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

Transmission Links in Chip Systems

Ultra Low Power (ULP) DSP Core

Low-Power Embedded Multi-Core Programming Tools

Messenger – Distributed Radio Transmitter System
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.

![Diagram showing the components of Streaming RPC](image)
Application Example: MP3
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`

```c
/* 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 $\delta A > \delta T$, the receiver is suspended frequently
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
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

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

Overhead of triggering the remote process

Time required for transferring $n$ streaming elements

Time required for receiver to process the first stream element

$$n \leq \left(T_r - o_t - l_A - \frac{\Delta}{B}\right) \times \delta_T$$
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
  - $Q = \{Q_i|i = 1 \ldots k\}$
  - $N_{Q_i}^{ub}$ is the upper bound of threshold of each channel $i$.
  - If the system can sustain at most $\chi$ elements.
  - If $\sum_{i=1}^{k} N_{Q_i}^{ub} > \chi$, the system could suffer huge overhead.
To provide a group of efficient threshold parameters for each channel under memory constraint

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

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

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

\[
\forall Q_i \forall Q_k (|\delta_{T_i} - \delta_{A_i}| > |\delta_{T_k} - \delta_{A_k}|) \rightarrow \Omega(Q_i, \pi) < \Omega(Q_k, \pi)
\]

\[
n_i = \frac{|\delta_{T_i} - \delta_{A_i}|}{\sum_{j=1}^{k} |\delta_{T_j} - \delta_{A_j}|} \times \chi
\]
Virtual Streaming Channel

- The partition scheduler schedules streaming channels when \(|\text{physical partition}| < |\text{streaming channel}|\)
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

```c
/**
 * Auto Generate Streaming Application
 * @generate
 */
void jpeg_app_InitThreadID 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 = 34;
  stream_create( rID0, STREAM_RECV, 150, 96 );
  stream_set_threshold( rID0, 30 );
}
```
Streaming RPC Design Patterns

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

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

- **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**
  - A stream data aggregator.
  - Display the output result of stream data.
How much can IDE help?

Streaming RPC Application Diagram and Code Generation

```c
void app_test_initial()
{
    THREADID pipe_test_0;
    THREADID pipe_test1_1;
    pthread_t p_source, p_sink;
    // Streaming Source
    pthread_create(&p_source, NULL, ...
    // Streaming Pattern: Pipe - Create
    pb_create(1, 2);
    pb_rpc| pipe_test_0 |;
    // Streaming Pattern: Pipe - Create
    pb_create(2, 2);
    pb_rpc| pipe_test1_1 |;
    // Streaming Sink
    pthread_create(&p_sink, NULL, ...
    pthread_join| p_source, NULL |
    pthread_join| p_sink, NULL |
}

void pipe_test()
{
    int sID = 11;
    stream_create( sID, 11 );
    stream_set_threshold( sID, 10 );
    int rID = 10;
    stream_create( rID, 10 );
    stream_set_threshold( rID, 10 );
    // Data "a" is using to transmit and receive.
    while( a != NULL )
    {
        // Put a to the streaming channel.
        stream_push( sID, a, 4 );
        stream_pop( rID, 4 );
        // Compute a 
        stream_push( sID, a, 4 );
        stream_pop( rID, 4 );
    }
    stream_flush(sID);
    }

void sink_test()
{
    int sID = 11;
    stream_create( sID, 11 );
    stream_set_threshold( sID, 10 );
    int rID = 10;
    stream_create( rID, 10 );
    stream_set_threshold( rID, 10 );
    // Data "a" is using to transmit and receive.
    while( a != NULL )
    {
        // Get a from the streaming channel.
        stream_get( sID, a, 4 );
        stream_pop( rID );
        // Compute a 
        stream_push( sID, a, 4 );
        stream_pop( rID );
    }
    stream_flush(sID);
}
Sid-Based Multicore ESL Simulation

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

Performance improvement of applications
Performance Improvement and Corresponding Internal Handshaking Times: MP3
Simulation System and Result of the Analytic Model for Memory Constrained System

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Machine Parameters</strong></td>
<td></td>
</tr>
<tr>
<td>Bus bandwidth</td>
<td>20 MB</td>
</tr>
<tr>
<td>Memory</td>
<td>64 KB</td>
</tr>
<tr>
<td>Memory limitation (•)</td>
<td>5 KB</td>
</tr>
<tr>
<td>Communication overhead (ο-)</td>
<td>25 μseconds</td>
</tr>
<tr>
<td><strong>Simulation Parameters</strong></td>
<td></td>
</tr>
<tr>
<td>Number of channel required</td>
<td>2, …, 8</td>
</tr>
<tr>
<td>Tasks generated for each run</td>
<td>50</td>
</tr>
<tr>
<td>Size of each streaming elements</td>
<td>2 B</td>
</tr>
<tr>
<td>Number of streaming operations of each task</td>
<td>10000</td>
</tr>
</tbody>
</table>

[Graph showing internal handshake times normalized to 100 for different numbers of channels used.]

- Unbounded memory
- Limited memory with analytic threshold distribution
- Limited memory with normalized threshold
- Limited memory with average threshold
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