Multicore Design Considerations

Objectives

In this module, the student will be able to:

- Understand the importance of multi-core parallel processing in current and future applications
- Define different types of Parallel processing and understand the possible limitations and dependencies
- Understand the importance of memory features and architecture and data movement for efficient parallel processing
  - Be Familiar with KeyStone special features that facilitate parallel processing
- Build a functional driven parallel project
  - Build and Run MCSDK Video demo
  - Analyze TI's H264 implementation
- Build a data driven parallel project
  - Build and run TI's very large FFT implementation. Understand the implementation
  - Build and Run openMP solution to the VLFFT problem

Definitions

- Paralell Processing –

  The usage of simultaneous processors to execute an application or multiple computational threads

- Multicore Paralell Processing –

  The usage of multiple cores in the same device to execute an application or multiple computational threads

Multicore: The Forefront of Computing Technology

- ***What Moore said: double number of transistors on a device every two years***
- ***Traditional interpretation – Double performances (smaller process, higher frequency, VLIW)***
- ***Multicore – achieve Moore's law by adding multiple cores***
  - ***The criterion – Watts per performances***
- ***The challenge – How to effectively use multiple-cores devices***

"We're not going to have faster processors. Instead, making software run faster in the future will mean using parallel programming techniques. This will be a huge shift."

-- Katherine Yelick, Lawrence Berkeley National Laboratory

from The Economist: Parallel Bars

Marketplace Challenges

- Increase of data rate
  - Think about Ethernet, from 10Mbps to 10Gbps
- Increase in algorithm complexity
  - Think about typical face recognition, finger prints
- Increase in development cost
  - Hardware and software development
- Multicore SOC devices are a solution
  - Fast peripherals part of the device
  - High performances, fixed point and floating point processing power. Paralell data movement.
  - Off the shelf devices
  - Elaborate set of software tools

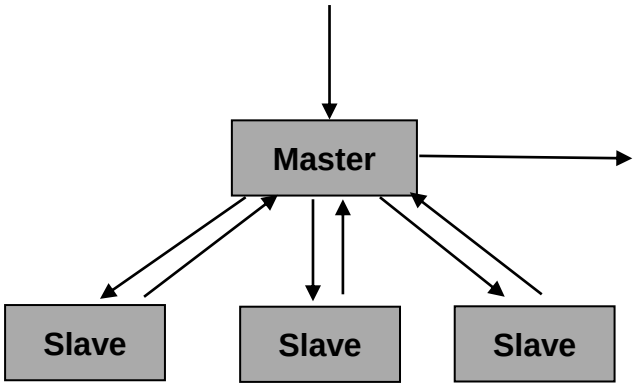
Common Use Cases

- Network gateway, speech/voice processing
  - Typically hundreds or thousands of channels
  - Each channel consumes about 30 MIPS
- Large, complex, floating point FFT (Radar applications and others)
- Video processing
- Medical imaging
- LTE, WiMAX, other wireless physical layers
- Scientific processing (Oil explorations)
  - Large complex matrix manipulations
- Your applications

Parallel Processing Models
Master Slave Model

- Centralized control and distributed execution
- Master responsible for execution scheduling and data availability
- Required fast and cheap (in terms of CPU resources) messages and data exchange between cores
- Typical applications consists of many small independent threads

Parallel Processing Models
Master Slave Model (2)

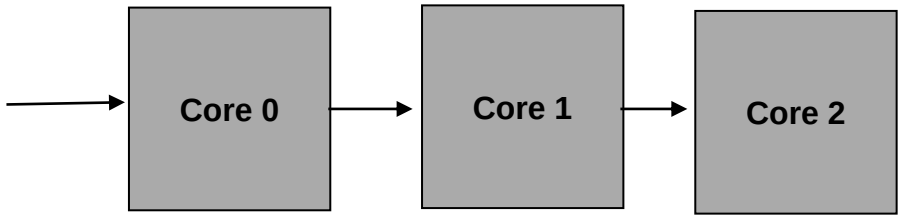


- Applications
  - Multiple media processing
  - Video encoder slice processing
  - JPEG 2000 – multiple frames
  - VLFFT

Parallel Processing Models
Data Flow Model

- Distributed Control and execution
- The algorithm is partitioned into multiple block, each block is processed by a core, and the output of one core is the input to the next core
  - Exchange data and messages between any cores
- Big challenge – partition blocks to optimize performances
  - Required loose link between cores (queue system)

Parallel Processing Models
Data Flow Model(2)



- Applications
  - High quality video encoder
  - Video decoder
  - Video transcoder
  - LTE physical layer

Partitioning Considerations

- Application is a set of algorithms. In order to partition an application into multiple cores, the system architect should understand-
  - Can a certain algorithm be executed on multiple cores in parallel?
    - Can the data be divided between two cores? FIR filter can be, IIR filter cannot
  - What are the dependencies between two (or more) algorithms. Can they be processed in parallel, can one algorithm must wait for the next?
    - Example – identification based on finger print and face recognition can be done in parallel. Pre filter and then image reconstruction in CT must be done in order
  - Can the application can run concurrently on two sets of data?
    - JPEG2000 Video encoder yes, H264 Video encoder no

Common Partitions Methods

- **Function driven**
  - Large tasks are divided into function blocks
  - Function blocks are assigned to each core
  - The output of one core is the input of the next core
  - Use cases: H.264 high quality encoding and decoding, LTE
- **Data driven**
  - Large data sets are divided into smaller data sets
  - All cores perform the same process on different blocks of data
  - Use cases: image processing, multi-channel speech processing, sliced-based encoder
- **Mixed Partition**
  - Both functional driven and data driven exists

Multicore SOC Design Challenges

- Getting large amount of data into the device and out of the device
- Ability to perform high performance computing – lots of operations on multiple cores
  - Powerful fixed point and floating point core
  - Efficient data sharing between cores and sending signals between cores without staling execution
  - Minimize multiple cores contentions of the shared resources

Input and output data

- Fast peripherals are needed to feed high bit rate data into the device and get high bit rate out
- KeyStone devices have variety of high bit rate peripherals such as
  - 100/1000G Ethernet
  - 10G Ethernet
  - SRIO
  - PCIe
  - AIF2
  - TSIP

C66 - Powerful Core

- 8 functional units of the C66 provide:
  - fixed point and floating point native instructions
  - Many SIMD instructions
  - Many Special purpose powerful instructions
  - Fast (0 wait state) L1 memory, fast L2 memory

Data Sharing Between Cores

- Shared memory
  - KeyStone family has a very fast large external DDR interface(s)
  - 32 to 36 bit address translation enables access of up to 10GB of DDR
  - Fast shared L2 memory as part of the sophisticated and fast MSMC
- Hardware provides ability to move data and signals between cores with minimal CPU resources
  - Powerful Multicore Navigator
  - Multiple instances of EDMA
- Other hardware mechanism that help facilitate messages and communications between cores
  - IPC registers
  - Semaphore blocks

Minimizing Resource Contention

- Each core has a dedicated port into the MSMC
- MSMC supports pre-fetching to speed up data load
- Shared L2 has multiple bank of memory
  - Support concurrent multiple access
- Wide and fast parallel Teranet switch fabric provides priority based parallel access
- Packet based HyperLink bus enables seamless connection of two KeyStone devices to increase performance while minimizing power and cost

Software Offering -System

- MCSDK is a complete set of software libraries and utilities developed for the KeyStone family
- Set of LLD supplemented by CSL utilities provide software access to all peripherals and coprocessors
- In particular, KeyStone has a set of software utilities to facilitate messages and communications between cores such as
  - IPC
  - LLD for the CPPI and QMSS
  - LLD for EDMA

Software Offering - Applications

- TI supports common parallel programming languishes :
  - OpenMP - part of the compiler release
  - OpenCL - plan to support in the next release

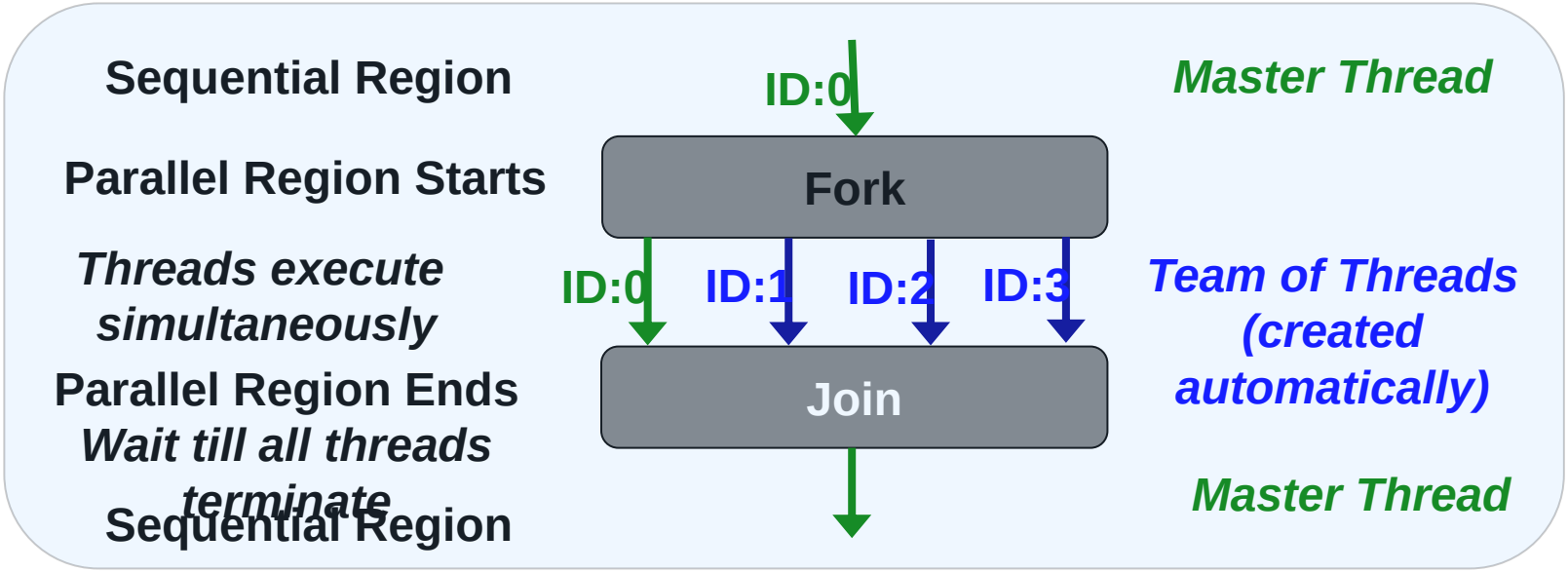
Software Support - OpenMP

OpenMP (Supported by TI compiler tools)

- An API for writing multi-threaded applications
- API includes compiler directives and library routines
- C, C++, and Fortran support
- Standardizes last 20 years of Shared-Memory Programming (SMP) practice

Software Support OpenMP (2)

- Create Teams of Threads
  - Fork-Join Model
  - Execute code in a parallel region
  - Implemented by using compiler directive #pragma omp parallel
  - Nesting ‘parallel’ directives is possible, allowing multilevel parallelism



Examples of Partitions Methods

- **Function driven**
  - H264 encoder – Example 1
- **Data driven**
  - Very Large FFT - Example 2

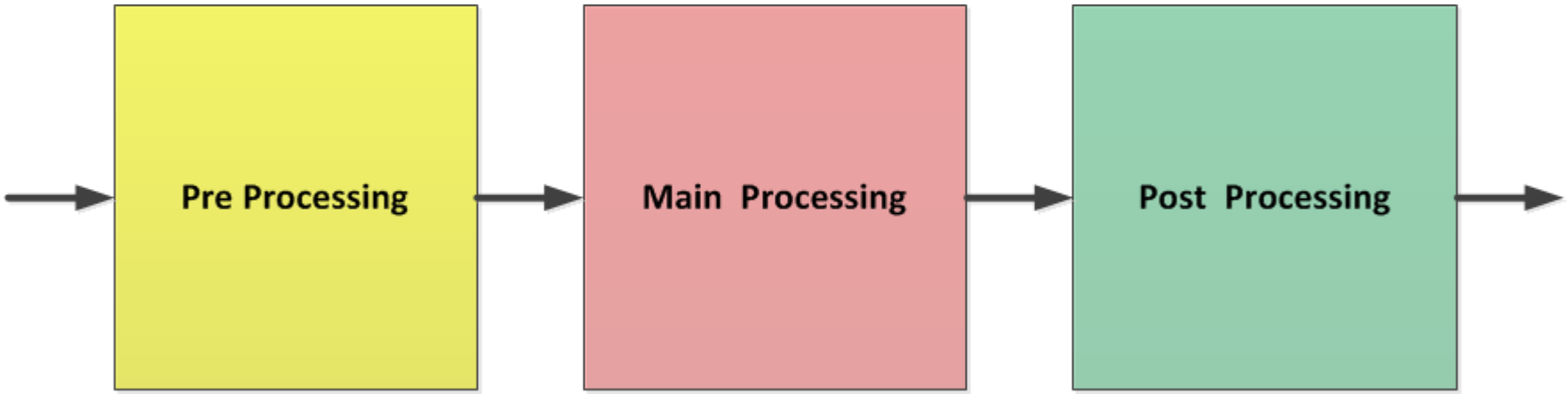
Example1:

High Def 1080i60 Video H264 Encoder

Data Flow Model
Function driven partition

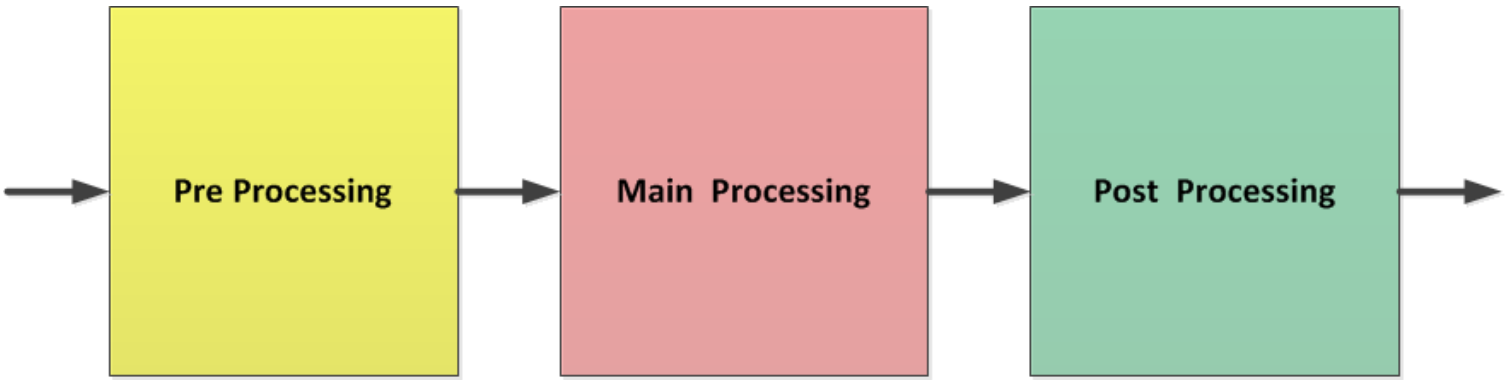
Video Compression Algorithm

- Video compression is done frame after frame
- Within a frame, the processing is done row after row
- Video compression is done on Macroblocks (16x16 pixels)
- Video compression can be divided into three parts – pre processing, main processing and post processing



Dependencies and limitations

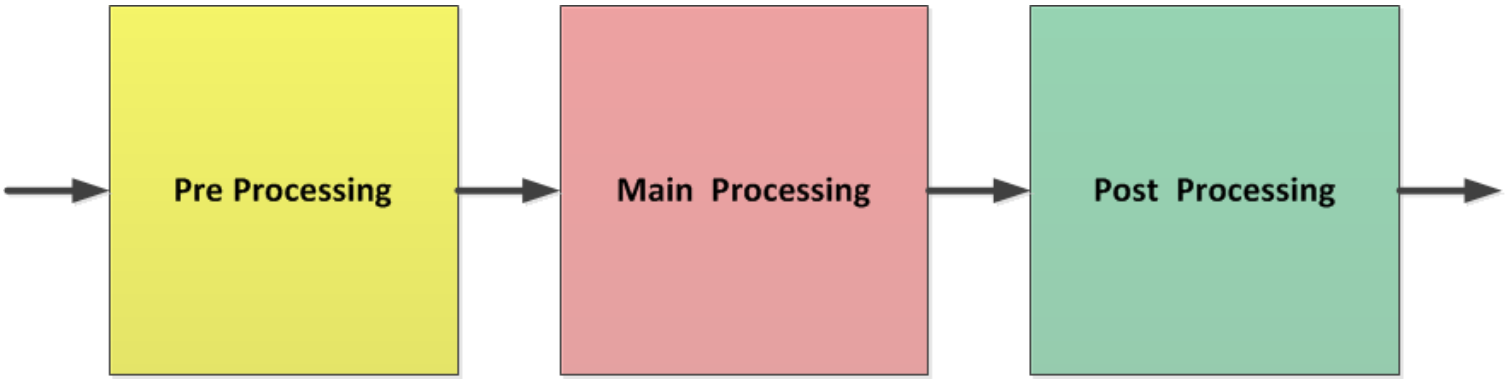
- Pre-processing can not work on frame (N) before frame (N-1) is done, but there is no dependency between macroblock, that is, multiple core can divide the input data for the preprocessing
- Main-processing can not work on frame (N) before frame (N-1) is done, and each macroblock depends on the macroblocks above and to left, that is, no way to use multiple cores on main processing
- Post processing must work on complete frame, but there is no dependency between consecutive frames, that is, post processing can process frame(N) before frame (N-1) is done



Video Encoder Processing Load

Coder	Width	Height	Frames/Second	MCycles/Second
D1(NTSC)	720	480	30	660
D1 (PAL)	720	576	25	660
720P30	1280	720	30	1850
1080i	1920	1080 (1088)	60 fields	3450

Module	Percentage	Approximate MIPS (1080i)/Second	Number of Cores
Pre=Processing	~50%	1750	2
Main Processing	~25%	875	1
Entropy Encoder	~25%	875	1



How Many Channels Can One C6678 Process?

- Looks like two channels;
  Each one uses four cores.
  - Two cores for pre-processing
  - One core for main-processing
  - One core for post-processing
- What other resources are needed?
  - Streaming data in and out of the system
  - Store and load data to and from DDR
  - Internal bus bandwidth
  - DMA availability
  - Synchronization between cores, especially if trying to minimize delay

What are the System Input Requirements?

- Stream data in and out of the system:
  - Raw data: 1920 * 1080 * 1.5 = 3,110,400 bytes per frame
    = 24.883200 bits per frame (~25M bits per frame)
  - At 30 frames per second, the input is 750 Mbps
  - NOTE: The order of raw data for a frame is Y component first, followed by U and V
- 750 Mbps input requires one of the following:
  - One SRIO lane (5 Gbps raw, about 3.5 Gbps of payload),
  - One PCIe lane (5 Gbps raw)
  - NOTE: KeyStone devices provide four SRIO lanes and two PCIe lanes
- Compressed data (e.g., 10 to 20 Mbps) can use SGMII (10M/100M/1G) or SRIO or PCIe.

How Many Accesses to the DDR?

- For purposes of this example, only consider frame-size accesses.
- All other accesses are negligible.
- Requirements for processing a single frame:
  - **Total DDR access for a single frame is less than 32 MB.**

How Does This Access Avoid Contention?

- **Total DDR access for a single frame is less than 32 MB.**
- The total DDR access for 30 frames per second (60 fields) is less than 32 * 30 = 960 MBps.
- The DDR3 raw bandwidth is more than 10 GBps (1333 MHz clock and 64 bits). 10% utilization reduces contention possibilities.
- DDR3 DMA uses TeraNet with clock/3 and 128 bits. TeraNet bandwidth is 400 MHz * 16B = 6.4 GBps.

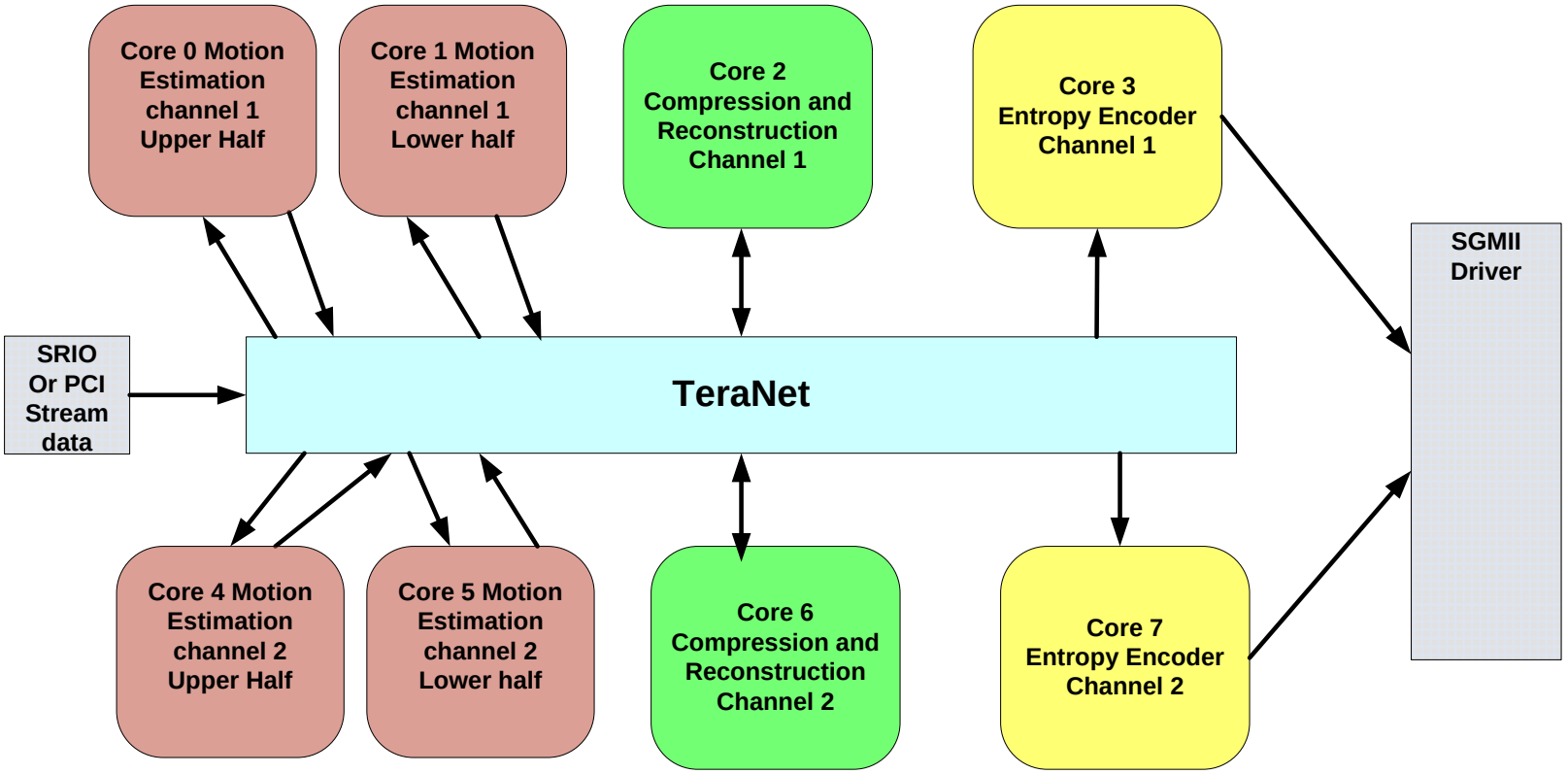
KeyStone SoC Architecture Resources

- 10 EDMA transfer controllers with 144 EDMA channels and 1152 PaRAM (parameter blocks)
  - The EDMA scheme must be designed by the user.
  - The LLD provides easy EDMA usage.
- In addition, Navigator has its own PKTDMA for each master.
- Data in and out of the system (SRIO or SGMII) is done using the Navigator or other master DMA (PCIe).
- All synchronization between cores and moving pointers to data between cores is done Using IPC

Conclusion

- Two H264 high-quality 1080i encoders can be processed on a single **TMS320C6678**

System Architecture



Example2:

Very large FFT (VLFFT) – 1M Floating point

Master/Slave Model
Data driven partition

Outlines

- Basic Algorithm for Parallelizing DFT
- Multi-core Implementation of DFT
- Review Benchmark Performance

Algorithm based on a paper:

Implement High-Performance Parallel FFT Algorithms for the HITACHI SR8000
Daisuke Takahashi
Information Technology Center, University of Tokyo 2-11-16 Yayoi, Bunkyo-ku, Tokyo 113-8658, Japan

TI VLFFT Software

- **Goal:**
  - To implement very large floating point fast DFT on TI multicore devices: The KeyStone Family
- **Requirements:**
  - FFT sizes: 4K – 1M samples
  - Configurable to run on different number of cores: 1, 2, 4, 8

Algorithm for Very Large DFT

- A generic discrete Fourier transform (DFT) is shown below,

$$y(n) = \sum_{n=0}^{N-1} x(n) e^{j\frac{2\pi}{N}k*n} \qquad k = 0, \ldots, N-1$$

- **Here *N* is the total size of DFT ,**

Develop The Algorithm step 1

Make a matrix of $N_{1(rows)} * N_{2(columns)} = N$ such that

$k = k_1 * N_1 + k_2$

$k_1 = 0, 1,..N_2-1$ $k_2 = 0, 1,..N_1-1$

It is easy to show that

$$Y(n) = \sum_{k_2=0}^{N_2-1} \sum_{k_1=0}^{N_1-1} x(k_1 N_2 + k_2) e^{((\frac{-2*\Pi}{(N1*N2)} * (k_1 N_2 + k_2) * n)}$$

Develop The Algorithm step 2

In a similar way, we can write that

$n = u_1 * N_1 + u_2$

$u_1 = 0, 1, ..N_2\text{-}1$ $u_2 = 0, 1, ..N_1\text{-}1$ and then

$$Y(u_1 * N_1 + u_2) = \sum_{k_2=0}^{N_2-1} \sum_{k_1=0}^{N_1-1} x(k_1 N_2 + k_2) e^{((\frac{-2*\Pi}{(N1*N2)} * (k_1 N_2 + k_2) * (u_1 * N_1 + u_2))}$$

Develop The Algorithm step 3

Next we observe that the exponent can be written as three terms. The forth term is always one ($e^{\frac{-2*\Pi}{}*k}$ =1)

$$e^{\frac{-2*\Pi}{N2}*(k_2)*(u_1)} \; e^{(\frac{-2*\Pi}{(N1*N2)}*(k_2)*(u_2))} \; e^{(\frac{-2*\Pi}{(N1)}*(k_1)*(u_2))}$$

$$Y(u_1 * N_1 + u_2) =$$

$$\sum_{k_2=0}^{N_2-1} \sum_{k_1=0}^{N_1-1} x(k_1 N_2 + k_2) \; e^{(\frac{-2*\Pi}{(N1)}*(k_1)*(u_2))} \; e^{(\frac{-2*\Pi}{(N1*N2)}*(k_2)*(u_2))} \; e^{\frac{-2*\Pi}{N2}*(k_2)*(u_1)}$$

Develop The Algorithm step 4

Look at the middle term, This is exactly FFT at the point u_2

For different K_2. Lets write it as $FFT_{K2}(u_2)$. There are N_2 different FFT, each of them is of size N_1.

$$Y(u_1 * N_1 + u_2) = \sum_{k_2=0}^{N_2-1} FFT_{K2}(U_2).\ e^{\left(\frac{-2*\Pi}{(N1*N2)}*(k_2)*(u_2)\right)}\ e^{\frac{-2*\Pi}{N2}*(k_2)*(u_1)}$$

Develop The Algorithm step 5

Look again at the middle term inside the sum, This is the FFT at the point u_2 for different K_2 multiply by a function (twiddle factor) of K_2 and u_2 . Lets write it as $Z_{u2}(k_2)$.

$$Y(u_1 * N_1 + u_2) = \sum_{k_2=0}^{N_2-1} Z_{u2}(k_2).\ e^{\frac{-2*\Pi}{N2}*(k_2)*(u_1)}$$

What we see is if we take from each FFT that we calculate before the u_2 element (after multiplying by the twiddle factor), we need to perform N_2 FFTs, each of them size N_1

Taking the u_2 element means transport the matrix, or “corner turn”

Algorithm for Very Large DFT

A vary large DFT of size *N=N1*N2* can be computed in the following steps:

1) Formulate input into N1xN2 matrix
2) Compute N2 FFTs size N1
3) Multiply Global twiddle factors
4) Matrix transpose: N2xN1 -> N1xN2
5) Compute N1 FFTs. Each is N2 size.

Implementing VLFFT on Multiple Cores

- Two iterations of computations
- 1st iteration
  - N2 FFTs are distributed across all the cores.
  - Each core implements matrix transpose and computes **N2/numCores FFTs and multiplying twiddle factor**.
- 2nd iteration
  - N1 FFTs of N2 size are distributed across all the cores
  - Each core computes **N1/numCores FFTs and** implements **matrix transpose before and after FFT computation**.

Data Buffers

- **DDR3: Three float complex arrays of size N**
  - Input buffer, output buffer, working buffer
- **L2 SRAM:**
  - Two ping-pong buffers, each buffer is the size of 16 FFT input/output
  - Some working buffer
  - Buffers for twiddle factors
    - Twiddle factors for N1 and N2 FFT
    - N2 global twiddle factors

Global Twiddle Factors

- **Global Twiddle Factors:**

$$e^{j\frac{2\pi}{N1*N2}k1*n2} \qquad n2 \in [0,\cdots,N2-1] \qquad k1 \in [0,\cdots,N1-1]$$

- **Total of N1*N2 global twiddle factors are required.**
- **N1 are actually pre-computed and saved.**

$$e^{j\frac{2\pi}{N1*N2}n2} \qquad n2 \in [0,\cdots,N2-1]$$

- **The rest are computed during run time.**

DMA Scheme

- Each core has dedicated in/out DMA channels
- Each core configures and triggers its own DMA channels for input/output
- On each core, the processing is divided into blocks of 8 FFT each.
- For each block on every core
  - DMA transfer 8 lines of FFT input
  - DSP computes FFT/transpose
  - DMA transfers 8 lines of FFT output

Matrix Transpose

- The transpose is required for the following matrixes from each core:
  - N1x8 -> 8xN1
  - N2x8 -> 8xN2
  - 8xN2 -> N2x8
- DSP computes matrix transpose from L2 SRAM
  - DMA bring samples from DDR to L2 SRAM
  - DSP implements transpose for matrixes in L2 SRAM
  - 32K L1 Cache

Major Kernels

- FFT: single precision floating point FFT from c66x DSPLIB
- Global twiddle factor compute and multiplication: 1 cycle per complex sample
- Transpose: 1 cycle per complex sample

Major Software Tools

- SYS BIOS 6
- CSL for EDMA configuration
- IPC for inter-processor communication

Conclusion

- After the demo …