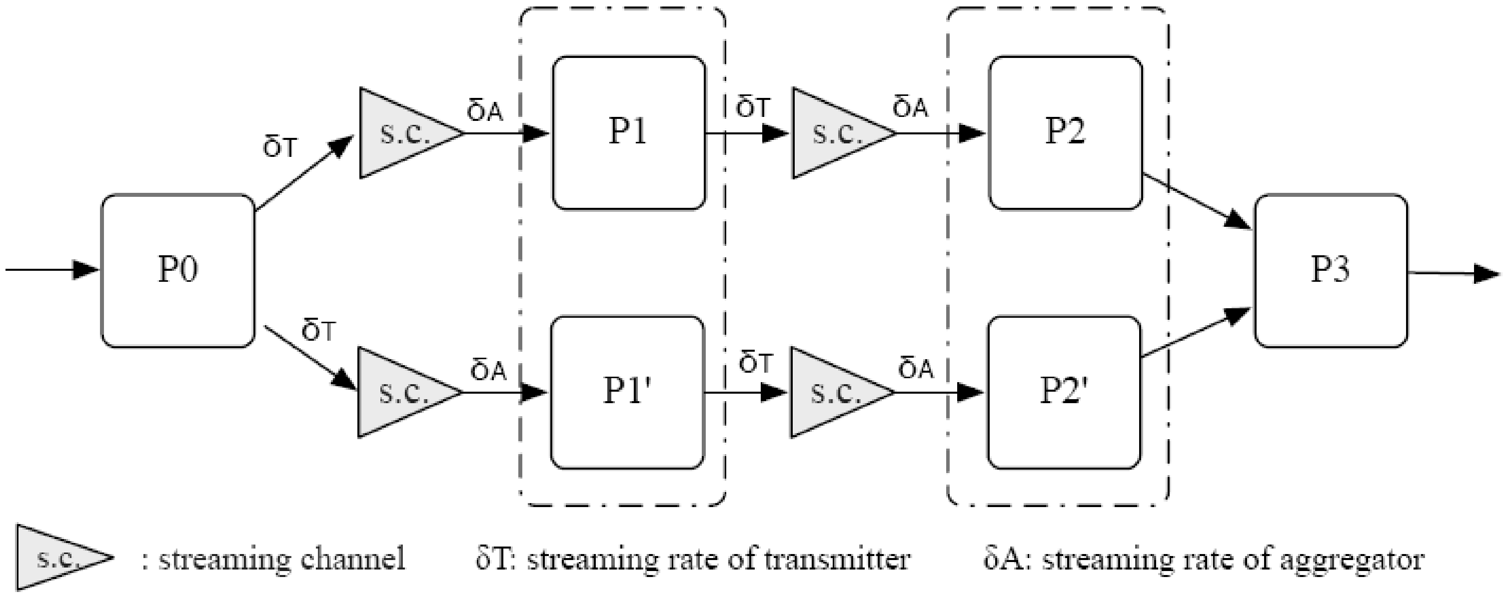
Wake Up

Streaming Channel

Receiver

reading & processing

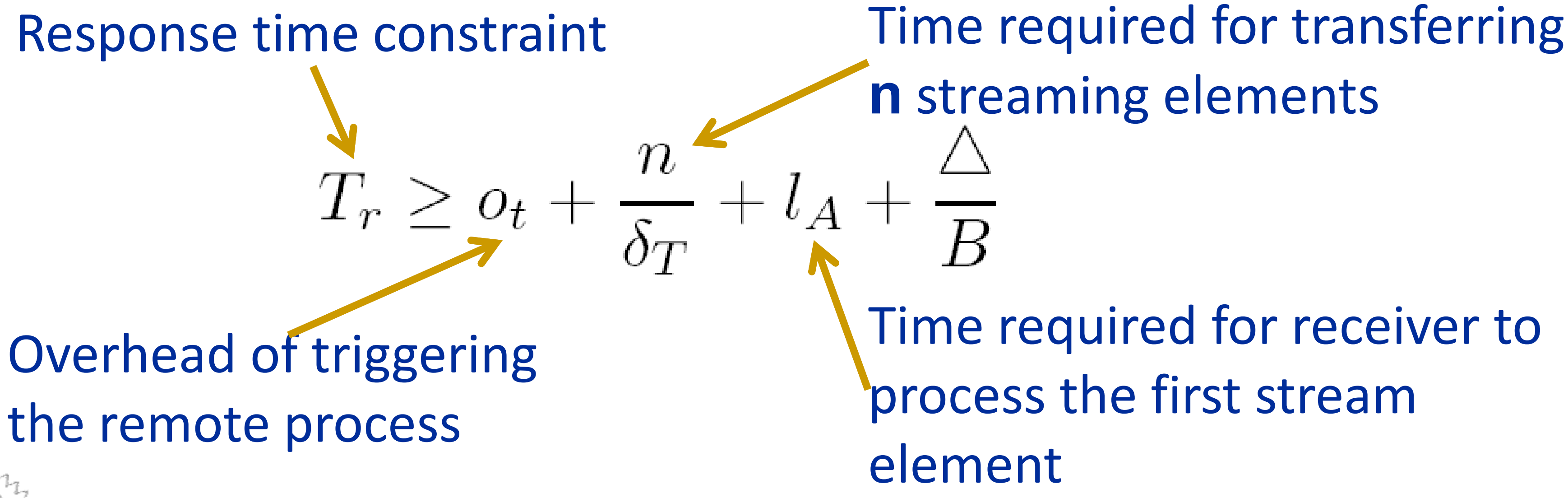
Analytic Model - Streaming Rate



- The streaming rate can be modeled for both transmitter/aggregator over the streaming channel.
- Parameters: Start Latency, Number of Streaming Elements, Size per element, communication bandwidth, overlapping ratio, speed in computation for handling one element.

Analytic Model for Deciding Threshold n

- To meet the response time constraint of the application
- Time of the first element to be processed after waiting for the sender to transmitting **n** stream elements must be less than the timing constraint



Response time constraint

Time required for transferring **n** streaming elements

$$T_r \geq o_t + \frac{n}{\delta_T} + l_A + \frac{\triangle}{B}$$

Overhead of triggering the remote process

Time required for receiver to process the first stream element

$$n \leq (T_r - o_t - l_A - \frac{\triangle}{B}) \times \delta_T$$

Optimization Issue: Memory Constraint

- The support of data streaming in streaming RPC is based on the technique of buffering
- There is an assignment problem for channel buffer when memory is limited.
- For a system that requires *k* streaming channels
  - $Q = \{Q_i | i = 1 \dots k\}$
  - $N_{Q_i}^{ub}$ is the upper bound of threshold of each channel *i*,
  - If the system can sustain at most χ elements
  - If $\sum_{i=1}^{k} N_{Qi}^{ub} > \chi$, the system could suffer huge overhead

Decision Equation

- To provide a group of efficient threshold parameters for each channel under memory constraint

$$\forall Q_i \in Q, 0 < n_i < N_{Q_i}^{ub}$$

$$\sum_{i=1}^{k} n_i < \chi$$

$$Max(\sum_{i=0}^{k} \Omega(Q_i, n_i))$$

Solving the Equation

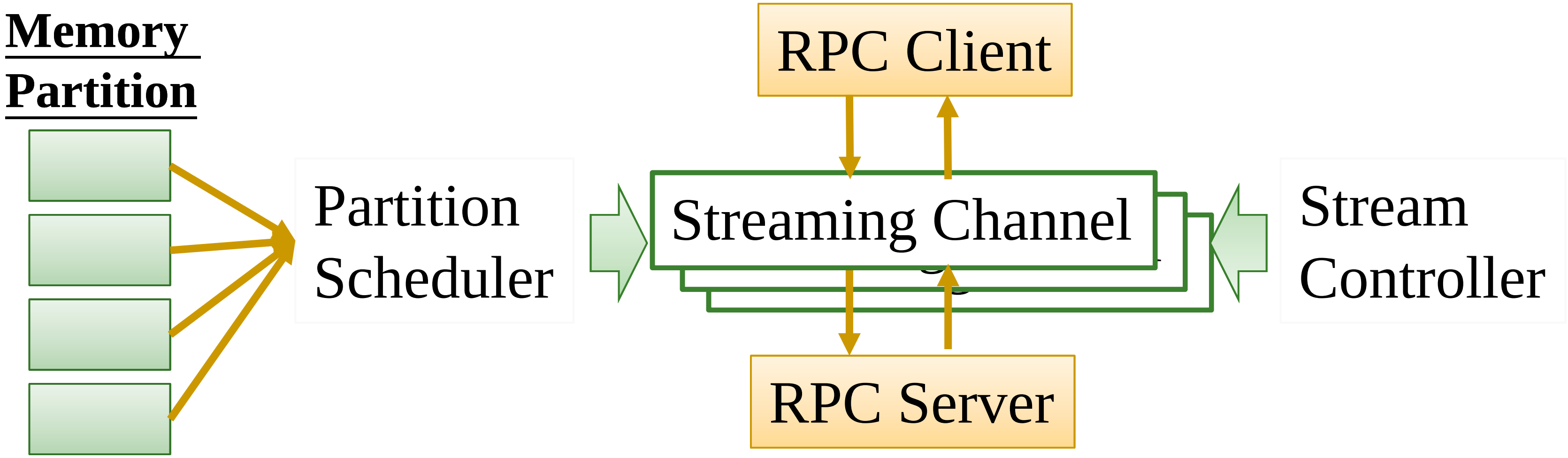
- The decision equation is NP-complete (Bounded Knapsack Problem).
- Observing from the experimental result, the internal communication time is proportional to the difference of the streaming rate between transmitter and aggregator.
- Thus, to simplify the problem, the threshold of each channel is decided by distribution of the available streaming elements

$$\forall Q_i \forall Q_k (|\delta_{Ti} - \delta_{Ai}| > |\delta_{Tk} - \delta_{Ak}|) \rightarrow \Omega(Q_i, \pi) < \Omega(Q_k, \pi)$$

$$n_i = \frac{|\delta_{Ti} - \delta_{Ai}|}{\sum_{j=1}^{k} |\delta_{Tj} - \delta_{Aj}|} * \chi$$

Virtual Streaming Channel

- The partition scheduler schedules streaming channels when |physical partition| < |streaming channel|



Memory Partition

Partition Scheduler

RPC Client

Streaming Channel

RPC Server

Stream Controller

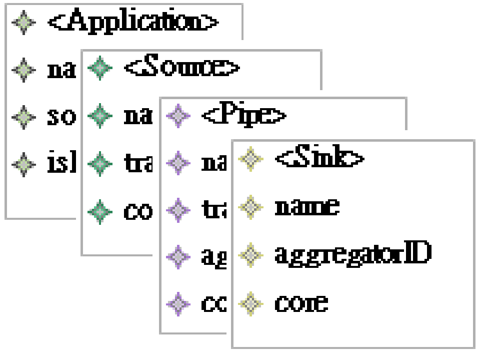
Scheduling Policy – Latency-aware (LaH)

- Find the channel which is almost done for scheduling.
- Scheduler finds a streaming channel with fastest response time and assigns ceiling priority

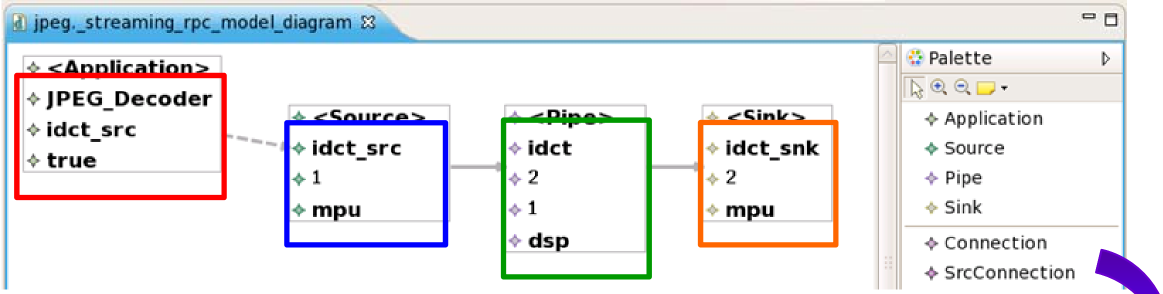
In general, voting schemes (TaH) can be used by considering latency, priorities, and job history.

Streaming RPC IDE Tools

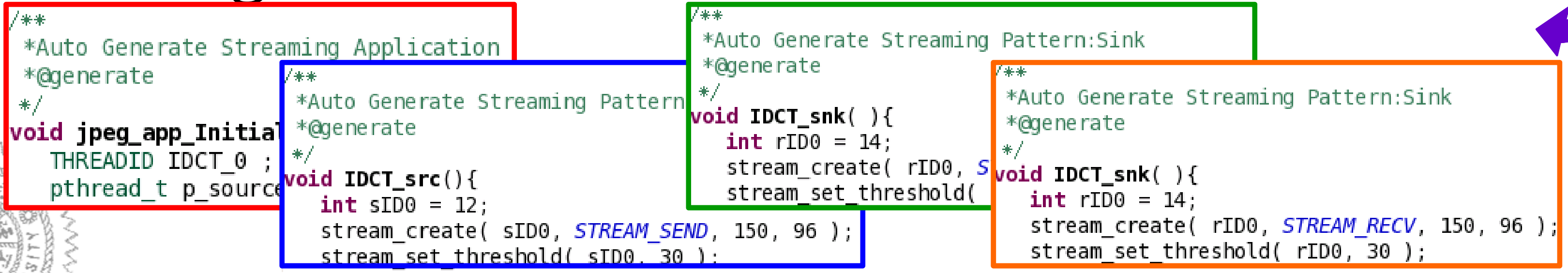
- Streaming RPC Design Patterns



- Streaming RPC Diagram



- Streaming RPC Code Transformation



```
/**
 *Auto Generate Streaming Application
 *@generate
 */
void jpeg_app_Initial
   THREADID IDCT_0 ;
   pthread_t p_source
```

```
/**
 *Auto Generate Streaming Pattern
 *@generate
 */
void IDCT_src(){
   int sID0 = 12;
   stream_create( sID0, STREAM_SEND, 150, 96 );
   stream_set_threshold( sID0, 30 );
```

```
/**
 *Auto Generate Streaming Pattern:Sink
 *@generate
 */
void IDCT_snk( ){
   int rID0 = 14;
   stream_create( rID0, S
   stream_set_threshold(
```

```
/**
 *Auto Generate Streaming Pattern:Sink
 *@generate
 */
void IDCT_snk( ){
   int rID0 = 14;
   stream_create( rID0, STREAM_RECV, 150, 96 );
   stream_set_threshold( rID0, 30 );
```

Streaming RPC Design Patterns

<Application>
name
sourceEntry
isMain

- Application
  - Stands for streaming applications.
  - Handling data communication and computation overlapping with *Source*, *Pipe*, and *Sink* patterns.

<Source>
name
transmitterID
core

- Source
  - A stream data transmitter, the originator of stream data.

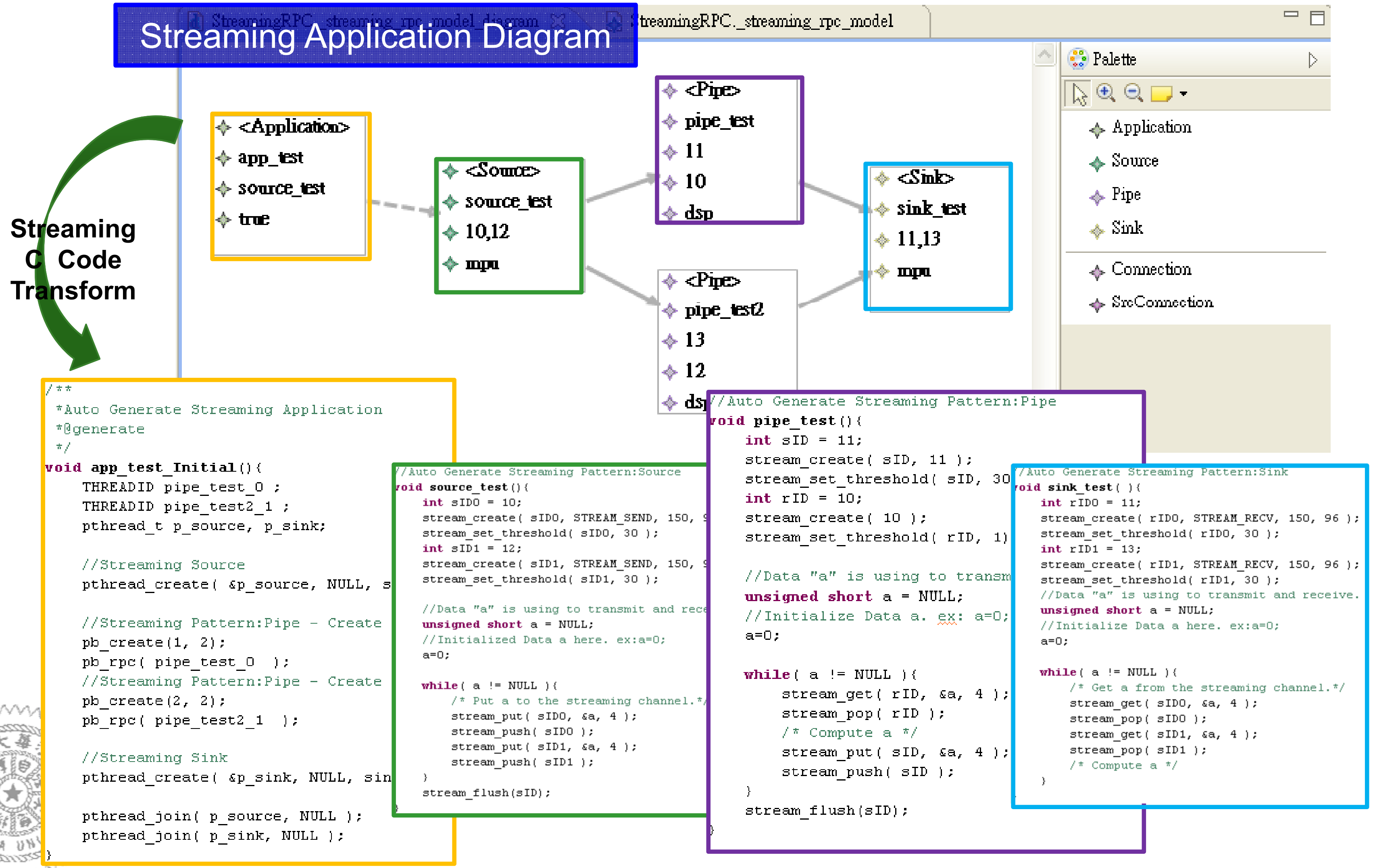
<Pipe>
name
transmitterID
aggregatorID
core

- Pipe
  - Parallel function stage of streaming application.
  - Gather stream data from streaming design pattern, do computation, and then transmit stream data to next streaming design pattern.

<Sink>
name
aggregatorID
core

- Sink
  - A stream data aggregator.
  - Display the output result of stream data.

How much can IDE help? Streaming RPC Application Diagram and Code Generation



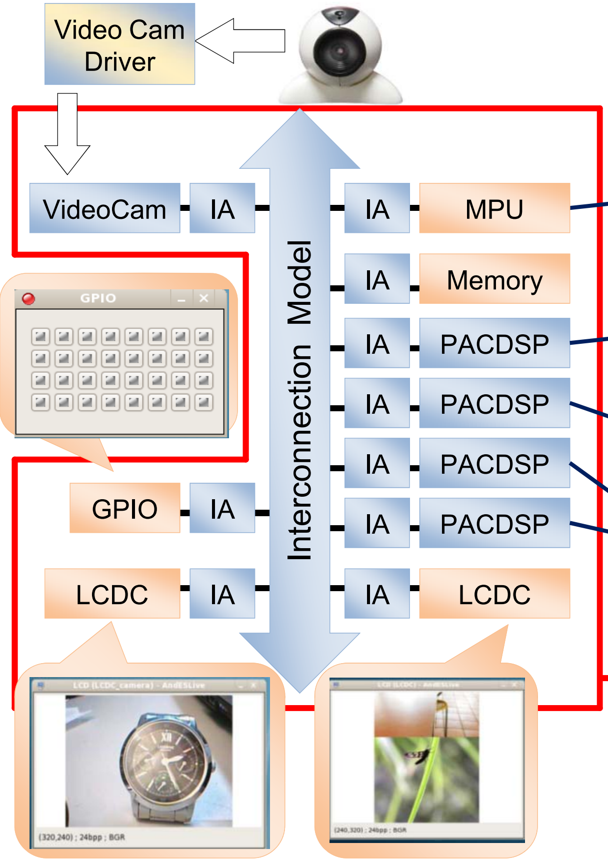
```
/**
 *Auto Generate Streaming Application
 *@generate
 */
void app_test_Initial(){
    THREADID pipe_test_0 ;
    THREADID pipe_test2_1 ;
    pthread_t p_source, p_sink;

    //Streaming Source
    pthread_create( &p_source, NULL, s

    //Streaming Pattern:Pipe - Create
    pb_create(1, 2);
    pb_rpc( pipe_test_0  );
    //Streaming Pattern:Pipe - Create
    pb_create(2, 2);
    pb_rpc( pipe_test2_1  );

    //Streaming Sink
    pthread_create( &p_sink, NULL, sin

    pthread_join( p_source, NULL );
    pthread_join( p_sink, NULL );
}
```

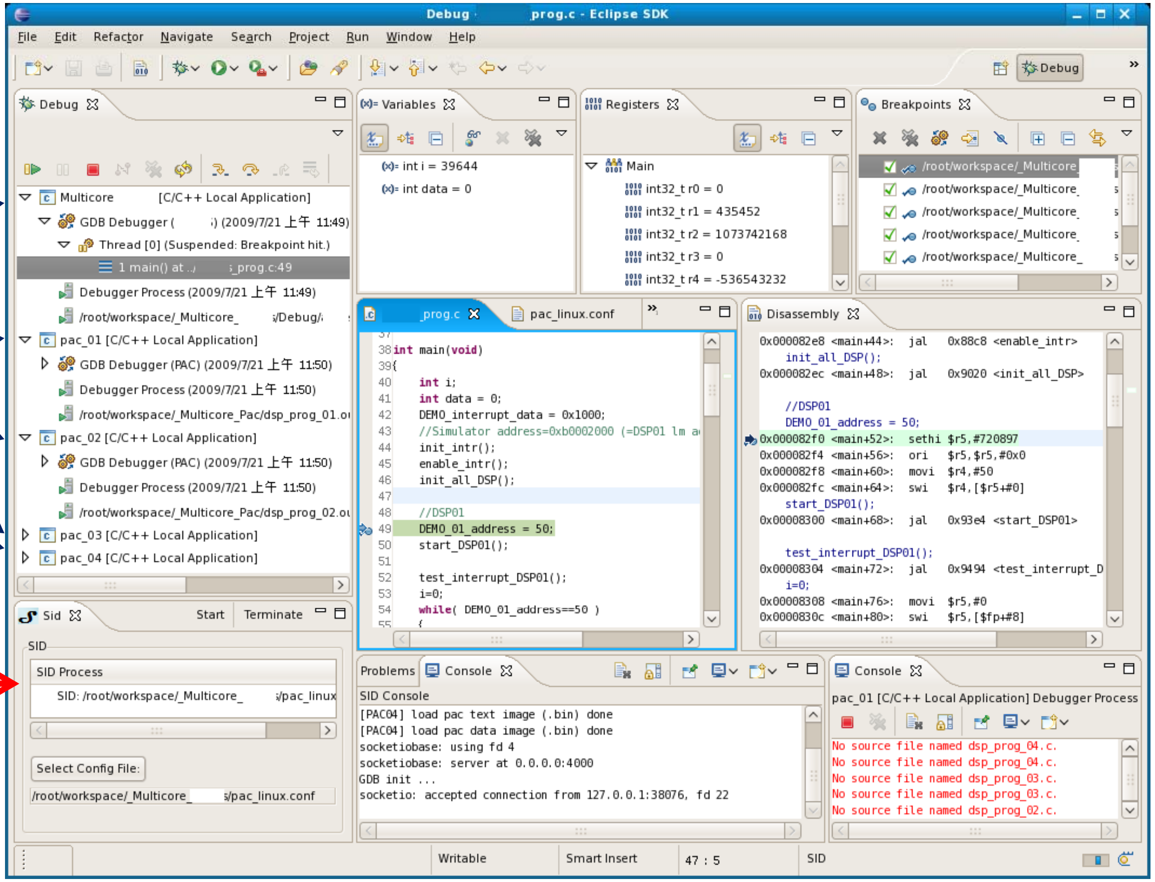
```
//Auto Generate Streaming Pattern:Source
void source_test(){
    int sID0 = 10;
    stream_create( sID0, STREAM_SEND, 150, 9
    stream_set_threshold( sID0, 30 );
    int sID1 = 12;
    stream_create( sID1, STREAM_SEND, 150, 9
    stream_set_threshold( sID1, 30 );

    //Data "a" is using to transmit and rece
    unsigned short a = NULL;
    //Initialized Data a here. ex:a=0;
    a=0;

    while( a != NULL ){
        /* Put a to the streaming channel.*/
        stream_put( sID0, &a, 4 );
        stream_push( sID0 );
        stream_put( sID1, &a, 4 );
        stream_push( sID1 );
    }
    stream_flush(sID);
}
```

```
//Auto Generate Streaming Pattern:Pipe
void pipe_test(){
    int sID = 11;
    stream_create( sID, 11 );
    stream_set_threshold( sID, 30
    int rID = 10;
    stream_create( 10 );
    stream_set_threshold( rID, 1)

    //Data "a" is using to transm
    unsigned short a = NULL;
    //Initialize Data a. ex: a=0;
    a=0;

    while( a != NULL ){
        stream_get( rID, &a, 4 );
        stream_pop( rID );
        /* Compute a */
        stream_put( sID, &a, 4 );
        stream_push( sID );
    }
    stream_flush(sID);
}
```

```
//Auto Generate Streaming Pattern:Sink
void sink_test( ){
    int rID0 = 11;
    stream_create( rID0, STREAM_RECV, 150, 96 );
    stream_set_threshold( rID0, 30 );
    int rID1 = 13;
    stream_create( rID1, STREAM_RECV, 150, 96 );
    stream_set_threshold( rID1, 30 );
    //Data "a" is using to transmit and receive.
    unsigned short a = NULL;
    //Initialize Data a here. ex:a=0;
    a=0;

    while( a != NULL ){
        /* Get a from the streaming channel.*/
        stream_get( sID0, &a, 4 );
        stream_pop( sID0 );
        stream_get( sID1, &a, 4 );
        stream_pop( sID1 );
        /* Compute a */
    }
```

Sid-Based Multicore ESL Simulation





```
Multicore [C/C++ Local Application]
  GDB Debugger ( ) (2009/7/21 上午 11:49)
    Thread [0] (Suspended: Breakpoint hit.)
      1 main() at _prog.c:49
  Debugger Process (2009/7/21 上午 11:49)
  /root/workspace/_Multicore_ /Debug/
pac_01 [C/C++ Local Application]
  GDB Debugger (PAC) (2009/7/21 上午 11:50)
  Debugger Process (2009/7/21 上午 11:50)
  /root/workspace/_Multicore_Pac/dsp_prog_01.o
pac_02 [C/C++ Local Application]
  GDB Debugger (PAC) (2009/7/21 上午 11:50)
  Debugger Process (2009/7/21 上午 11:50)
  /root/workspace/_Multicore_Pac/dsp_prog_02.o
pac_03 [C/C++ Local Application]
pac_04 [C/C++ Local Application]
```

```
int i = 39644
int data = 0
```

```
Main
  int32_t r0 = 0
  int32_t r1 = 435452
  int32_t r2 = 1073742168
  int32_t r3 = 0
  int32_t r4 = -536543232
```

```
☑ /root/workspace/_Multicore_
☑ /root/workspace/_Multicore_
☑ /root/workspace/_Multicore_
☑ /root/workspace/_Multicore_
☑ /root/workspace/_Multicore_
```

SID Process

SID: /root/workspace/_Multicore_ /pac_linux

Select Config File:

/root/workspace/_Multicore_ /pac_linux.conf

_prog.c   pac_linux.conf

```c
int main(void)
{
    int i;
    int data = 0;
    DEMO_interrupt_data = 0x1000;
    //Simulator address=0xb0002000 (=DSP01 lm a
    init_intr();
    enable_intr();
    init_all_DSP();

    //DSP01
    DEMO_01_address = 50;
    start_DSP01();

    test_interrupt_DSP01();
    i=0;
    while( DEMO_01_address==50 )
    {
```

```
0x000082e8 <main+44>:  jal   0x88c8 <enable_intr>
    init_all_DSP();
0x000082ec <main+48>:  jal   0x9020 <init_all_DSP>

    //DSP01
    DEMO_01_address = 50;
0x000082f0 <main+52>:  sethi $r5,#720897
0x000082f4 <main+56>:  ori   $r5,$r5,#0x0
0x000082f8 <main+60>:  movi  $r4,#50
0x000082fc <main+64>:  swi   $r4,[$r5+#0]
    start_DSP01();
0x00008300 <main+68>:  jal   0x93e4 <start_DSP01>

    test_interrupt_DSP01();
0x00008304 <main+72>:  jal   0x9494 <test_interrupt_D
    i=0;
0x00008308 <main+76>:  movi  $r5,#0
0x0000830c <main+80>:  swi   $r5,[$fp+#8]
```

SID Console

```
[PAC04] load pac text image (.bin) done
[PAC04] load pac data image (.bin) done
socketiobase: using fd 4
socketiobase: server at 0.0.0.0:4000
GDB init ...
socketio: accepted connection from 127.0.0.1:38076, fd 22
```

pac_01 [C/C++ Local Application] Debugger Process

```
No source file named dsp_prog_04.c.
No source file named dsp_prog_04.c.
No source file named dsp_prog_03.c.
No source file named dsp_prog_03.c.
No source file named dsp_prog_02.c.
```

Simulate on host PC: Intel Core 2 @ 2.0GHz

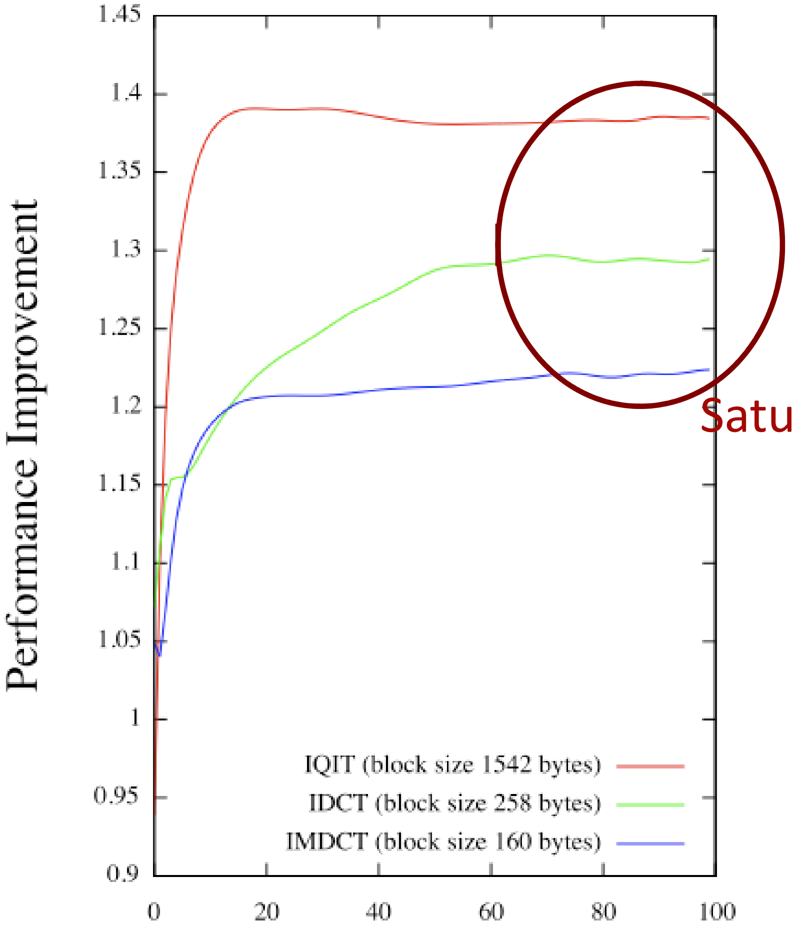
- 2-Core simulation
  - Work as 15Mhz physical platform
- 5-Core simulation
  - Work as 3Mhz physical platform

Experiments

- Dual-core platforms
  - PAC
  - OMAP 5912
- Three applications: JPEG, MP3, and H.264 decoders are used to demonstrate the performance.
- Three application kernels: IDCT, IMDCT, and IQ/IT are used to show the characteristics of streaming RPC.
- Effects of threshold values are evaluated.
- Experiments for assignment problems with memory constraints are given.
- Scheduling policies with virtual channel support are given.

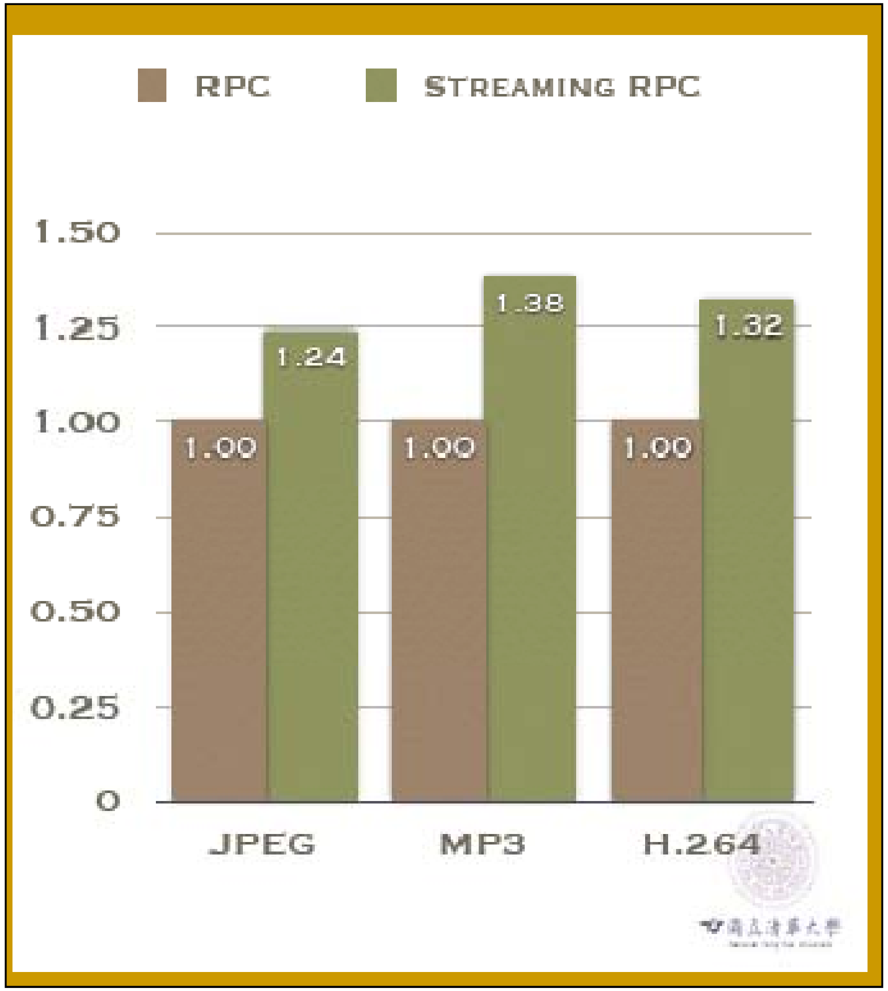
Performance Improvement on PAC

Performance evaluation
of different kernels

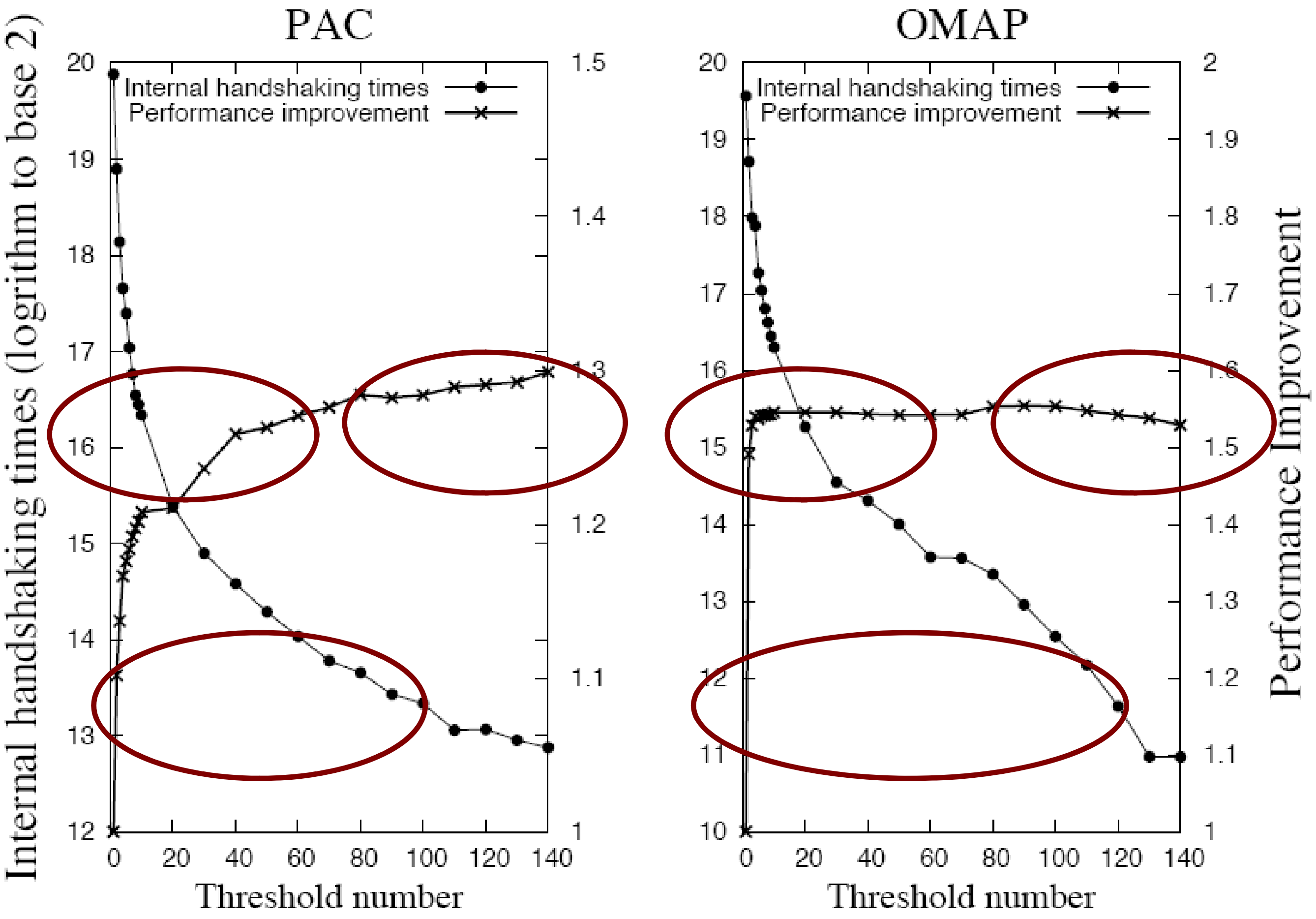


Saturated

Performance improvement of applications

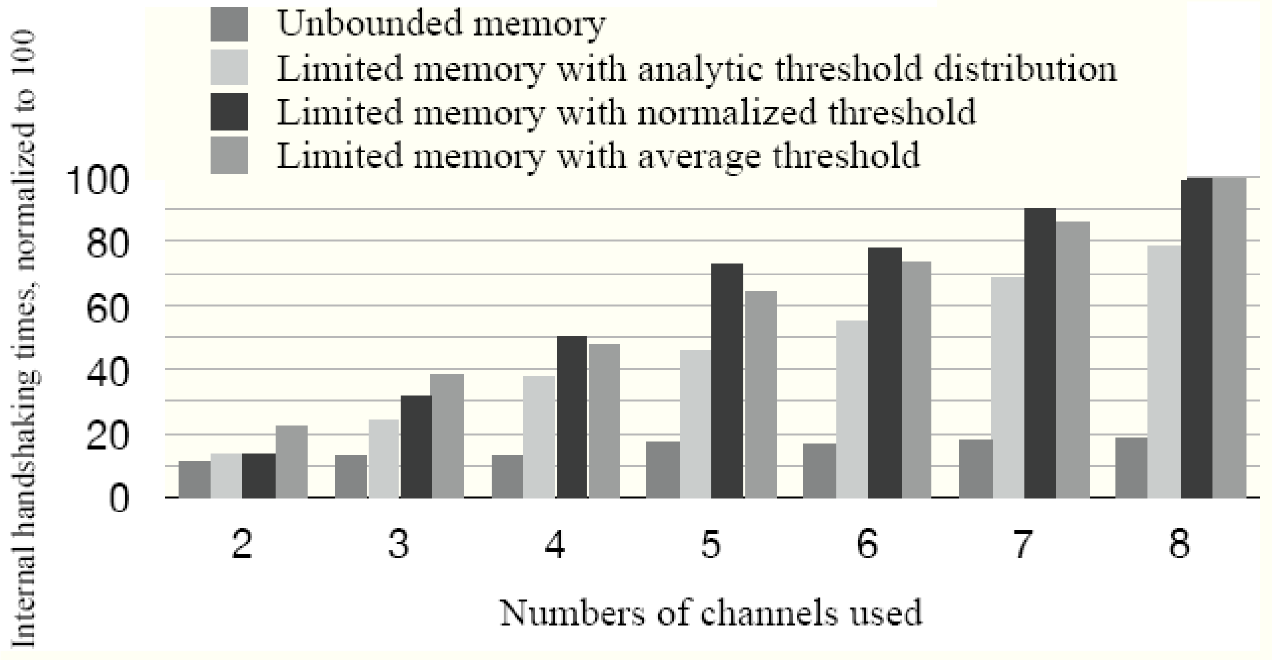


Performance Improvement and Corresponding Internal Handshaking Times: MP3



Simulation System and Result of the Analytic Model for Memory Constrained System

Parameters	Value
Machine Parameters	
Bus bandwidth	20 MB
Memory	64 KB
Memory limitation (•)	5 KB
Communication overhead (o_t)	25 μseconds
Simulation Parameters	
Number of channel required	2,. . .,8
Tasks generated for each run	50
Size of each streaming elements	2 B
Number of streaming operations of each task	10000



Summary

- We presented a stream programming model for embedded multi-core processors.
- Optimizations were done for internal hand-shaking and buffer assignments.
- Support for virtual channels with streaming RPC is also presented.
- Eclipse-based IDE tool is used to help streaming RPC programming.
- Related references can be seen in

  http://www.cs.nthu.edu.tw/~jklee